

Analysis

December 12, 2021

Contents

1	Linear Algebra	25
1.1	L2 Norm	25
1.2	Inner Product Spaces and Gradient Derivative	28
1.2.1	Real inner product spaces	28
1.2.2	Class instances	33
1.2.3	Gradient derivative	36
1.3	Cartesian Products as Vector Spaces	37
1.3.1	Product is a Module	38
1.3.2	Product is a Real Vector Space	38
1.3.3	Product is a Metric Space	39
1.3.4	Product is a Complete Metric Space	47
1.3.5	Product is a Normed Vector Space	47
1.3.6	Product is Finite Dimensional	51
1.4	Finite-Dimensional Inner Product Spaces	53
1.4.1	Interlude: Some properties of real sets	53
1.4.2	Type class of Euclidean spaces	53
1.4.3	Subclass relationships	57
1.4.4	Class instances	58
1.4.5	Locale instances	60
1.5	Elementary Linear Algebra on Euclidean Spaces	62
1.5.1	More interesting properties of the norm	64
1.5.2	Substandard Basis	65
1.5.3	Orthogonality	66
1.5.4	Orthogonality of a transformation	68
1.5.5	Bilinear functions	69
1.5.6	Adjoint	70
1.5.7	Euclidean Spaces as Typeclass	72
1.5.8	Linearity and Bilinearity continued	72
1.5.9	We continue	76
1.5.10	Infinity norm	81
1.5.11	Collinearity	85
1.5.12	Properties of special hyperplanes	88
1.5.13	Orthogonal bases and Gram-Schmidt process	90

1.5.14	Decomposing a vector into parts in orthogonal subspaces	93
1.5.15	Linear functions are (uniformly) continuous on any set	100
1.5.16	Topological properties of linear functions	100
1.6	Affine Sets	101
1.6.1	Affine set and affine hull	103
1.6.2	Affine Dependence	115
1.6.3	Some Properties of Affine Dependent Sets	119
1.6.4	Affine Dimension of a Set	124
1.7	Convex Sets and Functions	136
1.7.1	Convex Sets	136
1.7.2	Explicit expressions for convexity in terms of arbitrary sums	140
1.7.3	Convex Functions on a Set	143
1.7.4	Arithmetic operations on sets preserve convexity	146
1.7.5	Convexity of real functions	155
1.7.6	Some inequalities	157
1.7.7	Misc related lemmas	159
1.7.8	Cones	159
1.7.9	Connectedness of convex sets	163
1.7.10	Convex hull	164
1.7.11	Relations among closure notions and corresponding hulls	174
1.7.12	Caratheodory's theorem	175
1.7.13	Some Properties of subset of standard basis	178
1.7.14	Moving and scaling convex hulls	178
1.7.15	Convexity of cone hulls	179
1.7.16	Radon's theorem	180
1.7.17	Helly's theorem	183
1.7.18	Epigraphs of convex functions	184
1.7.19	A bound within a convex hull	186
1.8	Definition of Finite Cartesian Product Type	187
1.8.1	Finite Cartesian products, with indexing and lambdas .	188
1.8.2	Cardinality of vectors	189
1.8.3	Group operations and class instances	191
1.8.4	Basic componentwise operations on vectors	192
1.8.5	Real vector space	193
1.8.6	Topological space	194
1.8.7	Metric space	196
1.8.8	Normed vector space	199
1.8.9	Inner product space	200
1.8.10	Euclidean space	201
1.8.11	A naive proof procedure to lift really trivial arithmetic stuff from the basis of the vector space	203
1.8.12	Some frequently useful arithmetic lemmas over vectors	204
1.8.13	Matrix operations	207

1.8.14	Inverse matrices (not necessarily square)	212
1.9	Linear Algebra on Finite Cartesian Products	213
1.9.1	Type $(\text{'a'}, \text{'n'})$ <i>vec</i> and fields as vector spaces	213
1.9.2	Rank of a matrix	224
1.9.3	Lemmas for working on $\text{real}^1/2/3/4$	228
1.9.4	The collapse of the general concepts to dimension one	229
1.9.5	Routine results connecting the types $(\text{real}, 1)$ <i>vec</i> and <i>real</i>	229
1.9.6	Explicit vector construction from lists	230
1.9.7	lambda skolemization on cartesian products	231
1.9.8	Explicit formulas for low dimensions	233
1.9.9	Orthogonality of a matrix	234
1.9.10	Finding an Orthogonal Matrix	235
1.9.11	Scaling and isometry	237
1.9.12	Induction on matrix row operations	239
1.10	Traces and Determinants of Square Matrices	244
1.10.1	Trace	244
1.10.2	Relation to invertibility	259
1.10.3	Cramer's rule	262
1.10.4	Rotation, reflection, rotoinversion	265
2	Topology	269
2.1	Elementary Topology	269
2.1.1	Topological Basis	270
2.1.2	Countable Basis	272
2.1.3	Polish spaces	281
2.1.4	Limit Points	281
2.1.5	Interior of a Set	285
2.1.6	Closure of a Set	289
2.1.7	Frontier (also known as boundary)	292
2.1.8	Filters and the "eventually true" quantifier	294
2.1.9	Limits	295
2.1.10	Compactness	299
2.1.11	Cartesian products	313
2.1.12	Continuity	316
2.1.13	Homeomorphisms	319
2.1.14	On Linorder Topologies	323
2.2	Operators involving abstract topology	325
2.2.1	General notion of a topology as a value	325
2.2.2	The discrete topology	329
2.2.3	Subspace topology	330
2.2.4	The canonical topology from the underlying type class	334
2.2.5	Basic "localization" results are handy for connectedness.	334

2.2.6	Derived set (set of limit points)	337
2.2.7	Closure with respect to a topological space	340
2.2.8	Frontier with respect to topological space	349
2.2.9	Locally finite collections	353
2.2.10	Continuous maps	356
2.2.11	Open and closed maps (not a priori assumed continuous)	363
2.2.12	Quotient maps	367
2.2.13	Separated Sets	374
2.2.14	Homeomorphisms	376
2.2.15	Relation of homeomorphism between topological spaces	385
2.2.16	Connected topological spaces	386
2.2.17	Compact sets	396
2.2.18	Embedding maps	405
2.2.19	Retraction and section maps	407
2.2.20	Continuity	409
2.2.21	Half-global and completely global cases	410
2.2.22	The topology generated by some (open) subsets	414
2.2.23	Topology bases and sub-bases	416
2.2.24	Pullback topology	420
2.2.25	Proper maps (not a priori assumed continuous)	421
2.2.26	Perfect maps (proper, continuous and surjective)	426
2.3	Abstract Topology 2	427
2.3.1	Closure	429
2.3.2	Frontier	430
2.3.3	Compactness	431
2.3.4	Continuity	432
2.3.5	Equality of continuous functions on closure and related results	432
2.3.6	A function constant on a set	433
2.3.7	Continuity relative to a union.	433
2.3.8	Inverse function property for open/closed maps	434
2.3.9	Seperability	436
2.3.10	Closed Maps	437
2.3.11	Open Maps	438
2.3.12	Quotient maps	438
2.3.13	Pasting lemmas for functions, for of casewise definitions	441
2.3.14	Retractions	445
2.3.15	Retractions on a topological space	450
2.3.16	Paths and path-connectedness	452
2.3.17	Connected components	455
2.4	Connected Components	461
2.4.1	Connectedness	461
2.4.2	Connected components, considered as a connectedness relation or a set	462

2.4.3	The set of connected components of a set	467
2.4.4	Proving a function is constant on a connected set by proving that a level set is open	471
2.4.5	Preservation of Connectedness	471
2.4.6	Lemmas about components	474
2.4.7	Constancy of a function from a connected set into a finite, disconnected or discrete set	476
2.4.8	nhdsin and atin	477
2.4.9	Limits in a topological space	478
2.4.10	Pointwise continuity in topological spaces	481
2.4.11	Combining theorems for continuous functions into the reals	482
3	Functional Analysis	485
3.1	A decision procedure for metric spaces	485
3.2	Elementary Metric Spaces	487
3.2.1	Open and closed balls	487
3.2.2	Limit Points	493
3.2.3	Perfect Metric Spaces	494
3.2.4	?	494
3.2.5	Interior	495
3.2.6	Frontier	495
3.2.7	Limits	496
3.2.8	Continuity	498
3.2.9	Closure and Limit Characterization	501
3.2.10	Boundedness	503
3.2.11	Compactness	505
3.2.12	Banach fixed point theorem	508
3.2.13	Edelstein fixed point theorem	512
3.2.14	The diameter of a set	512
3.2.15	Metric spaces with the Heine-Borel property	517
3.2.16	Completeness	520
3.2.17	Finite intersection property	526
3.2.18	Properties of Balls and Spheres	527
3.2.19	Distance from a Set	527
3.2.20	Infimum Distance	528
3.2.21	Separation between Points and Sets	533
3.2.22	Uniform Continuity	535
3.2.23	Continuity on a Compact Domain Implies Uniform Con- tinuity	537
3.2.24	Theorems relating continuity and uniform continuity to closures	539

3.2.25	With Abstract Topology (TODO: move and remove dependency?)	544
3.2.26	Closed Nest	545
3.2.27	Making a continuous function avoid some value in a neighbourhood	547
3.2.28	Consequences for Real Numbers	548
3.2.29	The infimum of the distance between two sets	551
3.3	Elementary Normed Vector Spaces	555
3.3.1	Orthogonal Transformation of Balls	556
3.3.2	Various Lemmas Combining Imports	556
3.3.3	Support	557
3.3.4	Intervals	558
3.3.5	Limit Points	559
3.3.6	Balls and Spheres in Normed Spaces	561
3.3.7	Various Lemmas on Normed Algebras	564
3.3.8	Filters	565
3.3.9	Trivial Limits	565
3.3.10	Limits	565
3.3.11	Limit Point of Filter	568
3.3.12	Boundedness	568
3.3.13	Relations among convergence and absolute convergence for power series	572
3.3.14	Normed spaces with the Heine-Borel property	572
3.3.15	Intersecting chains of compact sets and the Baire property	573
3.3.16	Continuity	577
3.3.17	Arithmetic Preserves Topological Properties	579
3.3.18	Homeomorphisms	586
3.3.19	Discrete	588
3.3.20	Completeness of "Isometry" (up to constant bounds)	589
3.3.21	Connected Normed Spaces	590
3.4	Linear Decision Procedure for Normed Spaces	593
4	Vector Analysis	597
4.1	Elementary Topology in Euclidean Space	597
4.1.1	Continuity of the representation WRT an orthogonal basis	597
4.1.2	Balls in Euclidean Space	599
4.1.3	Boxes	604
4.1.4	General Intervals	617
4.1.5	Bounded Projections	621
4.1.6	Structural rules for pointwise continuity	621
4.1.7	Structural rules for setwise continuity	622
4.1.8	Openness of halfspaces.	622
4.1.9	Closure and Interior of halfspaces and hyperplanes	622

4.1.10	Some more convenient intermediate-value theorem formulations	625
4.1.11	Limit Component Bounds	625
4.1.12	Class Instances	631
4.1.13	Compact Boxes	633
4.1.14	Componentwise limits and continuity	635
4.1.15	Continuous Extension	637
4.1.16	Separability	639
4.1.17	Diameter	641
4.1.18	Relating linear images to open/closed/interior/closure/connected	642
4.1.19	"Isometry" (up to constant bounds) of Injective Linear Map	645
4.1.20	Some properties of a canonical subspace	648
4.1.21	Set Distance	650
4.2	Convex Sets and Functions on (Normed) Euclidean Spaces	652
4.2.1	Topological Properties of Convex Sets and Functions	652
4.2.2	Relative interior of a set	653
4.2.3	Openness and compactness are preserved by convex hull operation	667
4.2.4	Extremal points of a simplex are some vertices	672
4.2.5	Closest point of a convex set is unique, with a continuous projection	676
4.2.6	More convexity generalities	684
4.2.7	Convex set as intersection of halfspaces	685
4.2.8	Convexity of general and special intervals	685
4.2.9	On <i>real</i> , <i>is_interval</i> , <i>convex</i> and <i>connected</i> are all equivalent	687
4.2.10	Another intermediate value theorem formulation	688
4.2.11	A bound within an interval	689
4.2.12	Representation of any interval as a finite convex hull	691
4.2.13	Bounded convex function on open set is continuous	693
4.2.14	Upper bound on a ball implies upper and lower bounds	695
4.3	Operator Norm	697
4.4	Line Segment	702
4.4.1	Topological Properties of Convex Sets, Metric Spaces and Functions	702
4.4.2	Midpoint	706
4.4.3	Open and closed segments	707
4.4.4	Betweenness	724
4.5	Limits on the Extended Real Number Line	727
4.5.1	Extended-Real.thy	735
4.5.2	Extended-Nonnegative-Real.thy	748
4.5.3	monoset	749

4.5.4	Relate extended reals and the indicator function	767
4.6	Radius of Convergence and Summation Tests	768
4.6.1	Convergence tests for infinite sums	768
4.6.2	Radius of convergence	778
4.7	Uniform Limit and Uniform Convergence	788
4.7.1	Definition	788
4.7.2	Exchange limits	789
4.7.3	Uniform limit theorem	790
4.7.4	Weierstrass M-Test	794
4.7.5	Structural introduction rules	796
4.7.6	Power series and uniform convergence	802
4.8	Function Topology	804
4.8.1	The product topology	805
4.8.2	The Alexander subbase theorem	823
4.8.3	Open Pi-sets in the product topology	831
4.8.4	Relationship with connected spaces, paths, etc.	834
4.8.5	Projections from a function topology to a component	840
4.9	Bounded Linear Function	841
4.9.1	Intro rules for <i>bounded_linear</i>	841
4.9.2	declaration of derivative/continuous/tendsto introduction rules for bounded linear functions	842
4.9.3	Type of bounded linear functions	843
4.9.4	Type class instantiations	843
4.9.5	On Euclidean Space	848
4.9.6	concrete bounded linear functions	853
4.9.7	The strong operator topology on continuous linear operators	857
4.10	Derivative	859
4.10.1	Derivatives	859
4.10.2	Derivative with composed bilinear function	859
4.10.3	Differentiability	861
4.10.4	Frechet derivative and Jacobian matrix	863
4.10.5	Differentiability implies continuity	864
4.10.6	The chain rule	865
4.10.7	Composition rules stated just for differentiability	866
4.10.8	Uniqueness of derivative	866
4.10.9	Derivatives of local minima and maxima are zero	869
4.10.10	One-dimensional mean value theorem	871
4.10.11	More general bound theorems	873
4.10.12	Differentiability of inverse function (most basic form)	881
4.10.13	Uniformly convergent sequence of derivatives	887
4.10.14	Differentiation of a series	893
4.10.15	Derivative as a vector	896
4.10.16	Field differentiability	902

4.10.17	Field derivative	907
4.10.18	Relation between convexity and derivative	910
4.10.19	Partial derivatives	912
4.10.20	Differentiable case distinction	916
4.10.21	The Inverse Function Theorem	917
4.10.22	Piecewise differentiable functions	925
4.10.23	The concept of continuously differentiable	928
4.11	Finite Cartesian Products of Euclidean Spaces	935
4.11.1	Closures and interiors of halfspaces	936
4.11.2	Bounds on components etc. relative to operator norm	937
4.11.3	Convex Euclidean Space	945
4.11.4	Derivative	945
4.11.5	Routine results connecting the types (<i>real</i> , 1) <i>vec</i> and <i>real</i>	946
5	Unsorted	949
5.0.1	Shrinking towards the interior of a convex set	950
5.0.2	Some obvious but surprisingly hard simplex lemmas	954
5.0.3	Relative interior of convex set	962
5.0.4	The relative frontier of a set	969
5.0.5	Convexity on direct sums	999
5.0.6	Explicit formulas for interior and relative interior of convex hull	1006
5.0.7	Similar results for closure and (relative or absolute) frontier	1013
5.0.8	Coplanarity, and collinearity in terms of affine hull	1017
5.0.9	Basic lemmas about hyperplanes and halfspaces	1027
5.0.10	Use set distance for an easy proof of separation properties	1028
5.0.11	Connectedness of the intersection of a chain	1030
5.0.12	Proper maps, including projections out of compact sets	1033
5.0.13	Trivial fact: convexity equals connectedness for collinear sets	1035
5.0.14	Some stepping theorems	1044
5.0.15	General case without assuming closure and getting non-strict separation	1047
5.0.16	Some results on decomposing convex hulls: intersections, simplicial subdivision	1051
5.0.17	Lower-dimensional affine subsets are nowhere dense	1063
5.0.18	Parallel slices, etc	1065
5.0.19	Paracompactness	1069
5.0.20	Closed-graph characterization of continuity	1072
5.0.21	The union of two collinear segments is another segment	1074

5.0.22	Covering an open set by a countable chain of compact sets	1078
5.0.23	Orthogonal complement	1080
5.0.24	A non-injective linear function maps into a hyperplane.	1083
5.1	The binary product topology	1084
5.2	Product Topology	1085
5.2.1	Definition	1085
5.2.2	Continuity	1088
5.2.3	Homeomorphic maps	1095
5.3	T1 and Hausdorff spaces	1098
5.4	T1 spaces with equivalences to many naturally "nice" properties.	1098
5.4.1	Hausdorff Spaces	1103
5.5	Path-Connectedness	1114
5.5.1	Paths and Arcs	1114
5.5.2	Invariance theorems	1114
5.5.3	Basic lemmas about paths	1116
5.5.4	Path Images	1120
5.5.5	Simple paths with the endpoints removed	1123
5.5.6	The operations on paths	1123
5.5.7	Some reversed and "if and only if" versions of joining theorems	1126
5.5.8	The joining of paths is associative	1128
5.5.9	Subpath	1130
5.5.10	There is a subpath to the frontier	1134
5.5.11	Shift Path to Start at Some Given Point	1137
5.5.12	Straight-Line Paths	1139
5.5.13	Segments via convex hulls	1142
5.5.14	Bounding a point away from a path	1145
5.5.15	Path component	1146
5.5.16	Path connectedness of a space	1147
5.5.17	Lemmas about path-connectedness	1154
5.5.18	Path components	1156
5.5.19	Path components	1157
5.5.20	Sphere is path-connected	1167
5.5.21	Every annulus is a connected set	1174
5.5.22	Relations between components and path components	1175
5.5.23	Existence of unbounded components	1178
5.5.24	The <i>inside</i> and <i>outside</i> of a Set	1180
5.5.25	Condition for an open map's image to contain a ball	1197
5.5.26	Rectangular paths	1203
5.6	Bernstein-Weierstrass and Stone-Weierstrass	1204
5.6.1	Bernstein polynomials	1204
5.6.2	Explicit Bernstein version of the 1D Weierstrass approximation theorem	1205

5.6.3	General Stone-Weierstrass theorem	1207
5.6.4	Polynomial functions	1220
5.6.5	Stone-Weierstrass theorem for polynomial functions . .	1225
5.6.6	Polynomial functions as paths	1229

6 Measure and Integration Theory 1235

6.1	Sigma Algebra	1235
6.1.1	Families of sets	1235
6.1.2	Measure type	1265
6.1.3	The smallest σ -algebra regarding a function	1278
6.2	Measurability Prover	1284
6.2.1	Measurability for (co)inductive predicates	1292
6.3	Measure Spaces	1298
6.3.1	Relate extended reals and the indicator function	1299
6.3.2	Extend binary sets	1299
6.3.3	Properties of a premeasure μ	1300
6.3.4	Properties of <i>emeasure</i>	1308
6.3.5	μ -null sets	1318
6.3.6	The almost everywhere filter (i.e. quantifier)	1320
6.3.7	σ -finite Measures	1326
6.3.8	Measure space induced by distribution of $(\rightarrow_M)$ -functions	1329
6.3.9	Real measure values	1332
6.3.10	Set of measurable sets with finite measure	1336
6.3.11	Measurable sets formed by unions and intersections . .	1337
6.3.12	Measure spaces with <i>emeasure</i> M (<i>space</i> M) $< \infty$. . .	1344
6.3.13	Counting space	1349
6.3.14	Measure restricted to space	1353
6.3.15	Null measure	1356
6.3.16	Scaling a measure	1357
6.3.17	Complete lattice structure on measures	1358
6.4	Ordered Euclidean Space	1384
6.5	Borel Space	1390
6.5.1	Generic Borel spaces	1394
6.5.2	Borel spaces on order topologies	1402
6.5.3	Borel spaces on topological monoids	1409
6.5.4	Borel spaces on Euclidean spaces	1410
6.5.5	Borel measurable operators	1417
6.5.6	Borel space on the extended reals	1421
6.5.7	Borel space on the extended non-negative reals	1425
6.5.8	LIMSEQ is borel measurable	1427
6.6	Lebesgue Integration for Nonnegative Functions	1436
6.6.1	Approximating functions	1436
6.6.2	Simple function	1438

6.6.3	Simple integral	1448
6.6.4	Integral on nonnegative functions	1454
6.6.5	Integral under concrete measures	1475
6.7	Binary Product Measure	1495
6.7.1	Binary products	1495
6.7.2	Binary products of σ -finite emeasure spaces	1502
6.7.3	Fubini's theorem	1507
6.7.4	Products on counting spaces, densities and distributions	1509
6.7.5	Product of Borel spaces	1520
6.8	Finite Product Measure	1521
6.8.1	More about Function restricted by <i>extensional</i>	1522
6.8.2	Finite product spaces	1524
6.8.3	Measurability	1549
6.9	Caratheodory Extension Theorem	1552
6.9.1	Characterizations of Measures	1553
6.9.2	Caratheodory's theorem	1562
6.9.3	Volumes	1563
6.10	Bochner Integration for Vector-Valued Functions	1572
6.10.1	Restricted measure spaces	1622
6.10.2	Measure spaces with an associated density	1623
6.10.3	Distributions	1625
6.10.4	Lebesgue integration on <i>count_space</i>	1627
6.10.5	Point measure	1629
6.10.6	Lebesgue integration on <i>null_measure</i>	1629
6.10.7	Legacy lemmas for the real-valued Lebesgue integral	1630
6.10.8	Product measure	1636
6.11	Complete Measures	1648
6.12	Regularity of Measures	1674
6.13	Lebesgue Measure	1683
6.13.1	Measures defined by monotonous functions	1684
6.13.2	Lebesgue-Borel measure	1691
6.13.3	Borel measurability	1694
6.13.4	Measurability of continuous functions	1698
6.13.5	Affine transformation on the Lebesgue-Borel	1705
6.13.6	Lebesgue measurable sets	1713
6.13.7	Translation preserves Lebesgue measure	1714
6.13.8	A nice lemma for negligibility proofs	1716
6.13.9	<i>F_sigma</i> and <i>G_delta</i> sets.	1721
6.14	Tagged Divisions for Henstock-Kurzweil Integration	1724
6.14.1	Sundries	1724
6.14.2	Some useful lemmas about intervals	1725
6.14.3	Bounds on intervals where they exist	1726
6.14.4	The notion of a gauge — simply an open set containing the point	1728

6.14.5	Attempt a systematic general set of "offset" results for components	1729
6.14.6	Divisions	1729
6.14.7	Tagged (partial) divisions	1745
6.14.8	Functions closed on boxes: morphisms from boxes to monoids	1751
6.14.9	Special case of additivity we need for the FTC	1763
6.14.10	Fine-ness of a partition w.r.t. a gauge	1763
6.14.11	Some basic combining lemmas	1764
6.14.12	The set we're concerned with must be closed	1764
6.14.13	General bisection principle for intervals; might be useful elsewhere	1764
6.14.14	Cousin's lemma	1770
6.14.15	A technical lemma about "refinement" of division	1771
6.14.16	Division filter	1779
6.15	Henstock-Kurzweil Gauge Integration in Many Dimensions . .	1780
6.15.1	Content (length, area, volume...) of an interval	1780
6.15.2	Gauge integral	1786
6.15.3	Basic theorems about integrals	1787
6.15.4	Cauchy-type criterion for integrability	1801
6.15.5	Additivity of integral on abutting intervals	1803
6.15.6	A sort of converse, integrability on subintervals	1808
6.15.7	Bounds on the norm of Riemann sums and the integral itself	1813
6.15.8	Similar theorems about relationship among components . . .	1815
6.15.9	Uniform limit of integrable functions is integrable	1819
6.15.10	Negligible sets	1822
6.15.11	Some other trivialities about negligible sets	1830
6.15.12	Finite case of the spike theorem is quite commonly needed	1832
6.15.13	In particular, the boundary of an interval is negligible . .	1833
6.15.14	Integrability of continuous functions	1833
6.15.15	Specialization of additivity to one dimension	1836
6.15.16	A useful lemma allowing us to factor out the content size	1836
6.15.17	Fundamental theorem of calculus	1837
6.15.18	Taylor series expansion	1841
6.15.19	Only need trivial subintervals if the interval itself is trivial	1844
6.15.20	Integrability on subintervals	1846
6.15.21	Combining adjacent intervals in 1 dimension	1847
6.15.22	Reduce integrability to "local" integrability	1848
6.15.23	Second FTC or existence of antiderivative	1849
6.15.24	Combined fundamental theorem of calculus	1851
6.15.25	General "twiddling" for interval-to-interval function image	1852
6.15.26	Special case of a basic affine transformation	1854
6.15.27	Special case of stretching coordinate axes separately . . .	1858

6.15.28	even more special cases	1859
6.15.29	Stronger form of FCT; quite a tedious proof	1860
6.15.30	Stronger form with finite number of exceptional points	1869
6.15.31	This doesn't directly involve integration, but that gives an easy proof	1875
6.15.32	Generalize a bit to any convex set	1876
6.15.33	Integrating characteristic function of an interval	1879
6.15.34	Integrals on set differences	1885
6.15.35	More lemmas that are useful later	1888
6.15.36	Continuity of the integral (for a 1-dimensional interval)	1890
6.15.37	A straddling criterion for integrability	1894
6.15.38	Adding integrals over several sets	1898
6.15.39	Also tagged divisions	1901
6.15.40	Henstock's lemma	1902
6.15.41	Monotone convergence (bounded interval first)	1907
6.15.42	differentiation under the integral sign	1921
6.15.43	Exchange uniform limit and integral	1926
6.15.44	Integration by parts	1928
6.15.45	Integration by substitution	1929
6.15.46	Compute a double integral using iterated integrals and switching the order of integration	1931
6.15.47	Definite integrals for exponential and power function	1939
6.16	Radon-Nikodým Derivative	1946
6.16.1	Absolutely continuous	1949
6.16.2	Existence of the Radon-Nikodym derivative	1950
6.16.3	Uniqueness of densities	1959
6.16.4	Radon-Nikodym derivative	1966
6.17	Non-Denumerability of the Continuum	1996
6.17.1	Abstract	1997
6.18	Homotopy of Maps	2000
6.18.1	Trivial properties	2001
6.18.2	Homotopy with P is an equivalence relation	2003
6.18.3	Continuity lemmas	2005
6.18.4	Homotopy of paths, maintaining the same endpoints	2011
6.18.5	Group properties for homotopy of paths	2015
6.18.6	Homotopy of loops without requiring preservation of endpoints	2016
6.18.7	Relations between the two variants of homotopy	2018
6.18.8	Homotopy of "nearby" function, paths and loops	2022
6.18.9	Homotopy and subpaths	2024
6.18.10	Simply connected sets	2027
6.18.11	Contractible sets	2031
6.18.12	Starlike sets	2034
6.18.13	Local versions of topological properties in general	2037

6.18.14	An induction principle for connected sets	2042
6.18.15	Basic properties of local compactness	2044
6.18.16	Sura-Bura's results about compact components of sets	2052
6.18.17	Special cases of local connectedness and path connect- edness	2057
6.18.18	Relations between components and path components	2063
6.18.19	Components, continuity, openin, closedin	2067
6.18.20	Existence of isometry between subspaces of same dimen- sion	2069
6.18.21	Retracts, in a general sense, preserve (co)homotopic triviality)	2074
6.18.22	Homotopy equivalence	2077
6.18.23	Homotopy equivalence of topological spaces.	2077
6.18.24	Contractible spaces	2080
6.18.25	Misc other results	2091
6.18.26	Some Uncountable Sets	2092
6.18.27	Some simple positive connection theorems	2095
6.18.28	Self-homeomorphisms shuffling points about	2101
6.18.29	Nullhomotopic mappings	2120
6.19	Homeomorphism Theorems	2125
6.19.1	Homeomorphism of all convex compact sets with nonempty interior	2125
6.19.2	Homeomorphisms between punctured spheres and affine sets	2135
6.19.3	Locally compact sets in an open set	2141
6.19.4	Covering spaces and lifting results for them	2146
6.19.5	Lifting of general functions to covering space	2162
6.19.6	Homeomorphisms of arc images	2173
6.19.7	Equivalence Lebesgue integral on <i>lborel</i> and HK-integral	2181
6.19.8	Absolute integrability (this is the same as Lebesgue in- tegrability)	2193
6.19.9	Applications to Negligibility	2201
6.19.10	Negligibility of image under non-injective linear map	2208
6.19.11	Negligibility of a Lipschitz image of a negligible set	2210
6.19.12	Measurability of countable unions and intersections of various kinds.	2218
6.19.13	Negligibility is a local property	2221
6.19.14	Integral bounds	2222
6.19.15	Outer and inner approximation of measurable sets by well-behaved sets.	2234
6.19.16	Transformation of measure by linear maps	2237
6.19.17	Lemmas about absolute integrability	2244
6.19.18	Componentwise	2246
6.19.19	Dominated convergence	2255

6.19.20	Fundamental Theorem of Calculus for the Lebesgue integral	2258
6.19.21	Integration by parts	2261
6.19.22	Various common equivalent forms of function measurability	2265
6.19.23	Lebesgue sets and continuous images	2269
6.19.24	Affine lemmas	2272
6.19.25	More results on integrability	2274
6.19.26	Relation between Borel measurability and integrability.	2277
6.20	Complex Analysis Basics	2285
6.20.1	General lemmas	2285
6.20.2	Holomorphic functions	2288
6.20.3	Analyticity on a set	2293
6.20.4	Analyticity at a point	2297
6.20.5	Combining theorems for derivative with “analytic at” hypotheses	2298
6.20.6	Complex differentiation of sequences and series	2299
6.20.7	Taylor on Complex Numbers	2300
6.21	Complex Transcendental Functions	2305
6.21.1	Möbius transformations	2305
6.21.2	The Exponential Function	2306
6.21.3	Euler and de Moivre formulas	2306
6.21.4	Relationships between real and complex trigonometric and hyperbolic functions	2308
6.21.5	More on the Polar Representation of Complex Numbers	2309
6.21.6	Taylor series for complex exponential, sine and cosine	2320
6.21.7	The argument of a complex number (HOL Light version)	2324
6.21.8	Analytic properties of tangent function	2330
6.21.9	The principal branch of the Complex logarithm	2330
6.21.10	Relation to Real Logarithm	2331
6.21.11	Derivative of Ln away from the branch cut	2333
6.21.12	Quadrant-type results for Ln	2338
6.21.13	More Properties of Ln	2340
6.21.14	The Argument of a Complex Number	2344
6.21.15	The Unwinding Number and the Ln product Formula	2349
6.21.16	Relation between Ln and Arg2pi, and hence continuity of Arg2pi	2350
6.21.17	Complex Powers	2354
6.21.18	Some Limits involving Logarithms	2362
6.21.19	Relation between Square Root and exp/Ln, hence its derivative	2365
6.21.20	Complex arctangent	2368
6.21.21	Real arctangent	2373
6.21.22	Bounds on pi using real arctangent	2377

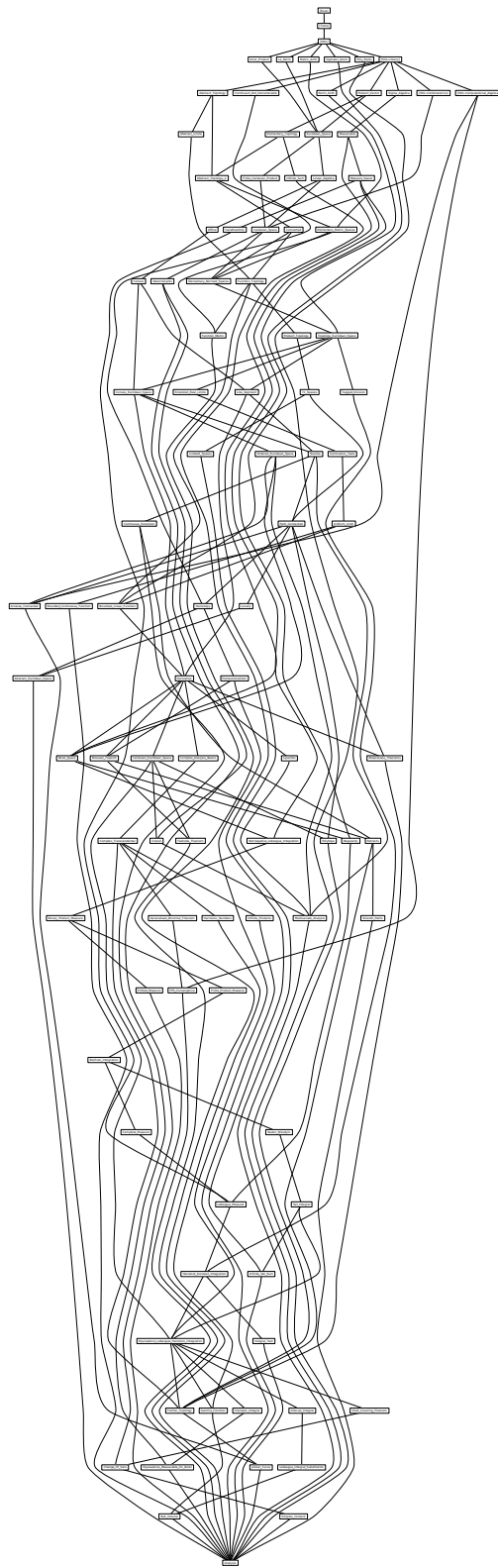
6.21.23	Inverse Sine	2377
6.21.24	Inverse Cosine	2381
6.21.25	Upper and Lower Bounds for Inverse Sine and Cosine	2384
6.21.26	Interrelations between Arcsin and Arccos	2385
6.21.27	Relationship with Arcsin on the Real Numbers	2387
6.21.28	Relationship with Arccos on the Real Numbers	2388
6.21.29	Some interrelationships among the real inverse trig functions	2389
6.21.30	Continuity results for arcsin and arccos	2390
6.21.31	Roots of unity	2392
6.22	Harmonic Numbers	2394
6.22.1	The Harmonic numbers	2394
6.22.2	The Euler-Mascheroni constant	2396
6.22.3	Bounds on the Euler-Mascheroni constant	2400
6.23	The Gamma Function	2406
6.23.1	The Euler form and the logarithmic Gamma function	2411
6.23.2	The Polygamma functions	2420
6.23.3	Basic properties	2431
6.23.4	Differentiability	2434
6.23.5	The complex Gamma function	2436
6.23.6	The real Gamma function	2442
6.23.7	The uniqueness of the real Gamma function	2449
6.23.8	The Beta function	2452
6.23.9	Legendre duplication theorem	2454
6.23.10	Limits and residues	2467
6.23.11	Alternative definitions	2468
6.23.12	The Weierstraß product formula for the sine	2485
6.23.13	The Solution to the Basel problem	2486
6.23.14	Approximating a (possibly infinite) interval	2489
6.23.15	Basic properties of integration over an interval	2492
6.23.16	Basic properties of integration over an interval wrt lebesgue measure	2495
6.23.17	General limit approximation arguments	2499
6.23.18	A slightly stronger Fundamental Theorem of Calculus	2502
6.23.19	The substitution theorem	2506
6.24	Integration by Substitution for the Lebesgue Integral	2514
6.25	The Volume of an n -Dimensional Ball	2523
6.26	Integral Test for Summability	2530
6.27	Continuity of the indefinite integral; improper integral theorem	2533
6.27.1	Equiintegrability	2533
6.27.2	Subinterval restrictions for equiintegrable families	2542
6.27.3	Continuity of the indefinite integral	2568
6.27.4	Second mean value theorem and corollaries	2576
6.28	Continuous Extensions of Functions	2585

6.28.1	Partitions of unity subordinate to locally finite open coverings	2585
6.28.2	Urysohn's Lemma for Euclidean Spaces	2587
6.28.3	Dugundji's Extension Theorem and Tietze Variants	2591
6.29	Equivalence Between Classical Borel Measurability and HOL Light's	2596
6.29.1	Austin's Lemma	2597
6.29.2	A differentiability-like property of the indefinite integral	2600
6.29.3	HOL Light measurability	2607
6.29.4	Composing continuous and measurable functions; a few variants	2612
6.29.5	Measurability on generalisations of the binary product	2631
6.30	Embedding Measure Spaces with a Function	2635
6.31	Brouwer's Fixed Point Theorem	2644
6.31.1	Retractions	2644
6.31.2	Kuhn Simplices	2648
6.31.3	Brouwer's fixed point theorem	2677
6.31.4	Applications	2684
6.32	Fashoda Meet Theorem	2698
6.32.1	Bijections between intervals	2698
6.32.2	Fashoda meet theorem	2700
6.32.3	Some slightly ad hoc lemmas I use below	2708
6.32.4	Useful Fashoda corollary pointed out to me by Tom Hales	2712
6.33	Vector Cross Products in 3 Dimensions	2715
6.33.1	Basic lemmas	2716
6.33.2	Preservation by rotation, or other orthogonal transformation up to sign	2719
6.33.3	Continuity	2720
6.34	Bounded Continuous Functions	2720
6.34.1	Definition	2720
6.34.2	Complete Space	2724
6.34.3	Supremum norm for a normed vector space	2724
6.34.4	(bounded) continuous extension	2726
6.35	Lindelöf spaces	2727
6.36	Infinite Products	2733
6.36.1	Preliminaries	2733
6.36.2	Definitions and basic properties	2734
6.36.3	Absolutely convergent products	2736
6.36.4	Ignoring initial segments	2741
6.36.5	More elementary properties	2743
6.36.6	Infinite products on ordered topological monoids	2752
6.36.7	Infinite products on topological spaces	2756
6.36.8	Infinite summability on real normed fields	2758
6.36.9	Exponentials and logarithms	2763

6.36.10	Embeddings from the reals into some complete real normed field	2772
6.37	Infinite sums	2772
6.37.1	Definition and syntax	2773
6.37.2	General properties	2774
6.37.3	Absolute convergence	2817
6.37.4	Extended reals and nats	2821
6.37.5	Real numbers	2824
6.37.6	Complex numbers	2825
6.38	Sums over Infinite Sets	2829
6.39	Faces, Extreme Points, Polytopes, Polyhedra etc	2859
6.39.1	Faces of a (usually convex) set	2859
6.39.2	Exposed faces	2873
6.39.3	Extreme points of a set: its singleton faces	2878
6.39.4	Facets	2880
6.39.5	Edges: faces of affine dimension 1	2881
6.39.6	Existence of extreme points	2881
6.39.7	Krein-Milman, the weaker form	2882
6.39.8	Applying it to convex hulls of explicitly indicated finite sets	2885
6.39.9	Polytopes	2893
6.39.10	Polyhedra	2895
6.39.11	Canonical polyhedron representation making facial structure explicit	2897
6.39.12	More general corollaries from the explicit representation	2909
6.39.13	Relation between polytopes and polyhedra	2915
6.39.14	Relative and absolute frontier of a polytope	2917
6.39.15	Special case of a triangle	2918
6.39.16	Subdividing a cell complex	2919
6.39.17	Simplexes	2924
6.39.18	Simplicial complexes and triangulations	2926
6.39.19	Refining a cell complex to a simplicial complex	2926
6.39.20	Some results on cell division with full-dimensional cells only	2941
6.40	Arcwise-Connected Sets	2944
6.40.1	The Brouwer reduction theorem	2944
6.40.2	Arcwise Connections	2946
6.40.3	Density of points with dyadic rational coordinates	2946
6.40.4	Accessibility of frontier points	2990
6.41	Absolute Retracts, Absolute Neighbourhood Retracts and Euclidean Neighbourhood Retracts	2994
6.41.1	Analogous properties of ENRs	3002
6.41.2	More advanced properties of ANRs and ENRs	3021
6.41.3	Original ANR material, now for ENRs	3026

6.41.4	Finally, spheres are ANRs and ENRs	3029
6.41.5	Spheres are connected, etc	3030
6.41.6	Borsuk homotopy extension theorem	3031
6.41.7	More extension theorems	3038
6.41.8	The complement of a set and path-connectedness . . .	3046
6.42	Extending Continous Maps, Invariance of Domain, etc	3049
6.42.1	A map from a sphere to a higher dimensional sphere is nullhomotopic	3049
6.42.2	Some technical lemmas about extending maps from cell complexes	3057
6.42.3	Special cases and corollaries involving spheres	3070
6.42.4	Extending maps to spheres	3074
6.42.5	Invariance of domain and corollaries	3091
6.42.6	Formulation of loop homotopy in terms of maps out of type complex	3109
6.42.7	Homeomorphism of simple closed curves to circles . . .	3113
6.42.8	Dimension-based conditions for various homeomorphisms	3114
6.42.9	more invariance of domain	3116
6.42.10	The power, squaring and exponential functions as cov- ering maps	3123
6.42.11	Hence the Borsukian results about mappings into circles	3131
6.42.12	Upper and lower hemicontinuous functions	3134
6.42.13	Complex logs exist on various "well-behaved" sets . . .	3138
6.42.14	Another simple case where sphere maps are nullhomotopic	3140
6.42.15	Holomorphic logarithms and square roots	3141
6.42.16	The "Borsukian" property of sets	3145
6.42.17	Unicoherence (closed)	3158
6.42.18	Several common variants of unicoherence	3163
6.42.19	Some separation results	3164
6.43	The Jordan Curve Theorem and Applications	3172
6.43.1	Janiszewski's theorem	3172
6.43.2	The Jordan Curve theorem	3176
6.44	Polynomial Functions: Extremal Behaviour and Root Counts .	3188
6.44.1	Basics about polynomial functions: extremal behaviour and root counts	3188
6.45	Generalised Binomial Theorem	3193
6.46	Vitali Covering Theorem and an Application to Negligibility .	3199
6.46.1	Vitali covering theorem	3204
6.47	Change of Variables Theorems	3213
6.47.1	Measurable Shear and Stretch	3213
6.47.2	Borel measurable Jacobian determinant	3239
6.47.3	Simplest case of Sard's theorem (we don't need conti- nuity of derivative)	3258

6.47.4	A one-way version of change-of-variables not assuming injectivity.	3265
6.47.5	Change-of-variables theorem	3275
6.47.6	Change of variables for integrals: special case of linear function	3291
6.47.7	Change of variable for measure	3292
6.48	Lipschitz Continuity	3293
6.48.1	Local Lipschitz continuity	3300
6.48.2	Local Lipschitz continuity (uniform for a family of functions)	3303
6.49	Volume of a Simplex	3311
6.50	Convergence of Formal Power Series	3317
6.50.1	Balls with extended real radius	3317
6.50.2	Basic properties of convergent power series	3318
6.50.3	Lower bounds on radius of convergence	3321
6.50.4	Evaluating power series	3326
6.50.5	Power series expansions of analytic functions	3330
6.51	Smooth paths	3339
6.51.1	Homeomorphisms of arc images	3339
6.51.2	Piecewise differentiability of paths	3340
6.51.3	Valid paths, and their start and finish	3345
6.52	Neighbourhood bases and Locally path-connected spaces	3350
6.52.1	Neighbourhood Bases	3350
6.52.2	Locally path-connected spaces	3352
6.53	Euclidean space and n-spheres, as subtopologies of n-dimensional space	3360
6.53.1	Euclidean spaces as abstract topologies	3360
6.53.2	n-dimensional spheres	3364
6.54	Metrics on product spaces	3370



Chapter 1

Linear Algebra

```
theory L2_Norm
imports Complex_Main
begin
```

1.1 L2 Norm

```
definition L2_set :: ('a  $\Rightarrow$  real)  $\Rightarrow$  'a set  $\Rightarrow$  real where
L2_set f A = sqrt ( $\sum_{i \in A}. (f i)^2$ )
```

```
lemma L2_set_cong:
 $\llbracket A = B; \bigwedge x. x \in B \implies f x = g x \rrbracket \implies L2\_set f A = L2\_set g B$ 
unfolding L2_set_def by simp
```

```
lemma L2_set_cong_simp:
 $\llbracket A = B; \bigwedge x. x \in B =_{simp} \implies f x = g x \rrbracket \implies L2\_set f A = L2\_set g B$ 
unfolding L2_set_def simp_implies_def by simp
```

```
lemma L2_set_infinite [simp]:  $\neg \text{finite } A \implies L2\_set f A = 0$ 
unfolding L2_set_def by simp
```

```
lemma L2_set_empty [simp]:  $L2\_set f \{\} = 0$ 
unfolding L2_set_def by simp
```

```
lemma L2_set_insert [simp]:
 $\llbracket \text{finite } F; a \notin F \rrbracket \implies$ 
 $L2\_set f (\text{insert } a F) = \text{sqrt } ((f a)^2 + (L2\_set f F)^2)$ 
unfolding L2_set_def by (simp add: sum_nonneg)
```

```
lemma L2_set_nonneg [simp]:  $0 \leq L2\_set f A$ 
unfolding L2_set_def by (simp add: sum_nonneg)
```

```
lemma L2_set_0':  $\forall a \in A. f a = 0 \implies L2\_set f A = 0$ 
unfolding L2_set_def by simp
```

lemma *L2_set_constant*: $L2\_set (\lambda x. y) A = \text{sqrt} (\text{of\_nat} (\text{card } A)) * |y|$
unfolding *L2_set_def* **by** (*simp add: real_sqrt_mult*)

lemma *L2_set_mono*:
assumes $\bigwedge i. i \in K \implies f i \leq g i$
assumes $\bigwedge i. i \in K \implies 0 \leq f i$
shows $L2\_set f K \leq L2\_set g K$
unfolding *L2_set_def*
by (*simp add: sum_nonneg sum_mono power_mono assms*)

lemma *L2_set_strict_mono*:
assumes *finite* *K* **and** $K \neq \{\}$
assumes $\bigwedge i. i \in K \implies f i < g i$
assumes $\bigwedge i. i \in K \implies 0 \leq f i$
shows $L2\_set f K < L2\_set g K$
unfolding *L2_set_def*
by (*simp add: sum_strict_mono power_strict_mono assms*)

lemma *L2_set_right_distrib*:
 $0 \leq r \implies r * L2\_set f A = L2\_set (\lambda x. r * f x) A$
unfolding *L2_set_def*
apply (*simp add: power_mult_distrib*)
apply (*simp add: sum_distrib_left [symmetric]*)
apply (*simp add: real_sqrt_mult sum_nonneg*)
done

lemma *L2_set_left_distrib*:
 $0 \leq r \implies L2\_set f A * r = L2\_set (\lambda x. f x * r) A$
unfolding *L2_set_def*
apply (*simp add: power_mult_distrib*)
apply (*simp add: sum_distrib_right [symmetric]*)
apply (*simp add: real_sqrt_mult sum_nonneg*)
done

lemma *L2_set_eq_0_iff*: $\text{finite } A \implies L2\_set f A = 0 \longleftrightarrow (\forall x \in A. f x = 0)$
unfolding *L2_set_def*
by (*simp add: sum_nonneg sum_nonneg_eq_0_iff*)

proposition *L2_set_triangle_ineq*:
 $L2\_set (\lambda i. f i + g i) A \leq L2\_set f A + L2\_set g A$
proof (*cases finite A*)
case *False*
thus ?thesis **by** *simp*
next
case *True*
thus ?thesis
proof (*induct set: finite*)
case *empty*
show ?case **by** *simp*

```

next
  case (insert x F)
  hence  $\sqrt{(f\ x + g\ x)^2 + (L2\_set\ (\lambda i. f\ i + g\ i)\ F)^2} \leq$ 
     $\sqrt{(f\ x + g\ x)^2 + (L2\_set\ f\ F + L2\_set\ g\ F)^2}$ 
  by (intro real_sqrt_le_mono add_left_mono power_mono insert
      L2_set_nonneg add_increasing zero_le_power2)
  also have
    ...  $\leq \sqrt{(f\ x)^2 + (L2\_set\ f\ F)^2} + \sqrt{(g\ x)^2 + (L2\_set\ g\ F)^2}$ 
  by (rule real_sqrt_sum_squares_triangle_ineq)
  finally show ?case
    using insert by simp
qed
qed

```

```

lemma L2_set_le_sum [rule_format]:
   $(\forall i \in A. 0 \leq f\ i) \longrightarrow L2\_set\ f\ A \leq \sum f\ A$ 
  apply (cases finite A)
  apply (induct set: finite)
  apply simp
  apply clarsimp
  apply (erule order_trans [OF sqrt_sum_squares_le_sum])
  apply simp
  apply simp
  apply simp
  done

```

```

lemma L2_set_le_sum_abs:  $L2\_set\ f\ A \leq (\sum i \in A. |f\ i|)$ 
  apply (cases finite A)
  apply (induct set: finite)
  apply simp
  apply simp
  apply (rule order_trans [OF sqrt_sum_squares_le_sum_abs])
  apply simp
  apply simp
  done

```

```

lemma L2_set_mult_ineq:  $(\sum i \in A. |f\ i| * |g\ i|) \leq L2\_set\ f\ A * L2\_set\ g\ A$ 
  apply (cases finite A)
  apply (induct set: finite)
  apply simp
  apply (rule power2_le_imp_le, simp)
  apply (rule order_trans)
  apply (rule power_mono)
  apply (erule add_left_mono)
  apply (simp add: sum_nonneg)
  apply (simp add: power2_sum)
  apply (simp add: power_mult_distrib)
  apply (simp add: distrib_left distrib_right)
  apply (rule ord_le_eq_trans)

```

```

apply (rule L2_set_mult_ineq_lemma)
apply simp_all
done

```

```

lemma member_le_L2_set:  $\llbracket \text{finite } A; i \in A \rrbracket \implies f\ i \leq L2\_set\ f\ A$ 
  unfolding L2_set_def
  by (auto intro!: member_le_sum_real_le_rsqr)

```

```

end

```

1.2 Inner Product Spaces and Gradient Derivative

```

theory Inner_Product
imports Complex_Main
begin

```

1.2.1 Real inner product spaces

Temporarily relax type constraints for *open*, *uniformity*, *dist*, and *norm*.

```

setup  $\langle \text{Sign.add\_const\_constraint}$ 
  (const\_name  $\langle \text{open} \rangle$ , SOME typ  $\langle 'a::\text{open\_set} \Rightarrow \text{bool} \rangle \rangle$ 

```

```

setup  $\langle \text{Sign.add\_const\_constraint}$ 
  (const\_name  $\langle \text{dist} \rangle$ , SOME typ  $\langle 'a::\text{dist} \Rightarrow 'a \Rightarrow \text{real} \rangle \rangle$ 

```

```

setup  $\langle \text{Sign.add\_const\_constraint}$ 
  (const\_name  $\langle \text{uniformity} \rangle$ , SOME typ  $\langle ('a::\text{uniformity} \times 'a)\ \text{filter} \rangle \rangle$ 

```

```

setup  $\langle \text{Sign.add\_const\_constraint}$ 
  (const\_name  $\langle \text{norm} \rangle$ , SOME typ  $\langle 'a::\text{norm} \Rightarrow \text{real} \rangle \rangle$ 

```

```

class real_inner = real_vector + sgn_div_norm + dist_norm + uniformity_dist
+ open_uniformity +
  fixes inner ::  $'a \Rightarrow 'a \Rightarrow \text{real}$ 
  assumes inner_commute:  $\text{inner } x\ y = \text{inner } y\ x$ 
  and inner_add_left:  $\text{inner } (x + y)\ z = \text{inner } x\ z + \text{inner } y\ z$ 
  and inner_scaleR_left [simp]:  $\text{inner } (\text{scaleR } r\ x)\ y = r * (\text{inner } x\ y)$ 
  and inner_ge_zero [simp]:  $0 \leq \text{inner } x\ x$ 
  and inner_eq_zero_iff [simp]:  $\text{inner } x\ x = 0 \iff x = 0$ 
  and norm_eq_sqrt_inner:  $\text{norm } x = \text{sqrt } (\text{inner } x\ x)$ 
begin

```

```

lemma inner_zero_left [simp]:  $\text{inner } 0\ x = 0$ 
  using inner_add_left [of  $0\ 0\ x$ ] by simp

```

```

lemma inner_minus_left [simp]:  $\text{inner } (-x)\ y = -\ \text{inner } x\ y$ 
  using inner_add_left [of  $x - x\ y$ ] by simp

```


lemma *inner_diff_left*: $\text{inner } (x - y) \ z = \text{inner } x \ z - \text{inner } y \ z$
using *inner_add_left* [of $x - y \ z$] **by** *simp*

lemma *inner_sum_left*: $\text{inner } (\sum_{x \in A}. f \ x) \ y = (\sum_{x \in A}. \text{inner } (f \ x) \ y)$
by (*cases* *finite* *A*, *induct* *set*: *finite*, *simp_all* *add*: *inner_add_left*)

lemma *all_zero_iff* [*simp*]: $(\forall u. \text{inner } x \ u = 0) \longleftrightarrow (x = 0)$
by *auto* (*use* *inner_eq_zero_iff* **in** *blast*)

Transfer distributivity rules to right argument.

lemma *inner_add_right*: $\text{inner } x \ (y + z) = \text{inner } x \ y + \text{inner } x \ z$
using *inner_add_left* [of $y \ z \ x$] **by** (*simp* *only*: *inner_commute*)

lemma *inner_scaleR_right* [*simp*]: $\text{inner } x \ (\text{scaleR } r \ y) = r * (\text{inner } x \ y)$
using *inner_scaleR_left* [of $r \ y \ x$] **by** (*simp* *only*: *inner_commute*)

lemma *inner_zero_right* [*simp*]: $\text{inner } x \ 0 = 0$
using *inner_zero_left* [of x] **by** (*simp* *only*: *inner_commute*)

lemma *inner_minus_right* [*simp*]: $\text{inner } x \ (-y) = - \text{inner } x \ y$
using *inner_minus_left* [of $y \ x$] **by** (*simp* *only*: *inner_commute*)

lemma *inner_diff_right*: $\text{inner } x \ (y - z) = \text{inner } x \ y - \text{inner } x \ z$
using *inner_diff_left* [of $y \ z \ x$] **by** (*simp* *only*: *inner_commute*)

lemma *inner_sum_right*: $\text{inner } x \ (\sum_{y \in A}. f \ y) = (\sum_{y \in A}. \text{inner } x \ (f \ y))$
using *inner_sum_left* [of $f \ A \ x$] **by** (*simp* *only*: *inner_commute*)

lemmas *inner_add* [*algebra_simps*] = *inner_add_left* *inner_add_right*
lemmas *inner_diff* [*algebra_simps*] = *inner_diff_left* *inner_diff_right*
lemmas *inner_scaleR* = *inner_scaleR_left* *inner_scaleR_right*

Legacy theorem names

lemmas *inner_left_distrib* = *inner_add_left*
lemmas *inner_right_distrib* = *inner_add_right*
lemmas *inner_distrib* = *inner_left_distrib* *inner_right_distrib*

lemma *inner_gt_zero_iff* [*simp*]: $0 < \text{inner } x \ x \longleftrightarrow x \neq 0$
by (*simp* *add*: *order_less_le*)

lemma *power2_norm_eq_inner*: $(\text{norm } x)^2 = \text{inner } x \ x$
by (*simp* *add*: *norm_eq_sqrt_inner*)

Identities involving real multiplication and division.

lemma *inner_mult_left*: $\text{inner } (\text{of\_real } m * a) \ b = m * (\text{inner } a \ b)$
by (*metis* *real_inner_class.inner_scaleR_left* *scaleR_conv_of_real*)

lemma *inner_mult_right*: $\text{inner } a \ (\text{of\_real } m * b) = m * (\text{inner } a \ b)$
by (*metis* *real_inner_class.inner_scaleR_right* *scaleR_conv_of_real*)

lemma *inner_mult_left'*: $\text{inner } (a * \text{of\_real } m) b = m * (\text{inner } a b)$
by (*simp add: of_real_def*)

lemma *inner_mult_right'*: $\text{inner } a (b * \text{of\_real } m) = (\text{inner } a b) * m$
by (*simp add: of_real_def real_inner_class.inner_scaleR_right*)

lemma *Cauchy_Schwarz_ineq*:
 $(\text{inner } x y)^2 \leq \text{inner } x x * \text{inner } y y$
proof (*cases*)
assume $y = 0$
thus *?thesis* **by** *simp*
next
assume $y \neq 0$
let $?r = \text{inner } x y / \text{inner } y y$
have $0 \leq \text{inner } (x - \text{scaleR } ?r y) (x - \text{scaleR } ?r y)$
by (*rule inner_ge_zero*)
also have $\dots = \text{inner } x x - \text{inner } y x * ?r$
by (*simp add: inner_diff*)
also have $\dots = \text{inner } x x - (\text{inner } x y)^2 / \text{inner } y y$
by (*simp add: power2_eq_square inner_commute*)
finally have $0 \leq \text{inner } x x - (\text{inner } x y)^2 / \text{inner } y y$.
hence $(\text{inner } x y)^2 / \text{inner } y y \leq \text{inner } x x$
by (*simp add: le_diff_eq*)
thus $(\text{inner } x y)^2 \leq \text{inner } x x * \text{inner } y y$
by (*simp add: pos_divide_le_eq y*)
qed

lemma *Cauchy_Schwarz_ineq2*:
 $|\text{inner } x y| \leq \text{norm } x * \text{norm } y$
proof (*rule power2_le_imp_le*)
have $(\text{inner } x y)^2 \leq \text{inner } x x * \text{inner } y y$
using *Cauchy_Schwarz_ineq* .
thus $|\text{inner } x y|^2 \leq (\text{norm } x * \text{norm } y)^2$
by (*simp add: power_mult_distrib power2_norm_eq_inner*)
show $0 \leq \text{norm } x * \text{norm } y$
unfolding *norm_eq_sqrt_inner*
by (*intro mult_nonneg_nonneg real_sqrt_ge_zero inner_ge_zero*)
qed

lemma *norm_cauchy_schwarz*: $\text{inner } x y \leq \text{norm } x * \text{norm } y$
using *Cauchy_Schwarz_ineq2* [*of x y*] **by** *auto*

subclass *real_normed_vector*

proof
fix $a :: \text{real}$ **and** $x y :: 'a$
show $\text{norm } x = 0 \longleftrightarrow x = 0$
unfolding *norm_eq_sqrt_inner* **by** *simp*
show $\text{norm } (x + y) \leq \text{norm } x + \text{norm } y$

```

proof (rule power2_le_imp_le)
  have inner x y ≤ norm x * norm y
    by (rule norm_cauchy_schwarz)
  thus (norm (x + y))2 ≤ (norm x + norm y)2
    unfolding power2_sum power2_norm_eq_inner
    by (simp add: inner_add inner_commute)
  show 0 ≤ norm x + norm y
    unfolding norm_eq_sqrt_inner by simp
qed

have sqrt (a2 * inner x x) = |a| * sqrt (inner x x)
  by (simp add: real_sqrt_mult)
then show norm (a *R x) = |a| * norm x
  unfolding norm_eq_sqrt_inner
  by (simp add: power2_eq_square mult.assoc)
qed

end

lemma square_bound_lemma:
  fixes x :: real
  shows x < (1 + x) * (1 + x)
proof -
  have (x + 1/2)2 + 3/4 > 0
    using zero_le_power2[of x+1/2] by arith
  then show ?thesis
    by (simp add: field_simps power2_eq_square)
qed

lemma square_continuous:
  fixes e :: real
  shows e > 0 ⇒ ∃ d. 0 < d ∧ (∀ y. |y - x| < d ⇒ |y * y - x * x| < e)
    using isCont_power[OF continuous_ident, of x, unfolded isCont_def LIM_eq,
rule_format, of e 2]
    by (force simp add: power2_eq_square)

lemma norm_le: norm x ≤ norm y ⟷ inner x x ≤ inner y y
  by (simp add: norm_eq_sqrt_inner)

lemma norm_lt: norm x < norm y ⟷ inner x x < inner y y
  by (simp add: norm_eq_sqrt_inner)

lemma norm_eq: norm x = norm y ⟷ inner x x = inner y y
  apply (subst order_eq_iff)
  apply (auto simp: norm_le)
  done

lemma norm_eq_1: norm x = 1 ⟷ inner x x = 1
  by (simp add: norm_eq_sqrt_inner)

```

```

lemma inner_divide_left:
  fixes a :: 'a :: {real_inner, real_div_algebra}
  shows inner (a / of_real m) b = (inner a b) / m
  by (metis (no_types) divide_inverse inner_commute inner_scaleR_right mult.left_neutral
mult.right_neutral mult_scaleR_right of_real_inverse scaleR_conv_of_real times_divide_eq_left)

```

```

lemma inner_divide_right:
  fixes a :: 'a :: {real_inner, real_div_algebra}
  shows inner a (b / of_real m) = (inner a b) / m
  by (metis inner_commute inner_divide_left)

```

Re-enable constraints for *open*, *uniformity*, *dist*, and *norm*.

```

setup <Sign.add_const_constraint
  (const_name <open>, SOME typ <'a::topological_space set  $\Rightarrow$  bool>)>

```

```

setup <Sign.add_const_constraint
  (const_name <uniformity>, SOME typ <('a::uniform_space  $\times$  'a) filter>)>

```

```

setup <Sign.add_const_constraint
  (const_name <dist>, SOME typ <'a::metric_space  $\Rightarrow$  'a  $\Rightarrow$  real>)>

```

```

setup <Sign.add_const_constraint
  (const_name <norm>, SOME typ <'a::real_normed_vector  $\Rightarrow$  real>)>

```

```

lemma bounded_bilinear_inner:
  bounded_bilinear (inner::'a::real_inner  $\Rightarrow$  'a  $\Rightarrow$  real)
proof
  fix x y z :: 'a and r :: real
  show inner (x + y) z = inner x z + inner y z
    by (rule inner_add_left)
  show inner x (y + z) = inner x y + inner x z
    by (rule inner_add_right)
  show inner (scaleR r x) y = scaleR r (inner x y)
    unfolding real_scaleR_def by (rule inner_scaleR_left)
  show inner x (scaleR r y) = scaleR r (inner x y)
    unfolding real_scaleR_def by (rule inner_scaleR_right)
  show  $\exists K. \forall x y::'a. \text{norm } (\text{inner } x y) \leq \text{norm } x * \text{norm } y * K$ 
  proof
    show  $\forall x y::'a. \text{norm } (\text{inner } x y) \leq \text{norm } x * \text{norm } y * 1$ 
      by (simp add: Cauchy_Schwarz_ineq2)
  qed
qed

```

```

lemmas tendsto_inner [tendsto_intros] =
  bounded_bilinear.tendsto [OF bounded_bilinear_inner]

```

```

lemmas isCont_inner [simp] =
  bounded_bilinear.isCont [OF bounded_bilinear_inner]

```

```

lemmas has_derivative_inner [derivative_intros] =
  bounded_bilinear.FDERIV [OF bounded_bilinear_inner]

lemmas bounded_linear_inner_left =
  bounded_bilinear.bounded_linear_left [OF bounded_bilinear_inner]

lemmas bounded_linear_inner_right =
  bounded_bilinear.bounded_linear_right [OF bounded_bilinear_inner]

lemmas bounded_linear_inner_left_comp = bounded_linear_inner_left [THEN
  bounded_linear_compose]

lemmas bounded_linear_inner_right_comp = bounded_linear_inner_right [THEN
  bounded_linear_compose]

lemmas has_derivative_inner_right [derivative_intros] =
  bounded_linear.has_derivative [OF bounded_linear_inner_right]

lemmas has_derivative_inner_left [derivative_intros] =
  bounded_linear.has_derivative [OF bounded_linear_inner_left]

lemma differentiable_inner [simp]:
  f differentiable (at x within s)  $\implies$  g differentiable at x within s  $\implies$  ( $\lambda x$ . inner (f
  x) (g x)) differentiable at x within s
  unfolding differentiable_def by (blast intro: has_derivative_inner)

```

1.2.2 Class instances

```

instantiation real :: real_inner
begin

```

```

definition inner_real_def [simp]: inner = (*)

```

```

instance

```

```

proof

```

```

  fix x y z r :: real
  show inner x y = inner y x
    unfolding inner_real_def by (rule mult.commute)
  show inner (x + y) z = inner x z + inner y z
    unfolding inner_real_def by (rule distrib_right)
  show inner (scaleR r x) y = r * inner x y
    unfolding inner_real_def real_scaleR_def by (rule mult.assoc)
  show 0  $\leq$  inner x x
    unfolding inner_real_def by simp
  show inner x x = 0  $\longleftrightarrow$  x = 0
    unfolding inner_real_def by simp
  show norm x = sqrt (inner x x)
    unfolding inner_real_def by simp

```

```

qed

```

end

lemma

shows $\text{real\_inner\_1\_left}[simp]: \text{inner } 1 \ x = x$
 and $\text{real\_inner\_1\_right}[simp]: \text{inner } x \ 1 = x$
 by simp_all

instantiation $\text{complex} :: \text{real\_inner}$
 begin

definition inner_complex_def :

$\text{inner } x \ y = \text{Re } x * \text{Re } y + \text{Im } x * \text{Im } y$

instance

proof

fix $x \ y \ z :: \text{complex}$ and $r :: \text{real}$
 show $\text{inner } x \ y = \text{inner } y \ x$
 unfolding inner_complex_def by (simp add: mult.commute)
 show $\text{inner } (x + y) \ z = \text{inner } x \ z + \text{inner } y \ z$
 unfolding inner_complex_def by (simp add: distrib_right)
 show $\text{inner } (\text{scaleR } r \ x) \ y = r * \text{inner } x \ y$
 unfolding inner_complex_def by (simp add: distrib_left)
 show $0 \leq \text{inner } x \ x$
 unfolding inner_complex_def by simp
 show $\text{inner } x \ x = 0 \longleftrightarrow x = 0$
 unfolding inner_complex_def
 by (simp add: add_nonneg_eq_0_iff complex_eq_iff)
 show $\text{norm } x = \text{sqrt } (\text{inner } x \ x)$
 unfolding inner_complex_def norm_complex_def
 by (simp add: power2_eq_square)

qed

end

lemma $\text{complex\_inner\_1} [simp]: \text{inner } 1 \ x = \text{Re } x$
 unfolding inner_complex_def by simp

lemma $\text{complex\_inner\_1\_right} [simp]: \text{inner } x \ 1 = \text{Re } x$
 unfolding inner_complex_def by simp

lemma $\text{complex\_inner\_i\_left} [simp]: \text{inner } i \ x = \text{Im } x$
 unfolding inner_complex_def by simp

lemma $\text{complex\_inner\_i\_right} [simp]: \text{inner } x \ i = \text{Im } x$
 unfolding inner_complex_def by simp

lemma $\text{dot\_square\_norm}: \text{inner } x \ x = (\text{norm } x)^2$

```

by (simp only: power2_norm_eq_inner)

lemma norm_eq_square: norm x = a  $\longleftrightarrow$  0  $\leq$  a  $\wedge$  inner x x = a2
  by (auto simp add: norm_eq_sqrt_inner)

lemma norm_le_square: norm x  $\leq$  a  $\longleftrightarrow$  0  $\leq$  a  $\wedge$  inner x x  $\leq$  a2
  apply (simp add: dot_square_norm abs_le_square_iff[symmetric])
  using norm_ge_zero[of x]
  apply arith
  done

lemma norm_ge_square: norm x  $\geq$  a  $\longleftrightarrow$  a  $\leq$  0  $\vee$  inner x x  $\geq$  a2
  apply (simp add: dot_square_norm abs_le_square_iff[symmetric])
  using norm_ge_zero[of x]
  apply arith
  done

lemma norm_lt_square: norm x < a  $\longleftrightarrow$  0 < a  $\wedge$  inner x x < a2
  by (metis not_le norm_ge_square)

lemma norm_gt_square: norm x > a  $\longleftrightarrow$  a < 0  $\vee$  inner x x > a2
  by (metis norm_le_square not_less)

Dot product in terms of the norm rather than conversely.

lemmas inner_simps = inner_add_left inner_add_right inner_diff_right inner_diff_left
  inner_scaleR_left inner_scaleR_right

lemma dot_norm: inner x y = ((norm (x + y))2 - (norm x)2 - (norm y)2) / 2
  by (simp only: power2_norm_eq_inner inner_simps inner_commute) auto

lemma dot_norm_neg: inner x y = (((norm x)2 + (norm y)2) - (norm (x - y))2) / 2
  by (simp only: power2_norm_eq_inner inner_simps inner_commute)
  (auto simp add: algebra_simps)

lemma of_real_inner_1 [simp]:
  inner (of_real x) (1 :: 'a :: {real_inner, real_normed_algebra_1}) = x
  by (simp add: of_real_def dot_square_norm)

lemma summable_of_real_iff:
  summable ( $\lambda$ x. of_real (f x)) :: 'a :: {real_normed_algebra_1, real_inner}  $\longleftrightarrow$ 
  summable f
proof
  assume *: summable ( $\lambda$ x. of_real (f x)) :: 'a
  interpret bounded_linear  $\lambda$ x::'a. inner x 1
  by (rule bounded_linear_inner_left)
  from summable [OF *] show summable f by simp
qed (auto intro: summable_of_real)

```

1.2.3 Gradient derivative

definition

gderiv ::
 $[a :: \text{real\_inner} \Rightarrow \text{real}, 'a, 'a] \Rightarrow \text{bool}$
 $((\text{GDERIV } \_)/ (\_)/ :> (\_)) [1000, 1000, 60] 60)$

where

$\text{GDERIV } f x :> D \longleftrightarrow \text{FDERIV } f x :> (\lambda h. \text{inner } h D)$

lemma *gderiv_deriv* [simp]: $\text{GDERIV } f x :> D \longleftrightarrow \text{DERIV } f x :> D$

by (simp only: *gderiv_def has_field_derivative_def inner_real_def mult_commute_abs*)

lemma *GDERIV_DERIV_compose*:

$\llbracket \text{GDERIV } f x :> df; \text{DERIV } g (f x) :> dg \rrbracket$
 $\implies \text{GDERIV } (\lambda x. g (f x)) x :> \text{scaleR } dg df$

unfolding *gderiv_def has_field_derivative_def*

apply (*drule* (1) *has_derivative_compose*)

apply (*simp add: ac_simps*)

done

lemma *has_derivative_subst*: $\llbracket \text{FDERIV } f x :> df; df = d \rrbracket \implies \text{FDERIV } f x :> d$

by *simp*

lemma *GDERIV_subst*: $\llbracket \text{GDERIV } f x :> df; df = d \rrbracket \implies \text{GDERIV } f x :> d$

by *simp*

lemma *GDERIV_const*: $\text{GDERIV } (\lambda x. k) x :> 0$

unfolding *gderiv_def inner_zero_right* **by** (*rule has_derivative_const*)

lemma *GDERIV_add*:

$\llbracket \text{GDERIV } f x :> df; \text{GDERIV } g x :> dg \rrbracket$
 $\implies \text{GDERIV } (\lambda x. f x + g x) x :> df + dg$

unfolding *gderiv_def inner_add_right* **by** (*rule has_derivative_add*)

lemma *GDERIV_minus*:

$\text{GDERIV } f x :> df \implies \text{GDERIV } (\lambda x. - f x) x :> - df$

unfolding *gderiv_def inner_minus_right* **by** (*rule has_derivative_minus*)

lemma *GDERIV_diff*:

$\llbracket \text{GDERIV } f x :> df; \text{GDERIV } g x :> dg \rrbracket$
 $\implies \text{GDERIV } (\lambda x. f x - g x) x :> df - dg$

unfolding *gderiv_def inner_diff_right* **by** (*rule has_derivative_diff*)

lemma *GDERIV_scaleR*:

$\llbracket \text{DERIV } f x :> df; \text{GDERIV } g x :> dg \rrbracket$
 $\implies \text{GDERIV } (\lambda x. \text{scaleR } (f x) (g x)) x$
 $:> (\text{scaleR } (f x) dg + \text{scaleR } df (g x))$

unfolding *gderiv_def has_field_derivative_def inner_add_right inner_scaleR_right*
apply (*rule has_derivative_subst*)


```

apply (erule (1) has_derivative_scaleR)
apply (simp add: ac_simps)
done

lemma GDERIV_mult:
  
$$\llbracket GDERIV\ f\ x\ :\>\ df;\ GDERIV\ g\ x\ :\>\ dg \rrbracket$$


$$\implies GDERIV\ (\lambda x. f\ x * g\ x)\ x\ :\>\ scaleR\ (f\ x)\ dg + scaleR\ (g\ x)\ df$$

unfolding gderiv_def
apply (rule has_derivative_subst)
apply (erule (1) has_derivative_mult)
apply (simp add: inner_add ac_simps)
done

lemma GDERIV_inverse:
  
$$\llbracket GDERIV\ f\ x\ :\>\ df;\ f\ x \neq 0 \rrbracket$$


$$\implies GDERIV\ (\lambda x. inverse\ (f\ x))\ x\ :\>\ -\ (inverse\ (f\ x))^2 *_{\mathbb{R}} df$$

by (metis DERIV_inverse GDERIV_DERIV_compose numerals(2))

lemma GDERIV_norm:
  assumes  $x \neq 0$  shows  $GDERIV\ (\lambda x. norm\ x)\ x\ :\>\ sgn\ x$ 
unfolding gderiv_def norm_eq_sqrt_inner
by (rule derivative_eq_intros | force simp add: inner_commute sgn_div_norm
  norm_eq_sqrt_inner assms)+

lemmas has_derivative_norm = GDERIV_norm [unfolded gderiv_def]

bundle inner_syntax begin
notation inner (infix  $\cdot$  70)
end

bundle no_inner_syntax begin
no_notation inner (infix  $\cdot$  70)
end

end

```

1.3 Cartesian Products as Vector Spaces

```

theory Product_Vector
imports
  Complex_Main
  HOL-Library.Product_Plus
begin

lemma Times_eq_image_sum:
  fixes  $S :: 'a :: comm\_monoid\_add\ set$  and  $T :: 'b :: comm\_monoid\_add\ set$ 
shows  $S \times T = \{u + v \mid u\ v. u \in (\lambda x. (x, 0))\ 'S \wedge v \in Pair\ 0\ 'T\}$ 
by force

```

1.3.1 Product is a Module

locale *module_prod* = *module_pair* **begin**

definition *scale* :: 'a $\Rightarrow$ 'b $\times$ 'c $\Rightarrow$ 'b $\times$ 'c
where *scale* a v = (s1 a (fst v), s2 a (snd v))

lemma *scale_prod*: *scale* x (a, b) = (s1 x a, s2 x b)
by (auto simp: *scale_def*)

sublocale *p*: *module* *scale*

proof qed (simp_all add: *scale_def*
m1.scale_left_distrib m1.scale_right_distrib m2.scale_left_distrib m2.scale_right_distrib)

lemma *subspace_Times*: *m1.subspace* A $\Longrightarrow$ *m2.subspace* B $\Longrightarrow$ *p.subspace* (A $\times$ B)

unfolding *m1.subspace_def m2.subspace_def p.subspace_def*
by (auto simp: *zero_prod_def scale_def*)

lemma *module_hom_fst*: *module_hom* *scale* s1 *fst*
by *unfold_locales* (auto simp: *scale_def*)

lemma *module_hom_snd*: *module_hom* *scale* s2 *snd*
by *unfold_locales* (auto simp: *scale_def*)

end

locale *vector_space_prod* = *vector_space_pair* **begin**

sublocale *module_prod* s1 s2
rewrites *module_hom* = *Vector_Spaces.linear*
by *unfold_locales* (fact *module_hom_eq_linear*)

sublocale *p*: *vector_space* *scale* **by** *unfold_locales* (auto simp: *algebra_simps*)

lemmas *linear_fst* = *module_hom_fst*
and *linear_snd* = *module_hom_snd*

end

1.3.2 Product is a Real Vector Space

instantiation *prod* :: (real_vector, real_vector) real_vector
begin

definition *scaleR_prod_def*:
scaleR r A = (*scaleR* r (fst A), *scaleR* r (snd A))

lemma *fst_scaleR* [simp]: *fst* (*scaleR* r A) = *scaleR* r (fst A)
unfolding *scaleR_prod_def* **by** *simp*

```

lemma snd_scaleR [simp]: snd (scaleR r A) = scaleR r (snd A)
  unfolding scaleR_prod_def by simp

```

```

proposition scaleR_Pair [simp]: scaleR r (a, b) = (scaleR r a, scaleR r b)
  unfolding scaleR_prod_def by simp

```

```

instance
proof
  fix a b :: real and x y :: 'a × 'b
  show scaleR a (x + y) = scaleR a x + scaleR a y
    by (simp add: prod_eq_iff scaleR_right_distrib)
  show scaleR (a + b) x = scaleR a x + scaleR b x
    by (simp add: prod_eq_iff scaleR_left_distrib)
  show scaleR a (scaleR b x) = scaleR (a * b) x
    by (simp add: prod_eq_iff)
  show scaleR 1 x = x
    by (simp add: prod_eq_iff)
qed

```

```

end

```

```

lemma module_prod_scale_eq_scaleR: module_prod.scale (*R) (*R) = scaleR
  apply (rule ext) apply (rule ext)
  apply (subst module_prod.scale_def)
  subgoal by unfold_locales
  by (simp add: scaleR_prod_def)

```

```

interpretation real_vector?: vector_space_prod scaleR::_⇒_⇒'a::real_vector scaleR::_⇒_⇒'b::real_vector
  rewrites scale = ((*R)::_⇒_⇒('a × 'b))
    and module.dependent (*R) = dependent
    and module.representation (*R) = representation
    and module.subspace (*R) = subspace
    and module.span (*R) = span
    and vector_space.extend_basis (*R) = extend_basis
    and vector_space.dim (*R) = dim
    and Vector_Spaces.linear (*R) (*R) = linear
  subgoal by unfold_locales
  subgoal by (fact module_prod_scale_eq_scaleR)
  unfolding dependent_raw_def representation_raw_def subspace_raw_def span_raw_def
    extend_basis_raw_def dim_raw_def linear_def
  by (rule refl)+

```

1.3.3 Product is a Metric Space

```

instantiation prod :: (metric_space, metric_space) dist
begin

```

```

definition dist_prod_def[code del]:

```

$$\text{dist } x \ y = \text{sqrt } ((\text{dist } (\text{fst } x) (\text{fst } y))^2 + (\text{dist } (\text{snd } x) (\text{snd } y))^2)$$

instance ..
end

instantiation prod :: (uniformity, uniformity) uniformity begin

definition [code del]: $\langle (\text{uniformity} :: (('a \times 'b) \times ('a \times 'b)) \text{ filter}) =$
 $\text{filtermap } (\lambda((x1,x2),(y1,y2)). ((x1,y1),(x2,y2))) (\text{uniformity} \times_F \text{uniformity}) \rangle$

instance..
end

instantiation prod :: (uniform_space, uniform_space) uniform_space begin

instance

proof standard

fix $U :: \langle ('a \times 'b) \text{ set} \rangle$
show $\langle \text{open } U \longleftrightarrow (\forall x \in U. \forall_F (x', y) \text{ in uniformity. } x' = x \longrightarrow y \in U) \rangle$
proof (intro iffI ballI)
fix x **assume** $\langle \text{open } U \rangle$ **and** $\langle x \in U \rangle$
then obtain $A \ B$ **where** $\langle \text{open } A \rangle \langle \text{open } B \rangle \langle x \in A \times B \rangle \langle A \times B \subseteq U \rangle$
by (metis open_prod_elim)
define UA **where** $\langle UA = (\lambda(x'::'a,y). x' = \text{fst } x \longrightarrow y \in A) \rangle$
from $\langle \text{open } A \rangle \langle x \in A \times B \rangle$
have $\langle \text{eventually } UA \text{ uniformity} \rangle$
unfolding open_uniformity UA_def **by** auto
define UB **where** $\langle UB = (\lambda(x'::'b,y). x' = \text{snd } x \longrightarrow y \in B) \rangle$
from $\langle \text{open } A \rangle \langle \text{open } B \rangle \langle x \in A \times B \rangle$
have $\langle \text{eventually } UA \text{ uniformity} \rangle \langle \text{eventually } UB \text{ uniformity} \rangle$
unfolding open_uniformity UA_def UB_def **by** auto
then have $\langle \forall_F ((x'1, y1), (x'2, y2)) \text{ in uniformity} \times_F \text{uniformity. } (x'1, x'2)$
 $= x \longrightarrow (y1, y2) \in U \rangle$
apply (auto intro!: exI[of_ UA] exI[of_ UB] simp add: eventually_prod_filter)
using $\langle A \times B \subseteq U \rangle$ **by** (auto simp: UA_def UB_def)
then show $\langle \forall_F (x', y) \text{ in uniformity. } x' = x \longrightarrow y \in U \rangle$
by (simp add: uniformity_prod_def eventually_filtermap case_prod_unfold)
next
assume $asm: \langle \forall x \in U. \forall_F (x', y) \text{ in uniformity. } x' = x \longrightarrow y \in U \rangle$
show $\langle \text{open } U \rangle$
proof (unfold open_prod_def, intro ballI)
fix x **assume** $\langle x \in U \rangle$
with asm **have** $\langle \forall_F (x', y) \text{ in uniformity. } x' = x \longrightarrow y \in U \rangle$
by auto
then have $\langle \forall_F ((x'1, y1), (x'2, y2)) \text{ in uniformity} \times_F \text{uniformity. } (x'1, x'2)$
 $= x \longrightarrow (y1, y2) \in U \rangle$
by (simp add: uniformity_prod_def eventually_filtermap case_prod_unfold)
then obtain $UA \ UB$ **where** $\langle \text{eventually } UA \text{ uniformity} \rangle$ **and** $\langle \text{eventually } UB$
 $\text{uniformity} \rangle$

```

      and  $UA\_UB\_U$ :  $\langle UA(a1, a2) \implies UB(b1, b2) \implies (a1, b1) = x \implies (a2, b2) \in U \rangle$  for  $a1\ a2\ b1\ b2$ 
      apply atomize_elim by (simp add: case_prod_beta eventually_prod_filter)
      have  $\langle eventually(\lambda a. UA(fst\ x, a))\ (nhds(fst\ x)) \rangle$ 
      using  $\langle eventually\ UA\ uniformity \rangle eventually\_mono eventually\_nhds\_uniformity$ 
    by fastforce
      then obtain  $A$  where  $\langle open\ A \rangle$  and  $A\_UA$ :  $\langle A \subseteq \{a. UA(fst\ x, a)\} \rangle$  and
 $\langle fst\ x \in A \rangle$ 
      by (metis (mono_tags, lifting) eventually_nhds mem_Collect_eq subsetI)
      have  $\langle eventually(\lambda b. UB(snd\ x, b))\ (nhds(snd\ x)) \rangle$ 
      using  $\langle eventually\ UB\ uniformity \rangle eventually\_mono eventually\_nhds\_uniformity$ 
    by fastforce
      then obtain  $B$  where  $\langle open\ B \rangle$  and  $B\_UB$ :  $\langle B \subseteq \{b. UB(snd\ x, b)\} \rangle$  and
 $\langle snd\ x \in B \rangle$ 
      by (metis (mono_tags, lifting) eventually_nhds mem_Collect_eq subsetI)
      have  $\langle x \in A \times B \rangle$ 
      by (simp add:  $\langle fst\ x \in A \rangle \langle snd\ x \in B \rangle mem\_Times\_iff$ )
      have  $\langle A \times B \subseteq U \rangle$ 
      using  $A\_UA\ B\_UB\ UA\_UB\_U$  by fastforce
      show  $\langle \exists A\ B. open\ A \wedge open\ B \wedge x \in A \times B \wedge A \times B \subseteq U \rangle$ 
      using  $\langle A \times B \subseteq U \rangle \langle open\ A \rangle \langle open\ B \rangle \langle x \in A \times B \rangle$  by auto
    qed
  qed
next
  show  $\langle eventually\ E\ uniformity \implies E(x, x) \rangle$  for  $E$  and  $x :: 'a \times 'b$ 
  apply (simp add: uniformity_prod_def eventually_filtermap case_prod_unfold eventually_prod_filter)
  by (metis surj_pair uniformity_refl)
next
  show  $\langle eventually\ E\ uniformity \implies \forall_F (x :: 'a \times 'b, y) \text{ in } uniformity. E(y, x) \rangle$  for  $E$ 
  apply (simp only: uniformity_prod_def eventually_filtermap case_prod_unfold eventually_prod_filter)
  apply (erule exE, erule exE, rename_tac Pf Pg)
  apply (rule_tac  $x = \langle \lambda(x, y). Pf(y, x) \rangle$  in exI)
  apply (rule_tac  $x = \langle \lambda(x, y). Pg(y, x) \rangle$  in exI)
  by (auto simp add: uniformity_sym)
next
  show  $\langle \exists D. eventually\ D\ uniformity \wedge (\forall x\ y\ z. D(x :: 'a \times 'b, y) \longrightarrow D(y, z) \longrightarrow E(x, z)) \rangle$ 
  if  $\langle eventually\ E\ uniformity \rangle$  for  $E$ 
  proof -
    from that
    obtain  $EA\ EB$  where  $\langle eventually\ EA\ uniformity \rangle$  and  $\langle eventually\ EB\ uniformity \rangle$ 
      and  $EA\_EB\_E$ :  $\langle EA(a1, a2) \implies EB(b1, b2) \implies E((a1, b1), (a2, b2)) \rangle$  for  $a1\ a2\ b1\ b2$ 
    by (auto simp add: uniformity_prod_def eventually_filtermap case_prod_unfold eventually_prod_filter)

```

```

obtain  $DA$  where  $\langle \text{eventually } DA \text{ uniformity} \rangle$  and  $DA\_EA: \langle DA(x,y) \implies$ 
 $DA(y,z) \implies EA(x,z) \rangle$  for  $x\ y\ z$ 
  using  $\langle \text{eventually } EA \text{ uniformity} \rangle$   $\text{uniformity\_transE}$  by  $\text{blast}$ 
obtain  $DB$  where  $\langle \text{eventually } DB \text{ uniformity} \rangle$  and  $DB\_EB: \langle DB(x,y) \implies$ 
 $DB(y,z) \implies EB(x,z) \rangle$  for  $x\ y\ z$ 
  using  $\langle \text{eventually } EB \text{ uniformity} \rangle$   $\text{uniformity\_transE}$  by  $\text{blast}$ 
define  $D$  where  $\langle D = (\lambda((a1,b1),(a2,b2)). DA(a1,a2) \wedge DB(b1,b2)) \rangle$ 
have  $\langle \text{eventually } D \text{ uniformity} \rangle$ 
  using  $\langle \text{eventually } DA \text{ uniformity} \rangle$   $\langle \text{eventually } DB \text{ uniformity} \rangle$ 
by ( $\text{auto simp add: uniformity\_prod\_def eventually\_filtermap case\_prod\_unfold}$ 
 $\text{eventually\_prod\_filter } D\_def$ )
  moreover have  $\langle D((a1, b1), (a2, b2)) \implies D((a2, b2), (a3, b3)) \implies E$ 
 $((a1, b1), (a3, b3)) \rangle$  for  $a1\ b1\ a2\ b2\ a3\ b3$ 
    using  $DA\_EA\ DB\_EB\ D\_def\ EA\_EB\_E$  by  $\text{blast}$ 
  ultimately show  $?thesis$ 
    by  $\text{auto}$ 
qed
qed
end

```

instantiation $\text{prod} :: (\text{metric\_space}, \text{metric\_space}) \text{ uniformity\_dist}$ **begin**

instance

proof

show $\langle \text{uniformity} = (\text{INF } e \in \{0 < ..\}. \text{principal } \{(x::'a \times 'b, y). \text{dist } x\ y < e\}) \rangle$

proof ($\text{subst filter\_eq\_iff, intro allI iffI}$)

fix $P :: \langle ('a \times 'b) \times ('a \times 'b) \Rightarrow \text{bool} \rangle$

have $1: \langle \exists e \in \{0 < ..\}.$

$\{(x,y). \text{dist } x\ y < e\} \subseteq \{(x,y). \text{dist } x\ y < a\} \wedge$

$\{(x,y). \text{dist } x\ y < e\} \subseteq \{(x,y). \text{dist } x\ y < b\} \rangle$ **if** $\langle a > 0 \rangle \langle b > 0 \rangle$ **for** $a\ b$

apply ($\text{rule bexI[of\_ } \langle \min\ a\ b \rangle$)

using that by auto

have $2: \langle \text{mono } (\lambda P. \text{eventually } (\lambda x. P(Q\ x))\ F) \rangle$ **for** $F :: \langle 'z \text{ filter} \rangle$ **and** $Q ::$
 $\langle 'z \Rightarrow 'y \rangle$

unfolding mono_def **using** $\text{eventually\_mono le\_funD}$ **by** fastforce

have $\langle \forall_F ((x1::'a,y1),(x2::'b,y2)) \text{ in } \text{uniformity} \times_F \text{uniformity}. \text{dist } x1\ y1 <$
 $e/2 \wedge \text{dist } x2\ y2 < e/2 \rangle$ **if** $\langle e > 0 \rangle$ **for** e

by ($\text{auto intro!: eventually\_prodI exI[of\_ } \langle e/2 \rangle \text{ simp: case\_prod\_unfold}$
 $\text{eventually\_uniformity\_metric that}$)

then have $3: \langle \forall_F ((x1::'a,y1),(x2::'b,y2)) \text{ in } \text{uniformity} \times_F \text{uniformity}. \text{dist}$
 $(x1,x2)\ (y1,y2) < e \rangle$ **if** $\langle e > 0 \rangle$ **for** e

apply ($\text{rule eventually\_rev\_mp}$)

by ($\text{auto intro!: that eventuallyI simp: case\_prod\_unfold dist\_prod\_def}$
 $\text{sqr\_sum\_squares\_half\_less}$)

show $\langle \text{eventually } P (\text{INF } e \in \{0 < ..\}. \text{principal } \{(x, y). \text{dist } x\ y < e\}) \implies$
 $\text{eventually } P \text{ uniformity} \rangle$

apply ($\text{subst (asm) eventually\_INF\_base}$)

using $1\ 3$ **apply** ($\text{auto simp: uniformity\_prod\_def case\_prod\_unfold even-}$
 $\text{tually\_filtermap } 2 \text{ eventually\_principal}$)

```

    by (smt (verit, best) eventually_mono)
  next
    fix P :: ⟨('a × 'b) × ('a × 'b) ⇒ bool⟩
    assume ⟨eventually P uniformity⟩
    then obtain P1 P2 where ⟨eventually P1 uniformity⟩ ⟨eventually P2 uniformity⟩
  and P1P2P: ⟨P1 (x1, y1) ⇒ P2 (x2, y2) ⇒ P ((x1, x2), (y1, y2))⟩ for
x1 y1 x2 y2
    by (auto simp: eventually_filtermap case_prod_beta eventually_prod_filter
uniformity_prod_def)
    from ⟨eventually P1 uniformity⟩ obtain e1 where ⟨e1 > 0⟩ and e1P1: ⟨dist x
y < e1 ⇒ P1 (x,y)⟩ for x y
    using eventually_uniformity_metric by blast
    from ⟨eventually P2 uniformity⟩ obtain e2 where ⟨e2 > 0⟩ and e2P2: ⟨dist x
y < e2 ⇒ P2 (x,y)⟩ for x y
    using eventually_uniformity_metric by blast
    define e where ⟨e = min e1 e2⟩
    have ⟨e > 0⟩
    using ⟨0 < e1⟩ ⟨0 < e2⟩ e_def by auto
    have ⟨dist (x1,x2) (y1,y2) < e ⇒ dist x1 y1 < e1⟩ for x1 y1 :: 'a and x2
y2 :: 'b
    unfolding dist_prod_def e_def apply auto
    by (smt (verit, best) real_sqrt_sum_squares_ge1)
    moreover have ⟨dist (x1,x2) (y1,y2) < e ⇒ dist x2 y2 < e2⟩ for x1 y1 ::
'a and x2 y2 :: 'b
    unfolding dist_prod_def e_def apply auto
    by (smt (verit, best) real_sqrt_sum_squares_ge1)
    ultimately have *: ⟨dist (x1,x2) (y1,y2) < e ⇒ P ((x1, x2), (y1, y2))⟩ for
x1 y1 x2 y2
    using e1P1 e2P2 P1P2P by auto

    show ⟨eventually P (INF e ∈ {0<..}. principal {(x, y). dist x y < e})⟩
    apply (rule eventually_INF1[where i=e])
    using ⟨e > 0⟩ * by (auto simp: eventually_principal)
  qed
qed
end

declare uniformity_Abort[where 'a='a :: metric_space × 'b :: metric_space,
code]

instantiation prod :: (metric_space, metric_space) metric_space
begin

proposition dist_Pair_Pair: dist (a, b) (c, d) = sqrt ((dist a c)2 + (dist b d)2)
  unfolding dist_prod_def by simp

lemma dist_fst_le: dist (fst x) (fst y) ≤ dist x y
  unfolding dist_prod_def by (rule real_sqrt_sum_squares_ge1)

```

```

lemma dist_snd_le: dist (snd x) (snd y) ≤ dist x y
  unfolding dist_prod_def by (rule real_sqrt_sum_squares_ge2)

instance
proof
  fix x y :: 'a × 'b
  show dist x y = 0 ⟷ x = y
    unfolding dist_prod_def prod_eq_iff by simp
next
  fix x y z :: 'a × 'b
  show dist x y ≤ dist x z + dist y z
    unfolding dist_prod_def
    by (intro order_trans [OF real_sqrt_sum_squares_triangle_ineq]
      real_sqrt_le_mono add_mono power_mono dist_triangle2 zero_le_dist)
next
  fix S :: ('a × 'b) set
  have *: open S ⟷ (∀ x ∈ S. ∃ e > 0. ∀ y. dist y x < e ⟶ y ∈ S)
  proof
    assume open S show ∀ x ∈ S. ∃ e > 0. ∀ y. dist y x < e ⟶ y ∈ S
    proof
      fix x assume x ∈ S
      obtain A B where open A open B x ∈ A × B A × B ⊆ S
        using ⟨open S⟩ and ⟨x ∈ S⟩ by (rule open_prod_elim)
      obtain r where r: 0 < r ∀ y. dist y (fst x) < r ⟶ y ∈ A
        using ⟨open A⟩ and ⟨x ∈ A × B⟩ unfolding open_dist by auto
      obtain s where s: 0 < s ∀ y. dist y (snd x) < s ⟶ y ∈ B
        using ⟨open B⟩ and ⟨x ∈ A × B⟩ unfolding open_dist by auto
      let ?e = min r s
      have 0 < ?e ∧ (∀ y. dist y x < ?e ⟶ y ∈ S)
      proof (intro allI impI conjI)
        show 0 < min r s by (simp add: r(1) s(1))
      next
        fix y assume dist y x < min r s
        hence dist y x < r and dist y x < s
          by simp_all
        hence dist (fst y) (fst x) < r and dist (snd y) (snd x) < s
          by (auto intro: le_less_trans dist_fst_le dist_snd_le)
        hence fst y ∈ A and snd y ∈ B
          by (simp_all add: r(2) s(2))
        hence y ∈ A × B by (induct y, simp)
        with ⟨A × B ⊆ S⟩ show y ∈ S ..
      qed
      thus ∃ e > 0. ∀ y. dist y x < e ⟶ y ∈ S ..
    qed
  next
    assume *: ∀ x ∈ S. ∃ e > 0. ∀ y. dist y x < e ⟶ y ∈ S show open S
    proof (rule open_prod_intro)
      fix x assume x ∈ S

```



```

then obtain e where 0 < e and S:  $\forall y. \text{dist } y \ x < e \longrightarrow y \in S$ 
  using * by fast
define r where  $r = e / \text{sqrt } 2$ 
define s where  $s = e / \text{sqrt } 2$ 
from 0 < e have 0 < r and 0 < s
  unfolding r_def s_def by simp_all
from 0 < e have  $e = \text{sqrt } (r^2 + s^2)$ 
  unfolding r_def s_def by (simp add: power_divide)
define A where  $A = \{y. \text{dist } (\text{fst } x) \ y < r\}$ 
define B where  $B = \{y. \text{dist } (\text{snd } x) \ y < s\}$ 
have open A and open B
  unfolding A_def B_def by (simp_all add: open_ball)
moreover have  $x \in A \times B$ 
  unfolding A_def B_def mem_Times_iff
  using 0 < r and 0 < s by simp
moreover have  $A \times B \subseteq S$ 
proof (clarify)
  fix a b assume  $a \in A$  and  $b \in B$ 
  hence  $\text{dist } a \ (\text{fst } x) < r$  and  $\text{dist } b \ (\text{snd } x) < s$ 
    unfolding A_def B_def by (simp_all add: dist_commute)
  hence  $\text{dist } (a, b) \ x < e$ 
    unfolding dist_prod_def 0 < e = sqrt (r^2 + s^2)
    by (simp add: add_strict_mono power_strict_mono)
  thus  $(a, b) \in S$ 
    by (simp add: S)
qed
ultimately show  $\exists A \ B. \text{open } A \wedge \text{open } B \wedge x \in A \times B \wedge A \times B \subseteq S$  by
fast
qed
qed
qed

end

declare [[code abort:  $\text{dist}::('a::\text{metric\_space} * 'b::\text{metric\_space}) \Rightarrow ('a * 'b) \Rightarrow \text{real}$ ]]

lemma Cauchy_fst:  $\text{Cauchy } X \Longrightarrow \text{Cauchy } (\lambda n. \text{fst } (X \ n :: 'a::\text{metric\_space} \times 'b::\text{metric\_space}))$ 
  unfolding Cauchy_def by (fast elim: le_less_trans [OF dist_fst_le])

lemma Cauchy_snd:  $\text{Cauchy } X \Longrightarrow \text{Cauchy } (\lambda n. \text{snd } (X \ n :: 'a::\text{metric\_space} \times 'b::\text{metric\_space}))$ 
  unfolding Cauchy_def by (fast elim: le_less_trans [OF dist_snd_le])

lemma Cauchy_Pair:
  assumes  $\text{Cauchy } X$  and  $\text{Cauchy } Y$ 
  shows  $\text{Cauchy } (\lambda n. (X \ n :: 'a::\text{metric\_space}, Y \ n :: 'a::\text{metric\_space}))$ 
proof (rule metric_CauchyI)
  fix r :: real assume 0 < r

```

hence $0 < r / \text{sqrt } 2$ (is $0 < ?s$) by *simp*
 obtain M where $M: \forall m \geq M. \forall n \geq M. \text{dist } (X \ m) (X \ n) < ?s$
 using *metric_CauchyD* [*OF* $\langle \text{Cauchy } X \rangle \langle 0 < ?s \rangle$] ..
 obtain N where $N: \forall m \geq N. \forall n \geq N. \text{dist } (Y \ m) (Y \ n) < ?s$
 using *metric_CauchyD* [*OF* $\langle \text{Cauchy } Y \rangle \langle 0 < ?s \rangle$] ..
 have $\forall m \geq \max M \ N. \forall n \geq \max M \ N. \text{dist } (X \ m, Y \ m) (X \ n, Y \ n) < r$
 using $M \ N$ by (*simp add: real_sqrt_sum_squares_less dist_Pair_Pair*)
 then show $\exists n0. \forall m \geq n0. \forall n \geq n0. \text{dist } (X \ m, Y \ m) (X \ n, Y \ n) < r$..
 qed

Analogue to *uniformly_continuous_on_def* for two-argument functions.

lemma *uniformly_continuous_on_prod_metric*:
 fixes $f :: \langle 'a::\text{metric\_space} \times 'b::\text{metric\_space} \rangle \Rightarrow 'c::\text{metric\_space}$
 shows $\langle \text{uniformly\_continuous\_on } (S \times T) \ f \longleftrightarrow (\forall e > 0. \exists d > 0. \forall x \in S. \forall y \in S. \forall x' \in T. \forall y' \in T. \text{dist } x \ y < d \longrightarrow \text{dist } x' \ y' < d \longrightarrow \text{dist } (f \ (x, x')) (f \ (y, y')) < e) \rangle$
proof (*unfold uniformly_continuous_on_def, intro iffI impI allI*)
 fix $e :: \text{real}$
 assume $\langle e > 0 \rangle$ and $\langle \forall e > 0. \exists d > 0. \forall x \in S. \forall y \in S. \forall x' \in T. \forall y' \in T. \text{dist } x \ y < d \longrightarrow \text{dist } x' \ y' < d \longrightarrow \text{dist } (f \ (x, x')) (f \ (y, y')) < e \rangle$
 then obtain d where $\langle d > 0 \rangle$
 and $d: \langle \forall x \in S. \forall y \in S. \forall x' \in T. \forall y' \in T. \text{dist } x \ y < d \longrightarrow \text{dist } x' \ y' < d \longrightarrow \text{dist } (f \ (x, x')) (f \ (y, y')) < e \rangle$
 by *auto*
 show $\langle \exists d > 0. \forall x \in S \times T. \forall y \in S \times T. \text{dist } y \ x < d \longrightarrow \text{dist } (f \ y) (f \ x) < e \rangle$
 apply (*rule exI[of _ d]*)
 using $\langle d > 0 \rangle$ *d*[*rule_format*] apply *auto*
 by (*smt (verit, del_insts) distfst_le distsnd_le fst_conv snd_conv*)
 next
 fix $e :: \text{real}$
 assume $\langle e > 0 \rangle$ and $\langle \forall e > 0. \exists d > 0. \forall x \in S \times T. \forall x' \in S \times T. \text{dist } x' \ x < d \longrightarrow \text{dist } (f \ x') (f \ x) < e \rangle$
 then obtain d where $\langle d > 0 \rangle$ and $d: \langle \forall x \in S \times T. \forall x' \in S \times T. \text{dist } x' \ x < d \longrightarrow \text{dist } (f \ x') (f \ x) < e \rangle$
 by *auto*
 show $\langle \exists d > 0. \forall x \in S. \forall y \in S. \forall x' \in T. \forall y' \in T. \text{dist } x \ y < d \longrightarrow \text{dist } x' \ y' < d \longrightarrow \text{dist } (f \ (x, x')) (f \ (y, y')) < e \rangle$
proof (*intro exI conjI impI ballI*)
 from $\langle d > 0 \rangle$ show $\langle d / 2 > 0 \rangle$ by *auto*
 fix $x \ y \ x' \ y'$
 assume [*simp*]: $\langle x \in S \rangle \langle y \in S \rangle \langle x' \in T \rangle \langle y' \in T \rangle$
 assume $\langle \text{dist } x \ y < d / 2 \rangle$ and $\langle \text{dist } x' \ y' < d / 2 \rangle$
 then have $\langle \text{dist } (x, x') (y, y') < d \rangle$
 by (*simp add: dist_Pair_Pair sqrt_sum_squares_half_less*)
 with d show $\langle \text{dist } (f \ (x, x')) (f \ (y, y')) < e \rangle$
 by *auto*
 qed
 qed

Analogue to *isUCont_def* for two-argument functions.

```

lemma isUCont_prod_metric:
  fixes  $f :: \langle 'a::\text{metric\_space} \times 'b::\text{metric\_space} \Rightarrow 'c::\text{metric\_space} \rangle$ 
  shows  $\langle \text{isUCont } f \longleftrightarrow (\forall e>0. \exists d>0. \forall x. \forall y. \forall x'. \forall y'. \text{dist } x \ y < d \longrightarrow \text{dist } x' \ y' < d \longrightarrow \text{dist } (f \ (x, x')) \ (f \ (y, y')) < e) \rangle$ 
  using uniformly_continuous_on_prod_metric[of UNIV UNIV]
  by auto

```

This logically belong with the real vector spaces by we only have the necessary lemmas now.

```

lemma isUCont_plus[simp]:
  shows  $\langle \text{isUCont } (\lambda(x::'a::\text{real\_normed\_vector}, y). \ x+y) \rangle$ 
proof (rule isUCont_prod_metric[THEN iffD2], intro allI impI, simp)
  fix  $e :: \text{real}$  assume  $\langle 0 < e \rangle$ 
  show  $\langle \exists d>0. \forall x \ y :: 'a. \text{dist } x \ y < d \longrightarrow (\forall x' \ y'. \text{dist } x' \ y' < d \longrightarrow \text{dist } (x + x') \ (y + y') < e) \rangle$ 
  apply (rule exI[of _ <e/2>])
  using  $\langle 0 < e \rangle$  apply auto
  by (smt (verit, ccfv_SIG) dist_add_cancel dist_add_cancel2 dist_commute dist_triangle_lt)
qed

```

1.3.4 Product is a Complete Metric Space

```

instance prod :: (complete_space, complete_space) complete_space
proof
  fix  $X :: \text{nat} \Rightarrow 'a \times 'b$  assume Cauchy X
  have 1:  $(\lambda n. \text{fst } (X \ n)) \longrightarrow \lim (\lambda n. \text{fst } (X \ n))$ 
  using Cauchy_fst [OF <Cauchy X>]
  by (simp add: Cauchy_convergent_iff convergent_LIMSEQ_iff)
  have 2:  $(\lambda n. \text{snd } (X \ n)) \longrightarrow \lim (\lambda n. \text{snd } (X \ n))$ 
  using Cauchy_snd [OF <Cauchy X>]
  by (simp add: Cauchy_convergent_iff convergent_LIMSEQ_iff)
  have  $X \longrightarrow (\lim (\lambda n. \text{fst } (X \ n)), \lim (\lambda n. \text{snd } (X \ n)))$ 
  using tendsto_Pair [OF 1 2] by simp
  then show convergent X
  by (rule convergentI)
qed

```

1.3.5 Product is a Normed Vector Space

```

instantiation prod :: (real_normed_vector, real_normed_vector) real_normed_vector
begin

```

```

definition norm_prod_def[code del]:
   $\text{norm } x = \text{sqrt } ((\text{norm } (\text{fst } x))^2 + (\text{norm } (\text{snd } x))^2)$ 

```

```

definition sgn_prod_def:
   $\text{sgn } (x::'a \times 'b) = \text{scaleR } (\text{inverse } (\text{norm } x)) \ x$ 

```

proposition *norm_Pair*: $\text{norm } (a, b) = \text{sqrt } ((\text{norm } a)^2 + (\text{norm } b)^2)$
unfolding *norm_prod_def* **by** *simp*

instance

proof

```

fix r :: real and x y :: 'a × 'b
show  $\text{norm } x = 0 \longleftrightarrow x = 0$ 
  unfolding norm_prod_def
  by (simp add: prod_eq_iff)
show  $\text{norm } (x + y) \leq \text{norm } x + \text{norm } y$ 
  unfolding norm_prod_def
  apply (rule order_trans [OF _ real_sqrt_sum_squares_triangle_ineq])
  apply (simp add: add_mono power_mono norm_triangle_ineq)
  done
show  $\text{norm } (\text{scaleR } r \ x) = |r| * \text{norm } x$ 
  unfolding norm_prod_def
  apply (simp add: power_mult_distrib)
  apply (simp add: distrib_left [symmetric])
  apply (simp add: real_sqrt_mult)
  done
show  $\text{sgn } x = \text{scaleR } (\text{inverse } (\text{norm } x)) \ x$ 
  by (rule sgn_prod_def)
show  $\text{dist } x \ y = \text{norm } (x - y)$ 
  unfolding dist_prod_def norm_prod_def
  by (simp add: dist_norm)

```

qed

end

declare [[*code abort: norm::('a::real_normed_vector*'b::real_normed_vector) ⇒ real*]]

instance *prod* :: (*banach, banach*) *banach* ..

Pair operations are linear

lemma *bounded_linear_fst*: *bounded_linear fst*

using *fst_add fst_scaleR*

by (*rule bounded_linear_intro [where K=1], simp add: norm_prod_def*)

lemma *bounded_linear_snd*: *bounded_linear snd*

using *snd_add snd_scaleR*

by (*rule bounded_linear_intro [where K=1], simp add: norm_prod_def*)

lemmas *bounded_linear_fst_comp* = *bounded_linear_fst [THEN bounded_linear_compose]*

lemmas *bounded_linear_snd_comp* = *bounded_linear_snd [THEN bounded_linear_compose]*

lemma *bounded_linear_Pair*:

```

assumes f: bounded_linear f
assumes g: bounded_linear g
shows bounded_linear ( $\lambda x. (f x, g x)$ )
proof
  interpret f: bounded_linear f by fact
  interpret g: bounded_linear g by fact
  fix x y and r :: real
  show (f (x + y), g (x + y)) = (f x, g x) + (f y, g y)
    by (simp add: f.add g.add)
  show (f (r *R x), g (r *R x)) = r *R (f x, g x)
    by (simp add: f.scale g.scale)
  obtain Kf where 0 < Kf and norm_f:  $\bigwedge x. \text{norm } (f x) \leq \text{norm } x * Kf$ 
    using f.pos_bounded by fast
  obtain Kg where 0 < Kg and norm_g:  $\bigwedge x. \text{norm } (g x) \leq \text{norm } x * Kg$ 
    using g.pos_bounded by fast
  have  $\forall x. \text{norm } (f x, g x) \leq \text{norm } x * (Kf + Kg)$ 
    apply (rule allI)
    apply (simp add: norm_Pair)
    apply (rule order_trans [OF sqrt_add_le_add_sqrt], simp, simp)
    apply (simp add: distrib_left)
    apply (rule add_mono [OF norm_f norm_g])
    done
  then show  $\exists K. \forall x. \text{norm } (f x, g x) \leq \text{norm } x * K$  ..
qed

```

Frechet derivatives involving pairs

```

proposition has_derivative_Pair [derivative_intros]:
  assumes f: (f has_derivative f') (at x within s)
    and g: (g has_derivative g') (at x within s)
  shows (( $\lambda x. (f x, g x)$ ) has_derivative ( $\lambda h. (f' h, g' h)$ )) (at x within s)
proof (rule has_derivativeI_sandwich[of 1])
  show bounded_linear ( $\lambda h. (f' h, g' h)$ )
    using f g by (intro bounded_linear_Pair has_derivative_bounded_linear)
  let ?Rf =  $\lambda y. f y - f x - f' (y - x)$ 
  let ?Rg =  $\lambda y. g y - g x - g' (y - x)$ 
  let ?R =  $\lambda y. ((f y, g y) - (f x, g x) - (f' (y - x), g' (y - x)))$ 

  show (( $\lambda y. \text{norm } (?Rf y) / \text{norm } (y - x) + \text{norm } (?Rg y) / \text{norm } (y - x)$ )
     $\longrightarrow 0$ ) (at x within s)
    using f g by (intro tendsto_add_zero) (auto simp: has_derivative_iff_norm)

  fix y :: 'a assume y  $\neq$  x
  show  $\text{norm } (?R y) / \text{norm } (y - x) \leq \text{norm } (?Rf y) / \text{norm } (y - x) + \text{norm } (?Rg y) / \text{norm } (y - x)$ 
    unfolding add_divide_distrib [symmetric]
    by (simp add: norm_Pair divide_right_mono order_trans [OF sqrt_add_le_add_sqrt])
qed simp

```

lemma *differentiable_Pair* [*simp*, *derivative_intros*]:
 f differentiable at x within $s \implies g$ differentiable at x within $s \implies$
 $(\lambda x. (f\ x, g\ x))$ differentiable at x within s
unfolding *differentiable_def* **by** (*blast intro: has_derivative_Pair*)

lemmas *has_derivative_fst* [*derivative_intros*] = *bounded_linear.has_derivative*
[*OF bounded_linear_fst*]

lemmas *has_derivative_snd* [*derivative_intros*] = *bounded_linear.has_derivative*
[*OF bounded_linear_snd*]

lemma *has_derivative_split* [*derivative_intros*]:
 $((\lambda p. f\ (fst\ p)\ (snd\ p))\ has\_derivative\ f')\ F \implies ((\lambda (a, b). f\ a\ b)\ has\_derivative\ f')\ F$
unfolding *split_beta'*.

Vector derivatives involving pairs

lemma *has_vector_derivative_Pair* [*derivative_intros*]:
assumes (f *has_vector_derivative* f') (*at* x *within* s)
(g *has_vector_derivative* g') (*at* x *within* s)
shows $((\lambda x. (f\ x, g\ x))\ has\_vector\_derivative\ (f', g'))\ (at\ x\ within\ s)$
using *assms*
by (*auto simp: has_vector_derivative_def intro!: derivative_eq_intros*)

lemma
fixes $x :: 'a :: real\_normed\_vector$
shows *norm_Pair1* [*simp*]: $norm\ (0, x) = norm\ x$
and *norm_Pair2* [*simp*]: $norm\ (x, 0) = norm\ x$
by (*auto simp: norm_Pair*)

lemma *norm_commute*: $norm\ (x, y) = norm\ (y, x)$
by (*simp add: norm_Pair*)

lemma *norm_fst_le*: $norm\ x \leq norm\ (x, y)$
by (*metis dist_fst_le fst_conv fst_zero norm_conv_dist*)

lemma *norm_snd_le*: $norm\ y \leq norm\ (x, y)$
by (*metis dist_snd_le snd_conv snd_zero norm_conv_dist*)

lemma *norm_Pair_le*:
shows $norm\ (x, y) \leq norm\ x + norm\ y$
unfolding *norm_Pair*
by (*metis norm_ge_zero sqrt_sum_squares_le_sum*)

lemma (*in* *vector_space_prod*) *span_Times_sing1*: $p.span\ (\{0\} \times B) = \{0\} \times vs2.span\ B$
apply (*rule p.span_unique*)
subgoal by (*auto intro!: vs1.span_base vs2.span_base*)
subgoal using *vs1.subspace_single_0 vs2.subspace_span* **by** (*rule subspace_Times*)

```

subgoal for T
proof safe
  fix b
  assume subset_T:  $\{0\} \times B \subseteq T$  and subspace:  $p.\text{subspace } T$  and b_span:  $b \in vs2.\text{span } B$ 
  then obtain t r where b:  $b = (\sum a \in t. r \ a * b \ a)$  and t:  $\text{finite } t \ t \subseteq B$ 
    by (auto simp: vs2.span_explicit)
  have  $(0, b) = (\sum b \in t. \text{scale } (r \ b) \ (0, b))$ 
    unfolding b scale_prod sum_prod
    by simp
  also have  $\dots \in T$ 
    using  $\langle t \subseteq B \rangle$  subset_T
    by (auto intro!: p.subspace_sum p.subspace_scale subspace)
  finally show  $(0, b) \in T$  .
qed
done

lemma (in vector_space_prod) span_Times_sing2:  $p.\text{span } (A \times \{0\}) = vs1.\text{span } A \times \{0\}$ 
proof
  apply (rule p.span_unique)
  subgoal by (auto intro!: vs1.span_base vs2.span_base)
  subgoal using vs1.subspace_span vs2.subspace_single_0 by (rule subspace_Times)
  subgoal for T
  proof safe
    fix a
    assume subset_T:  $A \times \{0\} \subseteq T$  and subspace:  $p.\text{subspace } T$  and a_span:  $a \in vs1.\text{span } A$ 
    then obtain t r where a:  $a = (\sum a \in t. r \ a * a \ a)$  and t:  $\text{finite } t \ t \subseteq A$ 
      by (auto simp: vs1.span_explicit)
    have  $(a, 0) = (\sum a \in t. \text{scale } (r \ a) \ (a, 0))$ 
      unfolding a scale_prod sum_prod
      by simp
    also have  $\dots \in T$ 
      using  $\langle t \subseteq A \rangle$  subset_T
      by (auto intro!: p.subspace_sum p.subspace_scale subspace)
    finally show  $(a, 0) \in T$  .
  qed
done

```

1.3.6 Product is Finite Dimensional

```

lemma (in finite_dimensional_vector_space) zero_not_in_Basis[simp]:  $0 \notin \text{Basis}$ 
  using dependent_zero local.independent_Basis by blast

```

```

locale finite_dimensional_vector_space_prod = vector_space_prod + finite_dimensional_vector_space_pair
begin

```

```

definition Basis_pair =  $B1 \times \{0\} \cup \{0\} \times B2$ 

```

```

sublocale p: finite_dimensional_vector_space scale Basis_pair
proof unfold_locales
  show finite Basis_pair
    by (auto intro!: finite_cartesian_product vs1.finite_Basis vs2.finite_Basis
simp: Basis_pair_def)
  show p.independent Basis_pair
    unfolding p.dependent_def Basis_pair_def
  proof safe
    fix a
    assume a:  $a \in B1$ 
    assume  $(a, 0) \in p.\text{span } (B1 \times \{0\} \cup \{0\} \times B2 - \{(a, 0)\})$ 
    also have  $B1 \times \{0\} \cup \{0\} \times B2 - \{(a, 0)\} = (B1 - \{a\}) \times \{0\} \cup \{0\} \times$ 
B2
      by auto
    finally show False
      using a vs1.dependent_def vs1.independent_Basis
      by (auto simp: p.span_Un span_Times_sing1 span_Times_sing2)
  next
    fix b
    assume b:  $b \in B2$ 
    assume  $(0, b) \in p.\text{span } (B1 \times \{0\} \cup \{0\} \times B2 - \{(0, b)\})$ 
    also have  $(B1 \times \{0\} \cup \{0\} \times B2 - \{(0, b)\}) = B1 \times \{0\} \cup \{0\} \times (B2 -$ 
 $\{b\})$ 
      by auto
    finally show False
      using b vs2.dependent_def vs2.independent_Basis
      by (auto simp: p.span_Un span_Times_sing1 span_Times_sing2)
  qed
show p.span Basis_pair = UNIV
  by (auto simp: p.span_Un span_Times_sing2 span_Times_sing1 vs1.span_Basis
vs2.span_Basis
Basis_pair_def)
qed

proposition dim_Times:
assumes vs1.subspace S vs2.subspace T
shows  $p.\text{dim}(S \times T) = vs1.\text{dim } S + vs2.\text{dim } T$ 
proof –
  interpret p1: Vector_Spaces.linear s1 scale  $(\lambda x. (x, 0))$ 
    by unfold_locales (auto simp: scale_def)
  interpret pair1: finite_dimensional_vector_space_pair  $(*a) B1$  scale Basis_pair
    by unfold_locales
  interpret p2: Vector_Spaces.linear s2 scale  $(\lambda x. (0, x))$ 
    by unfold_locales (auto simp: scale_def)
  interpret pair2: finite_dimensional_vector_space_pair  $(*b) B2$  scale Basis_pair
    by unfold_locales
  have ss: p.subspace  $((\lambda x. (x, 0)) \text{ ‘ } S)$  p.subspace  $(\text{Pair } 0 \text{ ‘ } T)$ 
    by (rule p1.subspace_image p2.subspace_image assms) +

```



```

  have  $p.\dim(S \times T) = p.\dim(\{u + v \mid u \in (\lambda x. (x, 0)) \text{ ' } S \wedge v \in \text{Pair } 0 \text{ ' } T\})$ 
  by (simp add: Times_eq_image_sum)
  moreover have  $p.\dim((\lambda x. (x, 0::'c)) \text{ ' } S) = vs1.\dim S \ p.\dim(\text{Pair } (0::'b) \text{ ' } T)$ 
  by (simp_all add: inj_on_def p1.linear_axioms pair1.dim_image_eq p2.linear_axioms pair2.dim_image_eq)
  moreover have  $p.\dim((\lambda x. (x, 0)) \text{ ' } S \cap \text{Pair } 0 \text{ ' } T) = 0$ 
  by (subst p.dim_eq_0) auto
  ultimately show ?thesis
  using p.dim_sums_Int [OF ss] by linarith
qed

lemma dimension_pair:  $p.\text{dimension} = vs1.\text{dimension} + vs2.\text{dimension}$ 
  using dim_Times[OF vs1.subspace_UNIV vs2.subspace_UNIV]
  by (auto simp: p.dimension_def vs1.dimension_def vs2.dimension_def)

end

end

```

1.4 Finite-Dimensional Inner Product Spaces

```

theory Euclidean_Space
imports
  L2_Norm
  Inner_Product
  Product_Vector
begin

```

1.4.1 Interlude: Some properties of real sets

```

lemma seq_mono_lemma:
  assumes  $\forall (n::nat) \geq m. (d\ n :: real) < e\ n$ 
  and  $\forall n \geq m. e\ n \leq e\ m$ 
  shows  $\forall n \geq m. d\ n < e\ m$ 
  using assms by force

```

1.4.2 Type class of Euclidean spaces

```

class euclidean_space = real_inner +
  fixes Basis :: 'a set
  assumes nonempty_Basis [simp]:  $\text{Basis} \neq \{\}$ 
  assumes finite_Basis [simp]:  $\text{finite Basis}$ 
  assumes inner_Basis:
     $\llbracket u \in \text{Basis}; v \in \text{Basis} \rrbracket \implies \text{inner } u\ v = (\text{if } u = v \text{ then } 1 \text{ else } 0)$ 
  assumes euclidean_all_zero_iff:
     $(\forall u \in \text{Basis}. \text{inner } x\ u = 0) \longleftrightarrow (x = 0)$ 

```

```

syntax _type_dimension :: type  $\Rightarrow$  nat ((1DIM/(1'(_'))))
translations DIM('a)  $\rightarrow$  CONST card (CONST Basis :: 'a set)
typed_print_translation  $\langle$ 
  [(const_syntax  $\langle$ card $\rangle$ ,
    fn ctxt  $\Rightarrow$  fn _  $\Rightarrow$  fn [Const (const_syntax  $\langle$ Basis $\rangle$ , Type (type_name  $\langle$ set $\rangle$ ,
      [T]))]  $\Rightarrow$ 
      Syntax.const syntax_const  $\langle$ _type_dimension $\rangle$  $ Syntax_Phases.term_of_typ
        ctxt T)]
   $\rangle$ 

```

```

lemma (in euclidean_space) norm_Basis[simp]:  $u \in \text{Basis} \implies \text{norm } u = 1$ 
unfolding norm_eq_sqrt_inner by (simp add: inner_Basis)

```

```

lemma (in euclidean_space) inner_same_Basis[simp]:  $u \in \text{Basis} \implies \text{inner } u \ u = 1$ 
by (simp add: inner_Basis)

```

```

lemma (in euclidean_space) inner_not_same_Basis:  $u \in \text{Basis} \implies v \in \text{Basis} \implies u \neq v \implies \text{inner } u \ v = 0$ 
by (simp add: inner_Basis)

```

```

lemma (in euclidean_space) sgn_Basis:  $u \in \text{Basis} \implies \text{sgn } u = u$ 
unfolding sgn_div_norm by (simp add: scaleR_one)

```

```

lemma (in euclidean_space) Basis_zero [simp]:  $0 \notin \text{Basis}$ 
proof
  assume  $0 \in \text{Basis}$  thus False
  using inner_Basis [of 0 0] by simp
qed

```

```

lemma (in euclidean_space) nonzero_Basis:  $u \in \text{Basis} \implies u \neq 0$ 
by clarsimp

```

```

lemma (in euclidean_space) SOME_Basis: (SOME  $i$ .  $i \in \text{Basis}$ )  $\in \text{Basis}$ 
by (metis ex_in_conv nonempty_Basis someI_ex)

```

```

lemma norm_some_Basis [simp]:  $\text{norm } (\text{SOME } i. i \in \text{Basis}) = 1$ 
by (simp add: SOME_Basis)

```

```

lemma (in euclidean_space) inner_sum_left_Basis[simp]:
   $b \in \text{Basis} \implies \text{inner } (\sum_{i \in \text{Basis}} f \ i *_{\mathbb{R}} i) \ b = f \ b$ 
by (simp add: inner_sum_left inner_Basis if_distrib comm_monoid_add_class.sum.If_cases)

```

```

lemma (in euclidean_space) euclidean_eqI:
  assumes  $b: \bigwedge b. b \in \text{Basis} \implies \text{inner } x \ b = \text{inner } y \ b$  shows  $x = y$ 
proof -
  from  $b$  have  $\forall b \in \text{Basis}. \text{inner } (x - y) \ b = 0$ 
  by (simp add: inner_diff_left)
  then show  $x = y$ 

```

by (simp add: euclidean_all_zero_iff)
qed

lemma (in euclidean_space) euclidean_eq_iff:
 $x = y \longleftrightarrow (\forall b \in \text{Basis}. \text{inner } x \ b = \text{inner } y \ b)$
 by (auto intro: euclidean_eqI)

lemma (in euclidean_space) euclidean_representation_sum:
 $(\sum_{i \in \text{Basis}} f \ i *_{\mathbb{R}} i) = b \longleftrightarrow (\forall i \in \text{Basis}. f \ i = \text{inner } b \ i)$
 by (subst euclidean_eq_iff) simp

lemma (in euclidean_space) euclidean_representation_sum':
 $b = (\sum_{i \in \text{Basis}} f \ i *_{\mathbb{R}} i) \longleftrightarrow (\forall i \in \text{Basis}. f \ i = \text{inner } b \ i)$
 by (auto simp add: euclidean_representation_sum[symmetric])

lemma (in euclidean_space) euclidean_representation: $(\sum_{b \in \text{Basis}} \text{inner } x \ b *_{\mathbb{R}} b) = x$
 unfolding euclidean_representation_sum by simp

lemma (in euclidean_space) euclidean_inner: $\text{inner } x \ y = (\sum_{b \in \text{Basis}} (\text{inner } x \ b) * (\text{inner } y \ b))$
 by (subst (1 2) euclidean_representation [symmetric])
 (simp add: inner_sum_right inner_Basis ac_simps)

lemma (in euclidean_space) choice_Basis_iff:
 fixes $P :: 'a \Rightarrow \text{real} \Rightarrow \text{bool}$
 shows $(\forall i \in \text{Basis}. \exists x. P \ i \ x) \longleftrightarrow (\exists x. \forall i \in \text{Basis}. P \ i \ (\text{inner } x \ i))$
 unfolding bchoice_iff
 proof safe
 fix f assume $\forall i \in \text{Basis}. P \ i \ (f \ i)$
 then show $\exists x. \forall i \in \text{Basis}. P \ i \ (\text{inner } x \ i)$
 by (auto intro!: exI[of _ $\sum_{i \in \text{Basis}} f \ i *_{\mathbb{R}} i$])
 qed auto

lemma (in euclidean_space) bchoice_Basis_iff:
 fixes $P :: 'a \Rightarrow \text{real} \Rightarrow \text{bool}$
 shows $(\forall i \in \text{Basis}. \exists x \in A. P \ i \ x) \longleftrightarrow (\exists x. \forall i \in \text{Basis}. \text{inner } x \ i \in A \wedge P \ i \ (\text{inner } x \ i))$
 by (simp add: choice_Basis_iff Bex_def)

lemma (in euclidean_space) euclidean_representation_sum_fun:
 $(\lambda x. \sum_{b \in \text{Basis}} \text{inner } (f \ x) \ b *_{\mathbb{R}} b) = f$
 by (rule ext) (simp add: euclidean_representation_sum)

lemma euclidean_isCont:
 assumes $\bigwedge b. b \in \text{Basis} \implies \text{isCont } (\lambda x. (\text{inner } (f \ x) \ b) *_{\mathbb{R}} b) \ x$
 shows $\text{isCont } f \ x$
 apply (subst euclidean_representation_sum_fun [symmetric])
 apply (rule isCont_sum)

apply (*blast intro: assms*)
done

lemma *DIM_positive* [*simp*]: $0 < DIM('a::euclidean\_space)$
by (*simp add: card_gt_0_iff*)

lemma *DIM_ge_Suc0* [*simp*]: $Suc\ 0 \leq card\ Basis$
by (*meson DIM_positive Suc_leI*)

lemma *sum_inner_Basis_scaleR* [*simp*]:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::real\_vector$
assumes $b \in Basis$ **shows** $(\sum i \in Basis. (inner\ i\ b) *_{\mathbb{R}} f\ i) = f\ b$
by (*simp add: comm_monoid_add_class.sum.remove [OF finite_Basis assms]*
assms inner_not_same_Basis comm_monoid_add_class.sum.neutral)

lemma *sum_inner_Basis_eq* [*simp*]:
assumes $b \in Basis$ **shows** $(\sum i \in Basis. (inner\ i\ b) * f\ i) = f\ b$
by (*simp add: comm_monoid_add_class.sum.remove [OF finite_Basis assms]*
assms inner_not_same_Basis comm_monoid_add_class.sum.neutral)

lemma *sum_if_inner* [*simp*]:
assumes $i \in Basis\ j \in Basis$
shows $inner\ (\sum k \in Basis. if\ k = i\ then\ f\ i *_{\mathbb{R}} i\ else\ g\ k *_{\mathbb{R}} k)\ j = (if\ j=i\ then\ f\ j\ else\ g\ j)$
proof (*cases i=j*)
case *True*
with *assms* **show** ?thesis
by (*auto simp: inner_sum_left if_distrib [of $\lambda x. inner\ x\ j$] inner_Basis cong: if_cong*)
next
case *False*
have $(\sum k \in Basis. inner\ (if\ k = i\ then\ f\ i *_{\mathbb{R}} i\ else\ g\ k *_{\mathbb{R}} k)\ j) =$
 $(\sum k \in Basis. if\ k = j\ then\ g\ k\ else\ 0)$
apply (*rule sum.cong*)
using *False assms* **by** (*auto simp: inner_Basis*)
also **have** $\dots = g\ j$
using *assms* **by** *auto*
finally **show** ?thesis
using *False* **by** (*auto simp: inner_sum_left*)
qed

lemma *norm_le_componentwise*:
 $(\bigwedge b. b \in Basis \implies abs(inner\ x\ b) \leq abs(inner\ y\ b)) \implies norm\ x \leq norm\ y$
by (*auto simp: norm_le euclidean_inner [of $x\ x$] euclidean_inner [of $y\ y$] abs_le_square_iff power2_eq_square intro!: sum_mono*)

lemma *Basis_le_norm*: $b \in Basis \implies |inner\ x\ b| \leq norm\ x$
by (*rule order_trans [OF Cauchy_Schwarz_ineq2] simp*)

lemma *norm_bound_Basis_le*: $b \in \text{Basis} \implies \text{norm } x \leq e \implies |\text{inner } x \ b| \leq e$
by (*metis Basis_le_norm order_trans*)

lemma *norm_bound_Basis_lt*: $b \in \text{Basis} \implies \text{norm } x < e \implies |\text{inner } x \ b| < e$
by (*metis Basis_le_norm le_less_trans*)

lemma *norm_le_l1*: $\text{norm } x \leq (\sum_{b \in \text{Basis}} |\text{inner } x \ b|)$
apply (*subst euclidean_representation[of x, symmetric]*)
apply (*rule order_trans[OF norm_sum]*)
apply (*auto intro!: sum_mono*)
done

lemma *sum_norm_allsubsets_bound*:
fixes $f :: 'a \Rightarrow 'n::\text{euclidean\_space}$
assumes fP : *finite P*
and fPs : $\bigwedge Q. Q \subseteq P \implies \text{norm } (\text{sum } f \ Q) \leq e$
shows $(\sum_{x \in P} \text{norm } (f \ x)) \leq 2 * \text{real } \text{DIM}('n) * e$
proof –
have $(\sum_{x \in P} \text{norm } (f \ x)) \leq (\sum_{x \in P} \sum_{b \in \text{Basis}} |\text{inner } (f \ x) \ b|)$
by (*rule sum_mono*) (*rule norm_le_l1*)
also have $(\sum_{x \in P} \sum_{b \in \text{Basis}} |\text{inner } (f \ x) \ b|) = (\sum_{b \in \text{Basis}} \sum_{x \in P} |\text{inner } (f \ x) \ b|)$
by (*rule sum.swap*)
also have $\dots \leq \text{of\_nat } (\text{card } (\text{Basis} :: 'n \ \text{set})) * (2 * e)$
proof (*rule sum_bounded_above*)
fix $i :: 'n$
assume i : $i \in \text{Basis}$
have $\text{norm } (\sum_{x \in P} |\text{inner } (f \ x) \ i|) \leq$
 $\text{norm } (\text{inner } (\sum_{x \in P \cap -\{x. \text{inner } (f \ x) \ i < 0\}} f \ x) \ i) + \text{norm } (\text{inner } (\sum_{x \in P \cap \{x. \text{inner } (f \ x) \ i < 0\}} f \ x) \ i)$
by (*simp add: abs_real_def sum.If_cases[OF fP] sum_negf norm_triangle_ineq4 inner_sum_left del: real_norm_def*)
also have $\dots \leq e + e$
unfolding *real_norm_def*
by (*intro add_mono norm_bound_Basis_le i fPs*) *auto*
finally show $(\sum_{x \in P} |\text{inner } (f \ x) \ i|) \leq 2 * e$ **by** *simp*
qed
also have $\dots = 2 * \text{real } \text{DIM}('n) * e$ **by** *simp*
finally show *?thesis* .
qed

1.4.3 Subclass relationships

instance *euclidean_space* $\subseteq$ *perfect_space*

proof

fix $x :: 'a$ **show** $\neg \text{open } \{x\}$

proof

```

    assume open {x}
    then obtain e where 0 < e and e:  $\forall y. \text{dist } y \ x < e \longrightarrow y = x$ 
      unfolding open_dist by fast
    define y where  $y = x + \text{scaleR } (e/2) \ (SOME \ b. \ b \in \text{Basis})$ 
    have [simp]:  $(SOME \ b. \ b \in \text{Basis}) \in \text{Basis}$ 
      by (rule someI_ex) (auto simp: ex_in_conv)
    from  $\langle 0 < e \rangle$  have  $y \neq x$ 
      unfolding y_def by (auto intro!: nonzero_Basis)
    from  $\langle 0 < e \rangle$  have  $\text{dist } y \ x < e$ 
      unfolding y_def by (simp add: dist_norm)
    from  $\langle y \neq x \rangle$  and  $\langle \text{dist } y \ x < e \rangle$  show False
      using e by simp
  qed
qed

```

1.4.4 Class instances

Type *real*

instantiation *real* :: *euclidean_space*
begin

definition

[simp]: $\text{Basis} = \{1 :: \text{real}\}$

instance

by standard auto

end

lemma *DIM_real*[simp]: $\text{DIM}(\text{real}) = 1$

by simp

Type *complex*

instantiation *complex* :: *euclidean_space*
begin

definition *Basis_complex_def*: $\text{Basis} = \{1, i\}$

instance

by standard (auto simp add: *Basis_complex_def* intro: *complex_eqI* split: *if_split_asm*)

end

lemma *DIM_complex*[simp]: $\text{DIM}(\text{complex}) = 2$

unfolding *Basis_complex_def* by simp

lemma *complex_Basis_1* [iff]: $(1 :: \text{complex}) \in \text{Basis}$

by (simp add: *Basis_complex_def*)

lemma *complex_Basis_i [iff]: $i \in \text{Basis}$*
by (*simp add: Basis_complex_def*)

Type $'a \times 'b$

instantiation *prod* :: (*real_inner*, *real_inner*) *real_inner*
begin

definition *inner_prod_def*:
 $\text{inner } x \ y = \text{inner } (\text{fst } x) \ (\text{fst } y) + \text{inner } (\text{snd } x) \ (\text{snd } y)$

lemma *inner_Pair [simp]: $\text{inner } (a, b) \ (c, d) = \text{inner } a \ c + \text{inner } b \ d$*
unfolding *inner_prod_def* **by** *simp*

instance

proof

fix *r* :: *real*
fix *x y z* :: $'a::\text{real\_inner} \times 'b::\text{real\_inner}$
show $\text{inner } x \ y = \text{inner } y \ x$
unfolding *inner_prod_def*
by (*simp add: inner_commute*)
show $\text{inner } (x + y) \ z = \text{inner } x \ z + \text{inner } y \ z$
unfolding *inner_prod_def*
by (*simp add: inner_add_left*)
show $\text{inner } (\text{scaleR } r \ x) \ y = r * \text{inner } x \ y$
unfolding *inner_prod_def*
by (*simp add: distrib_left*)
show $0 \leq \text{inner } x \ x$
unfolding *inner_prod_def*
by (*intro add_nonneg_nonneg inner_ge_zero*)
show $\text{inner } x \ x = 0 \longleftrightarrow x = 0$
unfolding *inner_prod_def prod_eq_iff*
by (*simp add: add_nonneg_eq_0_iff*)
show $\text{norm } x = \text{sqrt } (\text{inner } x \ x)$
unfolding *norm_prod_def inner_prod_def*
by (*simp add: power2_norm_eq_inner*)

qed

end

lemma *inner_Pair_0: $\text{inner } x \ (0, b) = \text{inner } (\text{snd } x) \ b$ $\text{inner } x \ (a, 0) = \text{inner } (\text{fst } x) \ a$*
by (*cases x, simp*)**+**

instantiation *prod* :: (*euclidean_space*, *euclidean_space*) *euclidean_space*
begin

definition

$$\text{Basis} = (\lambda u. (u, 0)) \text{ 'Basis} \cup (\lambda v. (0, v)) \text{ 'Basis}$$

```

lemma sum_Basis_prod_eq:
  fixes f::('a*'b)⇒('a*'b)
  shows sum f Basis = sum (λi. f (i, 0)) Basis + sum (λi. f (0, i)) Basis
proof -
  have inj_on (λu. (u::'a, 0::'b)) Basis inj_on (λu. (0::'a, u::'b)) Basis
  by (auto intro!: inj_onI Pair_inject)
  thus ?thesis
  unfolding Basis_prod_def
  by (subst sum.union_disjoint) (auto simp: Basis_prod_def sum.reindex)
qed

```

```

instance proof
  show (Basis :: ('a × 'b) set) ≠ {}
  unfolding Basis_prod_def by simp
next
  show finite (Basis :: ('a × 'b) set)
  unfolding Basis_prod_def by simp
next
  fix u v :: 'a × 'b
  assume u ∈ Basis and v ∈ Basis
  thus inner u v = (if u = v then 1 else 0)
  unfolding Basis_prod_def inner_prod_def
  by (auto simp add: inner_Basis split: if_split_asm)
next
  fix x :: 'a × 'b
  show (∀ u ∈ Basis. inner x u = 0) ⟷ x = 0
  unfolding Basis_prod_def ball_Un ball_simps
  by (simp add: inner_prod_def prod_eq_iff euclidean_all_zero_iff)
qed

```

```

lemma DIM_prod[simp]: DIM('a × 'b) = DIM('a) + DIM('b)
  unfolding Basis_prod_def
  by (subst card_Un_disjoint) (auto intro!: card_image arg_cong2[where f=(+)]
  inj_onI)

```

end

1.4.5 Locale instances

```

lemma finite_dimensional_vector_space_euclidean:
  finite_dimensional_vector_space (*R) Basis
proof unfold_locales
  show finite (Basis::'a set) by (metis finite_Basis)
  show real_vector.independent (Basis::'a set)
  unfolding dependent_def dependent_raw_def[symmetric]
  apply (subst span_finite)
  apply simp

```



```

    apply clarify
    apply (drule_tac f=inner a in arg_cong)
    apply (simp add: inner_Basis inner_sum_right eq_commute)
    done
show module.span (*R) Basis = UNIV
  unfolding span_finite [OF finite_Basis] span_raw_def[symmetric]
  by (auto intro!: euclidean_representation[symmetric])
qed

interpretation eucl?: finite_dimensional_vector_space scaleR :: real => 'a =>
'a::euclidean_space Basis
  rewrites module.dependent (*R) = dependent
    and module.representation (*R) = representation
    and module.subspace (*R) = subspace
    and module.span (*R) = span
    and vector_space.extend_basis (*R) = extend_basis
    and vector_space.dim (*R) = dim
    and Vector_Spaces.linear (*R) (*R) = linear
    and Vector_Spaces.linear (*) (*R) = linear
    and finite_dimensional_vector_space.dimension Basis = DIM('a)
    and dimension = DIM('a)
  by (auto simp add: dependent_raw_def representation_raw_def
    subspace_raw_def span_raw_def extend_basis_raw_def dim_raw_def lin-
ear_def
    real_scaleR_def[abs_def]
    finite_dimensional_vector_space.dimension_def
    intro!: finite_dimensional_vector_space.dimension_def
    finite_dimensional_vector_space_euclidean)

interpretation eucl?: finite_dimensional_vector_space_pair_1
scaleR::real=>'a::euclidean_space=>'a Basis
scaleR::real=>'b::real_vector => 'b
  by unfold_locales

interpretation eucl?: finite_dimensional_vector_space_prod scaleR scaleR Basis
Basis
  rewrites Basis_pair = Basis
    and module_prod.scale (*R) (*R) = (scaleR::_=>_=>('a × 'b))
proof –
  show finite_dimensional_vector_space_prod (*R) (*R) Basis Basis
    by unfold_locales
  interpret finite_dimensional_vector_space_prod (*R) (*R) Basis::'a set Ba-
sis::'b set
    by fact
  show Basis_pair = Basis
    unfolding Basis_pair_def Basis_prod_def by auto
  show module_prod.scale (*R) (*R) = scaleR
    by (fact module_prod_scale_eq_scaleR)
qed

```

end

1.5 Elementary Linear Algebra on Euclidean Spaces

theory *Linear_Algebra*

imports

Euclidean_Space

HOL-Library.Infinite_Set

begin

lemma *linear_simps*:

assumes *bounded_linear f*

shows

$$f (a + b) = f a + f b$$

$$f (a - b) = f a - f b$$

$$f 0 = 0$$

$$f (- a) = - f a$$

$$f (s *_R v) = s *_R (f v)$$

proof –

interpret *f*: *bounded_linear f* **by** *fact*

show $f (a + b) = f a + f b$ **by** (*rule f.add*)

show $f (a - b) = f a - f b$ **by** (*rule f.diff*)

show $f 0 = 0$ **by** (*rule f.zero*)

show $f (- a) = - f a$ **by** (*rule f.neg*)

show $f (s *_R v) = s *_R (f v)$ **by** (*rule f.scale*)

qed

lemma *finite_Atleast_Atmost_nat[simp]*: *finite* $\{f x \mid x. x \in (UNIV::'a::finite\ set)\}$

using *finite_finite_image_set* **by** *blast*

lemma *substdbasis_expansion_unique*:

includes *inner_syntax*

assumes $d: d \subseteq Basis$

shows $(\sum i \in d. f i *_R i) = (x::'a::euclidean\_space) \longleftrightarrow$

$$(\forall i \in Basis. (i \in d \longrightarrow f i = x \cdot i) \wedge (i \notin d \longrightarrow x \cdot i = 0))$$

proof –

have $*$: $\bigwedge x a b P. x * (if P then a else b) = (if P then x * a else x * b)$

by *auto*

have $**$: *finite d*

by (*auto intro: finite_subset[OF assms]*)

have $***$: $\bigwedge i. i \in Basis \implies (\sum i \in d. f i *_R i) \cdot i = (\sum x \in d. if x = i then f x else 0)$

using *d*

by (*auto intro!: sum.cong simp: inner_Basis inner_sum_left*)

show *?thesis*

unfolding *euclidean_eq_iff* [**where** $'a = 'a$] **by** (*auto simp: sum.delta[OF **]*)

$***$)

qed

lemma *independent_subdbasis*: $d \subseteq \text{Basis} \implies \text{independent } d$
by (rule *independent_mono*[OF *independent_Basis*])

lemma *subset_translation_eq* [simp]:
fixes $a :: 'a::\text{real\_vector}$ **shows** $(+) a \text{ ` } s \subseteq (+) a \text{ ` } t \longleftrightarrow s \subseteq t$
by *auto*

lemma *translate_inj_on*:
fixes $A :: 'a::\text{ab\_group\_add set}$
shows *inj_on* $(\lambda x. a + x) A$
unfolding *inj_on_def* **by** *auto*

lemma *translation_assoc*:
fixes $a b :: 'a::\text{ab\_group\_add}$
shows $(\lambda x. b + x) \text{ ` } ((\lambda x. a + x) \text{ ` } S) = (\lambda x. (a + b) + x) \text{ ` } S$
by *auto*

lemma *translation_invert*:
fixes $a :: 'a::\text{ab\_group\_add}$
assumes $(\lambda x. a + x) \text{ ` } A = (\lambda x. a + x) \text{ ` } B$
shows $A = B$
proof –
have $(\lambda x. -a + x) \text{ ` } ((\lambda x. a + x) \text{ ` } A) = (\lambda x. -a + x) \text{ ` } ((\lambda x. a + x) \text{ ` } B)$
using *assms* **by** *auto*
then show *?thesis*
using *translation_assoc*[of $-a$ a A] *translation_assoc*[of $-a$ a B] **by** *auto*
qed

lemma *translation_galois*:
fixes $a :: 'a::\text{ab\_group\_add}$
shows $T = ((\lambda x. a + x) \text{ ` } S) \longleftrightarrow S = ((\lambda x. (- a) + x) \text{ ` } T)$
using *translation_assoc*[of $-a$ a S]
apply *auto*
using *translation_assoc*[of a $-a$ T]
apply *auto*
done

lemma *translation_inverse_subset*:
assumes $((\lambda x. -a + x) \text{ ` } V) \leq (S :: 'n::\text{ab\_group\_add set})$
shows $V \leq ((\lambda x. a + x) \text{ ` } S)$
proof –
{
fix x
assume $x \in V$
then have $x - a \in S$ **using** *assms* **by** *auto*
then have $x \in \{a + v \mid v. v \in S\}$
apply *auto*
apply (rule *exI*[of $x - a$], *simp*)
}

```

    done
  then have  $x \in ((\lambda x. a+x) ' S)$  by auto
}
then show ?thesis by auto
qed

```

1.5.1 More interesting properties of the norm

unbundle *inner_syntax*

Equality of vectors in terms of $(\cdot)$ products.

```

lemma linear_componentwise:
  fixes  $f :: 'a :: euclidean\_space \Rightarrow 'b :: real\_inner$ 
  assumes  $lf: linear\ f$ 
  shows  $(f\ x) \cdot j = (\sum_{i \in Basis. (x \cdot i) * (f\ i \cdot j))$  (is ?lhs = ?rhs)
proof -
  interpret linear f by fact
  have ?rhs =  $(\sum_{i \in Basis. (x \cdot i) *_{\mathbb{R}} (f\ i) \cdot j)$ 
    by (simp add: inner_sum_left)
  then show ?thesis
    by (simp add: euclidean_representation sum[symmetric] scale[symmetric])
qed

```

```

lemma vector_eq:  $x = y \iff x \cdot x = x \cdot y \wedge y \cdot y = x \cdot x$ 
  (is ?lhs  $\iff$  ?rhs)

```

```

proof
  assume ?lhs
  then show ?rhs by simp
next
  assume ?rhs
  then have  $x \cdot x - x \cdot y = 0 \wedge x \cdot y - y \cdot y = 0$ 
    by simp
  then have  $x \cdot (x - y) = 0 \wedge y \cdot (x - y) = 0$ 
    by (simp add: inner_diff inner_commute)
  then have  $(x - y) \cdot (x - y) = 0$ 
    by (simp add: field_simps inner_diff inner_commute)
  then show  $x = y$  by simp
qed

```

```

lemma norm_triangle_half_r:
   $norm\ (y - x1) < e / 2 \implies norm\ (y - x2) < e / 2 \implies norm\ (x1 - x2) < e$ 
  using dist_triangle_half_r unfolding dist_norm[symmetric] by auto

```

```

lemma norm_triangle_half_l:
  assumes  $norm\ (x - y) < e / 2$ 
  and  $norm\ (x' - y) < e / 2$ 
  shows  $norm\ (x - x') < e$ 
  using dist_triangle_half_l[OF assms[unfolded dist_norm[symmetric]]]
  unfolding dist_norm[symmetric] .

```

```

lemma abs_triangle_half_r:
  fixes y :: 'a::linordered_field
  shows abs (y - x1) < e / 2  $\implies$  abs (y - x2) < e / 2  $\implies$  abs (x1 - x2) < e
  by linarith

```

```

lemma abs_triangle_half_l:
  fixes y :: 'a::linordered_field
  assumes abs (x - y) < e / 2
    and abs (x' - y) < e / 2
  shows abs (x - x') < e
  using assms by linarith

```

```

lemma sum_clauses:
  shows sum f {} = 0
    and finite S  $\implies$  sum f (insert x S) = (if x  $\in$  S then sum f S else f x + sum f S)
  by (auto simp add: insert_absorb)

```

```

lemma vector_eq_ldot: ( $\forall x. x \cdot y = x \cdot z$ )  $\longleftrightarrow$  y = z
proof
  assume  $\forall x. x \cdot y = x \cdot z$ 
  then have  $\forall x. x \cdot (y - z) = 0$ 
    by (simp add: inner_diff)
  then have  $(y - z) \cdot (y - z) = 0$  ..
  then show y = z by simp
qed simp

```

```

lemma vector_eq_rdot: ( $\forall z. x \cdot z = y \cdot z$ )  $\longleftrightarrow$  x = y
proof
  assume  $\forall z. x \cdot z = y \cdot z$ 
  then have  $\forall z. (x - y) \cdot z = 0$ 
    by (simp add: inner_diff)
  then have  $(x - y) \cdot (x - y) = 0$  ..
  then show x = y by simp
qed simp

```

1.5.2 Substandard Basis

```

lemma ex_card:
  assumes n  $\leq$  card A
  shows  $\exists S \subseteq A. \text{card } S = n$ 
proof (cases finite A)
  case True
  from ex_bij_betw_nat_finite[OF this] obtain f where f: bij_betw f {0.. $\text{card } A$ } A ..
  moreover from f  $\langle n \leq \text{card } A \rangle$  have  $\{.. $n$ \} \subseteq \{.. $\text{card } A$ \} \text{inj\_on } f \{.. $n$ \}$ 
    by (auto simp: bij_betw_def intro: subset_inj_on)
  ultimately have f ' $\{.. $n$ \} \subseteq A$  card (f ' $\{.. $n$ \}) = n$ 

```

```

    by (auto simp: bij_betw_def card_image)
  then show ?thesis by blast
next
  case False
  with ⟨ $n \leq \text{card } A$ ⟩ show ?thesis by force
qed

```

```

lemma subspace_substandard: subspace { $x::'a::\text{euclidean\_space}.$   $(\forall i \in \text{Basis}. P\ i \longrightarrow x \cdot i = 0)$ }
  by (auto simp: subspace_def inner_add_left)

```

```

lemma dim_substandard:
  assumes  $d: d \subseteq \text{Basis}$ 
  shows  $\dim \{x::'a::\text{euclidean\_space}.$   $\forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\} = \text{card } d$  (is
 $\dim\ ?A = \_$ )
proof (rule dim_unique)
  from  $d$  show  $d \subseteq ?A$ 
    by (auto simp: inner_Basis)
  from  $d$  show independent  $d$ 
    by (rule independent_mono [OF independent_Basis])
  have  $x \in \text{span } d$  if  $\forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0$  for  $x$ 
  proof -
    have finite  $d$ 
      by (rule finite_subset [OF  $d$  finite_Basis])
    then have  $(\sum i \in d. (x \cdot i) *_{\mathbb{R}} i) \in \text{span } d$ 
      by (simp add: span_sum span_clauses)
    also have  $(\sum i \in d. (x \cdot i) *_{\mathbb{R}} i) = (\sum i \in \text{Basis}. (x \cdot i) *_{\mathbb{R}} i)$ 
      by (rule sum.mono_neutral_cong_left [OF finite_Basis  $d$ ]) (auto simp: that)
    finally show  $x \in \text{span } d$ 
      by (simp only: euclidean_representation)
  qed
  then show  $?A \subseteq \text{span } d$  by auto
qed simp

```

1.5.3 Orthogonality

definition (in *real_inner*) *orthogonal* $x\ y \longleftrightarrow x \cdot y = 0$

context *real_inner*
begin

```

lemma orthogonal_self: orthogonal  $x\ x \longleftrightarrow x = 0$ 
  by (simp add: orthogonal_def)

```

```

lemma orthogonal_clauses:
  orthogonal  $a\ 0$ 
  orthogonal  $a\ x \implies$  orthogonal  $a\ (c *_{\mathbb{R}} x)$ 
  orthogonal  $a\ x \implies$  orthogonal  $a\ (-x)$ 
  orthogonal  $a\ x \implies$  orthogonal  $a\ y \implies$  orthogonal  $a\ (x + y)$ 

```

```

    orthogonal a x  $\implies$  orthogonal a y  $\implies$  orthogonal a (x - y)
    orthogonal 0 a
    orthogonal x a  $\implies$  orthogonal (c *R x) a
    orthogonal x a  $\implies$  orthogonal (- x) a
    orthogonal x a  $\implies$  orthogonal y a  $\implies$  orthogonal (x + y) a
    orthogonal x a  $\implies$  orthogonal y a  $\implies$  orthogonal (x - y) a
    unfolding orthogonal_def inner_add inner_diff by auto

end

lemma orthogonal_commute: orthogonal x y  $\longleftrightarrow$  orthogonal y x
by (simp add: orthogonal_def inner_commute)

lemma orthogonal_scaleR [simp]: c  $\neq$  0  $\implies$  orthogonal (c *R x) = orthogonal x
by (rule ext) (simp add: orthogonal_def)

lemma pairwise_ortho_scaleR:
  pairwise ( $\lambda i j.$  orthogonal (f i) (g j)) B
   $\implies$  pairwise ( $\lambda i j.$  orthogonal (a i *R f i) (a j *R g j)) B
by (auto simp: pairwise_def orthogonal_clauses)

lemma orthogonal_rvsum:
   $\llbracket \text{finite } s; \bigwedge y. y \in s \implies \text{orthogonal } x (f y) \rrbracket \implies \text{orthogonal } x (\text{sum } f s)$ 
by (induction s rule: finite_induct) (auto simp: orthogonal_clauses)

lemma orthogonal_lvsum:
   $\llbracket \text{finite } s; \bigwedge x. x \in s \implies \text{orthogonal } (f x) y \rrbracket \implies \text{orthogonal } (\text{sum } f s) y$ 
by (induction s rule: finite_induct) (auto simp: orthogonal_clauses)

lemma norm_add_Pythagorean:
  assumes orthogonal a b
  shows norm(a + b)  $^2$  = norm a  $^2$  + norm b  $^2$ 
proof -
  from assms have (a - (0 - b))  $\cdot$  (a - (0 - b)) = a  $\cdot$  a - (0 - b  $\cdot$  b)
  by (simp add: algebra_simps orthogonal_def inner_commute)
  then show ?thesis
  by (simp add: power2_norm_eq_inner)
qed

lemma norm_sum_Pythagorean:
  assumes finite I pairwise ( $\lambda i j.$  orthogonal (f i) (f j)) I
  shows (norm (sum f I)) $^2$  = ( $\sum i \in I. (\text{norm } (f i))^2$ )
using assms
proof (induction I rule: finite_induct)
  case empty then show ?case by simp
next
  case (insert x I)
  then have orthogonal (f x) (sum f I)
  by (metis pairwise_insert orthogonal_rvsum)

```

```

with insert show ?case
  by (simp add: pairwise_insert norm_add_Pythagorean)
qed

```

1.5.4 Orthogonality of a transformation

definition $\text{orthogonal\_transformation } f \longleftrightarrow \text{linear } f \wedge (\forall v \ w. f \ v \cdot f \ w = v \cdot w)$

```

lemma orthogonal_transformation:
  orthogonal_transformation f  $\longleftrightarrow$  linear f  $\wedge$  ( $\forall v. \text{norm } (f \ v) = \text{norm } v$ )
unfolding orthogonal_transformation_def
apply auto
apply (erule_tac x=v in allE)+
apply (simp add: norm_eq_sqrt_inner)
apply (simp add: dot_norm linear_add[symmetric])
done

```

```

lemma orthogonal_transformation_id [simp]: orthogonal_transformation ( $\lambda x. x$ )
  by (simp add: linear_iff orthogonal_transformation_def)

```

```

lemma orthogonal_orthogonal_transformation:
  orthogonal_transformation f  $\implies$  orthogonal (f x) (f y)  $\longleftrightarrow$  orthogonal x y
  by (simp add: orthogonal_def orthogonal_transformation_def)

```

```

lemma orthogonal_transformation_compose:
   $\llbracket \text{orthogonal\_transformation } f; \text{orthogonal\_transformation } g \rrbracket \implies \text{orthogonal\_transformation}(f \circ g)$ 
  by (auto simp: orthogonal_transformation_def linear_compose)

```

```

lemma orthogonal_transformation_neg:
  orthogonal_transformation( $\lambda x. -(f \ x)$ )  $\longleftrightarrow$  orthogonal_transformation f
  by (auto simp: orthogonal_transformation_def dest: linear_compose_neg)

```

```

lemma orthogonal_transformation_scaleR: orthogonal_transformation f  $\implies f \ (c *_{\mathbb{R}} v) = c *_{\mathbb{R}} f \ v$ 
  by (simp add: linear_iff orthogonal_transformation_def)

```

```

lemma orthogonal_transformation_linear:
  orthogonal_transformation f  $\implies$  linear f
  by (simp add: orthogonal_transformation_def)

```

```

lemma orthogonal_transformation_inj:
  orthogonal_transformation f  $\implies$  inj f
unfolding orthogonal_transformation_def inj_on_def
by (metis vector_eq)

```

```

lemma orthogonal_transformation_surj:
  orthogonal_transformation f  $\implies$  surj f
for f :: 'a::euclidean_space  $\Rightarrow$  'a::euclidean_space

```


by (simp add: linear_injective_imp_surjective orthogonal_transformation_inj
orthogonal_transformation_linear)

lemma orthogonal_transformation_bij:
 orthogonal_transformation $f \implies \text{bij } f$
 for $f :: 'a::\text{euclidean\_space} \Rightarrow 'a::\text{euclidean\_space}$
 by (simp add: bij_def orthogonal_transformation_inj orthogonal_transformation_surj)

lemma orthogonal_transformation_inv:
 orthogonal_transformation $f \implies \text{orthogonal\_transformation } (\text{inv } f)$
 for $f :: 'a::\text{euclidean\_space} \Rightarrow 'a::\text{euclidean\_space}$
 by (metis (no_types, opaque_lifting) bijection.inv_right bijection_def inj_linear_imp_inv_linear
orthogonal_transformation orthogonal_transformation_bij orthogonal_transformation_inj)

lemma orthogonal_transformation_norm:
 orthogonal_transformation $f \implies \text{norm } (f \ x) = \text{norm } x$
 by (metis orthogonal_transformation)

1.5.5 Bilinear functions

definition
 bilinear $:: ('a::\text{real\_vector} \Rightarrow 'b::\text{real\_vector} \Rightarrow 'c::\text{real\_vector}) \Rightarrow \text{bool}$ **where**
 bilinear $f \iff (\forall x. \text{linear } (\lambda y. f \ x \ y)) \wedge (\forall y. \text{linear } (\lambda x. f \ x \ y))$

lemma bilinear_ladd: bilinear $h \implies h \ (x + y) \ z = h \ x \ z + h \ y \ z$
 by (simp add: bilinear_def linear_iff)

lemma bilinear_radd: bilinear $h \implies h \ x \ (y + z) = h \ x \ y + h \ x \ z$
 by (simp add: bilinear_def linear_iff)

lemma bilinear_times:
 fixes $c::'a::\text{real\_algebra}$ **shows** bilinear $(\lambda x \ y::'a. x*y)$
 by (auto simp: bilinear_def distrib_left distrib_right intro!: linearI)

lemma bilinear_lmul: bilinear $h \implies h \ (c *_R x) \ y = c *_R h \ x \ y$
 by (simp add: bilinear_def linear_iff)

lemma bilinear_rmul: bilinear $h \implies h \ x \ (c *_R y) = c *_R h \ x \ y$
 by (simp add: bilinear_def linear_iff)

lemma bilinear_lneg: bilinear $h \implies h \ (- x) \ y = - h \ x \ y$
 by (drule bilinear_lmul [of _ - 1]) simp

lemma bilinear_rneg: bilinear $h \implies h \ x \ (- y) = - h \ x \ y$
 by (drule bilinear_rmul [of _ _ - 1]) simp

lemma (in ab_group_add) eq_add_iff: $x = x + y \iff y = 0$
 using add_left_imp_eq[of $x \ y \ 0$] by auto

```

lemma bilinear_lzero:
  assumes bilinear h
  shows  $h\ 0\ x = 0$ 
  using bilinear_ladd [OF assms, of 0 0 x] by (simp add: eq_add_iff field_simps)

```

```

lemma bilinear_rzero:
  assumes bilinear h
  shows  $h\ x\ 0 = 0$ 
  using bilinear_radd [OF assms, of x 0 0] by (simp add: eq_add_iff field_simps)

```

```

lemma bilinear_lsub: bilinear h  $\implies h\ (x - y)\ z = h\ x\ z - h\ y\ z$ 
  using bilinear_ladd [of h x - y] by (simp add: bilinear_neg)

```

```

lemma bilinear_rsub: bilinear h  $\implies h\ z\ (x - y) = h\ z\ x - h\ z\ y$ 
  using bilinear_radd [of h _ x - y] by (simp add: bilinear_neg)

```

```

lemma bilinear_sum:
  assumes bilinear h
  shows  $h\ (\text{sum } f\ S)\ (\text{sum } g\ T) = \text{sum } (\lambda(i,j). h\ (f\ i)\ (g\ j))\ (S \times T)$ 
proof -
  interpret l: linear  $\lambda x. h\ x\ y$  for y using assms by (simp add: bilinear_def)
  interpret r: linear  $\lambda y. h\ x\ y$  for x using assms by (simp add: bilinear_def)
  have  $h\ (\text{sum } f\ S)\ (\text{sum } g\ T) = \text{sum } (\lambda x. h\ (f\ x)\ (\text{sum } g\ T))\ S$ 
    by (simp add: l.sum)
  also have  $\dots = \text{sum } (\lambda x. \text{sum } (\lambda y. h\ (f\ x)\ (g\ y))\ T)\ S$ 
    by (rule sum.cong) (simp_all add: r.sum)
  finally show ?thesis
    unfolding sum.cartesian_product .
qed

```

1.5.6 Adjoints

definition *adjoint* :: $((a::\text{real\_inner}) \Rightarrow (b::\text{real\_inner})) \Rightarrow 'b \Rightarrow 'a$ **where**
adjoint $f = (\text{SOME } f'. \forall x\ y. f\ x \cdot y = x \cdot f'\ y)$

```

lemma adjoint_unique:
  assumes  $\forall x\ y. \text{inner } (f\ x)\ y = \text{inner } x\ (g\ y)$ 
  shows adjoint  $f = g$ 
  unfolding adjoint_def
proof (rule some_equality)
  show  $\forall x\ y. \text{inner } (f\ x)\ y = \text{inner } x\ (g\ y)$ 
    by (rule assms)
next
  fix h
  assume  $\forall x\ y. \text{inner } (f\ x)\ y = \text{inner } x\ (h\ y)$ 
  then have  $\forall x\ y. \text{inner } x\ (g\ y) = \text{inner } x\ (h\ y)$ 
    using assms by simp
  then have  $\forall x\ y. \text{inner } x\ (g\ y - h\ y) = 0$ 
    by (simp add: inner_diff_right)

```

```

then have  $\forall y. \text{inner } (g \ y - h \ y) \ (g \ y - h \ y) = 0$ 
by simp
then have  $\forall y. h \ y = g \ y$ 
by simp
then show  $h = g$  by (simp add: ext)
qed

```

TODO: The following lemmas about adjoints should hold for any Hilbert space (i.e. complete inner product space). (see https://en.wikipedia.org/wiki/Hermitian_adjoint)

```

lemma adjoint_works:
  fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'm::\text{euclidean\_space}$ 
  assumes  $lf: \text{linear } f$ 
  shows  $x \cdot \text{adjoint } f \ y = f \ x \cdot y$ 
proof -
  interpret linear f by fact
  have  $\forall y. \exists w. \forall x. f \ x \cdot y = x \cdot w$ 
  proof (intro allI exI)
    fix  $y :: 'm$  and  $x$ 
    let  $?w = (\sum i \in \text{Basis}. (f \ i \cdot y) *_{\mathbb{R}} i) :: 'n$ 
    have  $f \ x \cdot y = f \ (\sum i \in \text{Basis}. (x \cdot i) *_{\mathbb{R}} i) \cdot y$ 
      by (simp add: euclidean_representation)
    also have  $\dots = (\sum i \in \text{Basis}. (x \cdot i) *_{\mathbb{R}} f \ i) \cdot y$ 
      by (simp add: sum scale)
    finally show  $f \ x \cdot y = x \cdot ?w$ 
      by (simp add: inner_sum_left inner_sum_right mult.commute)
  qed
  then show ?thesis
    unfolding adjoint_def choice_iff
    by (intro someI2_ex[where Q= $\lambda f'. x \cdot f' \ y = f \ x \cdot y$ ] auto)
qed

```

```

lemma adjoint_clauses:
  fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'm::\text{euclidean\_space}$ 
  assumes  $lf: \text{linear } f$ 
  shows  $x \cdot \text{adjoint } f \ y = f \ x \cdot y$ 
    and  $\text{adjoint } f \ y \cdot x = y \cdot f \ x$ 
  by (simp_all add: adjoint_works[OF lf] inner_commute)

```

```

lemma adjoint_linear:
  fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'm::\text{euclidean\_space}$ 
  assumes  $lf: \text{linear } f$ 
  shows linear (adjoint f)
  by (simp add: lf linear_iff euclidean_eq_iff[where 'a='n] euclidean_eq_iff[where 'a='m]
    adjoint_clauses[OF lf] inner_distrib)

```

```

lemma adjoint_adjoint:
  fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'm::\text{euclidean\_space}$ 

```

```

assumes lf: linear f
shows adjoint (adjoint f) = f
by (rule adjoint_unique, simp add: adjoint_clauses [OF lf])

```

1.5.7 Euclidean Spaces as Typeclass

```

lemma independent_Basis: independent Basis
  by (rule independent_Basis)

```

```

lemma span_Basis [simp]: span Basis = UNIV
  by (rule span_Basis)

```

```

lemma in_span_Basis: x ∈ span Basis
  unfolding span_Basis ..

```

1.5.8 Linearity and Bilinearity continued

```

lemma linear_bounded:
  fixes f :: 'a::euclidean_space ⇒ 'b::real_normed_vector
  assumes lf: linear f
  shows ∃ B. ∀ x. norm (f x) ≤ B * norm x
proof
  interpret linear f by fact
  let ?B = ∑ b∈Basis. norm (f b)
  show ∀ x. norm (f x) ≤ ?B * norm x
  proof
    fix x :: 'a
    let ?g = λ b. (x • b) *R f b
    have norm (f x) = norm (f (∑ b∈Basis. (x • b) *R b))
      unfolding euclidean_representation ..
    also have ... = norm (sum ?g Basis)
      by (simp add: sum_scale)
    finally have th0: norm (f x) = norm (sum ?g Basis) .
    have th: norm (?g i) ≤ norm (f i) * norm x if i ∈ Basis for i
    proof -
      from Basis_le_norm[OF that, of x]
      show norm (?g i) ≤ norm (f i) * norm x
      unfolding norm_scaleR by (metis mult.commute mult_left_mono norm_ge_zero)
    qed
    from sum_norm_le[of _ ?g, OF th]
    show norm (f x) ≤ ?B * norm x
      unfolding th0 sum_distrib_right by metis
  qed
qed

```

```

lemma linear_conv_bounded_linear:
  fixes f :: 'a::euclidean_space ⇒ 'b::real_normed_vector
  shows linear f ⟷ bounded_linear f
proof
  assume linear f

```

```

then interpret f: linear f .
show bounded_linear f
proof
  have  $\exists B. \forall x. \text{norm } (f x) \leq B * \text{norm } x$ 
    using  $\langle \text{linear } f \rangle$  by (rule linear_bounded)
  then show  $\exists K. \forall x. \text{norm } (f x) \leq \text{norm } x * K$ 
    by (simp add: mult.commute)
qed
next
assume bounded_linear f
then interpret f: bounded_linear f .
show linear f ..
qed

lemmas linear_linear = linear_conv_bounded_linear[symmetric]

lemma inj_linear_imp_inv_bounded_linear:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'a
  shows  $\llbracket \text{bounded\_linear } f; \text{inj } f \rrbracket \Longrightarrow \text{bounded\_linear } (\text{inv } f)$ 
  by (simp add: inj_linear_imp_inv_linear linear_linear)

lemma linear_bounded_pos:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::real_normed_vector
  assumes lf: linear f
  obtains B where  $B > 0 \wedge x. \text{norm } (f x) \leq B * \text{norm } x$ 
proof -
  have  $\exists B > 0. \forall x. \text{norm } (f x) \leq \text{norm } x * B$ 
    using lf unfolding linear_conv_bounded_linear
    by (rule bounded_linear.pos_bounded)
  with that show ?thesis
    by (auto simp: mult.commute)
qed

lemma linear_invertible_bounded_below_pos:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::euclidean_space
  assumes linear f linear g  $g \circ f = \text{id}$ 
  obtains B where  $B > 0 \wedge x. B * \text{norm } x \leq \text{norm}(f x)$ 
proof -
  obtain B where  $B > 0$  and B:  $\wedge x. \text{norm } (g x) \leq B * \text{norm } x$ 
    using linear_bounded_pos [OF  $\langle \text{linear } g \rangle$ ] by blast
  show thesis
  proof
    show  $0 < 1/B$ 
      by (simp add:  $\langle B > 0 \rangle$ )
    show  $1/B * \text{norm } x \leq \text{norm } (f x)$  for x
    proof -
      have  $1/B * \text{norm } x = 1/B * \text{norm } (g (f x))$ 
        using assms by (simp add: pointfree_idE)
      also have  $\dots \leq \text{norm } (f x)$ 

```

```

      using B [of f x] by (simp add: ⟨B > 0⟩ mult.commute pos_divide_le_eq)
    finally show ?thesis .
  qed
qed
qed

```

```

lemma linear_inj_bounded_below_pos:
  fixes f :: 'a::real_normed_vector ⇒ 'b::euclidean_space
  assumes linear f inj f
  obtains B where B > 0 ∧ x. B * norm x ≤ norm(f x)
  using linear_injective_left_inverse [OF assms]
    linear_invertible_bounded_below_pos assms by blast

```

```

lemma bounded_linearI':
  fixes f :: 'a::euclidean_space ⇒ 'b::real_normed_vector
  assumes ∧ x y. f (x + y) = f x + f y
    and ∧ c x. f (c *R x) = c *R f x
  shows bounded_linear f
  using assms linearI linear_conv_bounded_linear by blast

```

```

lemma bilinear_bounded:
  fixes h :: 'm::euclidean_space ⇒ 'n::euclidean_space ⇒ 'k::real_normed_vector
  assumes bh: bilinear h
  shows ∃ B. ∀ x y. norm (h x y) ≤ B * norm x * norm y
proof (clarify intro!: exI[of _ ∑ i∈Basis. ∑ j∈Basis. norm (h i j)])
  fix x :: 'm
  fix y :: 'n
  have norm (h x y) = norm (h (sum (λ i. (x • i) *R i) Basis) (sum (λ i. (y • i)
*_R i) Basis))
  by (simp add: euclidean_representation)
  also have ... = norm (sum (λ (i,j). h ((x • i) *R i) ((y • j) *R j)) (Basis ×
Basis))
  unfolding bilinear_sum[OF bh] ..
  finally have th: norm (h x y) = ... .
  have ∧ i j. [i ∈ Basis; j ∈ Basis]
    ⇒ |x • i| * (|y • j| * norm (h i j)) ≤ norm x * (norm y * norm (h i j))
  by (auto simp add: zero_le_mult_iff Basis_le_norm mult_mono)
  then show norm (h x y) ≤ (∑ i∈Basis. ∑ j∈Basis. norm (h i j)) * norm x *
norm y
  unfolding sum_distrib_right th sum.cartesian_product
  by (clarsimp simp add: bilinear_rmul[OF bh] bilinear_lmul[OF bh]
    field_simps simp del: scaleR_scaleR intro!: sum_norm_le)
qed

```

```

lemma bilinear_conv_bounded_bilinear:
  fixes h :: 'a::euclidean_space ⇒ 'b::euclidean_space ⇒ 'c::real_normed_vector
  shows bilinear h ⟷ bounded_bilinear h
proof
  assume bilinear h

```

```

show bounded_bilinear h
proof
  fix x y z
  show h (x + y) z = h x z + h y z
    using ⟨bilinear h⟩ unfolding bilinear_def linear_iff by simp
next
  fix x y z
  show h x (y + z) = h x y + h x z
    using ⟨bilinear h⟩ unfolding bilinear_def linear_iff by simp
next
  show h (scaleR r x) y = scaleR r (h x y) h x (scaleR r y) = scaleR r (h x y)
for r x y
  using ⟨bilinear h⟩ unfolding bilinear_def linear_iff
  by simp_all
next
  have  $\exists B. \forall x y. \text{norm } (h x y) \leq B * \text{norm } x * \text{norm } y$ 
    using ⟨bilinear h⟩ by (rule bilinear_bounded)
  then show  $\exists K. \forall x y. \text{norm } (h x y) \leq \text{norm } x * \text{norm } y * K$ 
    by (simp add: ac_simps)
qed
next
assume bounded_bilinear h
then interpret h: bounded_bilinear h .
show bilinear h
  unfolding bilinear_def linear_conv_bounded_linear
  using h.bounded_linear_left h.bounded_linear_right by simp
qed

```

```

lemma bilinear_bounded_pos:
  fixes h :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'c::real_normed_vector
  assumes bh: bilinear h
  shows  $\exists B > 0. \forall x y. \text{norm } (h x y) \leq B * \text{norm } x * \text{norm } y$ 
proof -
  have  $\exists B > 0. \forall x y. \text{norm } (h x y) \leq \text{norm } x * \text{norm } y * B$ 
    using bh [unfolded bilinear_conv_bounded_bilinear]
    by (rule bounded_bilinear.pos_bounded)
  then show ?thesis
    by (simp only: ac_simps)
qed

```

```

lemma bounded_linear_imp_has_derivative: bounded_linear f  $\implies$  (f has_derivative
f) net
  by (auto simp add: has_derivative_def linear_diff linear_linear linear_def
dest: bounded_linear.linear)

```

```

lemma linear_imp_has_derivative:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::real_normed_vector
  shows linear f  $\implies$  (f has_derivative f) net
  by (simp add: bounded_linear_imp_has_derivative linear_conv_bounded_linear)

```

lemma *bounded_linear_imp_differentiable*: *bounded_linear f $\implies$ f differentiable net*

using *bounded_linear_imp_has_derivative differentiable_def* **by** *blast*

lemma *linear_imp_differentiable*:

fixes *f :: 'a::euclidean_space $\Rightarrow$ 'b::real_normed_vector*

shows *linear f $\implies$ f differentiable net*

by (*metis linear_imp_has_derivative differentiable_def*)

lemma *of_real_differentiable* [*simp, derivative_intros*]: *of_real differentiable F*

by (*simp add: bounded_linear_imp_differentiable bounded_linear_of_real*)

1.5.9 We continue

lemma *independent_bound*:

fixes *S :: 'a::euclidean_space set*

shows *independent S $\implies$ finite S $\wedge$ card S $\leq$ DIM('a)*

by (*metis dim_subset_UNIV finiteI_independent dim_span_eq_card_independent*)

lemmas *independent_imp_finite = finiteI_independent*

corollary *independent_card_le*:

fixes *S :: 'a::euclidean_space set*

assumes *independent S*

shows *card S $\leq$ DIM('a)*

using *assms independent_bound* **by** *auto*

lemma *dependent_biggerset*:

fixes *S :: 'a::euclidean_space set*

shows (*finite S $\implies$ card S $>$ DIM('a)*) $\implies$ *dependent S*

by (*metis independent_bound not_less*)

Picking an orthogonal replacement for a spanning set.

lemma *vector_sub_project_orthogonal*:

fixes *b x :: 'a::euclidean_space*

shows *b $\cdot$ (x - ((b $\cdot$ x) / (b $\cdot$ b)) *_R b) = 0*

unfolding *inner_simps* **by** *auto*

lemma *pairwise_orthogonal_insert*:

assumes *pairwise orthogonal S*

and $\bigwedge y. y \in S \implies$ *orthogonal x y*

shows *pairwise orthogonal (insert x S)*

using *assms unfolding pairwise_def*

by (*auto simp add: orthogonal_commute*)

lemma *basis_orthogonal*:

fixes *B :: 'a::real_inner set*

assumes *fB: finite B*


```

shows  $\exists C. \text{finite } C \wedge \text{card } C \leq \text{card } B \wedge \text{span } C = \text{span } B \wedge \text{pairwise orthogonal } C$ 
(is  $\exists C. ?P \ B \ C$ )
using fB
proof (induct rule: finite_induct)
  case empty
  then show ?case
    apply (rule exI[where x={}])
    apply (auto simp add: pairwise_def)
    done
  next
    case (insert a B)
    note fB = ⟨finite B⟩ and aB = ⟨a ∉ B⟩
    from  $\langle \exists C. \text{finite } C \wedge \text{card } C \leq \text{card } B \wedge \text{span } C = \text{span } B \wedge \text{pairwise orthogonal } C \rangle$ 
    obtain C where C: finite C card C ≤ card B
      span C = span B pairwise orthogonal C by blast
    let ?a = a - sum (λx. (x • a / (x • x)) *R x) C
    let ?C = insert ?a C
    from C(1) have fC: finite ?C
      by simp
    from fB aB C(1,2) have cC: card ?C ≤ card (insert a B)
      by (simp add: card_insert_if)
    {
      fix x k
      have th0:  $\bigwedge (a::'a) \ b \ c. a - (b - c) = c + (a - b)$ 
        by (simp add: field_simps)
      have  $x - k *_{\mathbb{R}} (a - (\sum_{x \in C. (x \cdot a / (x \cdot x)) *_{\mathbb{R}} x)) \in \text{span } C \longleftrightarrow x - k *_{\mathbb{R}} a \in \text{span } C$ 
        apply (simp only: scaleR_right_diff_distrib th0)
        apply (rule span_add_eq)
        apply (rule span_scale)
        apply (rule span_sum)
        apply (rule span_scale)
        apply (rule span_base)
        apply assumption
        done
      }
    then have SC: span ?C = span (insert a B)
      unfolding set_eq_iff span_breakdown_eq C(3)[symmetric] by auto
    {
      fix y
      assume yC: y ∈ C
      then have Cy: C = insert y (C - {y})
        by blast
      have fth: finite (C - {y})
        using C by simp
      have orthogonal ?a y
        unfolding orthogonal_def

```

```

    unfolding inner_diff inner_sum_left right_minus_eq
    unfolding sum.remove [OF ⟨finite C⟩ ⟨y ∈ C⟩]
    apply (clarsimp simp add: inner_commute[of y a])
    apply (rule sum.neutral)
    apply clarsimp
    apply (rule C(4)[unfolded pairwise_def orthogonal_def, rule_format])
    using ⟨y ∈ C⟩ by auto
  }
  with ⟨pairwise orthogonal C⟩ have CPO: pairwise orthogonal ?C
  by (rule pairwise_orthogonal_insert)
  from fC cC SC CPO have ?P (insert a B) ?C
  by blast
  then show ?case by blast
qed

```

```

lemma orthogonal_basis_exists:
  fixes V :: ('a::euclidean_space) set
  shows  $\exists B. \text{independent } B \wedge B \subseteq \text{span } V \wedge V \subseteq \text{span } B \wedge$ 
    ( $\text{card } B = \dim V$ )  $\wedge$  pairwise orthogonal B
proof -
  from basis_exists[of V] obtain B where
    B:  $B \subseteq V$  independent B  $V \subseteq \text{span } B$   $\text{card } B = \dim V$ 
  by force
  from B have fB: finite B  $\text{card } B = \dim V$ 
  using independent_bound by auto
  from basis_orthogonal[OF fB(1)] obtain C where
    C: finite C  $\text{card } C \leq \text{card } B$   $\text{span } C = \text{span } B$  pairwise orthogonal C
  by blast
  from C B have CSV:  $C \subseteq \text{span } V$ 
  by (metis span_superset span_mono subset_trans)
  from span_mono[OF B(3)] C have SVC:  $\text{span } V \subseteq \text{span } C$ 
  by (simp add: span_span)
  from card_le_dim_spanning[OF CSV SVC C(1)] C(2,3) fB
  have iC: independent C
  by (simp)
  from C fB have card C  $\leq \dim V$ 
  by simp
  moreover have  $\dim V \leq \text{card } C$ 
  using span_card_ge_dim[OF CSV SVC C(1)]
  by simp
  ultimately have CdV:  $\text{card } C = \dim V$ 
  using C(1) by simp
  from C B CSV CdV iC show ?thesis
  by auto
qed

```

Low-dimensional subset is in a hyperplane (weak orthogonal complement).

```

lemma span_not_univ_orthogonal:
  fixes S :: ('a::euclidean_space) set

```

```

    assumes  $sU$ :  $\text{span } S \neq \text{UNIV}$ 
    shows  $\exists a::'a. a \neq 0 \wedge (\forall x \in \text{span } S. a \cdot x = 0)$ 
  proof -
    from  $sU$  obtain  $a$  where  $a: a \notin \text{span } S$ 
    by blast
    from  $\text{orthogonal\_basis\_exists}$  obtain  $B$  where
       $B$ :  $\text{independent } B$   $B \subseteq \text{span } S$   $S \subseteq \text{span } B$ 
       $\text{card } B = \text{dim } S$   $\text{pairwise\_orthogonal } B$ 
    by blast
    from  $B$  have  $fB$ :  $\text{finite } B$   $\text{card } B = \text{dim } S$ 
    using  $\text{independent\_bound}$  by auto
    from  $\text{span\_mono}[OF\ B(2)]$   $\text{span\_mono}[OF\ B(3)]$ 
    have  $sSB$ :  $\text{span } S = \text{span } B$ 
    by (simp add:  $\text{span\_span}$ )
    let  $?a = a - \text{sum } (\lambda b. (a \cdot b / (b \cdot b)) *_{\mathbb{R}} b)$   $B$ 
    have  $\text{sum } (\lambda b. (a \cdot b / (b \cdot b)) *_{\mathbb{R}} b) \in \text{span } S$ 
    unfolding  $sSB$ 
    apply (rule  $\text{span\_sum}$ )
    apply (rule  $\text{span\_scale}$ )
    apply (rule  $\text{span\_base}$ )
    apply assumption
    done
    with  $a$  have  $a0$ :  $?a \neq 0$ 
    by auto
    have  $?a \cdot x = 0$  if  $x \in \text{span } B$  for  $x$ 
    proof (rule  $\text{span\_induct}$  [OF  $\text{that}$ ])
      show  $\text{subspace } \{x. ?a \cdot x = 0\}$ 
      by (auto simp add:  $\text{subspace\_def}$   $\text{inner\_add}$ )
    next
      {
        fix  $x$ 
        assume  $x$ :  $x \in B$ 
        from  $x$  have  $B'$ :  $B = \text{insert } x (B - \{x\})$ 
        by blast
        have  $fth$ :  $\text{finite } (B - \{x\})$ 
        using  $fB$  by simp
        have  $?a \cdot x = 0$ 
        apply (subst  $B'$ )
        using  $fB$   $fth$ 
        unfolding  $\text{sum\_clauses}(2)[OF\ fth]$ 
        apply simp unfolding  $\text{inner\_sims}$ 
        apply (clarsimp simp add:  $\text{inner\_add}$   $\text{inner\_sum\_left}$ )
        apply (rule  $\text{sum.neutral}$ , rule  $\text{ballI}$ )
        apply (simp only:  $\text{inner\_commute}$ )
        apply (auto simp add:  $\text{field\_sims}$ 
          intro:  $B(5)[\text{unfolded pairwise\_def orthogonal\_def, rule\_format}]$ )
        done
      }
    then show  $?a \cdot x = 0$  if  $x \in B$  for  $x$ 

```

```

      using that by blast
    qed
  with a0 show ?thesis
    unfolding sSB by (auto intro: exI[where x=?a])
  qed

```

```

lemma span_not_univ_subset_hyperplane:
  fixes S :: 'a::euclidean_space set
  assumes SU: span S ≠ UNIV
  shows ∃ a. a ≠ 0 ∧ span S ⊆ {x. a • x = 0}
  using span_not_univ_orthogonal[OF SU] by auto

```

```

lemma lowdim_subset_hyperplane:
  fixes S :: 'a::euclidean_space set
  assumes d: dim S < DIM('a)
  shows ∃ a::'a. a ≠ 0 ∧ span S ⊆ {x. a • x = 0}
proof -
  {
    assume span S = UNIV
    then have dim (span S) = dim (UNIV :: ('a) set)
      by simp
    then have dim S = DIM('a)
      by (metis Euclidean_Space.dim_UNIV dim_span)
    with d have False by arith
  }
  then have th: span S ≠ UNIV
    by blast
  from span_not_univ_subset_hyperplane[OF th] show ?thesis .
qed

```

```

lemma linear_eq_stdbasis:
  fixes f :: 'a::euclidean_space ⇒ _
  assumes lf: linear f
    and lg: linear g
    and fg: ⋀ b. b ∈ Basis ⇒ f b = g b
  shows f = g
  using linear_eq_on_span[OF lf lg, of Basis] fg
  by auto

```

Similar results for bilinear functions.

```

lemma bilinear_eq:
  assumes bf: bilinear f
    and bg: bilinear g
    and SB: S ⊆ span B
    and TC: T ⊆ span C
    and x∈S y∈T
    and fg: ⋀ x y. [x ∈ B; y ∈ C] ⇒ f x y = g x y
  shows f x y = g x y
proof -

```

```

let ?P = {x.  $\forall y \in \text{span } C. f\ x\ y = g\ x\ y$ }
from bf bg have sp: subspace ?P
  unfolding bilinear_def linear_iff subspace_def bf bg
  by (auto simp add: span_zero bilinear_lzero[OF bf] bilinear_lzero[OF bg]
    span_add Ball_def
    intro: bilinear_ladd[OF bf])
have sfg:  $\bigwedge x. x \in B \implies \text{subspace } \{a. f\ x\ a = g\ x\ a\}$ 
  apply (auto simp add: subspace_def)
  using bf bg unfolding bilinear_def linear_iff
  apply (auto simp add: span_zero bilinear_rzero[OF bf] bilinear_rzero[OF bg]
    span_add Ball_def
    intro: bilinear_ladd[OF bf])
done
have  $\forall y \in \text{span } C. f\ x\ y = g\ x\ y$  if  $x \in \text{span } B$  for x
  apply (rule span_induct [OF that sp])
  using fg sfg span_induct by blast
then show ?thesis
  using SB TC assms by auto
qed

```

```

lemma bilinear_eq_stdbasis:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  _
  assumes bf: bilinear f
    and bg: bilinear g
    and fg:  $\bigwedge i\ j. i \in \text{Basis} \implies j \in \text{Basis} \implies f\ i\ j = g\ i\ j$ 
  shows f = g
  using bilinear_eq[OF bf bg equalityD2[OF span_Basis] equalityD2[OF span_Basis]]
  fg by blast

```

1.5.10 Infinity norm

definition *infnorm* (x::'a::euclidean_space) = Sup { $|x \cdot b|$ | b. b $\in$ Basis}

```

lemma infnorm_set_image:
  fixes x :: 'a::euclidean_space
  shows  $\{|x \cdot i| \mid i. i \in \text{Basis}\} = (\lambda i. |x \cdot i|) \text{ ` Basis}$ 
  by blast

```

```

lemma infnorm_Max:
  fixes x :: 'a::euclidean_space
  shows infnorm x = Max (( $\lambda i. |x \cdot i|$ ) ` Basis)
  by (simp add: infnorm_def infnorm_set_image cSup_eq_Max)

```

```

lemma infnorm_set_lemma:
  fixes x :: 'a::euclidean_space
  shows finite  $\{|x \cdot i| \mid i. i \in \text{Basis}\}$ 
    and  $\{|x \cdot i| \mid i. i \in \text{Basis}\} \neq \{\}$ 
  unfolding infnorm_set_image
  by auto

```

```

lemma infnorm_pos_le:
  fixes x :: 'a::euclidean_space
  shows  $0 \leq \text{infnorm } x$ 
  by (simp add: infnorm_Max Max_ge_iff ex_in_conv)

```

```

lemma infnorm_triangle:
  fixes x :: 'a::euclidean_space
  shows  $\text{infnorm } (x + y) \leq \text{infnorm } x + \text{infnorm } y$ 
proof -
  have *:  $\bigwedge a \ b \ c \ d :: \text{real}. |a| \leq c \implies |b| \leq d \implies |a + b| \leq c + d$ 
  by simp
  show ?thesis
  by (auto simp: infnorm_Max inner_add_left intro!: *)
qed

```

```

lemma infnorm_eq_0:
  fixes x :: 'a::euclidean_space
  shows  $\text{infnorm } x = 0 \longleftrightarrow x = 0$ 
proof -
  have  $\text{infnorm } x \leq 0 \longleftrightarrow x = 0$ 
  unfolding infnorm_Max by (simp add: euclidean_all_zero_iff)
  then show ?thesis
  using infnorm_pos_le[of x] by simp
qed

```

```

lemma infnorm_0:  $\text{infnorm } 0 = 0$ 
  by (simp add: infnorm_eq_0)

```

```

lemma infnorm_neg:  $\text{infnorm } (-x) = \text{infnorm } x$ 
  unfolding infnorm_def by simp

```

```

lemma infnorm_sub:  $\text{infnorm } (x - y) = \text{infnorm } (y - x)$ 
  by (metis infnorm_neg minus_diff_eq)

```

```

lemma absdiff_infnorm:  $|\text{infnorm } x - \text{infnorm } y| \leq \text{infnorm } (x - y)$ 
proof -
  have *:  $\bigwedge (nx::\text{real}) \ n \ ny. nx \leq n + ny \implies ny \leq n + nx \implies |nx - ny| \leq n$ 
  by arith
  show ?thesis
  proof (rule *)
    from infnorm_triangle[of x - y y] infnorm_triangle[of x - y -x]
    show  $\text{infnorm } x \leq \text{infnorm } (x - y) + \text{infnorm } y$ 
     $\text{infnorm } y \leq \text{infnorm } (x - y) + \text{infnorm } x$ 
    by (simp_all add: field_simps infnorm_neg)
  qed
qed

```

```

lemma real_abs_infnorm:  $|\text{infnorm } x| = \text{infnorm } x$ 

```

using *infnorm_pos_le*[of *x*] by *arith*

lemma *Basis_le_infnorm*:

fixes *x* :: '*a*::euclidean_space

shows $b \in \text{Basis} \implies |x \cdot b| \leq \text{infnorm } x$

by (*simp add: infnorm_Max*)

lemma *infnorm_mul*: $\text{infnorm } (a *_R x) = |a| * \text{infnorm } x$

unfolding *infnorm_Max*

proof (*safe intro!: Max_eqI*)

let *?B* = $(\lambda i. |x \cdot i|)$ ' *Basis*

{ fix *b* :: '*a*

assume $b \in \text{Basis}$

then show $|a *_R x \cdot b| \leq |a| * \text{Max } ?B$

by (*simp add: abs_mult mult_left_mono*)

next

from *Max_in*[of *?B*] obtain *b* where $b \in \text{Basis}$ $\text{Max } ?B = |x \cdot b|$

by (*auto simp del: Max_in*)

then show $|a| * \text{Max } ((\lambda i. |x \cdot i|) \text{ ' } \text{Basis}) \in (\lambda i. |a *_R x \cdot i|) \text{ ' } \text{Basis}$

by (*intro image_eqI*[where $x=b$]) (*auto simp: abs_mult*)

}

qed *simp*

lemma *infnorm_mul_lemma*: $\text{infnorm } (a *_R x) \leq |a| * \text{infnorm } x$

unfolding *infnorm_mul* ..

lemma *infnorm_pos_lt*: $\text{infnorm } x > 0 \longleftrightarrow x \neq 0$

using *infnorm_pos_le*[of *x*] *infnorm_eq_0*[of *x*] by *arith*

Prove that it differs only up to a bound from Euclidean norm.

lemma *infnorm_le_norm*: $\text{infnorm } x \leq \text{norm } x$

by (*simp add: Basis_le_infnorm infnorm_Max*)

lemma *norm_le_infnorm*:

fixes *x* :: '*a*::euclidean_space

shows $\text{norm } x \leq \sqrt{\text{DIM } ('a)} * \text{infnorm } x$

unfolding *norm_eq_sqrt_inner id_def*

proof (*rule real_le_sqrt[OF inner_ge_zero]*)

show $\sqrt{\text{DIM } ('a)} * \text{infnorm } x \geq 0$

by (*simp add: zero_le_mult_iff infnorm_pos_le*)

have $x \cdot x \leq (\sum b \in \text{Basis}. x \cdot b * (x \cdot b))$

by (*metis euclidean_inner_order_refl*)

also have $\dots \leq \text{DIM } ('a) * |\text{infnorm } x|^2$

by (*rule sum_bounded_above*) (*metis Basis_le_infnorm abs_le_square_iff power2_eq_square real_abs_infnorm*)

also have $\dots \leq (\sqrt{\text{DIM } ('a)} * \text{infnorm } x)^2$

by (*simp add: power_mult_distrib*)

finally show $x \cdot x \leq (\sqrt{\text{DIM } ('a)} * \text{infnorm } x)^2$.

qed

```

lemma tendsto_infnorm [tendsto_intros]:
  assumes ( $f \longrightarrow a$ )  $F$ 
  shows  $((\lambda x. \text{infnorm } (f x)) \longrightarrow \text{infnorm } a) F$ 
proof (rule tendsto_compose [OF LIM_I assms])
  fix  $r :: \text{real}$ 
  assume  $r > 0$ 
  then show  $\exists s > 0. \forall x. x \neq a \wedge \text{norm } (x - a) < s \longrightarrow \text{norm } (\text{infnorm } x - \text{infnorm } a) < r$ 
    by (metis real_norm_def le_less_trans absdiff_infnorm infnorm_le_norm)
qed

```

Equality in Cauchy-Schwarz and triangle inequalities.

```

lemma norm_cauchy_schwarz_eq:  $x \cdot y = \text{norm } x * \text{norm } y \longleftrightarrow \text{norm } x *_R y = \text{norm } y *_R x$ 
  (is  $?lhs \longleftrightarrow ?rhs$ )
proof (cases x=0)
  case True
  then show  $?thesis$ 
    by auto
next
  case False
  from inner_eq_zero_iff [of norm y *_R x - norm x *_R y]
  have  $?rhs \longleftrightarrow$ 
     $(\text{norm } y * (\text{norm } y * \text{norm } x * \text{norm } x - \text{norm } x * (x \cdot y)) - \text{norm } x * (\text{norm } y * (y \cdot x) - \text{norm } x * \text{norm } y * \text{norm } y) = 0)$ 
  using False unfolding inner_simps
  by (auto simp add: power2_norm_eq_inner[symmetric] power2_eq_square inner_commute field_simps)
  also have  $\dots \longleftrightarrow (2 * \text{norm } x * \text{norm } y * (\text{norm } x * \text{norm } y - x \cdot y) = 0)$ 
  using False by (simp add: field_simps inner_commute)
  also have  $\dots \longleftrightarrow ?lhs$ 
  using False by auto
  finally show  $?thesis$  by metis
qed

```

```

lemma norm_cauchy_schwarz_abs_eq:
   $|x \cdot y| = \text{norm } x * \text{norm } y \longleftrightarrow$ 
     $\text{norm } x *_R y = \text{norm } y *_R x \vee \text{norm } x *_R y = - \text{norm } y *_R x$ 
  (is  $?lhs \longleftrightarrow ?rhs$ )
proof -
  have  $th: \bigwedge (x::\text{real}) a. a \geq 0 \implies |x| = a \longleftrightarrow x = a \vee x = -a$ 
  by arith
  have  $?rhs \longleftrightarrow \text{norm } x *_R y = \text{norm } y *_R x \vee \text{norm } (-x) *_R y = \text{norm } y *_R (-x)$ 
  by simp
  also have  $\dots \longleftrightarrow (x \cdot y = \text{norm } x * \text{norm } y \vee (-x) \cdot y = \text{norm } x * \text{norm } y)$ 
  unfolding norm_cauchy_schwarz_eq[symmetric]
  unfolding norm_minus_cancel norm_scaleR ..

```



```

    also have ...  $\longleftrightarrow$  ?lhs
    unfolding th[OF mult_nonneg_nonneg, OF norm_ge_zero[of x] norm_ge_zero[of
y]] inner_simps
    by auto
    finally show ?thesis ..
qed

lemma norm_triangle_eq:
  fixes x y :: 'a::real_inner
  shows norm (x + y) = norm x + norm y  $\longleftrightarrow$  norm x *R y = norm y *R x
proof (cases x = 0  $\vee$  y = 0)
  case True
  then show ?thesis
    by force
next
  case False
  then have n: norm x > 0 norm y > 0
    by auto
  have norm (x + y) = norm x + norm y  $\longleftrightarrow$  (norm (x + y))2 = (norm x +
norm y)2
    by simp
  also have ...  $\longleftrightarrow$  norm x *R y = norm y *R x
    unfolding norm_cauchy_schwarz_eq[symmetric]
    unfolding power2_norm_eq_inner inner_simps
    by (simp add: power2_norm_eq_inner[symmetric] power2_eq_square inner_commute
field_simps)
  finally show ?thesis .
qed

lemma dist_triangle_eq:
  fixes x y z :: 'a::real_inner
  shows dist x z = dist x y + dist y z  $\longleftrightarrow$ 
    norm (x - y) *R (y - z) = norm (y - z) *R (x - y)
proof -
  have *: x - y + (y - z) = x - z by auto
  show ?thesis unfolding dist_norm norm_triangle_eq[of x - y y - z, unfolded
*]
    by (auto simp: norm_minus_commute)
qed

```

1.5.11 Collinearity

```

definition collinear :: 'a::real_vector set  $\Rightarrow$  bool
  where collinear S  $\longleftrightarrow$  ( $\exists u. \forall x \in S. \forall y \in S. \exists c. x - y = c *_{\mathbf{R}} u$ )

lemma collinear_alt:
  collinear S  $\longleftrightarrow$  ( $\exists u v. \forall x \in S. \exists c. x = u + c *_{\mathbf{R}} v$ ) (is ?lhs = ?rhs)
proof
  assume ?lhs

```

```

    then show ?rhs
      unfolding collinear_def by (metis Groups.add_ac(2) diff_add_cancel)
next
  assume ?rhs
  then obtain u v where *:  $\bigwedge x. x \in S \implies \exists c. x = u + c *_R v$ 
    by (auto simp: )
  have  $\exists c. x - y = c *_R v$  if  $x \in S \ y \in S$  for  $x \ y$ 
    by (metis *[OF ⟨x ∈ S⟩] *[OF ⟨y ∈ S⟩] scaleR_left.diff_add_diff_cancel_left)
  then show ?lhs
    using collinear_def by blast
qed

```

```

lemma collinear:
  fixes S :: 'a::{perfect_space,real_vector} set
  shows collinear S  $\longleftrightarrow (\exists u. u \neq 0 \wedge (\forall x \in S. \forall y \in S. \exists c. x - y = c *_R u))$ 
proof -
  have  $\exists v. v \neq 0 \wedge (\forall x \in S. \forall y \in S. \exists c. x - y = c *_R v)$ 
    if  $\forall x \in S. \forall y \in S. \exists c. x - y = c *_R u$  for  $u$ 
  proof -
    have  $\forall x \in S. \forall y \in S. x - y = 0$ 
      using that by auto
    moreover
    obtain v::'a where  $v \neq 0$ 
      using UNIV_not_singleton [of 0] by auto
    ultimately have  $\forall x \in S. \forall y \in S. \exists c. x - y = c *_R v$ 
      by auto
    then show ?thesis
      using ⟨v ≠ 0⟩ by blast
  qed
  then show ?thesis
    apply (clarsimp simp: collinear_def)
    by (metis scaleR_zero_right vector_fraction_eq_iff)
qed

```

```

lemma collinear_subset:  $\llbracket \text{collinear } T; S \subseteq T \rrbracket \implies \text{collinear } S$ 
  by (meson collinear_def subsetCE)

```

```

lemma collinear_empty [iff]: collinear {}
  by (simp add: collinear_def)

```

```

lemma collinear_sing [iff]: collinear {x}
  by (simp add: collinear_def)

```

```

lemma collinear_2 [iff]: collinear {x, y}
  apply (simp add: collinear_def)
  apply (rule exI[where x=x - y])
  by (metis minus_diff_eq scaleR_left.minus scaleR_one)

```

```

lemma collinear_lemma: collinear {0, x, y}  $\longleftrightarrow x = 0 \vee y = 0 \vee (\exists c. y = c$ 

```

```

*_R x)
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof (cases  $x = 0 \vee y = 0$ )
  case True
  then show ?thesis
    by (auto simp: insert_commute)
next
case False
show ?thesis
proof
  assume h: ?lhs
  then obtain u where u:  $\forall x \in \{0, x, y\}. \forall y \in \{0, x, y\}. \exists c. x - y = c *_R u$ 
    unfolding collinear_def by blast
  from u[rule_format, of x 0] u[rule_format, of y 0]
  obtain cx and cy where
    cx:  $x = cx *_R u$  and cy:  $y = cy *_R u$ 
  by auto
  from cx cy False have cx0:  $cx \neq 0$  and cy0:  $cy \neq 0$  by auto
  let ?d =  $cy / cx$ 
  from cx cy cx0 have  $y = ?d *_R x$ 
  by simp
  then show ?rhs using False by blast
next
assume h: ?rhs
then obtain c where c:  $y = c *_R x$ 
  using False by blast
show ?lhs
  unfolding collinear_def c
  apply (rule exI[where x=x])
  apply auto
  apply (rule exI[where x=- 1], simp)
  apply (rule exI[where x=-c], simp)
  apply (rule exI[where x=1], simp)
  apply (rule exI[where x=1 - c], simp add: scaleR_left_diff_distrib)
  apply (rule exI[where x=c - 1], simp add: scaleR_left_diff_distrib)
  done
qed
qed

lemma collinear_iff_Reals:  $\text{collinear } \{0::\text{complex}, w, z\} \longleftrightarrow z/w \in \mathbb{R}$ 
proof
  show  $z/w \in \mathbb{R} \implies \text{collinear } \{0, w, z\}$ 
    by (metis Reals_cases collinear_lemma nonzero_divide_eq_eq scaleR_conv_of_real)
qed (auto simp: collinear_lemma scaleR_conv_of_real)

lemma collinear_scaleR_iff:  $\text{collinear } \{0, \alpha *_R w, \beta *_R z\} \longleftrightarrow \text{collinear } \{0, w, z\}$ 
 $\vee \alpha=0 \vee \beta=0$ 
  (is ?lhs = ?rhs)
proof (cases  $\alpha=0 \vee \beta=0$ )

```

```

    case False
    then have  $(\exists c. \beta *_R z = (c *_R \alpha) *_R w) = (\exists c. z = c *_R w)$ 
      by (metis mult.commute scaleR_scaleR vector_fraction_eq_iff)
    then show ?thesis
      by (auto simp add: collinear_lemma)
qed (auto simp: collinear_lemma)

lemma norm_cauchy_schwarz_equal:  $|x \cdot y| = \text{norm } x * \text{norm } y \longleftrightarrow \text{collinear } \{0, x, y\}$ 
proof (cases  $x=0$ )
  case True
  then show ?thesis
    by (auto simp: insert_commute)
next
  case False
  then have nnz:  $\text{norm } x \neq 0$ 
    by auto
  show ?thesis
  proof
    assume  $|x \cdot y| = \text{norm } x * \text{norm } y$ 
    then show collinear  $\{0, x, y\}$ 
      unfolding norm_cauchy_schwarz_abs_eq collinear_lemma
      by (meson eq_vector_fraction_iff nnz)
  next
    assume collinear  $\{0, x, y\}$ 
    with False show  $|x \cdot y| = \text{norm } x * \text{norm } y$ 
      unfolding norm_cauchy_schwarz_abs_eq collinear_lemma by (auto simp:
abs_if)
  qed
qed

lemma norm_triangle_eq_imp_collinear:
  fixes  $x y :: 'a::\text{real\_inner}$ 
  assumes  $\text{norm } (x + y) = \text{norm } x + \text{norm } y$ 
  shows collinear $\{0,x,y\}$ 
proof (cases  $x = 0 \vee y = 0$ )
  case False
  with assms show ?thesis
    by (meson norm_cauchy_schwarz_abs_eq norm_cauchy_schwarz_equal norm_triangle_eq)
qed (use collinear_lemma in blast)

```

1.5.12 Properties of special hyperplanes

```

lemma subspace_hyperplane: subspace  $\{x. a \cdot x = 0\}$ 
  by (simp add: subspace_def inner_right_distrib)

lemma subspace_hyperplane2: subspace  $\{x. x \cdot a = 0\}$ 
  by (simp add: inner_commute inner_right_distrib subspace_def)

```

```

lemma special_hyperplane_span:
  fixes S :: 'n::euclidean_space set
  assumes k ∈ Basis
  shows {x. k • x = 0} = span (Basis - {k})
proof -
  have *: x ∈ span (Basis - {k}) if k • x = 0 for x
  proof -
    have x = (∑ b ∈ Basis. (x • b) *R b)
      by (simp add: euclidean_representation)
    also have ... = (∑ b ∈ Basis - {k}. (x • b) *R b)
      by (auto simp: sum.remove [of _ k] inner_commute assms that)
    finally have x = (∑ b ∈ Basis - {k}. (x • b) *R b) .
    then show ?thesis
      by (simp add: span_finite)
  qed
qed
show ?thesis
  apply (rule span_subspace [symmetric])
  using assms
  apply (auto simp: inner_not_same_Basis intro: * subspace_hyperplane)
  done
qed

```

```

lemma dim_special_hyperplane:
  fixes k :: 'n::euclidean_space
  shows k ∈ Basis ⟹ dim {x. k • x = 0} = DIM('n) - 1
apply (simp add: special_hyperplane_span)
apply (rule dim_unique [OF subset_refl])
apply (auto simp: independent_substdbasis)
apply (metis member_remove remove_def span_base)
done

```

```

proposition dim_hyperplane:
  fixes a :: 'a::euclidean_space
  assumes a ≠ 0
  shows dim {x. a • x = 0} = DIM('a) - 1
proof -
  have span0: span {x. a • x = 0} = {x. a • x = 0}
    by (rule span_unique) (auto simp: subspace_hyperplane)
  then obtain B where independent B
    and Bsub: B ⊆ {x. a • x = 0}
    and subspB: {x. a • x = 0} ⊆ span B
    and card0: (card B = dim {x. a • x = 0})
    and ortho: pairwise orthogonal B
  using orthogonal_basis_exists by metis
  with assms have a ∉ span B
    by (metis (mono_tags, lifting) span_eq inner_eq_zero_iff mem_Collect_eq span0)
  then have ind: independent (insert a B)
    by (simp add: ⟨independent B⟩ independent_insert)

```

```

have finite B
  using ⟨independent B⟩ independent_bound by blast
have UNIV ⊆ span (insert a B)
proof fix y::'a
  obtain r z where z: y = r *R a + z a • z = 0
  apply (rule_tac r=(a • y) / (a • a) and z = y - ((a • y) / (a • a)) *R a in
that)
  using assms
  by (auto simp: algebra_simps)
show y ∈ span (insert a B)
  by (metis (mono_tags, lifting) z Bsub span_eq_iff
    add_diff_cancel_left' mem_Collect_eq span0 span_breakdown_eq span_subspace
subspB)
qed
then have dima: DIM('a) = dim(insert a B)
  by (metis independent_Basis span_Basis dim_eq_card top.extremum_uniqueI)
then show ?thesis
  by (metis (mono_tags, lifting) Bsub Diff_insert_absorb ⟨a ∉ span B⟩ ind_card0
    card_Diff_singleton dim_span indep_card_eq_dim_span insertI1 subsetCE
subspB)
qed

```

```

lemma lowdim_eq_hyperplane:
  fixes S :: 'a::euclidean_space set
  assumes dim S = DIM('a) - 1
  obtains a where a ≠ 0 and span S = {x. a • x = 0}
proof -
  have dimS: dim S < DIM('a)
    by (simp add: assms)
  then obtain b where b: b ≠ 0 span S ⊆ {a. b • a = 0}
    using lowdim_subset_hyperplane [of S] by fastforce
  show ?thesis
    apply (rule that[OF b(1)])
    apply (rule subspace_dim_equal)
    by (auto simp: assms b dim_hyperplane subspace_hyperplane)
qed

```

```

lemma dim_eq_hyperplane:
  fixes S :: 'n::euclidean_space set
  shows dim S = DIM('n) - 1 ⟷ (∃ a. a ≠ 0 ∧ span S = {x. a • x = 0})
by (metis One_nat_def dim_hyperplane dim_span lowdim_eq_hyperplane)

```

1.5.13 Orthogonal bases and Gram-Schmidt process

```

lemma pairwise_orthogonal_independent:
  assumes pairwise orthogonal S and 0 ∉ S
  shows independent S
proof -
  have 0: ∧ x y. [x ≠ y; x ∈ S; y ∈ S] ⟹ x • y = 0

```

```

    using assms by (simp add: pairwise_def orthogonal_def)
  have False if  $a \in S$  and  $a: a \in \text{span } (S - \{a\})$  for  $a$ 
  proof -
    obtain  $T U$  where  $T \subseteq S - \{a\}$   $a = (\sum_{v \in T}. U v *_R v)$ 
    using  $a$  by (force simp: span_explicit)
    then have  $a \cdot a = a \cdot (\sum_{v \in T}. U v *_R v)$ 
    by simp
    also have  $\dots = 0$ 
    apply (simp add: inner_sum_right)
    apply (rule comm_monoid_add_class.sum.neutral)
    by (metis 0 DiffE  $\langle T \subseteq S - \{a\} \rangle$  mult_not_zero singletonI subsetCE  $\langle a \in S \rangle$ )
    finally show ?thesis
    using  $\langle 0 \notin S \rangle \langle a \in S \rangle$  by auto
  qed
  then show ?thesis
  by (force simp: dependent_def)
qed

lemma pairwise_orthogonal_imp_finite:
  fixes  $S :: 'a::euclidean\_space$  set
  assumes pairwise_orthogonal  $S$ 
  shows finite  $S$ 
proof -
  have independent  $(S - \{0\})$ 
  apply (rule pairwise_orthogonal_independent)
  apply (metis Diff_iff assms pairwise_def)
  by blast
  then show ?thesis
  by (meson independent_imp_finite infinite_remove)
qed

lemma subspace_orthogonal_to_vector: subspace  $\{y. \text{orthogonal } x y\}$ 
  by (simp add: subspace_def orthogonal_clauses)

lemma subspace_orthogonal_to_vectors: subspace  $\{y. \forall x \in S. \text{orthogonal } x y\}$ 
  by (simp add: subspace_def orthogonal_clauses)

lemma orthogonal_to_span:
  assumes  $a: a \in \text{span } S$  and  $x: \bigwedge y. y \in S \implies \text{orthogonal } x y$ 
  shows orthogonal  $x a$ 
  by (metis a orthogonal_clauses(1,2,4) span_induct_alt x)

proposition Gram_Schmidt_step:
  fixes  $S :: 'a::euclidean\_space$  set
  assumes  $S$ : pairwise_orthogonal  $S$  and  $x: x \in \text{span } S$ 
  shows orthogonal  $x (a - (\sum_{b \in S}. (b \cdot a / (b \cdot b)) *_R b))$ 
proof -

```

```

have finite S
  by (simp add: S pairwise_orthogonal_imp_finite)
have orthogonal (a - (∑ b∈S. (b · a / (b · b)) *R b)) x
  if x ∈ S for x
proof -
  have a · x = (∑ y∈S. if y = x then y · a else 0)
    by (simp add: ⟨finite S⟩ inner_commute that)
  also have ... = (∑ b∈S. b · a * (b · x) / (b · b))
    apply (rule sum.cong [OF refl], simp)
    by (meson S orthogonal_def pairwise_def that)
  finally show ?thesis
    by (simp add: orthogonal_def algebra_simps inner_sum_left)
qed
then show ?thesis
  using orthogonal_to_span orthogonal_commute x by blast
qed

lemma orthogonal_extension_aux:
  fixes S :: 'a::euclidean_space set
  assumes finite T finite S pairwise_orthogonal S
  shows ∃ U. pairwise_orthogonal (S ∪ U) ∧ span (S ∪ U) = span (S ∪ T)
using assms
proof (induction arbitrary: S)
  case empty then show ?case
    by simp (metis sup_bot_right)
next
  case (insert a T)
  have 0: ∧x y. ‖x ≠ y; x ∈ S; y ∈ S‖ ⇒ x · y = 0
    using insert by (simp add: pairwise_def orthogonal_def)
  define a' where a' = a - (∑ b∈S. (b · a / (b · b)) *R b)
  obtain U where orthU: pairwise_orthogonal (S ∪ insert a' U)
    and spanU: span (insert a' S ∪ U) = span (insert a' S ∪ T)
  by (rule exE [OF insert.IH [of insert a' S]])
    (auto simp: Gram_Schmidt_step a'_def insert.premis orthogonal_commute
      pairwise_orthogonal_insert span_clauses)
  have orthS: ∧x. x ∈ S ⇒ a' · x = 0
    apply (simp add: a'_def)
    using Gram_Schmidt_step [OF ⟨pairwise_orthogonal S⟩]
    apply (force simp: orthogonal_def inner_commute span_superset [THEN sub-
      setD])
  done
  have span (S ∪ insert a' U) = span (insert a' (S ∪ T))
    using spanU by simp
  also have ... = span (insert a (S ∪ T))
    apply (rule eq_span_insert_eq)
    apply (simp add: a'_def span_neg span_sum span_base span_mul)
  done
  also have ... = span (S ∪ insert a T)

```



```

  by simp
  finally show ?case
  by (rule_tac x=insert a' U in exI) (use orthU in auto)
qed

```

```

proposition orthogonal_extension:
  fixes S :: 'a::euclidean_space set
  assumes S: pairwise orthogonal S
  obtains U where pairwise orthogonal (S ∪ U) span (S ∪ U) = span (S ∪ T)
proof -
  obtain B where finite B span B = span T
  using basis_subspace_exists [of span T] subspace_span by metis
  with orthogonal_extension_aux [of B S]
  obtain U where pairwise orthogonal (S ∪ U) span (S ∪ U) = span (S ∪ B)
  using assms pairwise_orthogonal_imp_finite by auto
  with ⟨span B = span T⟩ show ?thesis
  by (rule_tac U=U in that) (auto simp: span_Un)
qed

```

```

corollary orthogonal_extension_strong:
  fixes S :: 'a::euclidean_space set
  assumes S: pairwise orthogonal S
  obtains U where U ∩ (insert 0 S) = {} pairwise orthogonal (S ∪ U)
    span (S ∪ U) = span (S ∪ T)
proof -
  obtain U where pairwise orthogonal (S ∪ U) span (S ∪ U) = span (S ∪ T)
  using orthogonal_extension assms by blast
  then show ?thesis
  apply (rule_tac U = U - (insert 0 S) in that)
  apply blast
  apply (force simp: pairwise_def)
  apply (metis Un_Diff_cancel Un_insert_left span_redundant span_zero)
  done
qed

```

1.5.14 Decomposing a vector into parts in orthogonal subspaces

existence of orthonormal basis for a subspace.

```

lemma orthogonal_spanningset_subspace:
  fixes S :: 'a :: euclidean_space set
  assumes subspace S
  obtains B where B ⊆ S pairwise orthogonal B span B = S
proof -
  obtain B where B ⊆ S independent B S ⊆ span B card B = dim S
  using basis_exists by blast
  with orthogonal_extension [of {} B]
  show ?thesis

```

by (metis Un_empty_left assms pairwise_empty span_superset span_subspace that)
qed

lemma *orthogonal_basis_subspace*:

fixes $S :: 'a :: \text{euclidean\_space}$ set

assumes *subspace* S

obtains B where $0 \notin B$ $B \subseteq S$ pairwise orthogonal B independent B
card $B = \dim S$ span $B = S$

proof –

obtain B where $B \subseteq S$ pairwise orthogonal B span $B = S$

using *assms orthogonal_spanningset_subspace* by blast

then show ?thesis

apply (rule_tac $B = B - \{0\}$ in that)

apply (auto simp: indep_card_eq_dim_span pairwise_subset pairwise_orthogonal_independent elim: pairwise_subset)

done

qed

proposition *orthonormal_basis_subspace*:

fixes $S :: 'a :: \text{euclidean\_space}$ set

assumes *subspace* S

obtains B where $B \subseteq S$ pairwise orthogonal B

and $\bigwedge x. x \in B \implies \text{norm } x = 1$

and independent B card $B = \dim S$ span $B = S$

proof –

obtain B where $0 \notin B$ $B \subseteq S$

and *orth*: pairwise orthogonal B

and independent B card $B = \dim S$ span $B = S$

by (blast intro: *orthogonal_basis_subspace* [OF *assms*])

have 1: $(\lambda x. x /_R \text{norm } x) \text{ ` } B \subseteq S$

using $\langle \text{span } B = S \rangle$ span_superset span_mul by fastforce

have 2: pairwise orthogonal $((\lambda x. x /_R \text{norm } x) \text{ ` } B)$

using *orth* by (force simp: pairwise_def orthogonal_clauses)

have 3: $\bigwedge x. x \in (\lambda x. x /_R \text{norm } x) \text{ ` } B \implies \text{norm } x = 1$

by (metis (no_types, lifting) $0 \notin B$ image_iff norm_sgn sgn_div_norm)

have 4: independent $((\lambda x. x /_R \text{norm } x) \text{ ` } B)$

by (metis 2 3 norm_zero pairwise_orthogonal_independent zero_neq_one)

have *inj_on* $(\lambda x. x /_R \text{norm } x)$ B

proof

fix x y

assume $x \in B$ $y \in B$ $x /_R \text{norm } x = y /_R \text{norm } y$

moreover have $\bigwedge i. i \in B \implies \text{norm } (i /_R \text{norm } i) = 1$

using 3 by blast

ultimately show $x = y$

by (metis norm_eq_1 *orth* orthogonal_clauses(7) orthogonal_commute orthogonal_def pairwise_def zero_neq_one)

qed

then have 5: card $((\lambda x. x /_R \text{norm } x) \text{ ` } B) = \dim S$

```

  by (metis ‹card B = dim S› card_image)
  have 6: span ((λx. x /R norm x) ` B) = S
  by (metis 1 4 5 assms card_eq_dim independent_imp_finite span_subspace)
  show ?thesis
  by (rule that [OF 1 2 3 4 5 6])
qed

```

proposition *orthogonal_to_subspace_exists_gen*:

```

  fixes S :: 'a :: euclidean_space set
  assumes span S ⊆ span T
  obtains x where x ≠ 0 x ∈ span T ∧ y. y ∈ span S ⇒ orthogonal x y
proof -
  obtain B where B ⊆ span S and orthB: pairwise orthogonal B
    and ∧x. x ∈ B ⇒ norm x = 1
    and independent B card B = dim S span B = span S
  by (rule orthonormal_basis_subspace [of span S, OF subspace_span]) (auto)
  with assms obtain u where spanBT: span B ⊆ span T and u ∉ span B u ∈
span T
  by auto
  obtain C where orthBC: pairwise orthogonal (B ∪ C) and spanBC: span (B
∪ C) = span (B ∪ {u})
  by (blast intro: orthogonal_extension [OF orthB])
  show thesis
proof (cases C ⊆ insert 0 B)
  case True
  then have C ⊆ span B
    using span_eq
    by (metis span_insert_0 subset_trans)
  moreover have u ∈ span (B ∪ C)
    using ‹span (B ∪ C) = span (B ∪ {u})› span_superset by force
  ultimately show ?thesis
    using True ‹u ∉ span B›
    by (metis Un_insert_left span_insert_0 sup.orderE)
next
  case False
  then obtain x where x ∈ C x ≠ 0 x ∉ B
    by blast
  then have x ∈ span T
    by (metis (no_types, lifting) Un_insert_right Un_upper2 ‹u ∈ span T›
spanBT spanBC
    ‹u ∈ span T› insert_subset span_superset span_mono
    span_span subsetCE subset_trans sup_bot.comm_neutral)
  moreover have orthogonal x y if y ∈ span B for y
    using that
  proof (rule span_induct)
    show subspace {a. orthogonal x a}
      by (simp add: subspace_orthogonal_to_vector)
    show ∧b. b ∈ B ⇒ orthogonal x b

```

```

    by (metis Un_iff ⟨x ∈ C⟩ ⟨x ∉ B⟩ orthBC pairwise_def)
  qed
  ultimately show ?thesis
    using ⟨x ≠ 0⟩ that ⟨span B = span S⟩ by auto
  qed
qed

corollary orthogonal_to_subspace_exists:
  fixes S :: 'a :: euclidean_space set
  assumes dim S < DIM('a)
  obtains x where x ≠ 0 ∧ y. y ∈ span S ⇒ orthogonal x y
proof -
  have span S ⊂ UNIV
  by (metis (mono_tags) UNIV_I assms inner_eq_zero_iff less_le lowdim_subset_hyperplane
    mem_Collect_eq top.extremum_strict top.not_eq_extremum)
  with orthogonal_to_subspace_exists_gen [of S UNIV] that show ?thesis
    by (auto)
qed

corollary orthogonal_to_vector_exists:
  fixes x :: 'a :: euclidean_space
  assumes 2 ≤ DIM('a)
  obtains y where y ≠ 0 orthogonal x y
proof -
  have dim {x} < DIM('a)
    using assms by auto
  then show thesis
    by (rule orthogonal_to_subspace_exists) (simp add: orthogonal_commute span_base
    that)
qed

proposition orthogonal_subspace_decomp_exists:
  fixes S :: 'a :: euclidean_space set
  obtains y z
  where y ∈ span S
    and ∧w. w ∈ span S ⇒ orthogonal z w
    and x = y + z
proof -
  obtain T where 0 ∉ T T ⊆ span S pairwise orthogonal T independent T
    card T = dim (span S) span T = span S
    using orthogonal_basis_subspace subspace_span by blast
  let ?a = ∑ b∈T. (b • x / (b • b)) *R b
  have orth: orthogonal (x - ?a) w if w ∈ span S for w
    by (simp add: Gram_Schmidt_step ⟨pairwise orthogonal T⟩ ⟨span T = span
    S⟩
    orthogonal_commute that)
  show ?thesis
    apply (rule_tac y = ?a and z = x - ?a in that)
    apply (meson ⟨T ⊆ span S⟩ span_scale span_sum subsetCE)

```

```

    apply (fact orth, simp)
  done
qed

```

```

lemma orthogonal_subspace_decomp_unique:
  fixes S :: 'a :: euclidean_space set
  assumes x + y = x' + y'
    and ST: x ∈ span S x' ∈ span S y ∈ span T y' ∈ span T
    and orth:  $\bigwedge a b. [a \in S; b \in T] \implies \text{orthogonal } a b$ 
  shows x = x'  $\wedge$  y = y'
proof -
  have x + y - y' = x'
    by (simp add: assms)
  moreover have  $\bigwedge a b. [a \in \text{span } S; b \in \text{span } T] \implies \text{orthogonal } a b$ 
    by (meson orth orthogonal_commute orthogonal_to_span)
  ultimately have 0 = x' - x
    by (metis (full_types) add_diff_cancel_left' ST diff_right_commute orthogonal_clauses(10) orthogonal_clauses(5) orthogonal_self)
  with assms show ?thesis by auto
qed

```

```

lemma vector_in_orthogonal_spanningset:
  fixes a :: 'a::euclidean_space
  obtains S where a ∈ S pairwise orthogonal S span S = UNIV
  by (metis UNIV_I Un_iff empty_iff insert_subset orthogonal_extension pairwise_def
    pairwise_orthogonal_insert span_UNIV subsetI subset_antisym)

```

```

lemma vector_in_orthogonal_basis:
  fixes a :: 'a::euclidean_space
  assumes a ≠ 0
  obtains S where a ∈ S 0 ∉ S pairwise orthogonal S independent S finite S
    span S = UNIV card S = DIM('a)

```

```

proof -
  obtain S where S: a ∈ S pairwise orthogonal S span S = UNIV
    using vector_in_orthogonal_spanningset .
  show thesis
  proof
    show pairwise orthogonal (S - {0})
      using pairwise_mono S(2) by blast
    show independent (S - {0})
      by (simp add: ⟨pairwise orthogonal (S - {0})⟩ pairwise_orthogonal_independent)
    show finite (S - {0})
      using ⟨independent (S - {0})⟩ independent_imp_finite by blast
    show card (S - {0}) = DIM('a)
      using span_delete_0 [of S] S
      by (simp add: ⟨independent (S - {0})⟩ indep_card_eq_dim_span)
  qed (use S ⟨a ≠ 0⟩ in auto)
qed

```

```

lemma vector_in_orthonormal_basis:
  fixes a :: 'a::euclidean_space
  assumes norm a = 1
  obtains S where a ∈ S pairwise orthogonal S ∧ x. x ∈ S ⇒ norm x = 1
    independent S card S = DIM('a) span S = UNIV
proof -
  have a ≠ 0
  using assms by auto
  then obtain S where a ∈ S 0 ∉ S finite S
    and S: pairwise orthogonal S independent S span S = UNIV card S =
DIM('a)
  by (metis vector_in_orthogonal_basis)
  let ?S = (λx. x /R norm x) ' S
  show thesis
  proof
    show a ∈ ?S
    using ⟨a ∈ S⟩ assms image_iff by fastforce
  next
    show pairwise orthogonal ?S
    using ⟨pairwise orthogonal S⟩ by (auto simp: pairwise_def orthogonal_def)
    show ∧x. x ∈ (λx. x /R norm x) ' S ⇒ norm x = 1
    using ⟨0 ∉ S⟩ by (auto simp: field_split_simps)
    then show independent ?S
    by (metis ⟨pairwise orthogonal ((λx. x /R norm x) ' S)⟩ norm_zero pairwise_orthogonal_independent zero_neq_one)
    have inj_on (λx. x /R norm x) S
    unfolding inj_on_def
    by (metis (full_types) S(1) ⟨0 ∉ S⟩ inverse_nonzero_iff_nonzero norm_eq_zero
orthogonal_scaleR orthogonal_self pairwise_def)
    then show card ?S = DIM('a)
    by (simp add: card_image S)
    show span ?S = UNIV
    by (metis (no_types) ⟨0 ∉ S⟩ ⟨finite S⟩ ⟨span S = UNIV⟩
field_class.field_inverse_zero inverse_inverse_eq less_irrefl span_image_scale
zero_less_norm_iff)
  qed
qed

```

```

proposition dim_orthogonal_sum:
  fixes A :: 'a::euclidean_space set
  assumes ∧x y. [x ∈ A; y ∈ B] ⇒ x · y = 0
  shows dim(A ∪ B) = dim A + dim B
proof -
  have 1: ∧x y. [x ∈ span A; y ∈ B] ⇒ x · y = 0
  by (erule span_induct [OF_ subspace_hyperplane2]; simp add: assms)
  have ∧x y. [x ∈ span A; y ∈ span B] ⇒ x · y = 0
  using 1 by (simp add: span_induct [OF_ subspace_hyperplane])
  then have 0: ∧x y. [x ∈ span A; y ∈ span B] ⇒ x · y = 0

```

```

  by simp
  have  $\dim(A \cup B) = \dim(\text{span}(A \cup B))$ 
  by (simp)
  also have  $\text{span}(A \cup B) = ((\lambda(a, b). a + b) \text{ ` } (\text{span } A \times \text{span } B))$ 
  by (auto simp add: span_Un image_def)
  also have  $\dim \dots = \dim \{x + y \mid x \in \text{span } A \wedge y \in \text{span } B\}$ 
  by (auto intro!: arg_cong [where f=dim])
  also have  $\dots = \dim \{x + y \mid x \in \text{span } A \wedge y \in \text{span } B\} + \dim(\text{span } A \cap \text{span } B)$ 
  by (auto simp: dest: 0)
  also have  $\dots = \dim(\text{span } A) + \dim(\text{span } B)$ 
  by (rule dim_sums_Int) (auto)
  also have  $\dots = \dim A + \dim B$ 
  by (simp)
  finally show ?thesis .
qed

```

lemma *dim_subspace_orthogonal_to_vectors:*

```

  fixes  $A :: 'a::\text{euclidean\_space}$  set
  assumes subspace  $A$  subspace  $B$   $A \subseteq B$ 
  shows  $\dim \{y \in B. \forall x \in A. \text{orthogonal } x \ y\} + \dim A = \dim B$ 
proof -
  have  $\dim(\text{span}(\{y \in B. \forall x \in A. \text{orthogonal } x \ y\} \cup A)) = \dim(\text{span } B)$ 
  proof (rule arg_cong [where f=dim, OF subset_antisym])
    show  $\text{span}(\{y \in B. \forall x \in A. \text{orthogonal } x \ y\} \cup A) \subseteq \text{span } B$ 
    by (simp add:  $\langle A \subseteq B \rangle$  Collect_restrict span_mono)
  next
    have *:  $x \in \text{span}(\{y \in B. \forall x \in A. \text{orthogonal } x \ y\} \cup A)$ 
    if  $x \in B$  for  $x$ 
    proof -
      obtain  $y \ z$  where  $x = y + z$   $y \in \text{span } A$  and orth:  $\bigwedge w. w \in \text{span } A \implies \text{orthogonal } z \ w$ 
      using orthogonal_subspace_decomp_exists [of  $A \ x$ ] that by auto
      have  $y \in \text{span } B$ 
      using  $\langle y \in \text{span } A \rangle$  assms(3) span_mono by blast
      then have  $z \in \{a \in B. \forall x. x \in A \longrightarrow \text{orthogonal } x \ a\}$ 
      apply simp
      using  $\langle x = y + z \rangle$  assms(1) assms(2) orth orthogonal_commute span_add_eq
      span_eq_iff that by blast
      then have  $z: z \in \text{span} \{y \in B. \forall x \in A. \text{orthogonal } x \ y\}$ 
      by (meson span_superset subset_iff)
      then show ?thesis
      apply (auto simp: span_Un image_def  $\langle x = y + z \rangle \langle y \in \text{span } A \rangle$ )
      using  $\langle y \in \text{span } A \rangle$  add.commute by blast
    qed
  show  $\text{span } B \subseteq \text{span}(\{y \in B. \forall x \in A. \text{orthogonal } x \ y\} \cup A)$ 
  by (rule span_minimal) (auto intro: * span_minimal)
qed
then show ?thesis

```

```

    by (metis (no_types, lifting) dim_orthogonal_sum dim_span mem_Collect_eq
        orthogonal_commute orthogonal_def)
qed

```

1.5.15 Linear functions are (uniformly) continuous on any set

1.5.16 Topological properties of linear functions

```

lemma linear_lim_0:
  assumes bounded_linear f
  shows (f ⟶ 0) (at (0))
proof -
  interpret f: bounded_linear f by fact
  have (f ⟶ f 0) (at 0)
    using tendsto_ident_at by (rule f.tendsto)
  then show ?thesis unfolding f.zero .
qed

```

```

lemma linear_continuous_at:
  assumes bounded_linear f
  shows continuous (at a) f
  unfolding continuous_at using assms
  apply (rule bounded_linear.tendsto)
  apply (rule tendsto_ident_at)
  done

```

```

lemma linear_continuous_within:
  bounded_linear f ⟹ continuous (at x within s) f
  using continuous_at_imp_continuous_at_within linear_continuous_at by blast

```

```

lemma linear_continuous_on:
  bounded_linear f ⟹ continuous_on s f
  using continuous_at_imp_continuous_on[of s f] using linear_continuous_at[of
    f] by auto

```

```

lemma Lim_linear:
  fixes f :: 'a::euclidean_space ⟹ 'b::euclidean_space and h :: 'b ⟹ 'c::real_normed_vector
  assumes (f ⟶ l) F linear h
  shows ((λx. h(f x)) ⟶ h l) F
proof -
  obtain B where B: B > 0 ∧ x. norm (h x) ≤ B * norm x
    using linear_bounded_pos [OF ⟨linear h⟩] by blast
  show ?thesis
    unfolding tendsto_iff
  proof (intro allI impI)
    show ∀ F x in F. dist (h (f x)) (h l) < e if e > 0 for e
    proof -
      have ∀ F x in F. dist (f x) l < e/B
        by (simp add: ⟨0 < B⟩ assms(1) tendstoD that)

```



```

    then show ?thesis
      unfolding dist_norm
    proof (rule eventually_mono)
      show norm (h (f x) - h l) < e if norm (f x - l) < e / B for x
        using that B
        apply (simp add: field_split_simps)
        by (metis ‹linear h› le_less_trans linear_diff)
    qed
  qed
qed
qed

lemma linear_continuous_compose:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space and g :: 'b  $\Rightarrow$  'c::real_normed_vector
  assumes continuous F f linear g
  shows continuous F ( $\lambda x. g(f x)$ )
  using assms unfolding continuous_def by (rule Lim_linear)

lemma linear_continuous_on_compose:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space and g :: 'b  $\Rightarrow$  'c::real_normed_vector
  assumes continuous_on S f linear g
  shows continuous_on S ( $\lambda x. g(f x)$ )
  using assms by (simp add: continuous_on_eq_continuous_within linear_continuous_compose)

Also bilinear functions, in composition form

lemma bilinear_continuous_compose:
  fixes h :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'c::real_normed_vector
  assumes continuous F f continuous F g bilinear h
  shows continuous F ( $\lambda x. h(f x)(g x)$ )
  using assms bilinear_conv_bounded_bilinear bounded_bilinear.continuous by
blast

lemma bilinear_continuous_on_compose:
  fixes h :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'c::real_normed_vector
  and f :: 'd::t2_space  $\Rightarrow$  'a
  assumes continuous_on S f continuous_on S g bilinear h
  shows continuous_on S ( $\lambda x. h(f x)(g x)$ )
  using assms by (simp add: continuous_on_eq_continuous_within bilinear_continuous_compose)

end

```

1.6 Affine Sets

```

theory Affine
imports Linear_Algebra
begin

```

```

lemma if_smult: (if P then x else (y::real)) *R v = (if P then x *R v else y *R v)
  by (fact if_distrib)

```

```

lemma sum_delta_notmem:
  assumes  $x \notin s$ 
  shows  $\text{sum } (\lambda y. \text{if } (y = x) \text{ then } P \ x \text{ else } Q \ y) \ s = \text{sum } Q \ s$ 
    and  $\text{sum } (\lambda y. \text{if } (x = y) \text{ then } P \ x \text{ else } Q \ y) \ s = \text{sum } Q \ s$ 
    and  $\text{sum } (\lambda y. \text{if } (y = x) \text{ then } P \ y \text{ else } Q \ y) \ s = \text{sum } Q \ s$ 
    and  $\text{sum } (\lambda y. \text{if } (x = y) \text{ then } P \ y \text{ else } Q \ y) \ s = \text{sum } Q \ s$ 
  apply (rule_tac [!] sum.cong)
  using assms
  apply auto
  done

lemmas independent_finite = independent_imp_finite

lemma span_substd_basis:
  assumes  $d: d \subseteq \text{Basis}$ 
  shows  $\text{span } d = \{x. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\}$ 
  (is  $\_ = ?B$ )
proof -
  have  $d \subseteq ?B$ 
    using  $d$  by (auto simp: inner_Basis)
  moreover have  $s: \text{subspace } ?B$ 
    using subspace_substandard[of  $\lambda i. i \notin d$ ] .
  ultimately have  $\text{span } d \subseteq ?B$ 
    using span_mono[of  $d \ ?B$ ] span_eq_iff[of  $?B$ ] by blast
  moreover have  $*$ :  $\text{card } d \leq \text{dim } (\text{span } d)$ 
    using independent_card_le_dim[of  $d \ \text{span } d$ ] independent_substdbasis[OF assms]
    span_superset[of  $d$ ]
  by auto
  moreover from  $*$  have  $\text{dim } ?B \leq \text{dim } (\text{span } d)$ 
    using dim_substandard[OF assms] by auto
  ultimately show  $?thesis$ 
    using  $s \ \text{subspace\_dim\_equal}$ [of  $\text{span } d \ ?B$ ] subspace_span[of  $d$ ] by auto
qed

lemma basis_to_substdbasis_subspace_isomorphism:
  fixes  $B :: 'a::\text{euclidean\_space} \text{ set}$ 
  assumes independent  $B$ 
  shows  $\exists f \ d :: 'a \text{ set}. \text{card } d = \text{card } B \wedge \text{linear } f \wedge f' B = d \wedge$ 
 $f' \text{span } B = \{x. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\} \wedge \text{inj\_on } f \ (\text{span } B) \wedge d \subseteq$ 
 $\text{Basis}$ 
proof -
  have  $B: \text{card } B = \text{dim } B$ 
    using dim_unique[of  $B \ B \ \text{card } B$ ] assms span_superset[of  $B$ ] by auto
  have  $\text{dim } B \leq \text{card } (\text{Basis} :: 'a \text{ set})$ 
    using dim_subset_UNIV[of  $B$ ] by simp
  from ex_card[OF this] obtain  $d :: 'a \text{ set}$  where  $d: d \subseteq \text{Basis}$  and  $t: \text{card } d =$ 
 $\text{dim } B$ 
  by auto

```

```

let ?t = {x::'a::euclidean_space.  $\forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0$ }
have  $\exists f. \text{linear } f \wedge f \text{ ` } B = d \wedge f \text{ ` } \text{span } B = ?t \wedge \text{inj\_on } f \text{ (span } B)$ 
proof (intro basis_to_basis_subspace_isomorphism subspace_span subspace_substandard
span_superset)
  show  $d \subseteq \{x. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\}$ 
  using d_inner_not_same_Basis by blast
qed (auto simp: span_substd_basis independent_substdbasis dim_substandard d
t B assms)
with t <card B = dim B> d show ?thesis by auto
qed

```

1.6.1 Affine set and affine hull

```

definition affine :: 'a::real_vector set  $\Rightarrow$  bool
  where affine s  $\longleftrightarrow (\forall x \in s. \forall y \in s. \forall u v. u + v = 1 \longrightarrow u *_R x + v *_R y \in s)$ 

```

```

lemma affine_alt: affine s  $\longleftrightarrow (\forall x \in s. \forall y \in s. \forall u::real. (1 - u) *_R x + u *_R y \in s)$ 
unfolding affine_def by (metis eq_diff_eq')

```

```

lemma affine_empty [iff]: affine {}
unfolding affine_def by auto

```

```

lemma affine_sing [iff]: affine {x}
unfolding affine_alt by (auto simp: scaleR_left_distrib [symmetric])

```

```

lemma affine_UNIV [iff]: affine UNIV
unfolding affine_def by auto

```

```

lemma affine_Inter [intro]:  $(\bigwedge s. s \in f \Longrightarrow \text{affine } s) \Longrightarrow \text{affine } (\bigcap f)$ 
unfolding affine_def by auto

```

```

lemma affine_Int[intro]: affine s  $\Longrightarrow$  affine t  $\Longrightarrow$  affine (s  $\cap$  t)
unfolding affine_def by auto

```

```

lemma affine_scaling: affine s  $\Longrightarrow$  affine (image  $(\lambda x. c *_R x)$  s)
apply (clarsimp simp add: affine_def)
apply (rule_tac x=u *_R x + v *_R y in image_eqI)
apply (auto simp: algebra_simps)
done

```

```

lemma affine_affine_hull [simp]: affine(affine hull s)
unfolding hull_def
using affine_Inter[of {t. affine t  $\wedge$  s  $\subseteq$  t}] by auto

```

```

lemma affine_hull_eq[simp]: (affine hull s = s)  $\longleftrightarrow$  affine s
by (metis affine_affine_hull hull_same)

```

```

lemma affine_hyperplane: affine {x. a  $\cdot$  x = b}

```

by (simp add: affine_def algebra_simps) (metis distrib_right mult.left_neutral)

Some explicit formulations

Formalized by Lars Schewe.

lemma *affine*:

fixes $V::'a::\text{real\_vector\_set}$

shows *affine* $V \longleftrightarrow$

$(\forall S\ u. \text{finite } S \wedge S \neq \{\} \wedge S \subseteq V \wedge \text{sum } u\ S = 1 \longrightarrow (\sum_{x \in S} u\ x *_{\text{R}} x) \in V)$

proof –

have $u *_{\text{R}} x + v *_{\text{R}} y \in V$ if $x \in V\ y \in V\ u + v = (1::\text{real})$

and $*$: $\bigwedge S\ u. [\text{finite } S; S \neq \{\}; S \subseteq V; \text{sum } u\ S = 1] \implies (\sum_{x \in S} u\ x *_{\text{R}} x) \in V$ for $x\ y\ u\ v$

proof (cases $x = y$)

case *True*

then show ?thesis

using that by (metis scaleR_add_left scaleR_one)

next

case *False*

then show ?thesis

using that $*[\text{of } \{x, y\}\ \lambda w. \text{if } w = x \text{ then } u \text{ else } v]$ by auto

qed

moreover have $(\sum_{x \in S} u\ x *_{\text{R}} x) \in V$

if $*$: $\bigwedge x\ y\ u\ v. [x \in V; y \in V; u + v = 1] \implies u *_{\text{R}} x + v *_{\text{R}} y \in V$

and $\text{finite } S\ S \neq \{\}\ S \subseteq V\ \text{sum } u\ S = 1$ for $S\ u$

proof –

define n where $n = \text{card } S$

consider $\text{card } S = 0 \mid \text{card } S = 1 \mid \text{card } S = 2 \mid \text{card } S > 2$ by *linarith*

then show $(\sum_{x \in S} u\ x *_{\text{R}} x) \in V$

proof cases

assume $\text{card } S = 1$

then obtain a where $S = \{a\}$

by (auto simp: card_Suc_eq)

then show ?thesis

using that by simp

next

assume $\text{card } S = 2$

then obtain $a\ b$ where $S = \{a, b\}$

by (metis Suc_1 card_1_singletonE card_Suc_eq)

then show ?thesis

using $*[\text{of } a\ b]$ that

by (auto simp: sum_clauses(2))

next

assume $\text{card } S > 2$

then show ?thesis using that n_def

proof (induct n arbitrary: $u\ S$)

case 0

then show ?case by auto

```

next
case (Suc n u S)
have sum u S = card S if  $\neg (\exists x \in S. u x \neq 1)$ 
  using that unfolding card_eq_sum by auto
with Suc.premis obtain x where  $x \in S$  and  $x: u x \neq 1$  by force
have c:  $\text{card } (S - \{x\}) = \text{card } S - 1$ 
  by (simp add: Suc.premis(3)  $\langle x \in S \rangle$ )
have sum u (S - {x}) = 1 - u x
  by (simp add: Suc.premis sum_diff1  $\langle x \in S \rangle$ )
with x have eq1:  $\text{inverse } (1 - u x) * \text{sum } u (S - \{x\}) = 1$ 
  by auto
have inV:  $(\sum y \in S - \{x\}. \text{inverse } (1 - u x) * u y) *_R y \in V$ 
proof (cases  $\text{card } (S - \{x\}) > 2$ )
case True
then have S:  $S - \{x\} \neq \{\}$   $\text{card } (S - \{x\}) = n$ 
  using Suc.premis c by force+
show ?thesis
proof (rule Suc.hyps)
show  $(\sum a \in S - \{x\}. \text{inverse } (1 - u x) * u a) = 1$ 
  by (auto simp: eq1 sum_distrib_left[symmetric])
qed (use S Suc.premis True in auto)
next
case False
then have card (S - {x}) = Suc (Suc 0)
  using Suc.premis c by auto
then obtain a b where  $ab: (S - \{x\}) = \{a, b\}$   $a \neq b$ 
  unfolding card_Suc_eq by auto
then show ?thesis
  using eq1  $\langle S \subseteq V \rangle$ 
  by (auto simp: sum_distrib_left distrib_left intro!: Suc.premis(2)[of a b])
qed
have  $u x + (1 - u x) = 1 \implies$ 
 $u x *_R x + (1 - u x) *_R ((\sum y \in S - \{x\}. u y *_R y) /_R (1 - u x)) \in V$ 
  by (rule Suc.premis) (use  $\langle x \in S \rangle$  Suc.premis inV in  $\langle \text{auto simp:}$ 
scaleR_right.sum  $\rangle$ )
moreover have  $(\sum a \in S. u a *_R a) = u x *_R x + (\sum a \in S - \{x\}. u a *_R$ 
a)
  by (meson Suc.premis(3) sum.remove  $\langle x \in S \rangle$ )
ultimately show  $(\sum x \in S. u x *_R x) \in V$ 
  by (simp add: x)
qed
qed (use  $\langle S \neq \{\} \rangle$   $\langle \text{finite } S \rangle$  in auto)
qed
ultimately show ?thesis
  unfolding affine_def by meson
qed

```

lemma affine_hull_explicit:

```

    affine hull p = {y.  $\exists S u. \text{finite } S \wedge S \neq \{\} \wedge S \subseteq p \wedge \text{sum } u \ S = 1 \wedge \text{sum } (\lambda v. u \ v \ *_R \ v) \ S = y\}$ 
    (is _ = ?rhs)
  proof (rule hull_unique)
    show p  $\subseteq$  ?rhs
    proof (intro subsetI CollectI exI conjI)
      show  $\bigwedge x. \text{sum } (\lambda z. 1) \ \{x\} = 1$ 
        by auto
    qed auto
    show ?rhs  $\subseteq T$  if p  $\subseteq T$  affine T for T
      using that unfolding affine by blast
    show affine ?rhs
      unfolding affine_def
    proof clarify
      fix u v :: real and sx ux sy uy
      assume uv: u + v = 1
      and x: finite sx sx  $\neq \{\}$  sx  $\subseteq p$  sum ux sx = (1::real)
      and y: finite sy sy  $\neq \{\}$  sy  $\subseteq p$  sum uy sy = (1::real)
      have **: (sx  $\cup$  sy)  $\cap$  sx = sx (sx  $\cup$  sy)  $\cap$  sy = sy
        by auto
      show  $\exists S w. \text{finite } S \wedge S \neq \{\} \wedge S \subseteq p \wedge$ 
        sum w S = 1  $\wedge (\sum_{v \in S. w \ v \ *_R \ v} = u \ *_R (\sum_{v \in sx. ux \ v \ *_R \ v}) + v \ *_R$ 
        ( $\sum_{v \in sy. uy \ v \ *_R \ v}$ )
      proof (intro exI conjI)
        show finite (sx  $\cup$  sy)
          using x y by auto
        show sum  $(\lambda i. (\text{if } i \in sx \text{ then } u \ * \ ux \ i \ \text{else } 0) + (\text{if } i \in sy \text{ then } v \ * \ uy \ i \ \text{else } 0))$ 
        (sx  $\cup$  sy) = 1
          using x y uv
          by (simp add: sum_Un sum.distrib sum.inter_restrict[symmetric] sum_distrib_left
            [symmetric] **)
        have  $(\sum_{i \in sx \cup sy. ((\text{if } i \in sx \text{ then } u \ * \ ux \ i \ \text{else } 0) + (\text{if } i \in sy \text{ then } v \ * \ uy \ i \ \text{else } 0)) \ *_R \ i})$ 
          =  $(\sum_{i \in sx. (u \ * \ ux \ i) \ *_R \ i}) + (\sum_{i \in sy. (v \ * \ uy \ i) \ *_R \ i})$ 
          using x y
          unfolding scaleR_left_distrib scaleR_zero_left if_smult
          by (simp add: sum_Un sum.distrib sum.inter_restrict[symmetric] **)
        also have ... = u *_R ( $\sum_{v \in sx. ux \ v \ *_R \ v}$ ) + v *_R ( $\sum_{v \in sy. uy \ v \ *_R \ v}$ )
          unfolding scaleR_scaleR[symmetric] scaleR_right.sum [symmetric] by blast
        finally show  $(\sum_{i \in sx \cup sy. ((\text{if } i \in sx \text{ then } u \ * \ ux \ i \ \text{else } 0) + (\text{if } i \in sy \text{ then } v \ * \ uy \ i \ \text{else } 0)) \ *_R \ i})$ 
          = u *_R ( $\sum_{v \in sx. ux \ v \ *_R \ v}$ ) + v *_R ( $\sum_{v \in sy. uy \ v \ *_R \ v}$ ) .
      qed (use x y in auto)
    qed
  qed

```

lemma affine_hull_finite:

```

  assumes finite S
  shows affine hull S = {y.  $\exists u. \text{sum } u \ S = 1 \wedge \text{sum } (\lambda v. u \ v \ *_R \ v) \ S = y\}$ 

```

```

proof -
  have *:  $\exists h. \text{sum } h \ S = 1 \wedge (\sum_{v \in S}. h \ v \ *_R \ v) = x$ 
    if  $F \subseteq S$  finite  $F \ F \neq \{\}$  and  $\text{sum}: \text{sum } u \ F = 1$  and  $x: (\sum_{v \in F}. u \ v \ *_R \ v)$ 
= x for  $x \ F \ u$ 
  proof -
    have  $S \cap F = F$ 
      using that by auto
    show ?thesis
  proof (intro exI conjI)
    show  $(\sum_{x \in S}. \text{if } x \in F \text{ then } u \ x \text{ else } 0) = 1$ 
      by (metis (mono_tags, lifting)  $\langle S \cap F = F \rangle$  assms sum.inter_restrict sum)
    show  $(\sum_{v \in S}. (\text{if } v \in F \text{ then } u \ v \text{ else } 0) \ *_R \ v) = x$ 
      by (simp add: if_smult cong: if_cong) (metis (no_types)  $\langle S \cap F = F \rangle$ 
assms sum.inter_restrict x)
    qed
  qed
show ?thesis
  unfolding affine_hull_explicit using assms
  by (fastforce dest: *)
qed

```

Stepping theorems and hence small special cases

```

lemma affine_hull_empty[simp]: affine hull  $\{\}$  =  $\{\}$ 
  by simp

```

```

lemma affine_hull_finite_step:
  fixes  $y :: 'a::\text{real\_vector}$ 
  shows finite S  $\implies$ 
     $(\exists u. \text{sum } u \ (\text{insert } a \ S) = w \wedge \text{sum } (\lambda x. u \ x \ *_R \ x) \ (\text{insert } a \ S) = y) \longleftrightarrow$ 
     $(\exists v \ u. \text{sum } u \ S = w - v \wedge \text{sum } (\lambda x. u \ x \ *_R \ x) \ S = y - v \ *_R \ a) \ (\text{is } \_ \implies$ 
    ?lhs = ?rhs)
proof -
  assume fin: finite S
  show ?lhs = ?rhs
  proof
    assume ?lhs
    then obtain  $u$  where  $u: \text{sum } u \ (\text{insert } a \ S) = w \wedge (\sum_{x \in \text{insert } a \ S}. u \ x \ *_R$ 
 $x) = y$ 
    by auto
    show ?rhs
  proof (cases  $a \in S$ )
    case True
      then show ?thesis
      using  $u$  by (simp add: insert_absorb) (metis diff_zero real_vector.scale_zero_left)
    next
      case False
      show ?thesis
      by (rule exI [where  $x=u \ a$ ]) (use  $u$  fin False in auto)
  qed

```

```

    qed
  next
    assume ?rhs
    then obtain v u where vu: sum u S = w - v (∑ x∈S. u x *R x) = y - v
  *R a
    by auto
    have *: ∧ x M. (if x = a then v else M) *R x = (if x = a then v *R x else M
  *R x)
    by auto
    show ?lhs
    proof (cases a ∈ S)
      case True
      show ?thesis
        by (rule exI [where x=λx. (if x=a then v else 0) + u x])
          (simp add: True scaleR_left_distrib sum.distrib sum_clauses fin vu *
cong: if_cong)
    next
      case False
      then show ?thesis
        apply (rule_tac x=λx. if x=a then v else u x in exI)
        apply (simp add: vu sum_clauses(2)[OF fin] *)
        by (simp add: sum_delta_notmem(3) vu)
    qed
  qed
qed

```

```

lemma affine_hull_2:
  fixes a b :: 'a::real_vector
  shows affine_hull {a,b} = {u *R a + v *R b | u v. (u + v = 1)}
  (is ?lhs = ?rhs)
proof -
  have *:
    ∧ x y z. z = x - y ⟷ y + z = (x::real)
    ∧ x y z. z = x - y ⟷ y + z = (x::'a) by auto
  have ?lhs = {y. ∃ u. sum u {a, b} = 1 ∧ (∑ v∈{a, b}. u v *R v) = y}
    using affine_hull_finite[of {a,b}] by auto
  also have ... = {y. ∃ v u. u b = 1 - v ∧ u b *R b = y - v *R a}
    by (simp add: affine_hull_finite_step[of {b} a])
  also have ... = ?rhs unfolding * by auto
  finally show ?thesis by auto
qed

```

```

lemma affine_hull_3:
  fixes a b c :: 'a::real_vector
  shows affine_hull {a,b,c} = { u *R a + v *R b + w *R c | u v w. u + v + w =
1}
proof -
  have *:
    ∧ x y z. z = x - y ⟷ y + z = (x::real)

```



```

   $\wedge x\ y\ z. z = x - y \longleftrightarrow y + z = (x::'a)$  by auto
show ?thesis
  apply (simp add: affine_hull_finite affine_hull_finite_step)
  unfolding *
  apply safe
  apply (metis add.assoc)
  apply (rule_tac x=u in exI, force)
  done
qed

```

```

lemma mem_affine:
  assumes affine S x ∈ S y ∈ S u + v = 1
  shows  $u *_R x + v *_R y \in S$ 
  using assms affine_def[of S] by auto

```

```

lemma mem_affine_3:
  assumes affine S x ∈ S y ∈ S z ∈ S u + v + w = 1
  shows  $u *_R x + v *_R y + w *_R z \in S$ 
proof –
  have  $u *_R x + v *_R y + w *_R z \in \text{affine hull } \{x, y, z\}$ 
    using affine_hull_3[of x y z] assms by auto
  moreover
  have  $\text{affine hull } \{x, y, z\} \subseteq \text{affine hull } S$ 
    using hull_mono[of {x, y, z} S] assms by auto
  moreover
  have  $\text{affine hull } S = S$ 
    using assms affine_hull_eq[of S] by auto
  ultimately show ?thesis by auto
qed

```

```

lemma mem_affine_3_minus:
  assumes affine S x ∈ S y ∈ S z ∈ S
  shows  $x + v *_R (y - z) \in S$ 
  using mem_affine_3[of S x y z 1 v -v] assms
  by (simp add: algebra_simps)

```

```

corollary mem_affine_3_minus2:
   $\llbracket \text{affine } S; x \in S; y \in S; z \in S \rrbracket \implies x - v *_R (y - z) \in S$ 
  by (metis add_uminus_conv_diff mem_affine_3_minus real_vector.scale_minus_left)

```

Some relations between affine hull and subspaces

```

lemma affine_hull_insert_subset_span:
   $\text{affine hull } (\text{insert } a\ S) \subseteq \{a + v \mid v \in \text{span } \{x - a \mid x \in S\}\}$ 
proof –
  have  $\exists v\ T\ u. x = a + v \wedge (\text{finite } T \wedge T \subseteq \{x - a \mid x \in S\} \wedge (\sum_{v \in T} u\ v$ 
 $*_R v) = v)$ 
    if finite F F ≠ {} F ⊆ insert a S sum u F = 1  $(\sum_{v \in F} u\ v *_R v) = x$ 
    for  $x\ F\ u$ 

```

```

proof -
  have *:  $(\lambda x. x - a) \text{ ' } (F - \{a\}) \subseteq \{x - a \mid x. x \in S\}$ 
    using that by auto
  show ?thesis
proof (intro exI conjI)
  show finite  $((\lambda x. x - a) \text{ ' } (F - \{a\}))$ 
    by (simp add: that(1))
  show  $(\sum v \in (\lambda x. x - a) \text{ ' } (F - \{a\}). u(v+a) *_R v) = x - a$ 
    by (simp add: sum.reindex[unfolded inj_on_def] algebra_simps
      sum_subtractf scaleR_left.sum[symmetric] sum_diff1 that)
  qed (use  $\langle F \subseteq \text{insert } a \ S \rangle$  in auto)
qed
then show ?thesis
  unfolding affine_hull_explicit span_explicit by fast
qed

lemma affine_hull_insert_span:
  assumes  $a \notin S$ 
  shows affine_hull (insert  $a \ S$ ) =  $\{a + v \mid v. v \in \text{span } \{x - a \mid x. x \in S\}\}$ 
proof -
  have *:  $\exists G u. \text{finite } G \wedge G \neq \{\} \wedge G \subseteq \text{insert } a \ S \wedge \text{sum } u \ G = 1 \wedge (\sum v \in G. u \ v *_R v) = y$ 
    if  $v \in \text{span } \{x - a \mid x. x \in S\}$   $y = a + v$  for  $y \ v$ 
  proof -
    from that
    obtain  $T \ u$  where  $u: \text{finite } T \ T \subseteq \{x - a \mid x. x \in S\} \ a + (\sum v \in T. u \ v *_R v) = y$ 
    unfolding span_explicit by auto
    define  $F$  where  $F = (\lambda x. x + a) \text{ ' } T$ 
    have  $F: \text{finite } F \ F \subseteq S \ (\sum v \in F. u \ (v - a) *_R (v - a)) = y - a$ 
    unfolding  $F\_def$  using  $u$  by (auto simp: sum.reindex[unfolded inj_on_def])
    have *:  $F \cap \{a\} = \{\} \ F \cap -\{a\} = F$ 
    using  $F$  assms by auto
    show  $\exists G u. \text{finite } G \wedge G \neq \{\} \wedge G \subseteq \text{insert } a \ S \wedge \text{sum } u \ G = 1 \wedge (\sum v \in G. u \ v *_R v) = y$ 
    apply (rule_tac  $x = \text{insert } a \ F$  in exI)
    apply (rule_tac  $x = \lambda x. \text{if } x=a \text{ then } 1 - \text{sum } (\lambda x. u \ (x - a)) \ F \text{ else } u \ (x - a)$  in exI)
    using assms  $F$ 
    apply (auto simp: sum_clauses sum.If_cases if_smult sum_subtractf scaleR_left.sum algebra_simps *)
    done
  qed
show ?thesis
  by (intro subset_antisym affine_hull_insert_subset_span) (auto simp: affine_hull_explicit dest!: *)
qed

lemma affine_hull_span:

```

```

assumes  $a \in S$ 
shows  $\text{affine\_hull } S = \{a + v \mid v. v \in \text{span } \{x - a \mid x. x \in S - \{a\}\}\}$ 
using  $\text{affine\_hull\_insert\_span}[of\ a\ S - \{a\}, \text{unfolded insert\_Diff}[OF\ \text{assms}]]$ 
by auto

```

Parallel affine sets

```

definition  $\text{affine\_parallel} :: 'a::\text{real\_vector\_set} \Rightarrow 'a::\text{real\_vector\_set} \Rightarrow \text{bool}$ 
where  $\text{affine\_parallel } S\ T \longleftrightarrow (\exists a. T = (\lambda x. a + x) ` S)$ 

```

```

lemma  $\text{affine\_parallel\_expl\_aux}$ :
fixes  $S\ T :: 'a::\text{real\_vector\_set}$ 
assumes  $\bigwedge x. x \in S \longleftrightarrow a + x \in T$ 
shows  $T = (\lambda x. a + x) ` S$ 
proof -
have  $x \in ((\lambda x. a + x) ` S)$  if  $x \in T$  for  $x$ 
using that
by (simp add: image_iff) (metis add.commute diff_add_cancel assms)
moreover have  $T \supseteq (\lambda x. a + x) ` S$ 
using assms by auto
ultimately show ?thesis by auto
qed

```

```

lemma  $\text{affine\_parallel\_expl}$ :  $\text{affine\_parallel } S\ T \longleftrightarrow (\exists a. \forall x. x \in S \longleftrightarrow a + x \in T)$ 
by (auto simp add: affine_parallel_def)
(use affine_parallel_expl_aux [of S _ T] in blast)

```

```

lemma  $\text{affine\_parallel\_reflex}$ :  $\text{affine\_parallel } S\ S$ 
unfolding  $\text{affine\_parallel\_def}$ 
using  $\text{image\_add\_0}$  by blast

```

```

lemma  $\text{affine\_parallel\_commut}$ :
assumes  $\text{affine\_parallel } A\ B$ 
shows  $\text{affine\_parallel } B\ A$ 
proof -
from assms obtain  $a$  where  $B = (\lambda x. a + x) ` A$ 
unfolding  $\text{affine\_parallel\_def}$  by auto
have [simp]:  $(\lambda x. x - a) = \text{plus } (-\ a)$  by (simp add: fun_eq_iff)
from  $B$  show ?thesis
using  $\text{translation\_galois}[of\ B\ a\ A]$ 
unfolding  $\text{affine\_parallel\_def}$  by blast
qed

```

```

lemma  $\text{affine\_parallel\_assoc}$ :
assumes  $\text{affine\_parallel } A\ B$ 
and  $\text{affine\_parallel } B\ C$ 
shows  $\text{affine\_parallel } A\ C$ 
proof -

```

```

from assms obtain ab where  $B = (\lambda x. ab + x) \text{ ' } A$ 
  unfolding affine_parallel_def by auto
moreover
from assms obtain bc where  $C = (\lambda x. bc + x) \text{ ' } B$ 
  unfolding affine_parallel_def by auto
ultimately show ?thesis
  using translation_assoc[of bc ab A] unfolding affine_parallel_def by auto
qed

```

```

lemma affine_translation_aux:
  fixes a :: 'a::real_vector
  assumes affine  $((\lambda x. a + x) \text{ ' } S)$ 
  shows affine S
proof -
  {
    fix x y u v
    assume xy:  $x \in S \ y \in S \ (u :: real) + v = 1$ 
    then have  $(a + x) \in ((\lambda x. a + x) \text{ ' } S) \ (a + y) \in ((\lambda x. a + x) \text{ ' } S)$ 
      by auto
    then have h1:  $u *_R (a + x) + v *_R (a + y) \in (\lambda x. a + x) \text{ ' } S$ 
      using xy assms unfolding affine_def by auto
    have  $u *_R (a + x) + v *_R (a + y) = (u + v) *_R a + (u *_R x + v *_R y)$ 
      by (simp add: algebra_simps)
    also have  $\dots = a + (u *_R x + v *_R y)$ 
      using  $\langle u + v = 1 \rangle$  by auto
    ultimately have  $a + (u *_R x + v *_R y) \in (\lambda x. a + x) \text{ ' } S$ 
      using h1 by auto
    then have  $u *_R x + v *_R y \in S$  by auto
  }
  then show ?thesis unfolding affine_def by auto
qed

```

```

lemma affine_translation:
   $affine\ S \longleftrightarrow affine\ ((+) \ a \text{ ' } S)$  for a :: 'a::real_vector
proof
  show affine  $((+) \ a \text{ ' } S)$  if affine S
    using that translation_assoc [of  $- \ a \ a \ S$ ]
    by (auto intro: affine_translation_aux [of  $- \ a \ ((+) \ a \text{ ' } S)$ ])
  show affine S if affine  $((+) \ a \text{ ' } S)$ 
    using that by (rule affine_translation_aux)
qed

```

```

lemma parallel_is_affine:
  fixes S T :: 'a::real_vector set
  assumes affine S affine_parallel S T
  shows affine T
proof -
  from assms obtain a where  $T = (\lambda x. a + x) \text{ ' } S$ 
    unfolding affine_parallel_def by auto

```

```

then show ?thesis
  using affine_translation assms by auto
qed

```

```

lemma subspace_imp_affine: subspace s  $\implies$  affine s
  unfolding subspace_def affine_def by auto

```

```

lemma affine_hull_subset_span: (affine hull s)  $\subseteq$  (span s)
  by (metis hull_minimal span_superset subspace_imp_affine subspace_span)

```

Subspace parallel to an affine set

```

lemma subspace_affine: subspace S  $\longleftrightarrow$  affine S  $\wedge$  0  $\in$  S

```

```

proof -

```

```

  have h0: subspace S  $\implies$  affine S  $\wedge$  0  $\in$  S
    using subspace_imp_affine[of S] subspace_0 by auto
  {
    assume assm: affine S  $\wedge$  0  $\in$  S
    {
      fix c :: real
      fix x
      assume x: x  $\in$  S
      have c *R x = (1-c) *R 0 + c *R x by auto
      moreover
      have (1 - c) *R 0 + c *R x  $\in$  S
        using affine_alt[of S] assm x by auto
      ultimately have c *R x  $\in$  S by auto
    }
  }
  then have h1:  $\forall c. \forall x \in S. c *_{\mathbf{R}} x \in S$  by auto

```

```

  {
    fix x y
    assume xy: x  $\in$  S y  $\in$  S
    define u where u = (1 :: real)/2
    have (1/2) *R (x+y) = (1/2) *R (x+y)
      by auto
    moreover
    have (1/2) *R (x+y) = (1/2) *R x + (1-(1/2)) *R y
      by (simp add: algebra_simps)
    moreover
    have (1 - u) *R x + u *R y  $\in$  S
      using affine_alt[of S] assm xy by auto
    ultimately
    have (1/2) *R (x+y)  $\in$  S
      using u_def by auto
    moreover
    have x + y = 2 *R ((1/2) *R (x+y))
      by auto
    ultimately

```

```

      have  $x + y \in S$ 
      using  $h1[rule\_format, of (1/2) *_R (x+y) 2]$  by auto
    }
    then have  $\forall x \in S. \forall y \in S. x + y \in S$ 
    by auto
    then have subspace  $S$ 
    using  $h1$  assm unfolding subspace_def by auto
  }
  then show ?thesis using  $h0$  by metis
qed

```

```

lemma affine_diffs_subspace:
  assumes affine  $S$   $a \in S$ 
  shows subspace  $((\lambda x. (-a)+x) ' S)$ 
proof -
  have [simp]:  $(\lambda x. x - a) = plus (- a)$  by (simp add: fun_eq_iff)
  have affine  $((\lambda x. (-a)+x) ' S)$ 
    using affine_translation assms by blast
  moreover have  $0 \in ((\lambda x. (-a)+x) ' S)$ 
    using assms exI[of  $(\lambda x. x \in S \wedge -a+x = 0)$   $a$ ] by auto
  ultimately show ?thesis using subspace_affine by auto
qed

```

```

lemma affine_diffs_subspace_subtract:
  subspace  $((\lambda x. x - a) ' S)$  if affine  $S$   $a \in S$ 
  using that affine_diffs_subspace [of  $- a$ ] by simp

```

```

lemma parallel_subspace_explicit:
  assumes affine  $S$ 
  and  $a \in S$ 
  assumes  $L \equiv \{y. \exists x \in S. (-a) + x = y\}$ 
  shows subspace  $L \wedge$  affine_parallel  $S$   $L$ 
proof -
  from assms have  $L = plus (- a) ' S$  by auto
  then have par: affine_parallel  $S$   $L$ 
    unfolding affine_parallel_def ..
  then have affine  $L$  using assms parallel_is_affine by auto
  moreover have  $0 \in L$ 
    using assms by auto
  ultimately show ?thesis
    using subspace_affine par by auto
qed

```

```

lemma parallel_subspace_aux:
  assumes subspace  $A$ 
  and subspace  $B$ 
  and affine_parallel  $A$   $B$ 
  shows  $A \supseteq B$ 
proof -

```

```

from assms obtain a where a:  $\forall x. x \in A \longleftrightarrow a + x \in B$ 
using affine_parallel_expl[of A B] by auto
then have  $-a \in A$ 
using assms subspace_0[of B] by auto
then have  $a \in A$ 
using assms subspace_neg[of A -a] by auto
then show ?thesis
using assms a unfolding subspace_def by auto
qed

```

```

lemma parallel_subspace:
  assumes subspace A
  and subspace B
  and affine_parallel A B
  shows  $A = B$ 
proof
  show  $A \supseteq B$ 
  using assms parallel_subspace_aux by auto
  show  $A \subseteq B$ 
  using assms parallel_subspace_aux[of B A] affine_parallel_commut by auto
qed

```

```

lemma affine_parallel_subspace:
  assumes affine S S  $S \neq \{\}$ 
  shows  $\exists! L. \text{subspace } L \wedge \text{affine\_parallel } S L$ 
proof -
  have ex:  $\exists L. \text{subspace } L \wedge \text{affine\_parallel } S L$ 
  using assms parallel_subspace_explicit by auto
  {
    fix L1 L2
    assume ass:  $\text{subspace } L1 \wedge \text{affine\_parallel } S L1 \text{ subspace } L2 \wedge \text{affine\_parallel } S L2$ 
    then have affine_parallel L1 L2
    using affine_parallel_commut[of S L1] affine_parallel_assoc[of L1 S L2] by auto
    then have  $L1 = L2$ 
    using ass parallel_subspace by auto
  }
  then show ?thesis using ex by auto
qed

```

1.6.2 Affine Dependence

Formalized by Lars Schewe.

```

definition affine_dependent :: 'a::real_vector set  $\Rightarrow$  bool'
  where affine_dependent s  $\longleftrightarrow (\exists x \in s. x \in \text{affine hull } (s - \{x\}))$ 

```

```

lemma affine_dependent_imp_dependent: affine_dependent s  $\Longrightarrow$  dependent s
  unfolding affine_dependent_def dependent_def

```

using affine_hull_subset_span by auto

lemma affine_dependent_subset:

$\llbracket \text{affine\_dependent } s; s \subseteq t \rrbracket \implies \text{affine\_dependent } t$

apply (simp add: affine_dependent_def Bex_def)

apply (blast dest: hull_mono [OF Diff_mono [OF subset_refl]])

done

lemma affine_independent_subset:

shows $\llbracket \neg \text{affine\_dependent } t; s \subseteq t \rrbracket \implies \neg \text{affine\_dependent } s$

by (metis affine_dependent_subset)

lemma affine_independent_Diff:

$\neg \text{affine\_dependent } s \implies \neg \text{affine\_dependent}(s - t)$

by (meson Diff_subset affine_dependent_subset)

proposition affine_dependent_explicit:

$\text{affine\_dependent } p \longleftrightarrow$

$(\exists S \ u. \text{finite } S \wedge S \subseteq p \wedge \text{sum } u \ S = 0 \wedge (\exists v \in S. u \ v \neq 0) \wedge \text{sum } (\lambda v. u \ v *_{\mathbb{R}} v) \ S = 0)$

proof -

have $\exists S \ u. \text{finite } S \wedge S \subseteq p \wedge \text{sum } u \ S = 0 \wedge (\exists v \in S. u \ v \neq 0) \wedge (\sum w \in S. u \ w *_{\mathbb{R}} w) = 0$

if $(\sum w \in S. u \ w *_{\mathbb{R}} w) = x \ x \in p \ \text{finite } S \ S \neq \{\} \ S \subseteq p - \{x\} \ \text{sum } u \ S = 1$

for $x \ S \ u$

proof (intro exI conjI)

have $x \notin S$

using that **by** auto

then show $(\sum v \in \text{insert } x \ S. \text{if } v = x \text{ then } -1 \text{ else } u \ v) = 0$

using that **by** (simp add: sum_delta_notmem)

show $(\sum w \in \text{insert } x \ S. (\text{if } w = x \text{ then } -1 \text{ else } u \ w) *_{\mathbb{R}} w) = 0$

using that $\langle x \notin S \rangle$ **by** (simp add: if_smult sum_delta_notmem cong: if_cong)

qed (use that in auto)

moreover have $\exists x \in p. \exists S \ u. \text{finite } S \wedge S \neq \{\} \wedge S \subseteq p - \{x\} \wedge \text{sum } u \ S = 1$

$\wedge (\sum v \in S. u \ v *_{\mathbb{R}} v) = x$

if $(\sum v \in S. u \ v *_{\mathbb{R}} v) = 0 \ \text{finite } S \ S \subseteq p \ \text{sum } u \ S = 0 \ v \in S \ u \ v \neq 0$ **for** $S \ u \ v$

proof (intro bexI exI conjI)

have $S \neq \{v\}$

using that **by** auto

then show $S - \{v\} \neq \{\}$

using that **by** auto

show $(\sum x \in S - \{v\}. -(1 / u \ v) * u \ x) = 1$

unfolding sum_distrib_left[symmetric] sum_diff1[OF $\langle \text{finite } S \rangle$] **by** (simp add: that)

show $(\sum x \in S - \{v\}. -(1 / u \ v) * u \ x) *_{\mathbb{R}} x = v$

unfolding sum_distrib_left[symmetric] scaleR_scaleR[symmetric]

scaleR_right.sum[symmetric] sum_diff1[OF $\langle \text{finite } S \rangle$]

using that **by** auto

show $S - \{v\} \subseteq p - \{v\}$


```

    using that by auto
qed (use that in auto)
ultimately show ?thesis
  unfolding affine_dependent_def affine_hull_explicit by auto
qed

lemma affine_dependent_explicit_finite:
  fixes S :: 'a::real_vector set
  assumes finite S
  shows affine_dependent S  $\longleftrightarrow$ 
    ( $\exists u. \text{sum } u \text{ } S = 0 \wedge (\exists v \in S. u \text{ } v \neq 0) \wedge \text{sum } (\lambda v. u \text{ } v *_{\mathbb{R}} v) \text{ } S = 0$ )
    (is ?lhs = ?rhs)
proof
  have *:  $\bigwedge vt \ u \ v. (\text{if } vt \text{ then } u \text{ } v \text{ else } 0) *_{\mathbb{R}} v = (\text{if } vt \text{ then } (u \text{ } v) *_{\mathbb{R}} v \text{ else } 0::'a)$ 
    by auto
  assume ?lhs
  then obtain t u v where
    finite t  $t \subseteq S$   $\text{sum } u \text{ } t = 0$   $v \in t$   $u \text{ } v \neq 0$   $(\sum v \in t. u \text{ } v *_{\mathbb{R}} v) = 0$ 
    unfolding affine_dependent_explicit by auto
  then show ?rhs
    apply (rule_tac x= $\lambda x. \text{if } x \in t \text{ then } u \text{ } x \text{ else } 0$  in exI)
    apply (auto simp: * sum.inter_restrict[OF assms, symmetric] Int_absorb1[OF
       $t \subseteq S$ ])
    done
next
  assume ?rhs
  then obtain u v where  $\text{sum } u \text{ } S = 0$   $v \in S$   $u \text{ } v \neq 0$   $(\sum v \in S. u \text{ } v *_{\mathbb{R}} v) = 0$ 
    by auto
  then show ?lhs unfolding affine_dependent_explicit
    using assms by auto
qed

lemma dependent_imp_affine_dependent:
  assumes dependent {x - a | x . x  $\in$  s}
  and a  $\notin$  s
  shows affine_dependent (insert a s)
proof -
  from assms(1)[unfolded dependent_explicit] obtain S u v
    where obt: finite S  $S \subseteq \{x - a \mid x. x \in s\}$   $v \in S$   $u \text{ } v \neq 0$   $(\sum v \in S. u \text{ } v *_{\mathbb{R}} v) = 0$ 
  by auto
  define t where t =  $(\lambda x. x + a) ` S$ 

  have inj: inj_on  $(\lambda x. x + a)$  S
    unfolding inj_on_def by auto
  have 0  $\notin$  S
    using obt(2) assms(2) unfolding subset_eq by auto
  have fin: finite t and t  $\subseteq$  s
    unfolding t_def using obt(1,2) by auto

```

```

then have finite (insert a t) and insert a t  $\subseteq$  insert a s
by auto
moreover have *:  $\bigwedge P Q. (\sum x \in t. (\text{if } x = a \text{ then } P x \text{ else } Q x)) = (\sum x \in t. Q$ 
x)
  apply (rule sum.cong)
  using  $\langle a \notin s \rangle \langle t \subseteq s \rangle$ 
  apply auto
  done
have  $(\sum x \in \text{insert } a t. \text{if } x = a \text{ then } - (\sum x \in t. u (x - a)) \text{ else } u (x - a)) = 0$ 
  unfolding sum_clauses(2) [OF fin] * using  $\langle a \notin s \rangle \langle t \subseteq s \rangle$  by auto
moreover have  $\exists v \in \text{insert } a t. (\text{if } v = a \text{ then } - (\sum x \in t. u (x - a)) \text{ else } u (v -$ 
a))  $\neq 0$ 
  using obt(3,4)  $\langle 0 \notin S \rangle$ 
  by (rule_tac x=v + a in bexI) (auto simp: t_def)
moreover have *:  $\bigwedge P Q. (\sum x \in t. (\text{if } x = a \text{ then } P x \text{ else } Q x) *_R x) = (\sum x \in t.$ 
Q x *_R x)
  using  $\langle a \notin s \rangle \langle t \subseteq s \rangle$  by (auto intro!: sum.cong)
have  $(\sum x \in t. u (x - a)) *_R a = (\sum v \in t. u (v - a) *_R v)$ 
  unfolding scaleR_left.sum
  unfolding t_def and sum.reindex [OF inj] and o_def
  using obt(5)
  by (auto simp: sum.distrib scaleR_right_distrib)
then have  $(\sum v \in \text{insert } a t. (\text{if } v = a \text{ then } - (\sum x \in t. u (x - a)) \text{ else } u (v - a))$ 
*_R v)  $= 0$ 
  unfolding sum_clauses(2) [OF fin]
  using  $\langle a \notin s \rangle \langle t \subseteq s \rangle$ 
  by (auto simp: *)
ultimately show ?thesis
  unfolding affine_dependent_explicit
  apply (rule_tac x=insert a t in exI, auto)
  done
qed

```

```

lemma affine_dependent_biggerset:
  fixes s :: 'a::euclidean_space set'
  assumes finite s card s  $\geq \text{DIM}('a) + 2$ 
  shows affine_dependent s
proof -
  have s  $\neq \{\}$  using assms by auto
  then obtain a where a  $\in s$  by auto
  have *:  $\{x - a \mid x. x \in s - \{a\}\} = (\lambda x. x - a) ` (s - \{a\})$ 
  by auto
  have card  $\{x - a \mid x. x \in s - \{a\}\} = \text{card } (s - \{a\})$ 
  unfolding * by (simp add: card_image inj_on_def)
  also have ...  $> \text{DIM}('a)$  using assms(2)
  unfolding card_Diff_singleton [OF  $\langle a \in s \rangle$ ] by auto
  finally show ?thesis
  apply (subst insert_Diff [OF  $\langle a \in s \rangle$ , symmetric])
  apply (rule dependent_imp_affine_dependent)

```

```

    apply (rule dependent_biggerset, auto)
  done
qed

lemma affine_dependent_biggerset_general:
  assumes finite (S :: 'a::euclidean_space set)
    and card S  $\geq$  dim S + 2
  shows affine_dependent S
proof -
  from assms(2) have S  $\neq$  {} by auto
  then obtain a where a  $\in$  S by auto
  have *: {x - a | x. x  $\in$  S - {a}} = ( $\lambda$ x. x - a) ` (S - {a})
    by auto
  have **: card {x - a | x. x  $\in$  S - {a}} = card (S - {a})
    by (metis (no_types, lifting) * card_image diff_add_cancel inj_on_def)
  have dim {x - a | x. x  $\in$  S - {a}}  $\leq$  dim S
    using <a $\in$ S> by (auto simp: span_base span_diff intro: subset_le_dim)
  also have ... < dim S + 1 by auto
  also have ...  $\leq$  card (S - {a})
    using assms card_Diff_singleton[OF <a $\in$ S>] by auto
  finally show ?thesis
    apply (subst insert_Diff[OF <a $\in$ S>, symmetric])
    apply (rule dependent_imp_affine_dependent)
    apply (rule dependent_biggerset_general)
    unfolding **
    apply auto
  done
qed

```

1.6.3 Some Properties of Affine Dependent Sets

```

lemma affine_independent_0 [simp]:  $\neg$  affine_dependent {}
  by (simp add: affine_dependent_def)

lemma affine_independent_1 [simp]:  $\neg$  affine_dependent {a}
  by (simp add: affine_dependent_def)

lemma affine_independent_2 [simp]:  $\neg$  affine_dependent {a,b}
  by (simp add: affine_dependent_def insert_Diff_if hull_same)

lemma affine_hull_translation: affine_hull (( $\lambda$ x. a + x) ` S) = ( $\lambda$ x. a + x) `
  (affine_hull S)
proof -
  have affine (( $\lambda$ x. a + x) ` (affine_hull S))
    using affine_translation affine_affine_hull by blast
  moreover have ( $\lambda$ x. a + x) ` S  $\subseteq$  ( $\lambda$ x. a + x) ` (affine_hull S)
    using hull_subset[of S] by auto
  ultimately have h1: affine_hull (( $\lambda$ x. a + x) ` S)  $\subseteq$  ( $\lambda$ x. a + x) ` (affine_hull
  S)

```

```

    by (metis hull_minimal)
  have affine(( $\lambda x. -a + x$ ) ' ( $\text{affine hull } ((\lambda x. a + x) ' S)$ ))
    using affine_translation affine_affine_hull by blast
  moreover have ( $\lambda x. -a + x$ ) ' ( $\lambda x. a + x$ ) '  $S \subseteq (\lambda x. -a + x$ ) ' ( $\text{affine hull } ((\lambda x. a + x) ' S)$ )
    using hull_subset[of ( $\lambda x. a + x$ ) '  $S$ ] by auto
  moreover have  $S = (\lambda x. -a + x$ ) ' ( $\lambda x. a + x$ ) '  $S$ 
    using translation_assoc[of  $-a$   $a$ ] by auto
  ultimately have ( $\lambda x. -a + x$ ) ' ( $\text{affine hull } ((\lambda x. a + x) ' S)$ )  $\supseteq$  ( $\text{affine hull } S$ )
    by (metis hull_minimal)
  then have  $\text{affine hull } ((\lambda x. a + x) ' S) \supseteq (\lambda x. a + x) ' (\text{affine hull } S)$ 
    by auto
  then show ?thesis using h1 by auto
qed

```

```

lemma affine_dependent_translation:
  assumes affine_dependent  $S$ 
  shows affine_dependent (( $\lambda x. a + x$ ) '  $S$ )
proof -
  obtain  $x$  where  $x: x \in S \wedge x \in \text{affine hull } (S - \{x\})$ 
    using assms affine_dependent_def by auto
  have (+)  $a$  ' ( $S - \{x\}$ ) = (+)  $a$  '  $S - \{a + x\}$ 
    by auto
  then have  $a + x \in \text{affine hull } ((\lambda x. a + x) ' S - \{a + x\})$ 
    using affine_hull_translation[of  $a$   $S - \{x\}$ ]  $x$  by auto
  moreover have  $a + x \in (\lambda x. a + x) ' S$ 
    using  $x$  by auto
  ultimately show ?thesis
    unfolding affine_dependent_def by auto
qed

```

```

lemma affine_dependent_translation_eq:
  affine_dependent  $S \longleftrightarrow$  affine_dependent (( $\lambda x. a + x$ ) '  $S$ )
proof -
  {
    assume affine_dependent (( $\lambda x. a + x$ ) '  $S$ )
    then have affine_dependent  $S$ 
      using affine_dependent_translation[of (( $\lambda x. a + x$ ) '  $S$ )  $-a$   $a$ ]
        by auto
  }
  then show ?thesis
    using affine_dependent_translation by auto
qed

```

```

lemma affine_hull_0_dependent:
  assumes  $0 \in \text{affine hull } S$ 
  shows dependent  $S$ 

```

```

proof -
  obtain s u where s_u: finite s  $\wedge$  s  $\neq$  {}  $\wedge$  s  $\subseteq$  S  $\wedge$  sum u s = 1  $\wedge$  ( $\sum_{v \in s. u$ 
    v *R v) = 0
    using assms affine_hull_explicit[of S] by auto
  then have  $\exists v \in s. u \ v \neq 0$  by auto
  then have finite s  $\wedge$  s  $\subseteq$  S  $\wedge$  ( $\exists v \in s. u \ v \neq 0 \wedge (\sum_{v \in s. u \ v *_{\mathbf{R}} v) = 0$ )
    using s_u by auto
  then show ?thesis
    unfolding dependent_explicit[of S] by auto
qed

```

lemma affine_dependent_imp_dependent2:

assumes affine_dependent (insert 0 S)

shows dependent S

proof -

obtain x where x: x $\in$ insert 0 S $\wedge$ x $\in$ affine hull (insert 0 S - {x})

using affine_dependent_def[of (insert 0 S)] assms by blast

then have x $\in$ span (insert 0 S - {x})

using affine_hull_subset_span by auto

moreover have span (insert 0 S - {x}) = span (S - {x})

using insert_Diff_if[of 0 S {x}] span_insert_0[of S-{x}] by auto

ultimately have x $\in$ span (S - {x}) by auto

then have x $\neq 0 \implies$ dependent S

using x dependent_def by auto

moreover

{

assume x = 0

then have 0 $\in$ affine hull S

using x hull_mono[of S - {0} S] by auto

then have dependent S

using affine_hull_0_dependent by auto

}

ultimately show ?thesis by auto

qed

lemma affine_dependent_iff_dependent:

assumes a $\notin$ S

shows affine_dependent (insert a S) $\longleftrightarrow$ dependent (($\lambda x. -a + x$) ' S)

proof -

have ((+) (- a) ' S) = {x - a | x . x $\in$ S} by auto

then show ?thesis

using affine_dependent_translation_eq[of (insert a S) -a]

affine_dependent_imp_dependent2 assms

dependent_imp_affine_dependent[of a S]

by (auto simp del: uminus_add_conv_diff)

qed

lemma affine_dependent_iff_dependent2:

assumes a $\in$ S

shows $\text{affine\_dependent } S \longleftrightarrow \text{dependent } ((\lambda x. -a + x) \text{ ` } (S - \{a\}))$
proof –
 have $\text{insert } a \ (S - \{a\}) = S$
 using *assms* **by** *auto*
 then show *?thesis*
 using *assms* $\text{affine\_dependent\_iff\_dependent}[of \ a \ S - \{a\}]$ **by** *auto*
qed

lemma $\text{affine\_hull\_insert\_span\_gen}$:
 $\text{affine\_hull } (\text{insert } a \ s) = (\lambda x. a + x) \text{ ` } \text{span } ((\lambda x. -a + x) \text{ ` } s)$
proof –
 have $h1: \{x - a \mid x. x \in s\} = ((\lambda x. -a + x) \text{ ` } s)$
 by *auto*
 {
 assume $a \notin s$
 then have *?thesis*
 using $\text{affine\_hull\_insert\_span}[of \ a \ s] \ h1$ **by** *auto*
 }
 moreover
 {
 assume $a1: a \in s$
 have $\exists x. x \in s \wedge -a + x = 0$
 apply (rule $\text{exI}[of \ _ \ a]$)
 using $a1$
 apply *auto*
 done
 then have $\text{insert } 0 \ ((\lambda x. -a + x) \text{ ` } (s - \{a\})) = (\lambda x. -a + x) \text{ ` } s$
 by *auto*
 then have $\text{span } ((\lambda x. -a + x) \text{ ` } (s - \{a\})) = \text{span } ((\lambda x. -a + x) \text{ ` } s)$
 using $\text{span\_insert\_0}[of \ (+) \ (- \ a) \text{ ` } (s - \{a\})]$ **by** (*auto simp del: uminus_add_conv_diff*)
 moreover have $\{x - a \mid x. x \in (s - \{a\})\} = ((\lambda x. -a + x) \text{ ` } (s - \{a\}))$
 by *auto*
 moreover have $\text{insert } a \ (s - \{a\}) = \text{insert } a \ s$
 by *auto*
 ultimately have *?thesis*
 using $\text{affine\_hull\_insert\_span}[of \ a \ s - \{a\}]$ **by** *auto*
 }
 ultimately show *?thesis* **by** *auto*
qed

lemma affine_hull_span2 :
 assumes $a \in s$
 shows $\text{affine\_hull } s = (\lambda x. a + x) \text{ ` } \text{span } ((\lambda x. -a + x) \text{ ` } (s - \{a\}))$
 using $\text{affine\_hull\_insert\_span\_gen}[of \ a \ s - \{a\}]$, *unfolded insert_Diff[OF assms]*
 by *auto*

lemma $\text{affine\_hull\_span\_gen}$:
 assumes $a \in \text{affine\_hull } s$

```

  shows affine hull s = ( $\lambda x. a+x$ ) ‘ span (( $\lambda x. -a+x$ ) ‘ s)
proof –
  have affine hull (insert a s) = affine hull s
    using hull_redundant[of a affine s] assms by auto
  then show ?thesis
    using affine_hull_insert_span_gen[of a s] by auto
qed

lemma affine_hull_span_0:
  assumes 0  $\in$  affine hull S
  shows affine hull S = span S
  using affine_hull_span_gen[of 0 S] assms by auto

lemma extend_to_affine_basis_nonempty:
  fixes S V :: 'n::real_vector set
  assumes  $\neg$  affine_dependent S  $S \subseteq V$   $S \neq \{\}$ 
  shows  $\exists T. \neg$  affine_dependent T  $\wedge S \subseteq T \wedge T \subseteq V \wedge$  affine hull T = affine
  hull V
proof –
  obtain a where a:  $a \in S$ 
    using assms by auto
  then have h0: independent (( $\lambda x. -a+x$ ) ‘ (S-{a}))
    using affine_dependent_iff_dependent2 assms by auto
  obtain B where B:
    ( $\lambda x. -a+x$ ) ‘ (S - {a})  $\subseteq B \wedge B \subseteq (\lambda x. -a+x)$  ‘ V  $\wedge$  independent B  $\wedge (\lambda x. -a+x)$  ‘ V  $\subseteq$  span B
    using assms
    by (blast intro: maximal_independent_subset_extend[OF h0, of ( $\lambda x. -a+x$ ) ‘ V])
  define T where T = ( $\lambda x. a+x$ ) ‘ insert 0 B
  then have T = insert a (( $\lambda x. a+x$ ) ‘ B)
    by auto
  then have affine hull T = ( $\lambda x. a+x$ ) ‘ span B
    using affine_hull_insert_span_gen[of a (( $\lambda x. a+x$ ) ‘ B)] translation_assoc[of -a a B]
    by auto
  then have V  $\subseteq$  affine hull T
    using B assms translation_inverse_subset[of a V span B]
    by auto
  moreover have T  $\subseteq V$ 
    using T_def B a assms by auto
  ultimately have affine hull T = affine hull V
    by (metis Int_absorb1 Int_absorb2 hull_hull hull_mono)
  moreover have S  $\subseteq T$ 
    using T_def B translation_inverse_subset[of a S-{a} B]
    by auto
  moreover have  $\neg$  affine_dependent T
    using T_def affine_dependent_translation_eq[of insert 0 B]
    affine_dependent_imp_dependent2 B

```

```

    by auto
    ultimately show ?thesis using ⟨ $T \subseteq V$ ⟩ by auto
qed

```

```

lemma affine_basis_exists:
  fixes  $V :: 'n::real\_vector\ set$ 
  shows  $\exists B. B \subseteq V \wedge \neg \text{affine\_dependent } B \wedge \text{affine hull } V = \text{affine hull } B$ 
proof (cases  $V = \{\}$ )
  case True
  then show ?thesis
    using affine_independent_0 by auto
next
  case False
  then obtain  $x$  where  $x \in V$  by auto
  then show ?thesis
    using affine_dependent_def[of  $\{x\}$ ] extend_to_affine_basis_nonempty[of  $\{x\}$ 
 $V$ ]
    by auto
qed

```

```

proposition extend_to_affine_basis:
  fixes  $S V :: 'n::real\_vector\ set$ 
  assumes  $\neg \text{affine\_dependent } S \ S \subseteq V$ 
  obtains  $T$  where  $\neg \text{affine\_dependent } T \ S \subseteq T \ T \subseteq V \ \text{affine hull } T = \text{affine hull } V$ 
proof (cases  $S = \{\}$ )
  case True then show ?thesis
    using affine_basis_exists by (metis empty_subsetI that)
next
  case False
  then show ?thesis by (metis assms extend_to_affine_basis_nonempty that)
qed

```

1.6.4 Affine Dimension of a Set

```

definition aff_dim :: ('a::euclidean_space) set  $\Rightarrow$  int
  where aff_dim  $V =$ 
    (SOME  $d :: int.$ 
       $\exists B. \text{affine hull } B = \text{affine hull } V \wedge \neg \text{affine\_dependent } B \wedge \text{of\_nat } (\text{card } B) = d + 1$ )

```

```

lemma aff_dim_basis_exists:
  fixes  $V :: ('n::euclidean\_space) \ set$ 
  shows  $\exists B. \text{affine hull } B = \text{affine hull } V \wedge \neg \text{affine\_dependent } B \wedge \text{of\_nat } (\text{card } B) = \text{aff\_dim } V + 1$ 
proof -
  obtain  $B$  where  $\neg \text{affine\_dependent } B \wedge \text{affine hull } B = \text{affine hull } V$ 
    using affine_basis_exists[of  $V$ ] by auto
  then show ?thesis

```



```

unfolding aff_dim_def
  some_eq_ex[of  $\lambda d. \exists B. \text{affine hull } B = \text{affine hull } V \wedge \neg \text{affine\_dependent } B \wedge \text{of\_nat } (\text{card } B) = d + 1]$ ]
apply auto
apply (rule exI[of  $\_ \text{int } (\text{card } B) - (1 :: \text{int})$ ])
apply (rule exI[of  $\_ B]$ , auto)
done
qed

```

```

lemma affine_hull_eq_empty [simp]:  $\text{affine hull } S = \{\} \longleftrightarrow S = \{\}$ 
by (metis affine_empty subset_empty subset_hull)

```

```

lemma empty_eq_affine_hull [simp]:  $\{\} = \text{affine hull } S \longleftrightarrow S = \{\}$ 
by (metis affine_hull_eq_empty)

```

```

lemma aff_dim_parallel_subspace_aux:
  fixes  $B :: 'n::\text{euclidean\_space set}$ 
  assumes  $\neg \text{affine\_dependent } B \ a \in B$ 
  shows  $\text{finite } B \wedge ((\text{card } B) - 1 = \dim (\text{span } ((\lambda x. -a+x) \text{ ` } (B-\{a\}))))$ 
proof -
  have  $\text{independent } ((\lambda x. -a+x) \text{ ` } (B-\{a\}))$ 
    using affine_dependent_iff_dependent2 assms by auto
  then have fin:  $\dim (\text{span } ((\lambda x. -a+x) \text{ ` } (B-\{a\}))) = \text{card } ((\lambda x. -a+x) \text{ ` } (B-\{a\}))$ 
    finite  $((\lambda x. -a+x) \text{ ` } (B-\{a\}))$ 
    using indep_card_eq_dim_span[of  $(\lambda x. -a+x) \text{ ` } (B-\{a\})$ ] by auto
  show ?thesis
proof (cases  $(\lambda x. -a+x) \text{ ` } (B-\{a\}) = \{\}$ )
  case True
  have  $B = \text{insert } a ((\lambda x. a+x) \text{ ` } (\lambda x. -a+x) \text{ ` } (B-\{a\}))$ 
    using translation_assoc[of  $a -a (B-\{a\})$ ] assms by auto
  then have  $B = \{a\}$  using True by auto
  then show ?thesis using assms fin by auto
next
  case False
  then have  $\text{card } ((\lambda x. -a+x) \text{ ` } (B-\{a\})) > 0$ 
    using fin by auto
  moreover have h1:  $\text{card } ((\lambda x. -a+x) \text{ ` } (B-\{a\})) = \text{card } (B-\{a\})$ 
    by (rule card_image) (use translate_inj_on in blast)
  ultimately have  $\text{card } (B-\{a\}) > 0$  by auto
  then have  $*$ :  $\text{finite } (B-\{a\})$ 
    using card_gt_0_iff[of  $(B-\{a\})$ ] by auto
  then have  $\text{card } (B-\{a\}) = \text{card } B - 1$ 
    using card_Diff_singleton assms by auto
  with  $*$  show ?thesis using fin h1 by auto
qed
qed

```

```

lemma aff_dim_parallel_subspace:

```

```

fixes  $V\ L :: 'n::euclidean\_space\ set$ 
assumes  $V \neq \{\}$ 
and  $subspace\ L$ 
and  $affine\_parallel\ (affine\ hull\ V)\ L$ 
shows  $aff\_dim\ V = int\ (dim\ L)$ 
proof –
  obtain  $B$  where
     $B: affine\ hull\ B = affine\ hull\ V \wedge \neg\ affine\_dependent\ B \wedge int\ (card\ B) =$ 
 $aff\_dim\ V + 1$ 
    using  $aff\_dim\_basis\_exists$  by  $auto$ 
    then have  $B \neq \{\}$ 
    using  $assms\ B$ 
    by  $auto$ 
    then obtain  $a$  where  $a: a \in B$  by  $auto$ 
    define  $Lb$  where  $Lb = span\ ((\lambda x. -a+x)\ ` (B-\{a\}))$ 
    moreover have  $affine\_parallel\ (affine\ hull\ B)\ Lb$ 
    using  $Lb\_def\ B\ assms\ affine\_hull\_span2[of\ a\ B]\ a$ 
     $affine\_parallel\_commut[of\ Lb\ (affine\ hull\ B)]$ 
    unfolding  $affine\_parallel\_def$ 
    by  $auto$ 
    moreover have  $subspace\ Lb$ 
    using  $Lb\_def\ subspace\_span$  by  $auto$ 
    moreover have  $affine\ hull\ B \neq \{\}$ 
    using  $assms\ B$  by  $auto$ 
    ultimately have  $L = Lb$ 
    using  $assms\ affine\_parallel\_subspace[of\ affine\ hull\ B]\ affine\_affine\_hull[of\ B]$ 
 $B$ 
    by  $auto$ 
    then have  $dim\ L = dim\ Lb$ 
    by  $auto$ 
    moreover have  $card\ B - 1 = dim\ Lb$  and  $finite\ B$ 
    using  $Lb\_def\ aff\_dim\_parallel\_subspace\_aux\ a\ B$  by  $auto$ 
    ultimately show  $?thesis$ 
    using  $B\ \langle B \neq \{\} \rangle\ card\_gt\_0\_iff[of\ B]$  by  $auto$ 
qed

lemma  $aff\_independent\_finite$ :
  fixes  $B :: 'n::euclidean\_space\ set$ 
  assumes  $\neg\ affine\_dependent\ B$ 
  shows  $finite\ B$ 
proof –
  {
    assume  $B \neq \{\}$ 
    then obtain  $a$  where  $a \in B$  by  $auto$ 
    then have  $?thesis$ 
    using  $aff\_dim\_parallel\_subspace\_aux\ assms$  by  $auto$ 
  }
  then show  $?thesis$  by  $auto$ 
qed

```

```

lemma aff_dim_empty:
  fixes S :: 'n::euclidean_space set
  shows S = {}  $\longleftrightarrow$  aff_dim S = -1
proof -
  obtain B where *: affine hull B = affine hull S
  and  $\neg$  affine_dependent B
  and int (card B) = aff_dim S + 1
  using aff_dim_basis_exists by auto
  moreover
  from * have S = {}  $\longleftrightarrow$  B = {}
  by auto
  ultimately show ?thesis
  using aff_independent_finite[of B] card_gt_0_iff[of B] by auto
qed

lemma aff_dim_empty_eq [simp]: aff_dim ({}::'a::euclidean_space set) = -1
  by (simp add: aff_dim_empty [symmetric])

lemma aff_dim_affine_hull [simp]: aff_dim (affine hull S) = aff_dim S
  unfolding aff_dim_def using hull_hull[of _ S] by auto

lemma aff_dim_affine_hull2:
  assumes affine hull S = affine hull T
  shows aff_dim S = aff_dim T
  unfolding aff_dim_def using assms by auto

lemma aff_dim_unique:
  fixes B V :: 'n::euclidean_space set
  assumes affine hull B = affine hull V  $\wedge$   $\neg$  affine_dependent B
  shows of_nat (card B) = aff_dim V + 1
proof (cases B = {})
  case True
  then have V = {}
  using assms
  by auto
  then have aff_dim V = (-1::int)
  using aff_dim_empty by auto
  then show ?thesis
  using  $\langle B = \{\} \rangle$  by auto
next
  case False
  then obtain a where a: a  $\in$  B by auto
  define Lb where Lb = span (( $\lambda x. -a+x$ ) ` (B-{a}))
  have affine_parallel (affine hull B) Lb
  using Lb_def affine_hull_span2[of a B] a
  affine_parallel_commut[of Lb (affine hull B)]
  unfolding affine_parallel_def by auto

```

```

moreover have subspace Lb
  using Lb_def subspace_span by auto
ultimately have aff_dim B = int(dim Lb)
  using aff_dim_parallel_subspace[of B Lb] ‹B ≠ {}› by auto
moreover have (card B) - 1 = dim Lb finite B
  using Lb_def aff_dim_parallel_subspace_aux a assms by auto
ultimately have of_nat (card B) = aff_dim B + 1
  using ‹B ≠ {}› card_gt_0_iff[of B] by auto
then show ?thesis
  using aff_dim_affine_hull2 assms by auto
qed

```

```

lemma aff_dim_affine_independent:
  fixes B :: 'n::euclidean_space set
  assumes ¬ affine_dependent B
  shows of_nat (card B) = aff_dim B + 1
  using aff_dim_unique[of B B] assms by auto

```

```

lemma affine_independent_iff_card:
  fixes s :: 'a::euclidean_space set
  shows ¬ affine_dependent s ⟷ finite s ∧ aff_dim s = int(card s) - 1
  apply (rule iffI)
  apply (simp add: aff_dim_affine_independent affine_independent_finite)
  by (metis affine_basis_exists [of s] aff_dim_unique card_subset_eq diff_add_cancel
of_nat_eq_iff)

```

```

lemma aff_dim_sing [simp]:
  fixes a :: 'n::euclidean_space
  shows aff_dim {a} = 0
  using aff_dim_affine_independent[of {a}] affine_independent_1 by auto

```

```

lemma aff_dim_2 [simp]:
  fixes a :: 'n::euclidean_space
  shows aff_dim {a,b} = (if a = b then 0 else 1)
proof (clarsimp)
  assume a ≠ b
  then have aff_dim {a,b} = card {a,b} - 1
    using affine_independent_2 [of a b] aff_dim_affine_independent by fastforce
  also have ... = 1
    using ‹a ≠ b› by simp
  finally show aff_dim {a, b} = 1 .
qed

```

```

lemma aff_dim_inner_basis_exists:
  fixes V :: ('n::euclidean_space) set
  shows ∃ B. B ⊆ V ∧ affine hull B = affine hull V ∧
    ¬ affine_dependent B ∧ of_nat (card B) = aff_dim V + 1
proof -
  obtain B where B: ¬ affine_dependent B B ⊆ V affine hull B = affine hull V

```

```

    using affine_basis_exists[of V] by auto
    then have of_nat(card B) = aff_dim V + 1 using aff_dim_unique by auto
    with B show ?thesis by auto
qed

```

```

lemma aff_dim_le_card:
  fixes V :: 'n::euclidean_space set
  assumes finite V
  shows aff_dim V ≤ of_nat (card V) - 1
proof -
  obtain B where B: B ⊆ V of_nat (card B) = aff_dim V + 1
    using aff_dim_inner_basis_exists[of V] by auto
  then have card B ≤ card V
    using assms card_mono by auto
  with B show ?thesis by auto
qed

```

```

lemma aff_dim_parallel_eq:
  fixes S T :: 'n::euclidean_space set
  assumes affine_parallel (affine hull S) (affine hull T)
  shows aff_dim S = aff_dim T
proof -
  {
    assume T ≠ {} S ≠ {}
    then obtain L where L: subspace L ∧ affine_parallel (affine hull T) L
      using affine_parallel_subspace[of affine hull T]
        affine_affine_hull[of T]
      by auto
    then have aff_dim T = int (dim L)
      using aff_dim_parallel_subspace ⟨T ≠ {}⟩ by auto
    moreover have *: subspace L ∧ affine_parallel (affine hull S) L
      using L affine_parallel_assoc[of affine hull S affine hull T L] assms by auto
    moreover from * have aff_dim S = int (dim L)
      using aff_dim_parallel_subspace ⟨S ≠ {}⟩ by auto
    ultimately have ?thesis by auto
  }
  moreover
  {
    assume S = {}
    then have S = {} and T = {}
      using assms
      unfolding affine_parallel_def
      by auto
    then have ?thesis using aff_dim_empty by auto
  }
  moreover
  {
    assume T = {}
    then have S = {} and T = {}

```

```

    using assms
    unfolding affine_parallel_def
    by auto
  then have ?thesis
    using aff_dim_empty by auto
}
ultimately show ?thesis by blast
qed

```

```

lemma aff_dim_translation_eq:
  aff_dim ((+) a ' S) = aff_dim S for a :: 'n::euclidean_space
proof -
  have affine_parallel (affine hull S) (affine hull (( $\lambda x. a + x$ ) ' S))
    unfolding affine_parallel_def
    apply (rule exI[of _ a])
    using affine_hull_translation[of a S]
    apply auto
  done
  then show ?thesis
    using aff_dim_parallel_eq[of S ( $\lambda x. a + x$ ) ' S] by auto
qed

```

```

lemma aff_dim_translation_eq_subtract:
  aff_dim (( $\lambda x. x - a$ ) ' S) = aff_dim S for a :: 'n::euclidean_space
  using aff_dim_translation_eq [of - a] by (simp cong: image_cong_simp)

```

```

lemma aff_dim_affine:
  fixes S L :: 'n::euclidean_space set
  assumes S ≠ {}
    and affine S
    and subspace L
    and affine_parallel S L
  shows aff_dim S = int (dim L)
proof -
  have *: affine hull S = S
    using assms affine_hull_eq[of S] by auto
  then have affine_parallel (affine hull S) L
    using assms by (simp add: *)
  then show ?thesis
    using assms aff_dim_parallel_subspace[of S L] by blast
qed

```

```

lemma dim_affine_hull:
  fixes S :: 'n::euclidean_space set
  shows dim (affine hull S) = dim S
proof -
  have dim (affine hull S) ≥ dim S
    using dim_subset by auto
  moreover have dim (span S) ≥ dim (affine hull S)

```

```

    using dim_subset affine_hull_subset_span by blast
  moreover have  $\dim (\text{span } S) = \dim S$ 
    using dim_span by auto
  ultimately show ?thesis by auto
qed

```

```

lemma aff_dim_subspace:
  fixes  $S :: 'n::\text{euclidean\_space}$  set
  assumes subspace  $S$ 
  shows  $\text{aff\_dim } S = \text{int } (\dim S)$ 
proof (cases  $S = \{\}$ )
  case True with assms show ?thesis
    by (simp add: subspace_affine)
next
  case False
  with  $\text{aff\_dim\_affine}[of\ S\ S]$  assms subspace_imp_affine[ $of\ S$ ] affine_parallel_reflex[ $of\ S$ ]
    subspace_affine
  show ?thesis by auto
qed

```

```

lemma aff_dim_zero:
  fixes  $S :: 'n::\text{euclidean\_space}$  set
  assumes  $0 \in \text{affine hull } S$ 
  shows  $\text{aff\_dim } S = \text{int } (\dim S)$ 
proof -
  have subspace (affine hull  $S$ )
    using subspace_affine[ $of\ \text{affine hull } S$ ] affine_affine_hull assms
    by auto
  then have  $\text{aff\_dim } (\text{affine hull } S) = \text{int } (\dim (\text{affine hull } S))$ 
    using assms aff_dim_subspace[ $of\ \text{affine hull } S$ ] by auto
  then show ?thesis
    using  $\text{aff\_dim\_affine\_hull}[of\ S]$   $\dim\_affine\_hull[of\ S]$ 
    by auto
qed

```

```

lemma aff_dim_eq_dim:
   $\text{aff\_dim } S = \text{int } (\dim ((+) (- a) ' S))$  if  $a \in \text{affine hull } S$ 
  for  $S :: 'n::\text{euclidean\_space}$  set
proof -
  have  $0 \in \text{affine hull } (+) (- a) ' S$ 
    unfolding affine_hull_translation
    using that by (simp add: ac_simps)
  with aff_dim_zero show ?thesis
    by (metis aff_dim_translation_eq)
qed

```

```

lemma aff_dim_eq_dim_subtract:
   $\text{aff\_dim } S = \text{int } (\dim ((\lambda x. x - a) ' S))$  if  $a \in \text{affine hull } S$ 
  for  $S :: 'n::\text{euclidean\_space}$  set

```

using *aff_dim_eq_dim* [of *a*] that by (*simp cong: image_cong_simp*)

lemma *aff_dim_UNIV* [*simp*]: *aff_dim* (*UNIV* :: '*n*::*euclidean_space* set) = *int*(*DIM*('n'))
 using *aff_dim_subspace*[of (*UNIV* :: '*n*::*euclidean_space* set)]
dim_UNIV[where '*a*'=*n*::*euclidean_space*]
 by *auto*

lemma *aff_dim_geq*:
 fixes *V* :: '*n*::*euclidean_space* set
 shows *aff_dim V* ≥ -1
proof -
 obtain *B* where *affine hull B* = *affine hull V*
 and ¬ *affine_dependent B*
 and *int (card B)* = *aff_dim V* + 1
 using *aff_dim_basis_exists* by *auto*
 then show ?thesis by *auto*
qed

lemma *aff_dim_negative_iff* [*simp*]:
 fixes *S* :: '*n*::*euclidean_space* set
 shows *aff_dim S* < 0 $\longleftrightarrow$ *S* = {}
 by (*metis aff_dim_empty aff_dim_geq diff_0 eq_iff zle_diff1_eq*)

lemma *aff_lowdim_subset_hyperplane*:
 fixes *S* :: '*a*::*euclidean_space* set
 assumes *aff_dim S* < *DIM*('a')
 obtains *a b* where *a* ≠ 0 *S* ⊆ {*x*. *a* · *x* = *b*}
proof (*cases S*={})
 case *True*
 moreover
 have (*SOME b*. *b* ∈ *Basis*) ≠ 0
 by (*metis norm_some_Basis norm_zero zero_neq_one*)
 ultimately show ?thesis
 using *that* by *blast*
 next
 case *False*
 then obtain *c S'* where *c* ∉ *S'* *S* = *insert c S'*
 by (*meson equals0I mk_disjoint_insert*)
 have *dim ((+) (-c) 'S)* < *DIM*('a')
 by (*metis* ⟨*S* = *insert c S'*⟩ *aff_dim_eq_dim* *assms hull_inc insertI1 of_nat_less_imp_less*)
 then obtain *a* where *a* ≠ 0 *span ((+) (-c) 'S)* ⊆ {*x*. *a* · *x* = 0}
 using *lowdim_subset_hyperplane* by *blast*
 moreover
 have *a* · *w* = *a* · *c* if *span ((+) (-c) 'S)* ⊆ {*x*. *a* · *x* = 0} *w* ∈ *S* for *w*
proof -
 have *w - c* ∈ *span ((+) (-c) 'S)*
 by (*simp add: span_base* ⟨*w* ∈ *S*⟩)
 with *that* have *w - c* ∈ {*x*. *a* · *x* = 0}
 by *blast*


```

    then show ?thesis
      by (auto simp: algebra_simps)
  qed
  ultimately have  $S \subseteq \{x. a \cdot x = a \cdot c\}$ 
    by blast
  then show ?thesis
    by (rule that[OF  $\langle a \neq 0 \rangle$ ])
qed

lemma affine_independent_card_dim_diffs:
  fixes  $S :: 'a :: euclidean\_space$  set
  assumes  $\neg$  affine_dependent  $S$   $a \in S$ 
  shows  $\text{card } S = \text{dim } ((\lambda x. x - a) \text{ ` } S) + 1$ 
proof -
  have non:  $\neg$  affine_dependent (insert  $a$   $S$ )
    by (simp add: assms insert_absorb)
  have finite  $S$ 
    by (meson assms aff_independent_finite)
  with  $\langle a \in S \rangle$  have  $\text{card } S \neq 0$  by auto
  moreover have  $\text{dim } ((\lambda x. x - a) \text{ ` } S) = \text{card } S - 1$ 
    using aff_dim_eq_dim_subtract aff_dim_unique  $\langle a \in S \rangle$  hull_inc insert_absorb
  non by fastforce
  ultimately show ?thesis
    by auto
qed

lemma independent_card_le_aff_dim:
  fixes  $B :: 'n :: euclidean\_space$  set
  assumes  $B \subseteq V$ 
  assumes  $\neg$  affine_dependent  $B$ 
  shows  $\text{int } (\text{card } B) \leq \text{aff\_dim } V + 1$ 
proof -
  obtain  $T$  where  $T: \neg$  affine_dependent  $T \wedge B \subseteq T \wedge T \subseteq V \wedge$  affine hull  $T$ 
    = affine hull  $V$ 
    by (metis assms extend_to_affine_basis[of  $B$   $V$ ])
  then have  $\text{of\_nat } (\text{card } T) = \text{aff\_dim } V + 1$ 
    using aff_dim_unique by auto
  then show ?thesis
    using  $T$  card_mono[of  $T$   $B$ ] aff_independent_finite[of  $T$ ] by auto
qed

lemma aff_dim_subset:
  fixes  $S T :: 'n :: euclidean\_space$  set
  assumes  $S \subseteq T$ 
  shows  $\text{aff\_dim } S \leq \text{aff\_dim } T$ 
proof -
  obtain  $B$  where  $B: \neg$  affine_dependent  $B \wedge B \subseteq S$  affine hull  $B =$  affine hull  $S$ 
    of_nat  $(\text{card } B) = \text{aff\_dim } S + 1$ 
    using aff_dim_inner_basis_exists[of  $S$ ] by auto

```

```

    then have  $\text{int}(\text{card } B) \leq \text{aff\_dim } T + 1$ 
      using assms independent_card_le_aff_dim[of  $B \ T$ ] by auto
    with  $B$  show ?thesis by auto
  qed

```

```

lemma aff_dim_le_DIM:
  fixes  $S :: 'n::\text{euclidean\_space set}$ 
  shows  $\text{aff\_dim } S \leq \text{int}(\text{DIM}('n))$ 
proof -
  have  $\text{aff\_dim}(\text{UNIV} :: 'n::\text{euclidean\_space set}) = \text{int}(\text{DIM}('n))$ 
    using aff_dim_UNIV by auto
  then show  $\text{aff\_dim}(S :: 'n::\text{euclidean\_space set}) \leq \text{int}(\text{DIM}('n))$ 
    using aff_dim_subset[of  $S(\text{UNIV} :: ('n::\text{euclidean\_space set}))$ ] subset_UNIV
  by auto
qed

```

```

lemma affine_dim_equal:
  fixes  $S :: 'n::\text{euclidean\_space set}$ 
  assumes  $\text{affine } S$   $\text{affine } T$   $S \neq \{\}$   $S \subseteq T$   $\text{aff\_dim } S = \text{aff\_dim } T$ 
  shows  $S = T$ 
proof -
  obtain  $a$  where  $a \in S$  using assms by auto
  then have  $a \in T$  using assms by auto
  define  $LS$  where  $LS = \{y. \exists x \in S. (-a) + x = y\}$ 
  then have  $ls: \text{subspace } LS$  affine_parallel  $S \ LS$ 
    using assms parallel_subspace_explicit[of  $S \ a \ LS$ ]  $\langle a \in S \rangle$  by auto
  then have  $h1: \text{int}(\text{dim } LS) = \text{aff\_dim } S$ 
    using assms aff_dim_affine[of  $S \ LS$ ] by auto
  have  $T \neq \{\}$  using assms by auto
  define  $LT$  where  $LT = \{y. \exists x \in T. (-a) + x = y\}$ 
  then have  $lt: \text{subspace } LT \wedge \text{affine\_parallel } T \ LT$ 
    using assms parallel_subspace_explicit[of  $T \ a \ LT$ ]  $\langle a \in T \rangle$  by auto
  then have  $\text{int}(\text{dim } LT) = \text{aff\_dim } T$ 
    using assms aff_dim_affine[of  $T \ LT$ ]  $\langle T \neq \{\} \rangle$  by auto
  then have  $\text{dim } LS = \text{dim } LT$ 
    using  $h1$  assms by auto
  moreover have  $LS \leq LT$ 
    using LS_def LT_def assms by auto
  ultimately have  $LS = LT$ 
    using subspace_dim_equal[of  $LS \ LT$ ]  $ls \ lt$  by auto
  moreover have  $S = \{x. \exists y \in LS. a+y=x\}$ 
    using LS_def by auto
  moreover have  $T = \{x. \exists y \in LT. a+y=x\}$ 
    using LT_def by auto
  ultimately show ?thesis by auto
qed

```

```

lemma aff_dim_eq_0:
  fixes  $S :: 'a::\text{euclidean\_space set}$ 

```

```

  shows  $\text{aff\_dim } S = 0 \longleftrightarrow (\exists a. S = \{a\})$ 
proof (cases  $S = \{\}$ )
  case True
  then show ?thesis
    by auto
next
  case False
  then obtain  $a$  where  $a \in S$  by auto
  show ?thesis
  proof safe
    assume  $0: \text{aff\_dim } S = 0$ 
    have  $\neg \{a, b\} \subseteq S$  if  $b \neq a$  for  $b$ 
      by (metis  $0 \text{ aff\_dim\_2 aff\_dim\_subset not\_one\_le\_zero that}$ )
    then show  $\exists a. S = \{a\}$ 
      using  $\langle a \in S \rangle$  by blast
  qed auto
qed

lemma affine_hull_UNIV:
  fixes  $S :: 'n::\text{euclidean\_space set}$ 
  assumes  $\text{aff\_dim } S = \text{int}(\text{DIM}('n))$ 
  shows  $\text{affine hull } S = (\text{UNIV} :: ('n::\text{euclidean\_space}) \text{ set})$ 
proof -
  have  $S \neq \{\}$ 
    using  $\text{assms aff\_dim\_empty[of } S] \text{ by auto}$ 
  have  $h0: S \subseteq \text{affine hull } S$ 
    using  $\text{hull\_subset[of } S \_] \text{ by auto}$ 
  have  $h1: \text{aff\_dim } (\text{UNIV} :: ('n::\text{euclidean\_space}) \text{ set}) = \text{aff\_dim } S$ 
    using  $\text{aff\_dim\_UNIV assms by auto}$ 
  then have  $h2: \text{aff\_dim } (\text{affine hull } S) \leq \text{aff\_dim } (\text{UNIV} :: ('n::\text{euclidean\_space}) \text{ set})$ 
    using  $\text{aff\_dim\_le\_DIM[of affine hull } S] \text{ assms } h0 \text{ by auto}$ 
  have  $h3: \text{aff\_dim } S \leq \text{aff\_dim } (\text{affine hull } S)$ 
    using  $h0 \text{ aff\_dim\_subset[of } S \text{ affine hull } S] \text{ assms by auto}$ 
  then have  $h4: \text{aff\_dim } (\text{affine hull } S) = \text{aff\_dim } (\text{UNIV} :: ('n::\text{euclidean\_space}) \text{ set})$ 
    using  $h0 h1 h2 \text{ by auto}$ 
  then show ?thesis
    using  $\text{affine\_dim\_equal[of affine hull } S (\text{UNIV} :: ('n::\text{euclidean\_space}) \text{ set})]$ 
       $\text{affine\_affine\_hull[of } S] \text{ affine\_UNIV assms } h4 h0 \langle S \neq \{\} \rangle$ 
    by auto
qed

lemma disjoint_affine_hull:
  fixes  $s :: 'n::\text{euclidean\_space set}$ 
  assumes  $\neg \text{affine\_dependent } s \ t \subseteq s \ u \subseteq s \ t \cap u = \{\}$ 
  shows  $(\text{affine hull } t) \cap (\text{affine hull } u) = \{\}$ 
proof -
  from  $\text{assms}(1)$  have  $\text{finite } s$ 

```

```

    by (simp add: aff_independent_finite)
  with assms(2,3) have finite t finite u
    by (blast intro: finite_subset)+
  have False if  $y \in \text{affine hull } t$  and  $y \in \text{affine hull } u$  for  $y$ 
  proof -
    from that obtain a b
      where a1 [simp]:  $\text{sum } a \ t = 1$ 
      and [simp]:  $\text{sum } (\lambda v. a \ v \ *_R \ v) \ t = y$ 
      and [simp]:  $\text{sum } b \ u = 1 \ \text{sum } (\lambda v. b \ v \ *_R \ v) \ u = y$ 
      by (auto simp: affine_hull_finite ⟨finite t⟩ ⟨finite u⟩)
    define c where  $c \ x = (\text{if } x \in t \text{ then } a \ x \text{ else if } x \in u \text{ then } -(b \ x) \text{ else } 0)$  for  $x$ 
    from assms(2,3,4) have [simp]:  $s \cap t = t \ s \cap - \ t \cap u = u$ 
      by auto
    have  $\text{sum } c \ s = 0$ 
    by (simp add: c_def comm_monoid_add_class.sum.If_cases ⟨finite s⟩ sum_negf)
    moreover have  $\neg (\forall v \in s. \ c \ v = 0)$ 
      by (metis (no_types) IntD1 ⟨s ∩ t = t⟩ a1 c_def sum.neutral zero_neq_one)
    moreover have  $(\sum v \in s. \ c \ v \ *_R \ v) = 0$ 
    by (simp add: c_def if_smult sum_negf comm_monoid_add_class.sum.If_cases
      ⟨finite s⟩)
    ultimately show ?thesis
      using assms(1) ⟨finite s⟩ by (auto simp: affine_dependent_explicit)
  qed
  then show ?thesis by blast
qed
end

```

1.7 Convex Sets and Functions

```

theory Convex
imports
  Affine
  HOL-Library.Set_Algebras
begin

```

1.7.1 Convex Sets

```

definition convex :: 'a::real_vector set  $\Rightarrow$  bool
  where  $\text{convex } s \longleftrightarrow (\forall x \in s. \ \forall y \in s. \ \forall u \geq 0. \ \forall v \geq 0. \ u + v = 1 \longrightarrow u *_R x + v *_R y \in s)$ 

```

```

lemma convexI:
  assumes  $\bigwedge x \ y \ u \ v. \ x \in s \Longrightarrow y \in s \Longrightarrow 0 \leq u \Longrightarrow 0 \leq v \Longrightarrow u + v = 1 \Longrightarrow$ 
 $u *_R x + v *_R y \in s$ 
  shows  $\text{convex } s$ 
  using assms unfolding convex_def by fast

```

```

lemma convexD:

```

assumes *convex s* and $x \in s$ and $y \in s$ and $0 \leq u$ and $0 \leq v$ and $u + v = 1$
 shows $u *_R x + v *_R y \in s$
 using *assms* unfolding *convex_def* by *fast*

lemma *convex_alt*: $\text{convex } s \longleftrightarrow (\forall x \in s. \forall y \in s. \forall u. 0 \leq u \wedge u \leq 1 \longrightarrow ((1 - u) *_R x + u *_R y) \in s)$
 (is $\_ \longleftrightarrow ?alt$)
proof
 show *convex s* if *alt*: ?alt
proof –
 {
 fix $x\ y$ and $u\ v :: \text{real}$
 assume *mem*: $x \in s\ y \in s$
 assume $0 \leq u\ 0 \leq v$
 moreover
 assume $u + v = 1$
 then have $u = 1 - v$ by *auto*
 ultimately have $u *_R x + v *_R y \in s$
 using *alt* [*rule_format*, *OF mem*] by *auto*
 }
 then show ?thesis
 unfolding *convex_def* by *auto*
qed
 show ?alt if *convex s*
 using *that* by (*auto simp: convex_def*)
qed

lemma *convexD_alt*:
 assumes *convex s* $a \in s\ b \in s\ 0 \leq u\ u \leq 1$
 shows $((1 - u) *_R a + u *_R b) \in s$
 using *assms* unfolding *convex_alt* by *auto*

lemma *mem_convex_alt*:
 assumes *convex S* $x \in S\ y \in S\ u \geq 0\ v \geq 0\ u + v > 0$
 shows $((u/(u+v)) *_R x + (v/(u+v)) *_R y) \in S$
 using *assms*
 by (*simp add: convex_def zero_le_divide_iff add_divide_distrib [symmetric]*)

lemma *convex_empty*[*intro,simp*]: *convex* {}
 unfolding *convex_def* by *simp*

lemma *convex_singleton*[*intro,simp*]: *convex* {*a*}
 unfolding *convex_def* by (*auto simp: scaleR_left_distrib [symmetric]*)

lemma *convex_UNIV*[*intro,simp*]: *convex UNIV*
 unfolding *convex_def* by *auto*

lemma *convex_Inter*: $(\bigwedge s. s \in f \implies \text{convex } s) \implies \text{convex}(\bigcap f)$
 unfolding *convex_def* by *auto*

lemma *convex_Int*: $\text{convex } s \implies \text{convex } t \implies \text{convex } (s \cap t)$
unfolding *convex_def* **by** *auto*

lemma *convex_INT*: $(\bigwedge i. i \in A \implies \text{convex } (B \ i)) \implies \text{convex } (\bigcap_{i \in A} B \ i)$
unfolding *convex_def* **by** *auto*

lemma *convex_Times*: $\text{convex } s \implies \text{convex } t \implies \text{convex } (s \times t)$
unfolding *convex_def* **by** *auto*

lemma *convex_halfspace_le*: $\text{convex } \{x. \text{inner } a \ x \leq b\}$
unfolding *convex_def*
by (*auto simp: inner_add intro!: convex_bound_le*)

lemma *convex_halfspace_ge*: $\text{convex } \{x. \text{inner } a \ x \geq b\}$
proof –
have *: $\{x. \text{inner } a \ x \geq b\} = \{x. \text{inner } (-a) \ x \leq -b\}$
by *auto*
show ?thesis
unfolding * **using** *convex_halfspace_le*[of $-a \ -b$] **by** *auto*
qed

lemma *convex_halfspace_abs_le*: $\text{convex } \{x. |\text{inner } a \ x| \leq b\}$
proof –
have *: $\{x. |\text{inner } a \ x| \leq b\} = \{x. \text{inner } a \ x \leq b\} \cap \{x. -b \leq \text{inner } a \ x\}$
by *auto*
show ?thesis
unfolding * **by** (*simp add: convex_Int convex_halfspace_ge convex_halfspace_le*)
qed

lemma *convex_hyperplane*: $\text{convex } \{x. \text{inner } a \ x = b\}$
proof –
have *: $\{x. \text{inner } a \ x = b\} = \{x. \text{inner } a \ x \leq b\} \cap \{x. \text{inner } a \ x \geq b\}$
by *auto*
show ?thesis **using** *convex_halfspace_le convex_halfspace_ge*
by (*auto intro!: convex_Int simp: **)
qed

lemma *convex_halfspace_lt*: $\text{convex } \{x. \text{inner } a \ x < b\}$
unfolding *convex_def*
by (*auto simp: convex_bound_lt inner_add*)

lemma *convex_halfspace_gt*: $\text{convex } \{x. \text{inner } a \ x > b\}$
using *convex_halfspace_lt*[of $-a \ -b$] **by** *auto*

lemma *convex_halfspace_Re_ge*: $\text{convex } \{x. \text{Re } x \geq b\}$
using *convex_halfspace_ge*[of $b \ 1::\text{complex}$] **by** *simp*

lemma *convex_halfspace_Re_le*: $\text{convex } \{x. \text{Re } x \leq b\}$

```

using convex_halfspace_le[of 1::complex b] by simp

lemma convex_halfspace_Im_ge: convex {x. Im x ≥ b}
  using convex_halfspace_ge[of b i] by simp

lemma convex_halfspace_Im_le: convex {x. Im x ≤ b}
  using convex_halfspace_le[of i b] by simp

lemma convex_halfspace_Re_gt: convex {x. Re x > b}
  using convex_halfspace_gt[of b 1::complex] by simp

lemma convex_halfspace_Re_lt: convex {x. Re x < b}
  using convex_halfspace_lt[of 1::complex b] by simp

lemma convex_halfspace_Im_gt: convex {x. Im x > b}
  using convex_halfspace_gt[of b i] by simp

lemma convex_halfspace_Im_lt: convex {x. Im x < b}
  using convex_halfspace_lt[of i b] by simp

lemma convex_real_interval [iff]:
  fixes a b :: real
  shows convex {a..} and convex {..b}
    and convex {a<..} and convex {..<b}
    and convex {a..b} and convex {a<..b}
    and convex {a..

```

```

then show convex {a<..b}
  by (simp only: convex_Int 3 2)
have {a..b} = {a..}  $\cap$  {..b}
  by auto
then show convex {a..b}
  by (simp only: convex_Int 1 4)
have {a<..b} = {a<..}  $\cap$  {..b}
  by auto
then show convex {a<..b}
  by (simp only: convex_Int 3 4)
qed

```

```

lemma convex_Reals: convex  $\mathbb{R}$ 
  by (simp add: convex_def scaleR_conv_of_real)

```

1.7.2 Explicit expressions for convexity in terms of arbitrary sums

```

lemma convex_sum:
  fixes C :: 'a::real_vector set
  assumes finite S
  and convex C
  and  $(\sum i \in S. a\ i) = 1$ 
  assumes  $\bigwedge i. i \in S \implies a\ i \geq 0$ 
  and  $\bigwedge i. i \in S \implies y\ i \in C$ 
  shows  $(\sum j \in S. a\ j *_{\mathbb{R}} y\ j) \in C$ 
  using assms(1,3,4,5)
proof (induct arbitrary: a set: finite)
  case empty
  then show ?case by simp
next
  case (insert i S) note IH = this(3)
  have a i + sum a S = 1
  and 0  $\leq$  a i
  and  $\forall j \in S. 0 \leq a\ j$ 
  and y i  $\in$  C
  and  $\forall j \in S. y\ j \in C$ 
  using insert.hyps(1,2) insert.premis by simp_all
  then have 0  $\leq$  sum a S
  by (simp add: sum_nonneg)
  have a i *R y i +  $(\sum j \in S. a\ j *_{\mathbb{R}} y\ j) \in C$ 
  proof (cases sum a S = 0)
    case True
    with  $\langle a\ i + \text{sum } a\ S = 1 \rangle$  have a i = 1
    by simp
    from sum_nonneg_0 [OF  $\langle \text{finite } S \rangle$  _ True]  $\langle \forall j \in S. 0 \leq a\ j \rangle$  have  $\forall j \in S. a\ j = 0$ 
    by simp
    show ?thesis using  $\langle a\ i = 1 \rangle$  and  $\langle \forall j \in S. a\ j = 0 \rangle$  and  $\langle y\ i \in C \rangle$ 

```



```

    by simp
  next
    case False
    with ‹ $0 \leq \text{sum } a \ S$ › have  $0 < \text{sum } a \ S$ 
    by simp
    then have  $(\sum_{j \in S}. (a \ j / \text{sum } a \ S) *_{\mathbb{R}} y \ j) \in C$ 
    using ‹ $\forall j \in S. 0 \leq a \ j$ › and ‹ $\forall j \in S. y \ j \in C$ ›
    by (simp add: IH sum_divide_distrib [symmetric])
    from ‹convex C› and ‹ $y \ i \in C$ › and this and ‹ $0 \leq a \ i$ ›
    and ‹ $0 \leq \text{sum } a \ S$ › and ‹ $a \ i + \text{sum } a \ S = 1$ ›
    have  $a \ i *_{\mathbb{R}} y \ i + \text{sum } a \ S *_{\mathbb{R}} (\sum_{j \in S}. (a \ j / \text{sum } a \ S) *_{\mathbb{R}} y \ j) \in C$ 
    by (rule convexD)
    then show ?thesis
    by (simp add: scaleR_sum_right False)
  qed
  then show ?case using ‹finite S› and ‹ $i \notin S$ ›
  by simp
qed

lemma convex:
  convex S  $\longleftrightarrow (\forall (k::nat) \ u \ x. (\forall i. 1 \leq i \wedge i \leq k \longrightarrow 0 \leq u \ i \wedge x \ i \in S) \wedge (\text{sum } u \ \{1..k\} = 1) \longrightarrow \text{sum } (\lambda i. u \ i *_{\mathbb{R}} x \ i) \ \{1..k\} \in S)$ 
proof safe
  fix k :: nat
  fix u :: nat  $\Rightarrow$  real
  fix x
  assume convex S
   $\forall i. 1 \leq i \wedge i \leq k \longrightarrow 0 \leq u \ i \wedge x \ i \in S$ 
   $\text{sum } u \ \{1..k\} = 1$ 
  with convex_sum[of {1 .. k} S] show  $(\sum_{j \in \{1 .. k\}}. u \ j *_{\mathbb{R}} x \ j) \in S$ 
  by auto
next
  assume *:  $\forall k \ u \ x. (\forall i :: nat. 1 \leq i \wedge i \leq k \longrightarrow 0 \leq u \ i \wedge x \ i \in S) \wedge \text{sum } u \ \{1..k\} = 1 \longrightarrow (\sum_{i=1..k} u \ i *_{\mathbb{R}} (x \ i :: 'a)) \in S$ 
  {
    fix  $\mu :: real$ 
    fix x y :: 'a
    assume xy:  $x \in S \ y \in S$ 
    assume mu:  $\mu \geq 0 \ \mu \leq 1$ 
    let ?u =  $\lambda i. \text{if } (i :: nat) = 1 \text{ then } \mu \text{ else } 1 - \mu$ 
    let ?x =  $\lambda i. \text{if } (i :: nat) = 1 \text{ then } x \text{ else } y$ 
    have  $\{1 :: nat .. 2\} \cap - \{x. x = 1\} = \{2\}$ 
    by auto
    then have card:  $\text{card } (\{1 :: nat .. 2\} \cap - \{x. x = 1\}) = 1$ 
    by simp
    then have sum ?u {1 .. 2} = 1
    using sum.If_cases[of {(1 :: nat) .. 2}  $\lambda x. x = 1 \ \lambda x. \mu \ \lambda x. 1 - \mu$ ]

```

```

    by auto
  with  $*[rule\_format, of \ 2 \ ?u \ ?x]$  have  $S: (\sum j \in \{1..2\}. \ ?u \ j \ *_R \ ?x \ j) \in S$ 
    using  $mu \ xy$  by auto
  have  $grarr: (\sum j \in \{Suc \ (Suc \ 0)..2\}. \ ?u \ j \ *_R \ ?x \ j) = (1 - \mu) *_R y$ 
    using  $sum.atLeast\_Suc\_atMost[of \ Suc \ (Suc \ 0) \ 2 \ \lambda \ j. \ (1 - \mu) *_R y]$  by auto
  from  $sum.atLeast\_Suc\_atMost[of \ Suc \ 0 \ 2 \ \lambda \ j. \ ?u \ j \ *_R \ ?x \ j, simplified \ this]$ 
  have  $(\sum j \in \{1..2\}. \ ?u \ j \ *_R \ ?x \ j) = \mu *_R x + (1 - \mu) *_R y$ 
    by auto
  then have  $(1 - \mu) *_R y + \mu *_R x \in S$ 
    using  $S$  by (auto simp:  $add.commute$ )
}
then show  $convex \ S$ 
  unfolding  $convex\_alt$  by auto
qed

```

```

lemma  $convex\_explicit$ :
  fixes  $S :: 'a::real\_vector \ set$ 
  shows  $convex \ S \longleftrightarrow$ 
     $(\forall t \ u. \ finite \ t \wedge t \subseteq S \wedge (\forall x \in t. \ 0 \leq u \ x) \wedge sum \ u \ t = 1 \longrightarrow sum \ (\lambda x. \ u \ x \ *_R \ x) \ t \in S)$ 
proof safe
  fix  $t$ 
  fix  $u :: 'a \Rightarrow real$ 
  assume  $convex \ S$ 
  and  $finite \ t$ 
  and  $t \subseteq S \ \forall x \in t. \ 0 \leq u \ x \ sum \ u \ t = 1$ 
  then show  $(\sum x \in t. \ u \ x \ *_R \ x) \in S$ 
    using  $convex\_sum[of \ t \ S \ u \ \lambda \ x. \ x]$  by auto
next
  assume  $*: \forall t. \forall u. \ finite \ t \wedge t \subseteq S \wedge (\forall x \in t. \ 0 \leq u \ x) \wedge$ 
     $sum \ u \ t = 1 \longrightarrow (\sum x \in t. \ u \ x \ *_R \ x) \in S$ 
  show  $convex \ S$ 
    unfolding  $convex\_alt$ 
  proof safe
    fix  $x \ y$ 
    fix  $\mu :: real$ 
    assume  $**: \ x \in S \ y \in S \ 0 \leq \mu \ \mu \leq 1$ 
    show  $(1 - \mu) *_R x + \mu *_R y \in S$ 
    proof (cases  $x = y$ )
    case  $False$ 
    then show  $?thesis$ 
      using  $*[rule\_format, of \ \{x, y\} \ \lambda \ z. \ if \ z = x \ then \ 1 - \mu \ else \ \mu]$  **
      by auto
    next
    case  $True$ 
    then show  $?thesis$ 
      using  $*[rule\_format, of \ \{x, y\} \ \lambda \ z. \ 1]$  **
      by (auto simp:  $field\_simps \ real\_vector.scale\_left\_diff\_distrib$ )

```

qed
qed
qed

lemma *convex_finite*:

assumes *finite S*

shows $\text{convex } S \longleftrightarrow (\forall u. (\forall x \in S. 0 \leq u \ x) \wedge \text{sum } u \ S = 1 \longrightarrow \text{sum } (\lambda x. u \ x \ *_R \ x) \ S \in S)$

(**is** ?lhs = ?rhs)

proof

{ **have** *if_distrib_arg*: $\bigwedge P \ f \ g \ x. (if \ P \ then \ f \ else \ g) \ x = (if \ P \ then \ f \ x \ else \ g \ x)$
by *simp*

fix *T* :: 'a set **and** *u* :: 'a $\Rightarrow$ real

assume *sum*: $\forall u. (\forall x \in S. 0 \leq u \ x) \wedge \text{sum } u \ S = 1 \longrightarrow (\sum_{x \in S} u \ x \ *_R \ x) \in S$

assume *: $\forall x \in T. 0 \leq u \ x \ \text{sum } u \ T = 1$

assume *T* $\subseteq S$

then have *S* $\cap T = T$ **by** *auto*

with *sum* [*THEN spec* [**where** *x* = $\lambda x. if \ x \in T \ then \ u \ x \ else \ 0$]] * **have** $(\sum_{x \in T} u \ x \ *_R \ x) \in S$

by (*auto simp: assms sum.If_cases if_distrib if_distrib_arg*) }

moreover assume ?rhs

ultimately show ?lhs

unfolding *convex_explicit* **by** *auto*

qed (*auto simp: convex_explicit assms*)

1.7.3 Convex Functions on a Set

definition *convex_on* :: 'a::real_vector set $\Rightarrow$ ('a $\Rightarrow$ real) $\Rightarrow$ bool

where *convex_on S f* $\longleftrightarrow$

$(\forall x \in S. \forall y \in S. \forall u \geq 0. \forall v \geq 0. u + v = 1 \longrightarrow f \ (u \ *_R \ x + v \ *_R \ y) \leq u \ * \ f \ x + v \ * \ f \ y)$

definition *concave_on* :: 'a::real_vector set $\Rightarrow$ ('a $\Rightarrow$ real) $\Rightarrow$ bool

where *concave_on S f* $\equiv \text{convex\_on } S \ (\lambda x. - \ f \ x)$

lemma *concave_on_iff*:

concave_on S f $\longleftrightarrow$

$(\forall x \in S. \forall y \in S. \forall u \geq 0. \forall v \geq 0. u + v = 1 \longrightarrow f \ (u \ *_R \ x + v \ *_R \ y) \geq u \ * \ f \ x + v \ * \ f \ y)$

by (*auto simp: concave_on_def convex_on_def algebra_simps*)

lemma *convex_onI* [*intro?*]:

assumes $\bigwedge t \ x \ y. t > 0 \Longrightarrow t < 1 \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow$

$f \ ((1 - t) \ *_R \ x + t \ *_R \ y) \leq (1 - t) \ * \ f \ x + t \ * \ f \ y$

shows *convex_on A f*

unfolding *convex_on_def*

proof *clarify*

fix *x y*

```

fix u v :: real
assume A: x ∈ A y ∈ A u ≥ 0 v ≥ 0 u + v = 1
from A(5) have [simp]: v = 1 - u
  by (simp add: algebra_simps)
from A(1-4) show f (u *R x + v *R y) ≤ u * f x + v * f y
  using assms[of u y x]
  by (cases u = 0 ∨ u = 1) (auto simp: algebra_simps)
qed

```

```

lemma convex_on_linorderI [intro?]:
  fixes A :: ('a::{linorder,real_vector}) set
  assumes  $\bigwedge t x y. t > 0 \implies t < 1 \implies x \in A \implies y \in A \implies x < y \implies$ 
     $f ((1 - t) *_{\mathbf{R}} x + t *_{\mathbf{R}} y) \leq (1 - t) * f x + t * f y$ 
  shows convex_on A f
proof
  fix x y
  fix t :: real
  assume A: x ∈ A y ∈ A t > 0 t < 1
  with assms [of t x y] assms [of 1 - t y x]
  show f ((1 - t) *R x + t *R y) ≤ (1 - t) * f x + t * f y
    by (cases x y rule: linorder_cases) (auto simp: algebra_simps)
qed

```

```

lemma convex_onD:
  assumes convex_on A f
  shows  $\bigwedge t x y. t \geq 0 \implies t \leq 1 \implies x \in A \implies y \in A \implies$ 
     $f ((1 - t) *_{\mathbf{R}} x + t *_{\mathbf{R}} y) \leq (1 - t) * f x + t * f y$ 
  using assms by (auto simp: convex_on_def)

```

```

lemma convex_onD_Icc:
  assumes convex_on {x..y} f x ≤ (y :: _ :: {real_vector,preorder})
  shows  $\bigwedge t. t \geq 0 \implies t \leq 1 \implies$ 
     $f ((1 - t) *_{\mathbf{R}} x + t *_{\mathbf{R}} y) \leq (1 - t) * f x + t * f y$ 
  using assms(2) by (intro convex_onD [OF assms(1)]) simp_all

```

```

lemma convex_on_subset: convex_on t f  $\implies S \subseteq t \implies$  convex_on S f
  unfolding convex_on_def by auto

```

```

lemma convex_on_add [intro]:
  assumes convex_on S f
  and convex_on S g
  shows convex_on S (λx. f x + g x)
proof -
  {
    fix x y
    assume x ∈ S y ∈ S
    moreover
    fix u v :: real
    assume 0 ≤ u 0 ≤ v u + v = 1
  }

```

```

ultimately
  have  $f(u *_R x + v *_R y) + g(u *_R x + v *_R y) \leq (u * f x + v * f y) + (u$ 
 $* g x + v * g y)$ 
    using assms unfolding convex_on_def by (auto simp: add_mono)
  then have  $f(u *_R x + v *_R y) + g(u *_R x + v *_R y) \leq u * (f x + g x) + v$ 
 $* (f y + g y)$ 
    by (simp add: field_simps)
  }
  then show ?thesis
    unfolding convex_on_def by auto
qed

```

```

lemma convex_on_cmul [intro]:
  fixes  $c :: \text{real}$ 
  assumes  $0 \leq c$ 
    and convex_on  $S f$ 
  shows convex_on  $S (\lambda x. c * f x)$ 
proof -
  have *:  $u * (c * f x) + v * (c * f y) = c * (u * f x + v * f y)$ 
    for  $u c f x v f y :: \text{real}$ 
    by (simp add: field_simps)
  show ?thesis using assms(2) and mult_left_mono [OF assms(1)]
    unfolding convex_on_def and * by auto
qed

```

```

lemma convex_lower:
  assumes convex_on  $S f$ 
    and  $x \in S$ 
    and  $y \in S$ 
    and  $0 \leq u$ 
    and  $0 \leq v$ 
    and  $u + v = 1$ 
  shows  $f(u *_R x + v *_R y) \leq \max(f x) (f y)$ 
proof -
  let  $?m = \max(f x) (f y)$ 
  have  $u * f x + v * f y \leq u * \max(f x) (f y) + v * \max(f x) (f y)$ 
    using assms(4,5) by (auto simp: mult_left_mono add_mono)
  also have  $\dots = \max(f x) (f y)$ 
    using assms(6) by (simp add: distrib_right [symmetric])
  finally show ?thesis
    using assms unfolding convex_on_def by fastforce
qed

```

```

lemma convex_on_dist [intro]:
  fixes  $S :: 'a :: \text{real\_normed\_vector\_set}$ 
  shows convex_on  $S (\lambda x. \text{dist } a x)$ 
proof (auto simp: convex_on_def dist_norm)
  fix  $x y$ 
  assume  $x \in S \ y \in S$ 

```

```

fix  $u\ v :: \text{real}$ 
assume  $0 \leq u$ 
assume  $0 \leq v$ 
assume  $u + v = 1$ 
have  $a = u *_R a + v *_R a$ 
  unfolding scaleR_left_distrib [symmetric] and  $\langle u + v = 1 \rangle$  by simp
then have  $*$ :  $a - (u *_R x + v *_R y) = (u *_R (a - x)) + (v *_R (a - y))$ 
  by (auto simp: algebra_simps)
show  $\text{norm } (a - (u *_R x + v *_R y)) \leq u * \text{norm } (a - x) + v * \text{norm } (a - y)$ 
  unfolding  $*$  using norm_triangle_ineq [of  $u *_R (a - x)\ v *_R (a - y)$ ]
  using  $\langle 0 \leq u \rangle\ \langle 0 \leq v \rangle$  by auto
qed

```

1.7.4 Arithmetic operations on sets preserve convexity

```

lemma convex_linear_image:
  assumes linear f
  and convex S
  shows convex (f ` S)
proof –
  interpret  $f$ : linear f by fact
  from  $\langle \text{convex } S \rangle$  show convex (f ` S)
  by (simp add: convex_def f.scaleR [symmetric] f.add [symmetric])
qed

```

```

lemma convex_linear_vimage:
  assumes linear f
  and convex S
  shows convex (f  $^{-1}$  ` S)
proof –
  interpret  $f$ : linear f by fact
  from  $\langle \text{convex } S \rangle$  show convex (f  $^{-1}$  ` S)
  by (simp add: convex_def f.add f.scaleR)
qed

```

```

lemma convex_scaling:
  assumes convex S
  shows convex (( $\lambda x. c *_R x$ ) ` S)
proof –
  have linear ( $\lambda x. c *_R x$ )
  by (simp add: linearI scaleR_add_right)
  then show ?thesis
  using  $\langle \text{convex } S \rangle$  by (rule convex_linear_image)
qed

```

```

lemma convex_scaled:
  assumes convex S
  shows convex (( $\lambda x. x *_R c$ ) ` S)
proof –

```

```

have linear ( $\lambda x. x *_{\mathbb{R}} c$ )
  by (simp add: linearI scaleR_add_left)
then show ?thesis
  using <convex S> by (rule convex_linear_image)
qed

```

```

lemma convex_negations:
  assumes convex S
  shows convex ( $(\lambda x. - x) \text{ ` } S$ )
proof -
  have linear ( $\lambda x. - x$ )
    by (simp add: linearI)
  then show ?thesis
    using <convex S> by (rule convex_linear_image)
qed

```

```

lemma convex_sums:
  assumes convex S
  and convex T
  shows convex ( $\bigcup x \in S. \bigcup y \in T. \{x + y\}$ )
proof -
  have linear ( $\lambda(x, y). x + y$ )
    by (auto intro: linearI simp: scaleR_add_right)
  with assms have convex ( $(\lambda(x, y). x + y) \text{ ` } (S \times T)$ )
    by (intro convex_linear_image convex_Times)
  also have  $((\lambda(x, y). x + y) \text{ ` } (S \times T)) = (\bigcup x \in S. \bigcup y \in T. \{x + y\})$ 
    by auto
  finally show ?thesis .
qed

```

```

lemma convex_differences:
  assumes convex S convex T
  shows convex ( $\bigcup x \in S. \bigcup y \in T. \{x - y\}$ )
proof -
  have  $\{x - y \mid x y. x \in S \wedge y \in T\} = \{x + y \mid x y. x \in S \wedge y \in \text{uminus ` } T\}$ 
    by (auto simp: diff_conv_add_uminus simp del: add_uminus_conv_diff)
  then show ?thesis
    using convex_sums[OF assms(1) convex_negations[OF assms(2)]] by auto
qed

```

```

lemma convex_translation:
  convex ( $(+) a \text{ ` } S$ ) if convex S
proof -
  have  $(\bigcup x \in \{a\}. \bigcup y \in S. \{x + y\}) = (+) a \text{ ` } S$ 
    by auto
  then show ?thesis
    using convex_sums [OF convex_singleton [of a] that] by auto
qed

```

lemma *convex_translation_subtract*:
 $\text{convex } ((\lambda b. b - a) \text{ ' } S) \text{ if } \text{convex } S$
using *convex_translation* [of $S - a$] **that by** (*simp cong: image_cong_simp*)

lemma *convex_affinity*:
assumes *convex* S
shows $\text{convex } ((\lambda x. a + c *_R x) \text{ ' } S)$
proof –
have $(\lambda x. a + c *_R x) \text{ ' } S = (+) a \text{ ' } (*_R) c \text{ ' } S$
by *auto*
then show *?thesis*
using *convex_translation*[*OF convex_scaling*[*OF assms*], of $a\ c$] **by** *auto*
qed

lemma *convex_on_sum*:
fixes $a :: 'a \Rightarrow \text{real}$
and $y :: 'a \Rightarrow 'b::\text{real\_vector}$
and $f :: 'b \Rightarrow \text{real}$
assumes *finite* $s\ s \neq \{\}$
and *convex_on* $C\ f$
and *convex* C
and $(\sum i \in s. a\ i) = 1$
and $\bigwedge i. i \in s \implies a\ i \geq 0$
and $\bigwedge i. i \in s \implies y\ i \in C$
shows $f\ (\sum i \in s. a\ i *_R y\ i) \leq (\sum i \in s. a\ i * f\ (y\ i))$
using *assms*
proof (*induct* s *arbitrary: a rule: finite_ne_induct*)
case (*singleton* i)
then have $ai: a\ i = 1$
by *auto*
then show *?case*
by *auto*
next
case (*insert* $i\ s$)
then have *convex_on* $C\ f$
by *simp*
from *this*[*unfolded convex_on_def, rule_format*]
have *conv*: $\bigwedge x\ y\ \mu. x \in C \implies y \in C \implies 0 \leq \mu \implies \mu \leq 1 \implies$
 $f\ (\mu *_R x + (1 - \mu) *_R y) \leq \mu * f\ x + (1 - \mu) * f\ y$
by *simp*
show *?case*
proof (*cases* $a\ i = 1$)
case *True*
then have $(\sum j \in s. a\ j) = 0$
using *insert* **by** *auto*
then have $\bigwedge j. j \in s \implies a\ j = 0$
using *insert* **by** (*fastforce simp: sum_nonneg_eq_0_iff*)
then show *?thesis*
using *insert* **by** *auto*


```

next
case False
from insert have yai:  $y\ i \in C \wedge a\ i \geq 0$ 
by auto
have fis: finite (insert i s)
using insert by auto
then have ai1:  $a\ i \leq 1$ 
using sum_nonneg_leq_bound[of insert i s a] insert by simp
then have ai < 1
using False by auto
then have i0:  $1 - a\ i > 0$ 
by auto
let ?a =  $\lambda j. a\ j / (1 - a\ i)$ 
have a_nonneg:  $?a\ j \geq 0$  if  $j \in s$  for  $j$ 
using i0 insert that by fastforce
have (sum j in insert i s. a j) = 1
using insert by auto
then have (sum j in s. a j) =  $1 - a\ i$ 
using sum.insert insert by fastforce
then have (sum j in s. a j) / (1 - a i) = 1
using i0 by auto
then have a1: (sum j in s. ?a j) = 1
unfolding sum_divide_distrib by simp
have convex C using insert by auto
then have asum: (sum j in s. ?a j *R y j)  $\in C$ 
using insert convex_sum [OF <finite s> <convex C> a1 a_nonneg] by auto
have asum_le:  $f\ (\sum j \in s. ?a\ j *_{\mathbf{R}} y\ j) \leq (\sum j \in s. ?a\ j * f\ (y\ j))$ 
using a_nonneg a1 insert by blast
have f (sum j in insert i s. a j *R y j) =  $f\ ((\sum j \in s. a\ j *_{\mathbf{R}} y\ j) + a\ i *_{\mathbf{R}} y\ i)$ 
using sum.insert[of s i  $\lambda j. a\ j *_{\mathbf{R}} y\ j$ , OF <finite s> <i  $\notin$  s>] insert
by (auto simp only: add.commute)
also have ... =  $f\ (((1 - a\ i) * \text{inverse}\ (1 - a\ i)) *_{\mathbf{R}} (\sum j \in s. a\ j *_{\mathbf{R}} y\ j) + a\ i *_{\mathbf{R}} y\ i)$ 
using i0 by auto
also have ... =  $f\ ((1 - a\ i) *_{\mathbf{R}} (\sum j \in s. (a\ j * \text{inverse}\ (1 - a\ i)) *_{\mathbf{R}} y\ j) + a\ i *_{\mathbf{R}} y\ i)$ 
using scaleR_right.sum[of inverse (1 - a i)  $\lambda j. a\ j *_{\mathbf{R}} y\ j$ , symmetric]
by (auto simp: algebra_simps)
also have ... =  $f\ (((1 - a\ i) *_{\mathbf{R}} (\sum j \in s. ?a\ j *_{\mathbf{R}} y\ j) + a\ i *_{\mathbf{R}} y\ i)$ 
by (auto simp: divide_inverse)
also have ...  $\leq (1 - a\ i) *_{\mathbf{R}} f\ ((\sum j \in s. ?a\ j *_{\mathbf{R}} y\ j)) + a\ i *_{\mathbf{R}} f\ (y\ i)$ 
using conv[of y i (sum j in s. ?a j *R y j) a i, OF yai(1) asum yai(2) ai1]
by (auto simp: add.commute)
also have ...  $\leq (1 - a\ i) * (\sum j \in s. ?a\ j * f\ (y\ j)) + a\ i * f\ (y\ i)$ 
using add_right_mono [OF mult_left_mono [of _ 1 - a i,
OF asum_le less_imp_le[OF i0]], of a i * f (y i)]
by simp
also have ... =  $(\sum j \in s. (1 - a\ i) * ?a\ j * f\ (y\ j)) + a\ i * f\ (y\ i)$ 
unfolding sum_distrib_left[of 1 - a i  $\lambda j. ?a\ j * f\ (y\ j)$ ]

```

```

    using i0 by auto
    also have ... = (∑ j ∈ s. a j * f (y j)) + a i * f (y i)
    using i0 by auto
    also have ... = (∑ j ∈ insert i s. a j * f (y j))
    using insert by auto
    finally show ?thesis
    by simp
qed
qed

lemma convex_on_alt:
  fixes C :: 'a::real_vector set
  shows convex_on C f ⟷
    (∀ x ∈ C. ∀ y ∈ C. ∀ μ :: real. μ ≥ 0 ∧ μ ≤ 1 ⟶
      f (μ *R x + (1 - μ) *R y) ≤ μ * f x + (1 - μ) * f y)
proof safe
  fix x y
  fix μ :: real
  assume *: convex_on C f x ∈ C y ∈ C 0 ≤ μ ≤ 1
  from this[unfolded convex_on_def, rule_format]
  have 0 ≤ u ⟹ 0 ≤ v ⟹ u + v = 1 ⟹ f (u *R x + v *R y) ≤ u * f x + v
  * f y for u v
  by auto
  from this [of μ 1 - μ, simplified] *
  show f (μ *R x + (1 - μ) *R y) ≤ μ * f x + (1 - μ) * f y
  by auto
next
  assume *: ∀ x ∈ C. ∀ y ∈ C. ∀ μ. 0 ≤ μ ∧ μ ≤ 1 ⟶
    f (μ *R x + (1 - μ) *R y) ≤ μ * f x + (1 - μ) * f y
  {
    fix x y
    fix u v :: real
    assume **: x ∈ C y ∈ C u ≥ 0 v ≥ 0 u + v = 1
    then have[simp]: 1 - u = v by auto
    from *[rule_format, of x y u]
    have f (u *R x + v *R y) ≤ u * f x + v * f y
    using ** by auto
  }
  then show convex_on C f
  unfolding convex_on_def by auto
qed

```

```

lemma convex_on_diff:
  fixes f :: real ⇒ real
  assumes f: convex_on I f
  and I: x ∈ I y ∈ I
  and t: x < t < y
  shows (f x - f t) / (x - t) ≤ (f x - f y) / (x - y)
  and (f t - f y) / (x - y) ≤ (f t - f y) / (t - y)

```

```

proof -
  define a where a  $\equiv$  (t - y) / (x - y)
  with t have 0  $\leq$  a 0  $\leq$  1 - a
    by (auto simp: field_simps)
  with f  $\langle x \in I \rangle \langle y \in I \rangle$  have cvx: f (a * x + (1 - a) * y)  $\leq$  a * f x + (1 - a)
    * f y
    by (auto simp: convex_on_def)
  have a * x + (1 - a) * y = a * (x - y) + y
    by (simp add: field_simps)
  also have ... = t
    unfolding a_def using  $\langle x < t \rangle \langle t < y \rangle$  by simp
  finally have f t  $\leq$  a * f x + (1 - a) * f y
    using cvx by simp
  also have ... = a * (f x - f y) + f y
    by (simp add: field_simps)
  finally have f t - f y  $\leq$  a * (f x - f y)
    by simp
  with t show (f x - f t) / (x - t)  $\leq$  (f x - f y) / (x - y)
    by (simp add: le_divide_eq divide_le_eq field_simps a_def)
  with t show (f x - f y) / (x - y)  $\leq$  (f t - f y) / (t - y)
    by (simp add: le_divide_eq divide_le_eq field_simps)
qed

```

lemma pos_convex_function:

```

fixes f :: real  $\Rightarrow$  real
assumes convex C
  and leq:  $\bigwedge x y. x \in C \implies y \in C \implies f' x * (y - x) \leq f y - f x$ 
shows convex_on C f
  unfolding convex_on_alt
  using assms
proof safe
  fix x y  $\mu$  :: real
  let ?x =  $\mu *_R x + (1 - \mu) *_R y$ 
  assume *: convex C x  $\in$  C y  $\in$  C  $\mu \geq 0$   $\mu \leq 1$ 
  then have 1 -  $\mu \geq 0$  by auto
  then have xpos: ?x  $\in$  C
    using * unfolding convex_alt by fastforce
  have geg:  $\mu * (f x - f ?x) + (1 - \mu) * (f y - f ?x) \geq$ 
     $\mu * f' ?x * (x - ?x) + (1 - \mu) * f' ?x * (y - ?x)$ 
    using add_mono [OF mult_left_mono [OF leq [OF xpos *(2)]]  $\langle \mu \geq 0 \rangle$ ]
      mult_left_mono [OF leq [OF xpos *(3)]]  $\langle 1 - \mu \geq 0 \rangle$ ]
    by auto
  then have  $\mu * f x + (1 - \mu) * f y - f ?x \geq 0$ 
    by (auto simp: field_simps)
  then show f ( $\mu *_R x + (1 - \mu) *_R y$ )  $\leq$   $\mu * f x + (1 - \mu) * f y$ 
    by auto
qed

```

lemma atMostAtLeast_subset_convex:

```

fixes C :: real set
assumes convex C
  and x ∈ C y ∈ C x < y
shows {x .. y} ⊆ C
proof safe
  fix z assume z: z ∈ {x .. y}
  have less: z ∈ C if *: x < z z < y
  proof -
    let ?μ = (y - z) / (y - x)
    have 0 ≤ ?μ ?μ ≤ 1
      using assms * by (auto simp: field_simps)
    then have comb: ?μ * x + (1 - ?μ) * y ∈ C
      using assms iffD1[OF convex_alt, rule_format, of C y x ?μ]
      by (simp add: algebra_simps)
    have ?μ * x + (1 - ?μ) * y = (y - z) * x / (y - x) + (1 - (y - z) / (y -
x)) * y
      by (auto simp: field_simps)
    also have ... = ((y - z) * x + (y - x - (y - z)) * y) / (y - x)
      using assms by (simp only: add_divide_distrib) (auto simp: field_simps)
    also have ... = z
      using assms by (auto simp: field_simps)
    finally show ?thesis
      using comb by auto
  qed
show z ∈ C
  using z less assms by (auto simp: le_less)
qed

lemma f''_imp_f':
fixes f :: real ⇒ real
assumes convex C
  and f': ⋀x. x ∈ C ⇒ DERIV f x :> (f' x)
  and f'': ⋀x. x ∈ C ⇒ DERIV f' x :> (f'' x)
  and pos: ⋀x. x ∈ C ⇒ f'' x ≥ 0
  and x: x ∈ C
  and y: y ∈ C
shows f' x * (y - x) ≤ f y - f x
using assms
proof -
  have less_imp: f y - f x ≥ f' x * (y - x) f' y * (x - y) ≤ f x - f y
    if *: x ∈ C y ∈ C y > x for x y :: real
  proof -
    from * have ge: y - x > 0 y - x ≥ 0
      by auto
    from * have le: x - y < 0 x - y ≤ 0
      by auto
    then obtain z1 where z1: z1 > x z1 < y f y - f x = (y - x) * f' z1
      using subsetD[OF atMostAtLeast_subset_convex[OF ⟨convex C⟩ ⟨x ∈ C⟩ ⟨y
∈ C⟩ ⟨x < y⟩],

```

```

      THEN f', THEN MVT2[OF ⟨x < y⟩, rule_format, unfolded atLeastAt-
Most_iff[symmetric]]]
    by auto
  then have z1 ∈ C
    using atMostAtLeast_subset_convex ⟨convex C⟩ ⟨x ∈ C⟩ ⟨y ∈ C⟩ ⟨x < y⟩
    by fastforce
  from z1 have z1': f x - f y = (x - y) * f' z1
    by (simp add: field_simps)
  obtain z2 where z2: z2 > x z2 < z1 f' z1 - f' x = (z1 - x) * f'' z2
    using subsetD[OF atMostAtLeast_subset_convex[OF ⟨convex C⟩ ⟨x ∈ C⟩ ⟨z1
∈ C⟩ ⟨x < z1⟩],
      THEN f'', THEN MVT2[OF ⟨x < z1⟩, rule_format, unfolded atLeastAt-
Most_iff[symmetric]]] z1
    by auto
  obtain z3 where z3: z3 > z1 z3 < y f' y - f' z1 = (y - z1) * f'' z3
    using subsetD[OF atMostAtLeast_subset_convex[OF ⟨convex C⟩ ⟨z1 ∈ C⟩
⟨y ∈ C⟩ ⟨z1 < y⟩],
      THEN f'', THEN MVT2[OF ⟨z1 < y⟩, rule_format, unfolded atLeastAt-
Most_iff[symmetric]]] z1
    by auto
  have f' y - (f x - f y) / (x - y) = f' y - f' z1
    using * z1' by auto
  also have ... = (y - z1) * f'' z3
    using z3 by auto
  finally have cool': f' y - (f x - f y) / (x - y) = (y - z1) * f'' z3
    by simp
  have A': y - z1 ≥ 0
    using z1 by auto
  have z3 ∈ C
    using z3 * atMostAtLeast_subset_convex ⟨convex C⟩ ⟨x ∈ C⟩ ⟨z1 ∈ C⟩ ⟨x
< z1⟩
    by fastforce
  then have B': f'' z3 ≥ 0
    using assms by auto
  from A' B' have (y - z1) * f'' z3 ≥ 0
    by auto
  from cool' this have f' y - (f x - f y) / (x - y) ≥ 0
    by auto
  from mult_right_mono_neg[OF this le(2)]
  have f' y * (x - y) - (f x - f y) / (x - y) * (x - y) ≤ 0 * (x - y)
    by (simp add: algebra_simps)
  then have f' y * (x - y) - (f x - f y) ≤ 0
    using le by auto
  then have res: f' y * (x - y) ≤ f x - f y
    by auto
  have (f y - f x) / (y - x) - f' x = f' z1 - f' x
    using * z1 by auto
  also have ... = (z1 - x) * f'' z2
    using z2 by auto

```

```

finally have cool:  $(f\ y - f\ x) / (y - x) - f'\ x = (z1 - x) * f''\ z2$ 
by simp
have A:  $z1 - x \geq 0$ 
using z1 by auto
have z2  $\in C$ 
using z2 z1 * atMostAtLeast_subset_convex  $\langle \text{convex } C \rangle \langle z1 \in C \rangle \langle y \in C \rangle$ 
 $\langle z1 < y \rangle$ 
by fastforce
then have B:  $f''\ z2 \geq 0$ 
using assms by auto
from A B have  $(z1 - x) * f''\ z2 \geq 0$ 
by auto
with cool have  $(f\ y - f\ x) / (y - x) - f'\ x \geq 0$ 
by auto
from mult_right_mono[OF this ge(2)]
have  $(f\ y - f\ x) / (y - x) * (y - x) - f'\ x * (y - x) \geq 0 * (y - x)$ 
by (simp add: algebra_simps)
then have  $f\ y - f\ x - f'\ x * (y - x) \geq 0$ 
using ge by auto
then show  $f\ y - f\ x \geq f'\ x * (y - x)$ 
using res by auto
qed
show ?thesis
proof (cases  $x = y$ )
case True
with x y show ?thesis by auto
next
case False
with less_imp x y show ?thesis
by (auto simp: neq_iff)
qed
qed

```

```

lemma f''_ge0_imp_convex:
fixes f :: real  $\Rightarrow$  real
assumes conv: convex C
and f':  $\bigwedge x. x \in C \implies \text{DERIV } f\ x :> (f'\ x)$ 
and f'':  $\bigwedge x. x \in C \implies \text{DERIV } f'\ x :> (f''\ x)$ 
and 0:  $\bigwedge x. x \in C \implies f''\ x \geq 0$ 
shows convex_on C f
using f''_imp_f'[OF conv f' f'' 0] assms pos_convex_function
by fastforce

```

```

lemma f''_le0_imp_concave:
fixes f :: real  $\Rightarrow$  real
assumes convex C
and  $\bigwedge x. x \in C \implies \text{DERIV } f\ x :> (f'\ x)$ 
and  $\bigwedge x. x \in C \implies \text{DERIV } f'\ x :> (f''\ x)$ 
and  $\bigwedge x. x \in C \implies f''\ x \leq 0$ 

```

```

shows concave_on C f
unfolding concave_on_def
by (rule assms f''_ge0_imp_convex derivative_eq_intros | simp)+

```

```

lemma log_concave:
  fixes b :: real
  assumes b > 1
  shows concave_on {0<..} ( $\lambda x. \log b x$ )
  using assms
  by (intro f''_le0_imp_concave derivative_eq_intros | simp)+

```

```

lemma ln_concave: concave_on {0<..} ln
  unfolding log_ln by (simp add: log_concave)

```

```

lemma minus_log_convex:
  fixes b :: real
  assumes b > 1
  shows convex_on {0 <..} ( $\lambda x. - \log b x$ )
  using assms concave_on_def log_concave by blast

```

```

lemma powr_convex:
  assumes p ≥ 1 shows convex_on {0<..} ( $\lambda x. x \text{ powr } p$ )
  using assms
  by (intro f''_ge0_imp_convex derivative_eq_intros | simp)+

```

```

lemma exp_convex: convex_on UNIV exp
  by (intro f''_ge0_imp_convex derivative_eq_intros | simp)+

```

1.7.5 Convexity of real functions

```

lemma convex_on_realI:
  assumes connected A
  and  $\bigwedge x. x \in A \implies (f \text{ has\_real\_derivative } f' x) (at x)$ 
  and  $\bigwedge x y. x \in A \implies y \in A \implies x \leq y \implies f' x \leq f' y$ 
  shows convex_on A f

```

```

proof (rule convex_on_linorderI)
  fix t x y :: real
  assume t: t > 0 t < 1
  assume xy: x ∈ A y ∈ A x < y
  define z where z = (1 - t) * x + t * y
  with ⟨connected A⟩ and xy have ivl: {x..y} ⊆ A
  using connected_contains_Icc by blast

```

```

from xy t have xz: z > x
  by (simp add: z_def algebra_simps)
have y - z = (1 - t) * (y - x)
  by (simp add: z_def algebra_simps)
also from xy t have ... > 0
  by (intro mult_pos_pos) simp_all

```

finally have $yz: z < y$
by *simp*

from *assms* $xz\ yz\ ivl\ t$ **have** $\exists \xi. \xi > x \wedge \xi < z \wedge f\ z - f\ x = (z - x) * f'\ \xi$
by (*intro* *MVT2*) (*auto* *intro!*: *assms*(2))
then obtain ξ **where** $\xi: \xi > x \wedge \xi < z \wedge f'\ \xi = (f\ z - f\ x) / (z - x)$
by *auto*
from *assms* $xz\ yz\ ivl\ t$ **have** $\exists \eta. \eta > z \wedge \eta < y \wedge f\ y - f\ z = (y - z) * f'\ \eta$
by (*intro* *MVT2*) (*auto* *intro!*: *assms*(2))
then obtain η **where** $\eta: \eta > z \wedge \eta < y \wedge f'\ \eta = (f\ y - f\ z) / (y - z)$
by *auto*

from $\eta(3)$ **have** $(f\ y - f\ z) / (y - z) = f'\ \eta$..
also from $\xi\ \eta\ ivl$ **have** $\xi \in A \wedge \eta \in A$
by *auto*

with $\xi\ \eta$ **have** $f'\ \eta \geq f'\ \xi$
by (*intro* *assms*(3)) *auto*
also from $\xi(3)$ **have** $f'\ \xi = (f\ z - f\ x) / (z - x)$..
finally have $(f\ y - f\ z) * (z - x) \geq (f\ z - f\ x) * (y - z)$
using $xz\ yz$ **by** (*simp* *add*: *field_simps*)
also have $z - x = t * (y - x)$
by (*simp* *add*: *z_def algebra_simps*)
also have $y - z = (1 - t) * (y - x)$
by (*simp* *add*: *z_def algebra_simps*)
finally have $(f\ y - f\ z) * t \geq (f\ z - f\ x) * (1 - t)$
using xy **by** *simp*
then show $(1 - t) * f\ x + t * f\ y \geq f\ ((1 - t) * x + t * y)$
by (*simp* *add*: *z_def algebra_simps*)

qed

lemma *convex_on_inverse*:

assumes $A \subseteq \{0 < ..\}$

shows *convex_on* A (*inverse* :: *real* $\Rightarrow$ *real*)

proof (*rule* *convex_on_subset*[*OF* __ *assms*], *intro* *convex_on_realI*[*of* __ $\lambda x.$
 $-inverse\ (x^2)$])

fix $u\ v :: real$

assume $u \in \{0 < ..\} \wedge v \in \{0 < ..\} \wedge u \leq v$

with *assms* **show** $-inverse\ (u^2) \leq -inverse\ (v^2)$

by (*intro* *le_imp_neg_le* *le_imp_inverse_le* *power_mono*) (*simp_all*)

qed (*insert* *assms*, *auto* *intro!*: *derivative_eq_intros* *simp*: *field_split_simps* *power2_eq_square*)

lemma *convex_onD_Icc'*:

assumes *convex_on* $\{x..y\}$ $f\ c \in \{x..y\}$

defines $d \equiv y - x$

shows $f\ c \leq (f\ y - f\ x) / d * (c - x) + f\ x$

proof (*cases* $x\ y$ *rule*: *linorder_cases*)

case *less*

then have $d: d > 0$

by (*simp* *add*: *d_def*)


```

from assms(2) less have  $A: 0 \leq (c - x) / d \ (c - x) / d \leq 1$ 
  by (simp_all add: d_def field_split_simps)
have  $f\ c = f\ (x + (c - x) * 1)$ 
  by simp
also from less have  $1 = ((y - x) / d)$ 
  by (simp add: d_def)
also from d have  $x + (c - x) * \dots = (1 - (c - x) / d) *_R x + ((c - x) / d)$ 
 $*_R y$ 
  by (simp add: field_simps)
also have  $f\ \dots \leq (1 - (c - x) / d) * f\ x + (c - x) / d * f\ y$ 
  using assms less by (intro convex_onD_Icc) simp_all
also from d have  $\dots = (f\ y - f\ x) / d * (c - x) + f\ x$ 
  by (simp add: field_simps)
finally show ?thesis .
qed (insert assms(2), simp_all)

```

```

lemma convex_onD_Icc'':
  assumes convex_on  $\{x..y\}$   $f\ c \in \{x..y\}$ 
  defines  $d \equiv y - x$ 
  shows  $f\ c \leq (f\ x - f\ y) / d * (y - c) + f\ y$ 
proof (cases x y rule: linorder_cases)
  case less
  then have  $d: d > 0$ 
  by (simp add: d_def)
  from assms(2) less have  $A: 0 \leq (y - c) / d \ (y - c) / d \leq 1$ 
  by (simp_all add: d_def field_split_simps)
  have  $f\ c = f\ (y - (y - c) * 1)$ 
  by simp
  also from less have  $1 = ((y - x) / d)$ 
  by (simp add: d_def)
  also from d have  $y - (y - c) * \dots = (1 - (1 - (y - c) / d)) *_R x + (1 -$ 
 $(y - c) / d) *_R y$ 
  by (simp add: field_simps)
  also have  $f\ \dots \leq (1 - (1 - (y - c) / d)) * f\ x + (1 - (y - c) / d) * f\ y$ 
  using assms less by (intro convex_onD_Icc) (simp_all add: field_simps)
  also from d have  $\dots = (f\ x - f\ y) / d * (y - c) + f\ y$ 
  by (simp add: field_simps)
  finally show ?thesis .
qed (insert assms(2), simp_all)

```

1.7.6 Some inequalities

```

lemma Youngs_inequality_0:
  fixes  $a::real$ 
  assumes  $0 \leq \alpha \ 0 \leq \beta \ \alpha + \beta = 1 \ a > 0 \ b > 0$ 
  shows  $a\ \text{powr}\ \alpha * b\ \text{powr}\ \beta \leq \alpha * a + \beta * b$ 
proof -
  have  $\alpha * \ln\ a + \beta * \ln\ b \leq \ln\ (\alpha * a + \beta * b)$ 
  using assms ln_concave by (simp add: concave_on_iff)

```

```

moreover have  $0 < \alpha * a + \beta * b$ 
using assms by (smt (verit) mult_pos_pos split_mult_pos_le)
ultimately show ?thesis
using assms by (simp add: powr_def mult_exp_exp flip: ln_ge_iff)
qed

```

```

lemma Youngs_inequality:
  fixes p::real
  assumes  $p > 1$   $q > 1$   $1/p + 1/q = 1$   $a \geq 0$   $b \geq 0$ 
  shows  $a * b \leq a^{\text{powr } p} / p + b^{\text{powr } q} / q$ 
proof (cases a=0 ∨ b=0)
  case False
  then show ?thesis
  using Youngs_inequality_0 [of 1/p 1/q a^powr p b^powr q] assms
  by (simp add: powr_powr)
qed (use assms in auto)

```

```

lemma Cauchy_Schwarz_ineq_sum:
  fixes a :: 'a  $\Rightarrow$  'b::linordered_field
  shows  $(\sum i \in I. a \ i * b \ i)^2 \leq (\sum i \in I. (a \ i)^2) * (\sum i \in I. (b \ i)^2)$ 
proof (cases  $(\sum i \in I. (b \ i)^2) > 0$ )
  case False
  then consider  $\bigwedge i. i \in I \Rightarrow b \ i = 0$  | infinite I
  by (metis (mono_tags, lifting) sum_pos2 zero_le_power2 zero_less_power2)
  thus ?thesis
  by fastforce
next
  case True
  define r where  $r \equiv (\sum i \in I. a \ i * b \ i) / (\sum i \in I. (b \ i)^2)$ 
  with True have  $*$ :  $(\sum i \in I. a \ i * b \ i) = r * (\sum i \in I. (b \ i)^2)$ 
  by simp
  have  $0 \leq (\sum i \in I. (a \ i - r * b \ i)^2)$ 
  by (meson sum_nonneg zero_le_power2)
  also have  $\dots = (\sum i \in I. (a \ i)^2) - 2 * r * (\sum i \in I. a \ i * b \ i) + r^2 * (\sum i \in I. (b \ i)^2)$ 
  by (simp add: algebra_simps power2_eq_square sum_distrib_left flip: sum.distrib)
  also have  $\dots = (\sum i \in I. (a \ i)^2) - (\sum i \in I. a \ i * b \ i) * r$ 
  by (simp add: * power2_eq_square)
  also have  $\dots = (\sum i \in I. (a \ i)^2) - ((\sum i \in I. a \ i * b \ i)^2 / (\sum i \in I. (b \ i)^2))$ 
  by (simp add: r_def power2_eq_square)
  finally have  $0 \leq (\sum i \in I. (a \ i)^2) - ((\sum i \in I. a \ i * b \ i)^2 / (\sum i \in I. (b \ i)^2))$ 
  hence  $((\sum i \in I. a \ i * b \ i)^2 / (\sum i \in I. (b \ i)^2)) \leq (\sum i \in I. (a \ i)^2)$ 
  by (simp add: le_diff_eq)
  thus  $((\sum i \in I. a \ i * b \ i)^2 \leq (\sum i \in I. (a \ i)^2) * (\sum i \in I. (b \ i)^2))$ 
  by (simp add: pos_divide_le_eq True)
qed

```

1.7.7 Misc related lemmas

lemma *convex_translation_eq* [simp]:
 $\text{convex } ((+) a \text{ ' } s) \longleftrightarrow \text{convex } s$
by (metis *convex_translation_translation_galois*)

lemma *convex_translation_subtract_eq* [simp]:
 $\text{convex } ((\lambda b. b - a) \text{ ' } s) \longleftrightarrow \text{convex } s$
using *convex_translation_eq* [of $-a$] **by** (simp cong: *image_cong_simp*)

lemma *convex_linear_image_eq* [simp]:
fixes $f :: 'a::\text{real\_vector} \Rightarrow 'b::\text{real\_vector}$
shows $\llbracket \text{linear } f; \text{inj } f \rrbracket \implies \text{convex } (f \text{ ' } s) \longleftrightarrow \text{convex } s$
by (metis (no_types) *convex_linear_image_convex_linear_vimage_inj_vimage_image_eq*)

lemma *vector_choose_size*:
assumes $0 \leq c$
obtains $x :: 'a::\{\text{real\_normed\_vector}, \text{perfect\_space}\}$ **where** $\text{norm } x = c$
proof –
obtain $a :: 'a$ **where** $a \neq 0$
using *UNIV_not_singleton UNIV_eq_I set_zero singletonI* **by** *fastforce*
then show *?thesis*
by (rule_tac $x = \text{scaleR } (c / \text{norm } a) a$ **in** *that*) (simp add: *assms*)
qed

lemma *vector_choose_dist*:
assumes $0 \leq c$
obtains $y :: 'a::\{\text{real\_normed\_vector}, \text{perfect\_space}\}$ **where** $\text{dist } x y = c$
by (metis *add_diff_cancel_left' assms dist_commute dist_norm vector_choose_size*)

lemma *sum_delta''*:
fixes $s :: 'a::\text{real\_vector}$ *set*
assumes *finite s*
shows $(\sum x \in s. (\text{if } y = x \text{ then } f x \text{ else } 0) *_R x) = (\text{if } y \in s \text{ then } (f y) *_R y \text{ else } 0)$
proof –
have $*$: $\bigwedge x y. (\text{if } y = x \text{ then } f x \text{ else } (0::\text{real})) *_R x = (\text{if } x=y \text{ then } (f x) *_R x \text{ else } 0)$
by *auto*
show *?thesis*
unfolding $*$ **using** *sum.delta[OF assms, of $y \lambda x. f x *_R x$]* **by** *auto*
qed

1.7.8 Cones

definition *cone* :: $'a::\text{real\_vector}$ *set* $\Rightarrow$ *bool*
where $\text{cone } s \longleftrightarrow (\forall x \in s. \forall c \geq 0. c *_R x \in s)$

lemma *cone_empty*[*intro, simp*]: *cone* $\{\}$
unfolding *cone_def* **by** *auto*

lemma *cone_univ*[*intro*, *simp*]: *cone UNIV*
unfolding *cone_def* **by** *auto*

lemma *cone_Inter*[*intro*]: $\forall s \in f. \text{cone } s \implies \text{cone } (\bigcap f)$
unfolding *cone_def* **by** *auto*

lemma *subspace_imp_cone*: *subspace S* $\implies$ *cone S*
by (*simp add: cone_def subspace_scale*)

Conic hull

lemma *cone_cone_hull*: *cone (cone hull S)*
unfolding *hull_def* **by** *auto*

lemma *cone_hull_eq*: *cone hull S = S* $\longleftrightarrow$ *cone S*
by (*metis cone_cone_hull hull_same*)

lemma *mem_cone*:
assumes *cone S x* $\in S$ $c \geq 0$
shows $c *_{\mathbb{R}} x \in S$
using *assms cone_def*[*of S*] **by** *auto*

lemma *cone_contains_0*:
assumes *cone S*
shows $S \neq \{\}$ $\longleftrightarrow 0 \in S$
using *assms mem_cone* **by** *fastforce*

lemma *cone_0*: *cone {0}*
unfolding *cone_def* **by** *auto*

lemma *cone_Union*[*intro*]: $(\forall s \in f. \text{cone } s) \longrightarrow \text{cone } (\bigcup f)$
unfolding *cone_def* **by** *blast*

lemma *cone_iff*:
assumes $S \neq \{\}$
shows $\text{cone } S \longleftrightarrow 0 \in S \wedge (\forall c. c > 0 \longrightarrow ((*_R) c) ` S = S)$

proof –

```

{
  assume cone S
  {
    fix  $c :: \text{real}$ 
    assume  $c > 0$ 
    {
      fix  $x$ 
      assume  $x \in S$ 
      then have  $x \in ((*_R) c) ` S$ 
      unfolding image_def
      using  $\langle \text{cone } S \rangle \langle c > 0 \rangle$  mem_cone[of S x 1/c]
      exI[of ( $\lambda t. t \in S \wedge x = c *_R t$ ) ( $1 / c$ )  $*_R x$ ]
    }
  }
}
```

```

      by auto
    }
  moreover
  {
    fix x
    assume  $x \in ((*_R) c) \text{ ' } S$ 
    then have  $x \in S$ 
      using  $\langle 0 < c \rangle \langle \text{cone } S \rangle \text{ mem\_cone}$  by fastforce
    }
    ultimately have  $((*_R) c) \text{ ' } S = S$  by blast
  }
  then have  $0 \in S \wedge (\forall c. c > 0 \longrightarrow ((*_R) c) \text{ ' } S = S)$ 
    using  $\langle \text{cone } S \rangle \text{ cone\_contains\_0[of } S] \text{ assms}$  by auto
  }
  moreover
  {
    assume  $a: 0 \in S \wedge (\forall c. c > 0 \longrightarrow ((*_R) c) \text{ ' } S = S)$ 
    {
      fix x
      assume  $x \in S$ 
      fix  $c1 :: \text{real}$ 
      assume  $c1 \geq 0$ 
      then have  $c1 = 0 \vee c1 > 0$  by auto
      then have  $c1 *_R x \in S$  using  $a \langle x \in S \rangle$  by auto
    }
    then have  $\text{cone } S \text{ unfolding cone\_def}$  by auto
  }
  ultimately show ?thesis by blast
qed

lemma cone_hull_empty:  $\text{cone hull } \{\} = \{\}$ 
  by (metis cone_empty cone_hull_eq)

lemma cone_hull_empty_iff:  $S = \{\} \longleftrightarrow \text{cone hull } S = \{\}$ 
  by (metis bot_least cone_hull_empty hull_subset xtrans(5))

lemma cone_hull_contains_0:  $S \neq \{\} \longleftrightarrow 0 \in \text{cone hull } S$ 
  using cone_cone_hull[of S] cone_contains_0[of cone hull S] cone_hull_empty_iff[of S]
  by auto

lemma mem_cone_hull:
  assumes  $x \in S \ c \geq 0$ 
  shows  $c *_R x \in \text{cone hull } S$ 
  by (metis assms cone_cone_hull hull_inc mem_cone)

proposition cone_hull_expl:  $\text{cone hull } S = \{c *_R x \mid c \geq 0 \wedge x \in S\}$ 
  (is ?lhs = ?rhs)
proof -

```

```

{
  fix x
  assume x ∈ ?rhs
  then obtain cx :: real and xx where x: x = cx *R xx cx ≥ 0 xx ∈ S
    by auto
  fix c :: real
  assume c: c ≥ 0
  then have c *R x = (c * cx) *R xx
    using x by (simp add: algebra_simps)
  moreover
  have c * cx ≥ 0 using c x by auto
  ultimately
  have c *R x ∈ ?rhs using x by auto
}
then have cone ?rhs
  unfolding cone_def by auto
then have ?rhs ∈ Collect cone
  unfolding mem_Collect_eq by auto
{
  fix x
  assume x ∈ S
  then have 1 *R x ∈ ?rhs
    using zero_le_one by blast
  then have x ∈ ?rhs by auto
}
then have S ⊆ ?rhs by auto
then have ?lhs ⊆ ?rhs
  using ⟨?rhs ∈ Collect cone⟩ hull_minimal[of S ?rhs cone] by auto
moreover
{
  fix x
  assume x ∈ ?rhs
  then obtain cx :: real and xx where x: x = cx *R xx cx ≥ 0 xx ∈ S
    by auto
  then have xx ∈ cone hull S
    using hull_subset[of S] by auto
  then have x ∈ ?lhs
    using x cone_cone_hull[of S] cone_def[of cone hull S] by auto
}
ultimately show ?thesis by auto
qed

```

lemma *convex_cone*:

$convex\ s \wedge cone\ s \longleftrightarrow (\forall x \in s. \forall y \in s. (x + y) \in s) \wedge (\forall x \in s. \forall c \geq 0. (c *_{\mathbb{R}} x) \in s)$

(is ?lhs = ?rhs)

proof –

```

{
  fix x y

```

```

    assume  $x \in s$   $y \in s$  and ?lhs
    then have  $2 *_{\mathbb{R}} x \in s$   $2 *_{\mathbb{R}} y \in s$ 
      unfolding cone_def by auto
    then have  $x + y \in s$ 
      using ‹?lhs›[unfolded convex_def, THEN conjunct1]
      apply (erule_tac  $x = 2 *_{\mathbb{R}} x$  in ballE)
      apply (erule_tac  $x = 2 *_{\mathbb{R}} y$  in ballE)
      apply (erule_tac  $x = 1/2$  in allE, simp)
      apply (erule_tac  $x = 1/2$  in allE, auto)
    done
  }
  then show ?thesis
    unfolding convex_def cone_def by blast
qed

```

1.7.9 Connectedness of convex sets

```

lemma convex_connected:
  fixes  $S :: 'a :: \text{real\_normed\_vector\_set}$ 
  assumes convex S
  shows connected S
proof (rule connectedI)
  fix A B
  assume open A open B  $A \cap B \cap S = \{\}$   $S \subseteq A \cup B$ 
  moreover
  assume  $A \cap S \neq \{\}$   $B \cap S \neq \{\}$ 
  then obtain a b where  $a \in A$   $a \in S$  and  $b \in B$   $b \in S$  by auto
  define f where [abs_def]:  $f\ u = u *_{\mathbb{R}} a + (1 - u) *_{\mathbb{R}} b$  for u
  then have continuous_on  $\{0 .. 1\}$  f
    by (auto intro!: continuous_intros)
  then have connected (f ‹ $\{0 .. 1\}$ ›)
    by (auto intro!: connected_continuous_image)
  note connectedD[OF this, of A B]
  moreover have  $a \in A \cap f\ ‹\{0 .. 1\}›$ 
    using a by (auto intro!: image_eqI[of _ _ 1] simp: f_def)
  moreover have  $b \in B \cap f\ ‹\{0 .. 1\}›$ 
    using b by (auto intro!: image_eqI[of _ _ 0] simp: f_def)
  moreover have  $f\ ‹\{0 .. 1\}› \subseteq S$ 
    using ‹convex S› a b unfolding convex_def f_def by auto
  ultimately show False by auto
qed

```

```

corollary connected_UNIV[intro]: connected (UNIV :: 'a :: real_normed_vector set)
by (simp add: convex_connected)

```

```

lemma convex_prod:
  assumes  $\bigwedge i. i \in \text{Basis} \implies \text{convex } \{x. P\ i\ x\}$ 
  shows  $\text{convex } \{x. \forall i \in \text{Basis}. P\ i\ (x \cdot i)\}$ 
  using assms unfolding convex_def

```

by (auto simp: inner_add_left)

lemma *convex_positive_orthant*: *convex* {*x*::'a::euclidean_space. ($\forall i \in \text{Basis}. 0 \leq x \cdot i$)}

by (rule *convex_prod*) (simp flip: *atLeast_def*)

1.7.10 Convex hull

lemma *convex_convex_hull* [iff]: *convex* (*convex hull* *s*)

unfolding *hull_def*

using *convex_Inter*[of {*t*. *convex t* $\wedge$ *s* $\subseteq$ *t*}]

by *auto*

lemma *convex_hull_subset*:

$s \subseteq \text{convex hull } t \implies \text{convex hull } s \subseteq \text{convex hull } t$

by (simp add: *subset_hull*)

lemma *convex_hull_eq*: *convex hull* *s* = *s* $\longleftrightarrow$ *convex s*

by (metis *convex_convex_hull hull_same*)

Convex hull is "preserved" by a linear function

lemma *convex_hull_linear_image*:

assumes *f*: *linear f*

shows $f \text{ ` } (\text{convex hull } s) = \text{convex hull } (f \text{ ` } s)$

proof

show $\text{convex hull } (f \text{ ` } s) \subseteq f \text{ ` } (\text{convex hull } s)$

by (intro *hull_minimal image_mono hull_subset convex_linear_image assms convex_convex_hull*)

show $f \text{ ` } (\text{convex hull } s) \subseteq \text{convex hull } (f \text{ ` } s)$

proof (*unfold image_subset_iff_subset_vimage, rule hull_minimal*)

show $s \subseteq f \text{ - ` } (\text{convex hull } (f \text{ ` } s))$

by (fast intro: *hull_inc*)

show $\text{convex } (f \text{ - ` } (\text{convex hull } (f \text{ ` } s)))$

by (intro *convex_linear_vimage [OF f] convex_convex_hull*)

qed

qed

lemma *in_convex_hull_linear_image*:

assumes *linear f*

and $x \in \text{convex hull } s$

shows $f x \in \text{convex hull } (f \text{ ` } s)$

using *convex_hull_linear_image*[OF *assms*(1)] *assms*(2) by *auto*

lemma *convex_hull_Times*:

$\text{convex hull } (s \times t) = (\text{convex hull } s) \times (\text{convex hull } t)$

proof

show $\text{convex hull } (s \times t) \subseteq (\text{convex hull } s) \times (\text{convex hull } t)$

by (intro *hull_minimal Sigma_mono hull_subset convex_Times convex_convex_hull*)


```

have  $(x, y) \in \text{convex hull } (s \times t)$  if  $x: x \in \text{convex hull } s$  and  $y: y \in \text{convex hull } t$ 
for  $x\ y$ 
proof (rule hull_induct [OF x], rule hull_induct [OF y])
  fix  $x\ y$  assume  $x \in s$  and  $y \in t$ 
  then show  $(x, y) \in \text{convex hull } (s \times t)$ 
    by (simp add: hull_inc)
next
fix  $x$  let  $?S = ((\lambda y. (0, y)) - '(\lambda p. (- x, 0) + p) ' (\text{convex hull } s \times t))$ 
have  $\text{convex } ?S$ 
  by (intro convex_linear_vimage convex_translation convex_convex_hull,
    simp add: linear_iff)
also have  $?S = \{y. (x, y) \in \text{convex hull } (s \times t)\}$ 
  by (auto simp: image_def Bex_def)
finally show  $\text{convex } \{y. (x, y) \in \text{convex hull } (s \times t)\}$  .
next
show  $\text{convex } \{x. (x, y) \in \text{convex hull } s \times t\}$ 
proof -
  fix  $y$  let  $?S = ((\lambda x. (x, 0)) - '(\lambda p. (0, - y) + p) ' (\text{convex hull } s \times t))$ 
have  $\text{convex } ?S$ 
  by (intro convex_linear_vimage convex_translation convex_convex_hull,
    simp add: linear_iff)
also have  $?S = \{x. (x, y) \in \text{convex hull } (s \times t)\}$ 
  by (auto simp: image_def Bex_def)
finally show  $\text{convex } \{x. (x, y) \in \text{convex hull } (s \times t)\}$  .
qed
qed
then show  $(\text{convex hull } s) \times (\text{convex hull } t) \subseteq \text{convex hull } (s \times t)$ 
  unfolding subset_eq split_paired_Ball_Sigma by blast
qed

```

Stepping theorems for convex hulls of finite sets

```

lemma convex_hull_empty[simp]:  $\text{convex hull } \{\} = \{\}$ 
  by (rule hull_unique) auto

```

```

lemma convex_hull_singleton[simp]:  $\text{convex hull } \{a\} = \{a\}$ 
  by (rule hull_unique) auto

```

```

lemma convex_hull_insert:
  fixes  $S :: 'a::\text{real\_vector set}$ 
  assumes  $S \neq \{\}$ 
  shows  $\text{convex hull } (\text{insert } a\ S) =$ 
     $\{x. \exists u \geq 0. \exists v \geq 0. \exists b. (u + v = 1) \wedge b \in (\text{convex hull } S) \wedge (x = u *_R a$ 
     $+ v *_R b)\}$ 
  (is  $\_ = ?hull)$ 
proof (intro equalityI hull_minimal subsetI)
  fix  $x$ 
  assume  $x \in \text{insert } a\ S$ 
  then have  $\exists u \geq 0. \exists v \geq 0. u + v = 1 \wedge (\exists b. b \in \text{convex hull } S \wedge x = u *_R a +$ 

```

```

v *R b)
  unfolding insert_iff
  proof
    assume x = a
    then show ?thesis
      by (rule_tac x=1 in exI) (use assms hull_subset in fastforce)
  next
    assume x ∈ S
    with hull_subset[of S convex] show ?thesis
      by force
  qed
  then show x ∈ ?hull
    by simp
next
  fix x
  assume x ∈ ?hull
  then obtain u v b where obt: u ≥ 0 v ≥ 0 u + v = 1 b ∈ convex hull S x = u *R
a + v *R b
    by auto
  have a ∈ convex hull insert a S b ∈ convex hull insert a S
    using hull_mono[of S insert a S convex] hull_mono[of {a} insert a S convex]
  and obt(4)
    by auto
  then show x ∈ convex hull insert a S
    unfolding obt(5) using obt(1-3)
    by (rule convexD [OF convex_convex_hull])
next
  show convex ?hull
  proof (rule convexI)
    fix x y u v
    assume as: (0::real) ≤ u 0 ≤ v u + v = 1 and x: x ∈ ?hull and y: y ∈ ?hull
    from x obtain u1 v1 b1 where
      obt1: u1 ≥ 0 v1 ≥ 0 u1 + v1 = 1 b1 ∈ convex hull S and xeq: x = u1 *R a +
v1 *R b1
      by auto
    from y obtain u2 v2 b2 where
      obt2: u2 ≥ 0 v2 ≥ 0 u2 + v2 = 1 b2 ∈ convex hull S and yeq: y = u2 *R a +
v2 *R b2
      by auto
    have *:  $\bigwedge (x::'a) s1 s2. x - s1 *R x - s2 *R x = ((1::real) - (s1 + s2)) *R x$ 
      by (auto simp: algebra_simps)
    have  $\exists b \in \text{convex hull } S. u *R x + v *R y =$ 
      (u * u1) *R a + (v * u2) *R a + (b - (u * u1) *R b - (v * u2) *R b)
    proof (cases u * v1 + v * v2 = 0)
      case True
      have *:  $\bigwedge (x::'a) s1 s2. x - s1 *R x - s2 *R x = ((1::real) - (s1 + s2)) *R$ 
x
      by (auto simp: algebra_simps)
      have eq0: u * v1 = 0 v * v2 = 0

```

```

    using True mult_nonneg_nonneg[OF  $\langle u \geq 0 \rangle \langle v1 \geq 0 \rangle$ ] mult_nonneg_nonneg[OF
 $\langle v \geq 0 \rangle \langle v2 \geq 0 \rangle$ ]
    by arith+
  then have  $u * u1 + v * u2 = 1$ 
    using as(3) obt1(3) obt2(3) by auto
  then show ?thesis
    using * eq0 as obt1(4) xeq yeq by auto
next
case False
have  $1 - (u * u1 + v * u2) = (u + v) - (u * u1 + v * u2)$ 
  using as(3) obt1(3) obt2(3) by (auto simp: field_simps)
also have  $\dots = u * (v1 + u1 - u1) + v * (v2 + u2 - u2)$ 
  using as(3) obt1(3) obt2(3) by (auto simp: field_simps)
also have  $\dots = u * v1 + v * v2$ 
  by simp
finally have **:  $1 - (u * u1 + v * u2) = u * v1 + v * v2$  by auto
let ?b =  $((u * v1) / (u * v1 + v * v2)) *_R b1 + ((v * v2) / (u * v1 + v * v2)) *_R b2$ 
have zeroes:  $0 \leq u * v1 + v * v2 \ 0 \leq u * v1 \ 0 \leq u * v1 + v * v2 \ 0 \leq v * v2$ 
  using as(1,2) obt1(1,2) obt2(1,2) by auto
show ?thesis
proof
  show  $u *_R x + v *_R y = (u * u1) *_R a + (v * u2) *_R a + (?b - (u * u1) *_R ?b - (v * u2) *_R ?b)$ 
    unfolding xeq yeq * **
    using False by (auto simp: scaleR_left_distrib scaleR_right_distrib)
  show ?b  $\in$  convex hull S
    using False zeroes obt1(4) obt2(4)
    by (auto simp: convexD [OF convex_convex_hull] scaleR_left_distrib
scaleR_right_distrib add_divide_distrib[symmetric] zero_le_divide_iff)
  qed
qed
then obtain b where b:  $b \in$  convex hull S
   $u *_R x + v *_R y = (u * u1) *_R a + (v * u2) *_R a + (b - (u * u1) *_R b - (v * u2) *_R b) ..$ 

have u1:  $u1 \leq 1$ 
  unfolding obt1(3)[symmetric] and not_le using obt1(2) by auto
have u2:  $u2 \leq 1$ 
  unfolding obt2(3)[symmetric] and not_le using obt2(2) by auto
have  $u1 * u + u2 * v \leq \max u1 u2 * u + \max u1 u2 * v$ 
proof (rule add_mono)
  show  $u1 * u \leq \max u1 u2 * u \ u2 * v \leq \max u1 u2 * v$ 
    by (simp_all add: as mult_right_mono)
qed
also have  $\dots \leq 1$ 
  unfolding distrib_left[symmetric] and as(3) using u1 u2 by auto
finally have le1:  $u1 * u + u2 * v \leq 1$  .

```

```

show  $u *_R x + v *_R y \in ?hull$ 
proof (intro CollectI exI conjI)
  show  $0 \leq u * u1 + v * u2$ 
    by (simp add: as(1) as(2) obt1(1) obt2(1))
  show  $0 \leq 1 - u * u1 - v * u2$ 
    by (simp add: le1 diff_diff_add mult.commute)
  qed (use b in <auto simp: algebra_simps>)
qed
qed

lemma convex_hull_insert_alt:
  convex hull (insert a S) =
    (if S = {} then {a}
     else {(1 - u) *_R a + u *_R x | x u. 0 ≤ u ∧ u ≤ 1 ∧ x ∈ convex hull S})
apply (auto simp: convex_hull_insert)
using diff_eq_eq apply fastforce
using diff_add_cancel diff_ge_0_iff_ge by blast

```

Explicit expression for convex hull

```

proposition convex_hull_indexed:
  fixes  $S :: 'a::real\_vector\ set$ 
  shows convex hull S =
    { $y. \exists k\ u\ x. (\forall i \in \{1::nat .. k\}. 0 \leq u\ i \wedge x\ i \in S) \wedge$ 
      ( $\sum u\ \{1..k\} = 1$ )  $\wedge (\sum i = 1..k. u\ i *_R x\ i) = y$ }
    (is ?xyz = ?hull)
proof (rule hull_unique [OF _ convexI])
  show  $S \subseteq ?hull$ 
    by (clarsimp, rule_tac x=1 in exI, rule_tac x=λx. 1 in exI, auto)
next
  fix  $T$ 
  assume  $S \subseteq T\ convex\ T$ 
  then show  $?hull \subseteq T$ 
    by (blast intro: convex_sum)
next
  fix  $x\ y\ u\ v$ 
  assume  $uv: 0 \leq u\ 0 \leq v\ u + v = (1::real)$ 
  assume  $xy: x \in ?hull\ y \in ?hull$ 
  from  $xy$  obtain  $k1\ u1\ x1$  where
     $x\ [rule\_format]: \forall i \in \{1::nat..k1\}. 0 \leq u1\ i \wedge x1\ i \in S$ 
     $\sum u1\ \{Suc\ 0..k1\} = 1\ (\sum i = Suc\ 0..k1. u1\ i *_R x1\ i) = x$ 
    by auto
  from  $xy$  obtain  $k2\ u2\ x2$  where
     $y\ [rule\_format]: \forall i \in \{1::nat..k2\}. 0 \leq u2\ i \wedge x2\ i \in S$ 
     $\sum u2\ \{Suc\ 0..k2\} = 1\ (\sum i = Suc\ 0..k2. u2\ i *_R x2\ i) = y$ 
    by auto
  have  $*$ :  $\bigwedge P\ (x::'a)\ y\ s\ t\ i. (if\ P\ i\ then\ s\ else\ t) *_R (if\ P\ i\ then\ x\ else\ y) = (if\ P$ 
     $i\ then\ s *_R\ x\ else\ t *_R\ y)$ 
     $\{1..k1 + k2\} \cap \{1..k1\} = \{1..k1\}\ \{1..k1 + k2\} \cap -\{1..k1\} = (\lambda i. i +$ 

```

```

k1) ' {1..k2}
  by auto
  have inj: inj_on ( $\lambda i. i + k1$ ) {1..k2}
    unfolding inj_on_def by auto
  let ?uu =  $\lambda i. \text{if } i \in \{1..k1\} \text{ then } u * u1 \ i \text{ else } v * u2 \ (i - k1)$ 
  let ?xx =  $\lambda i. \text{if } i \in \{1..k1\} \text{ then } x1 \ i \text{ else } x2 \ (i - k1)$ 
  show  $u *_R x + v *_R y \in ?hull$ 
  proof (intro CollectI exI conjI ballI)
    show  $0 \leq ?uu \ i \ ?xx \ i \in S$  if  $i \in \{1..k1+k2\}$  for  $i$ 
      using that by (auto simp add: le_diff_conv uv(1) x(1) uv(2) y(1))
    show  $(\sum i = 1..k1 + k2. ?uu \ i) = 1 \ (\sum i = 1..k1 + k2. ?uu \ i *_R ?xx \ i) =$ 
 $u *_R x + v *_R y$ 
      unfolding * sum.If_cases[OF finite_atLeastAtMost[of 1 k1 + k2]]
      sum.reindex[OF inj] Collect_mem_eq o_def
    unfolding scaleR_scaleR[symmetric] scaleR_right.sum [symmetric] sum_distrib_left[symmetric]
      by (simp_all add: sum_distrib_left[symmetric] x(2,3) y(2,3) uv(3))
  qed
qed

lemma convex_hull_finite:
  fixes S :: 'a::real_vector set
  assumes finite S
  shows  $\text{convex\_hull } S = \{y. \exists u. (\forall x \in S. 0 \leq u \ x) \wedge \text{sum } u \ S = 1 \wedge \text{sum } (\lambda x. u$ 
 $x *_R x) \ S = y\}$ 
  (is ?HULL =  $\_$ )
  proof (rule hull_unique [OF  $\_$  convexI]; clarify)
    fix x
    assume  $x \in S$ 
    then show  $\exists u. (\forall x \in S. 0 \leq u \ x) \wedge \text{sum } u \ S = 1 \wedge (\sum x \in S. u \ x *_R x) = x$ 
      by (rule_tac  $x = \lambda y. \text{if } x = y \text{ then } 1 \text{ else } 0$  in exI) (auto simp: sum.delta'[OF
assms] sum_delta'[OF assms])
  next
    fix u v :: real
    assume uv:  $0 \leq u \ 0 \leq v \ u + v = 1$ 
    fix ux assume ux [rule_format]:  $\forall x \in S. 0 \leq ux \ x \ \text{sum } ux \ S = (1::real)$ 
    fix uy assume uy [rule_format]:  $\forall x \in S. 0 \leq uy \ x \ \text{sum } uy \ S = (1::real)$ 
    have  $0 \leq u * ux \ x + v * uy \ x$  if  $x \in S$  for  $x$ 
      by (simp add: that uv ux(1) uy(1))
    moreover
    have  $(\sum x \in S. u * ux \ x + v * uy \ x) = 1$ 
      unfolding sum.distrib and sum_distrib_left[symmetric] ux(2) uy(2)
      using uv(3) by auto
    moreover
    have  $(\sum x \in S. (u * ux \ x + v * uy \ x) *_R x) = u *_R (\sum x \in S. ux \ x *_R x) + v *_R$ 
 $(\sum x \in S. uy \ x *_R x)$ 
      unfolding scaleR_left_distrib sum.distrib scaleR_scaleR[symmetric] scaleR_right.sum
[symmetric]
      by auto
    ultimately

```

```

show  $\exists uc. (\forall x \in S. 0 \leq uc\ x) \wedge \text{sum } uc\ S = 1 \wedge$ 
 $(\sum x \in S. uc\ x *_R x) = u *_R (\sum x \in S. ux\ x *_R x) + v *_R (\sum x \in S. uy\ x$ 
 $*_R x)$ 
by (rule_tac  $x = \lambda x. u * ux\ x + v * uy\ x$  in exI, auto)
qed (use assms in  $\langle \text{auto simp: convex\_explicit} \rangle$ )

```

Another formulation

Formalized by Lars Schewe.

```

lemma convex_hull_explicit:
  fixes  $p :: 'a::real\_vector\ set$ 
  shows convex hull  $p =$ 
 $\{y. \exists S\ u. \text{finite } S \wedge S \subseteq p \wedge (\forall x \in S. 0 \leq u\ x) \wedge \text{sum } u\ S = 1 \wedge \text{sum } (\lambda v. u\ v$ 
 $*_R v)\ S = y\}$ 
  (is ?lhs = ?rhs)
proof -
  {
    fix  $x$ 
    assume  $x \in ?lhs$ 
    then obtain  $k\ u\ y$  where
 $\text{obt: } \forall i \in \{1..k\}. 0 \leq u\ i \wedge y\ i \in p \text{ sum } u\ \{1..k\} = 1\ (\sum i = 1..k. u\ i$ 
 $*_R y\ i) = x$ 
    unfolding convex_hull_indexed by auto

    have fin: finite  $\{1..k\}$  by auto
    have fin':  $\bigwedge v. \text{finite } \{i \in \{1..k\}. y\ i = v\}$  by auto
    {
      fix  $j$ 
      assume  $j \in \{1..k\}$ 
      then have  $y\ j \in p \wedge 0 \leq \text{sum } u\ \{i. \text{Suc } 0 \leq i \wedge i \leq k \wedge y\ i = y\ j\}$ 
        using obt(1) [THEN bspec [where  $x=j$ ]] and obt(2)
        by (metis (no_types, lifting) One_nat_def atLeastAtMost_iff mem_Collect_eq
obt(1) sum_nonneg)
      }
    moreover
    have  $(\sum v \in y\ ' \{1..k\}. \text{sum } u\ \{i \in \{1..k\}. y\ i = v\}) = 1$ 
      unfolding sum.image_gen [OF fin, symmetric] using obt(2) by auto
    moreover have  $(\sum v \in y\ ' \{1..k\}. \text{sum } u\ \{i \in \{1..k\}. y\ i = v\} *_R v) = x$ 
      using sum.image_gen [OF fin, of  $\lambda i. u\ i *_R y\ i\ y$ , symmetric]
      unfolding scaleR_left.sum using obt(3) by auto
    ultimately
    have  $\exists S\ u. \text{finite } S \wedge S \subseteq p \wedge (\forall x \in S. 0 \leq u\ x) \wedge \text{sum } u\ S = 1 \wedge (\sum v \in S. u$ 
 $v *_R v) = x$ 
    apply (rule_tac  $x = y\ ' \{1..k\}$  in exI)
    apply (rule_tac  $x = \lambda v. \text{sum } u\ \{i \in \{1..k\}. y\ i = v\}$  in exI, auto)
    done
    then have  $x \in ?rhs$  by auto
  }
moreover

```

```

{
  fix y
  assume  $y \in ?rhs$ 
  then obtain  $S \ u$  where
    obt:  $finite\ S \ S \subseteq p \ \forall x \in S. \ 0 \leq u\ x \ \text{sum } u\ S = 1 \ (\sum_{v \in S} u\ v *_{\mathcal{R}} v) = y$ 
    by auto

  obtain  $f$  where  $f: inj\_on\ f\ \{1..card\ S\} \ f' \ \{1..card\ S\} = S$ 
    using  $ex\_bij\_betw\_nat\_finite\_1[OF\ obt(1)]$  unfolding  $bij\_betw\_def$  by
auto
  {
    fix  $i :: nat$ 
    assume  $i \in \{1..card\ S\}$ 
    then have  $f\ i \in S$ 
      using  $f(2)$  by blast
    then have  $0 \leq u\ (f\ i) \ f\ i \in p$  using  $obt(2,3)$  by auto
  }
  moreover have *:  $finite\ \{1..card\ S\}$  by auto
  {
    fix y
    assume  $y \in S$ 
    then obtain  $i$  where  $i \in \{1..card\ S\} \ f\ i = y$ 
      using  $f$  using  $image\_iff[of\ y\ f\ \{1..card\ S\}]$ 
      by auto
    then have  $\{x. Suc\ 0 \leq x \wedge x \leq card\ S \wedge f\ x = y\} = \{i\}$ 
      using  $f(1)$   $inj\_onD$  by fastforce
    then have  $card\ \{x. Suc\ 0 \leq x \wedge x \leq card\ S \wedge f\ x = y\} = 1$  by auto
    then have  $(\sum_{x \in \{x \in \{1..card\ S\}. f\ x = y\}. u\ (f\ x)}) = u\ y$ 
       $(\sum_{x \in \{x \in \{1..card\ S\}. f\ x = y\}. u\ (f\ x) *_{\mathcal{R}} f\ x}) = u\ y *_{\mathcal{R}} y$ 
      by (auto simp:  $sum\_constant\_scaleR$ )
  }
  then have  $(\sum_{x = 1..card\ S} u\ (f\ x)) = 1 \ (\sum_{i = 1..card\ S} u\ (f\ i) *_{\mathcal{R}} f\ i) =$ 
 $y$ 
    unfolding  $sum.image\_gen[OF\ *(1), of\ \lambda x. u\ (f\ x) *_{\mathcal{R}} f\ x\ f]$ 
    and  $sum.image\_gen[OF\ *(1), of\ \lambda x. u\ (f\ x)\ f]$ 
    unfolding  $f$ 
    using  $sum.cong\ [of\ S\ S\ \lambda y. (\sum_{x \in \{x \in \{1..card\ S\}. f\ x = y\}. u\ (f\ x) *_{\mathcal{R}} f\ x})$ 
 $\lambda v. u\ v *_{\mathcal{R}} v]$ 
    using  $sum.cong\ [of\ S\ S\ \lambda y. (\sum_{x \in \{x \in \{1..card\ S\}. f\ x = y\}. u\ (f\ x))\ u]$ 
    unfolding  $obt(4,5)$ 
    by auto
  ultimately
  have  $\exists k\ u\ x. (\forall i \in \{1..k\}. 0 \leq u\ i \wedge x\ i \in p) \wedge sum\ u\ \{1..k\} = 1 \wedge$ 
 $(\sum_{i::nat = 1..k} u\ i *_{\mathcal{R}} x\ i) = y$ 
    apply (rule_tac  $x=card\ S$  in  $exI$ )
    apply (rule_tac  $x=u \circ f$  in  $exI$ )
    apply (rule_tac  $x=f$  in  $exI$ , fastforce)
    done
  then have  $y \in ?lhs$ 

```

```

      unfolding convex_hull_indexed by auto
    }
    ultimately show ?thesis
      unfolding set_eq_iff by blast
  qed

```

A stepping theorem for that expansion

```

lemma convex_hull_finite_step:
  fixes S :: 'a::real_vector set
  assumes finite S
  shows
    ( $\exists u. (\forall x \in \text{insert } a \ S. 0 \leq u \ x) \wedge \text{sum } u \ (\text{insert } a \ S) = w \wedge \text{sum } (\lambda x. u \ x \ *_R \ x) \ (\text{insert } a \ S) = y$ )
     $\longleftrightarrow$  ( $\exists v \geq 0. \exists u. (\forall x \in S. 0 \leq u \ x) \wedge \text{sum } u \ S = w - v \wedge \text{sum } (\lambda x. u \ x \ *_R \ x) \ S = y - v \ *_R \ a$ )
    (is ?lhs = ?rhs)
  proof (cases a ∈ S)
    case True
    then have *: insert a S = S by auto
    show ?thesis
    proof
      assume ?lhs
      then show ?rhs
        unfolding * by force
    next
      have fin: finite (insert a S) using assms by auto
      assume ?rhs
      then obtain v u where uv:  $v \geq 0 \ \forall x \in S. 0 \leq u \ x \ \text{sum } u \ S = w - v \ (\sum_{x \in S} u \ x \ *_R \ x) = y - v \ *_R \ a$ 
        by auto
      then show ?lhs
        using uv True assms
        apply (rule_tac x =  $\lambda x. (\text{if } a = x \text{ then } v \text{ else } 0) + u \ x$  in exI)
        apply (auto simp: sum_clauses scaleR_left_distrib sum.distrib sum_delta'' [OF fin])
      done
    qed
  next
    case False
    show ?thesis
    proof
      assume ?lhs
      then obtain u where u:  $\forall x \in \text{insert } a \ S. 0 \leq u \ x \ \text{sum } u \ (\text{insert } a \ S) = w \ (\sum_{x \in \text{insert } a \ S} u \ x \ *_R \ x) = y$ 
        by auto
      then show ?rhs
        using u ⟨a ∉ S⟩ by (rule_tac x = u a in exI) (auto simp: sum_clauses assms)
    next

```



```

    assume ?rhs
    then obtain v u where uv: v ≥ 0 ∀ x ∈ S. 0 ≤ u x sum u S = w - v (∑ x ∈ S.
u x *R x) = y - v *R a
      by auto
    moreover
    have (∑ x ∈ S. if a = x then v else u x) = sum u S (∑ x ∈ S. (if a = x then v
else u x) *R x) = (∑ x ∈ S. u x *R x)
      using False by (auto intro!: sum.cong)
    ultimately show ?lhs
      using False by (rule_tac x = λx. if a = x then v else u x in exI) (auto simp:
sum_clauses(2)[OF assms])
  qed
qed

```

Hence some special cases

```

lemma convex_hull_2: convex hull {a,b} = {u *R a + v *R b | u v. 0 ≤ u ∧ 0
≤ v ∧ u + v = 1}
  (is ?lhs = ?rhs)
proof -
  have **: finite {b} by auto
  have ∧ x v u. [0 ≤ v; v ≤ 1; (1 - v) *R b = x - v *R a]
    ⇒ ∃ u v. x = u *R a + v *R b ∧ 0 ≤ u ∧ 0 ≤ v ∧ u + v = 1
    by (metis add.commute diff_add_cancel diff_ge_0_iff_ge)
  moreover
  have ∧ u v. [0 ≤ u; 0 ≤ v; u + v = 1]
    ⇒ ∃ p ≥ 0. ∃ q. 0 ≤ q b ∧ q b = 1 - p ∧ q b *R b = u *R a + v *R
b - p *R a
    apply (rule_tac x = u in exI, simp)
    apply (rule_tac x = λx. v in exI, simp)
    done
  ultimately show ?thesis
    using convex_hull_finite_step[OF **, of a 1]
    by (auto simp add: convex_hull_finite)
qed

```

```

lemma convex_hull_2_alt: convex hull {a,b} = {a + u *R (b - a) | u. 0 ≤ u ∧
u ≤ 1}
  unfolding convex_hull_2
proof (rule Collect_cong)
  have *: ∧ x y :: real. x + y = 1 ⟷ x = 1 - y
    by auto
  fix x
  show (∃ v u. x = v *R a + u *R b ∧ 0 ≤ v ∧ 0 ≤ u ∧ v + u = 1) ⟷
    (∃ u. x = a + u *R (b - a) ∧ 0 ≤ u ∧ u ≤ 1)
    apply (simp add: *)
    by (rule ex_cong1) (auto simp: algebra_simps)
qed

```

```

lemma convex_hull_3:
  convex_hull {a,b,c} = { u *R a + v *R b + w *R c | u v w. 0 ≤ u ∧ 0 ≤ v ∧ 0
  ≤ w ∧ u + v + w = 1 }
proof -
  have fin: finite {a,b,c} finite {b,c} finite {c}
    by auto
  have *: ∧ x y z :: real. x + y + z = 1 ↔ x = 1 - y - z
    by (auto simp: field_simps)
  show ?thesis
    unfolding convex_hull_finite[OF fin(1)] and convex_hull_finite_step[OF
    fin(2)] and *
    unfolding convex_hull_finite_step[OF fin(3)]
    apply (rule Collect_cong, simp)
    apply auto
    apply (rule_tac x=v in exI)
    apply (rule_tac x=u c in exI, simp)
    apply (rule_tac x=1 - v - w in exI, simp)
    apply (rule_tac x=v in exI, simp)
    apply (rule_tac x=λx. w in exI, simp)
    done
qed

```

```

lemma convex_hull_3_alt:
  convex_hull {a,b,c} = { a + u *R (b - a) + v *R (c - a) | u v. 0 ≤ u ∧ 0 ≤ v
  ∧ u + v ≤ 1 }
proof -
  have *: ∧ x y z :: real. x + y + z = 1 ↔ x = 1 - y - z
    by auto
  show ?thesis
    unfolding convex_hull_3
    apply (auto simp: *)
    apply (rule_tac x=v in exI)
    apply (rule_tac x=w in exI)
    apply (simp add: algebra_simps)
    apply (rule_tac x=u in exI)
    apply (rule_tac x=v in exI)
    apply (simp add: algebra_simps)
    done
qed

```

1.7.11 Relations among closure notions and corresponding hulls

```

lemma affine_imp_convex: affine s ⇒ convex s
  unfolding affine_def convex_def by auto

```

```

lemma convex_affine_hull [simp]: convex (affine_hull S)
  by (simp add: affine_imp_convex)

```

```

lemma subspace_imp_convex: subspace s  $\implies$  convex s
  using subspace_imp_affine affine_imp_convex by auto

lemma convex_hull_subset_span: (convex hull s)  $\subseteq$  (span s)
  by (metis hull_minimal span_superset subspace_imp_convex subspace_span)

lemma convex_hull_subset_affine_hull: (convex hull s)  $\subseteq$  (affine hull s)
  by (metis affine_affine_hull affine_imp_convex hull_minimal hull_subset)

lemma aff_dim_convex_hull:
  fixes S :: 'n::euclidean_space set
  shows aff_dim (convex hull S) = aff_dim S
  using aff_dim_affine_hull[of S] convex_hull_subset_affine_hull[of S]
    hull_subset[of S convex] aff_dim_subset[of S convex hull S]
    aff_dim_subset[of convex hull S affine hull S]
  by auto

```

1.7.12 Caratheodory's theorem

```

lemma convex_hull_caratheodory_aff_dim:
  fixes p :: ('a::euclidean_space) set
  shows convex hull p =
    {y.  $\exists S u$ . finite S  $\wedge$  S  $\subseteq$  p  $\wedge$  card S  $\leq$  aff_dim p + 1  $\wedge$ 
      ( $\forall x \in S$ .  $0 \leq u x$ )  $\wedge$  sum u S = 1  $\wedge$  sum ( $\lambda v$ . u v *R v) S = y}
  unfolding convex_hull_explicit set_eq_iff mem_Collect_eq
proof (intro allI iffI)
  fix y
  let ?P =  $\lambda n$ .  $\exists S u$ . finite S  $\wedge$  card S = n  $\wedge$  S  $\subseteq$  p  $\wedge$  ( $\forall x \in S$ .  $0 \leq u x$ )  $\wedge$ 
    sum u S = 1  $\wedge$  ( $\sum_{v \in S} u v *_{\mathbf{R}} v$ ) = y
  assume  $\exists S u$ . finite S  $\wedge$  S  $\subseteq$  p  $\wedge$  ( $\forall x \in S$ .  $0 \leq u x$ )  $\wedge$  sum u S = 1  $\wedge$  ( $\sum_{v \in S} u v *_{\mathbf{R}} v$ ) = y
  then obtain N where ?P N by auto
  then have  $\exists n \leq N$ . ( $\forall k < n$ .  $\neg$  ?P k)  $\wedge$  ?P n
    by (rule_tac ex_least_nat_le, auto)
  then obtain n where ?P n and smallest:  $\forall k < n$ .  $\neg$  ?P k
    by blast
  then obtain S u where obt: finite S card S = n S  $\subseteq$  p  $\forall x \in S$ .  $0 \leq u x$ 
    sum u S = 1 ( $\sum_{v \in S} u v *_{\mathbf{R}} v$ ) = y by auto

  have card S  $\leq$  aff_dim p + 1
  proof (rule ccontr, simp only: not_le)
    assume aff_dim p + 1 < card S
    then have affine_dependent S
      using affine_dependent_biggerset[OF obt(1)] independent_card_le_aff_dim
    not_less obt(3)
    by blast
    then obtain w v where wv: sum w S = 0  $\forall v \in S$  w v  $\neq$  0 ( $\sum_{v \in S} w v *_{\mathbf{R}} v$ ) = 0
    using affine_dependent_explicit_finite[OF obt(1)] by auto

```

```

define i where i = (λv. (u v) / (- w v)) ‘ {v∈S. w v < 0}
define t where t = Min i
have ∃ x∈S. w x < 0
proof (rule ccontr, simp add: not_less)
  assume as: ∀ x∈S. 0 ≤ w x
  then have sum w (S - {v}) ≥ 0
    by (meson Diff_iff sum_nonneg)
  then have sum w S > 0
    using as obt(1) sum_nonneg_eq_0_iff wv by blast
  then show False using wv(1) by auto
qed
then have i ≠ {} unfolding i_def by auto
then have t ≥ 0
  using Min_ge_iff[of i 0] and obt(1)
  unfolding t_def i_def
  using obt(4)[unfolded le_less]
  by (auto simp: divide_le_0_iff)
have t: ∀ v∈S. u v + t * w v ≥ 0
proof
  fix v
  assume v ∈ S
  then have v: 0 ≤ u v
  using obt(4)[THEN bspec[where x=v]] by auto
  show 0 ≤ u v + t * w v
  proof (cases w v < 0)
    case False
    thus ?thesis using v ⟨t ≥ 0⟩ by auto
  next
    case True
    then have t ≤ u v / (- w v)
      using ⟨v∈S⟩ obt unfolding t_def i_def by (auto intro: Min_le)
    then show ?thesis
      unfolding real_0_le_add_iff
      using True neg_le_minus_divide_eq by auto
  qed
qed
obtain a where a ∈ S and t = (λv. (u v) / (- w v)) a and w a < 0
  using Min_in[OF_ ⟨i ≠ {}⟩] and obt(1) unfolding i_def t_def by auto
then have a: a ∈ S u a + t * w a = 0 by auto
have *: ⋀ f. sum f (S - {a}) = sum f S - ((f a)::'b::ab_group_add)
  unfolding sum.remove[OF obt(1) ⟨a∈S⟩] by auto
have (∑ v∈S. u v + t * w v) = 1
  unfolding sum.distrib wv(1) sum_distrib_left[symmetric] obt(5) by auto
moreover have (∑ v∈S. u v *R v + (t * w v) *R v) - (u a *R a + (t * w a)
*R a) = y
  unfolding sum.distrib obt(6) scaleR_scaleR[symmetric] scaleR_right.sum
[symmetric] wv(4)
  using a(2) [THEN eq_neg_iff_add_eq_0 [THEN iffD2]] by simp
ultimately have ?P (n - 1)

```

```

    apply (rule_tac x=(S - {a}) in exI)
    apply (rule_tac x= $\lambda v. u \ v + t * w \ v$  in exI)
    using obt(1-3) and t and a
    apply (auto simp: * scaleR_left_distrib)
    done
  then show False
    using smallest[THEN spec[where x=n - 1]] by auto
qed
  then show  $\exists S \ u. \text{finite } S \wedge S \subseteq p \wedge \text{card } S \leq \text{aff\_dim } p + 1 \wedge$ 
     $(\forall x \in S. 0 \leq u \ x) \wedge \text{sum } u \ S = 1 \wedge (\sum v \in S. u \ v *_{\mathbb{R}} v) = y$ 
    using obt by auto
qed auto

lemma caratheodory_aff_dim:
  fixes p :: ('a::euclidean_space) set
  shows convex hull p = {x.  $\exists S. \text{finite } S \wedge S \subseteq p \wedge \text{card } S \leq \text{aff\_dim } p + 1 \wedge x$ 
 $\in \text{convex hull } S$ }
    (is ?lhs = ?rhs)
proof
  have  $\bigwedge x \ S \ u. [\text{finite } S; S \subseteq p; \text{int } (\text{card } S) \leq \text{aff\_dim } p + 1; \forall x \in S. 0 \leq u \ x;$ 
 $\text{sum } u \ S = 1]$ 
     $\implies (\sum v \in S. u \ v *_{\mathbb{R}} v) \in \text{convex hull } S$ 
  by (simp add: hull_subset convex_explicit [THEN iffD1, OF convex_convex_hull])
  then show ?lhs  $\subseteq$  ?rhs
    by (subst convex_hull_caratheodory_aff_dim, auto)
qed (use hull_mono in auto)

lemma convex_hull_caratheodory:
  fixes p :: ('a::euclidean_space) set
  shows convex hull p =
    {y.  $\exists S \ u. \text{finite } S \wedge S \subseteq p \wedge \text{card } S \leq \text{DIM}('a) + 1 \wedge$ 
 $(\forall x \in S. 0 \leq u \ x) \wedge \text{sum } u \ S = 1 \wedge \text{sum } (\lambda v. u \ v *_{\mathbb{R}} v) \ S = y$ }
    (is ?lhs = ?rhs)
proof (intro set_eqI iffI)
  fix x
  assume x  $\in$  ?lhs then show x  $\in$  ?rhs
    unfolding convex_hull_caratheodory_aff_dim
    using aff_dim_le_DIM [of p] by fastforce
qed (auto simp: convex_hull_explicit)

theorem caratheodory:
  convex hull p =
    {x::'a::euclidean_space.  $\exists S. \text{finite } S \wedge S \subseteq p \wedge \text{card } S \leq \text{DIM}('a) + 1 \wedge x \in$ 
 $\text{convex hull } S$ }
proof safe
  fix x
  assume x  $\in$  convex hull p
  then obtain S u where finite S  $S \subseteq p$   $\text{card } S \leq \text{DIM}('a) + 1$ 
     $\forall x \in S. 0 \leq u \ x$   $\text{sum } u \ S = 1$   $(\sum v \in S. u \ v *_{\mathbb{R}} v) = x$ 

```

```

    unfolding convex_hull_caratheodory by auto
  then show  $\exists S. \text{finite } S \wedge S \subseteq p \wedge \text{card } S \leq \text{DIM}(a) + 1 \wedge x \in \text{convex hull } S$ 
    using convex_hull_finite by fastforce
qed (use hull_mono in force)

```

1.7.13 Some Properties of subset of standard basis

```

lemma affine_hull_substd_basis:
  assumes  $d \subseteq \text{Basis}$ 
  shows  $\text{affine hull } (\text{insert } 0 \ d) = \{x::'a::\text{euclidean\_space}. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\}$ 
    (is  $\text{affine hull } (\text{insert } 0 \ ?A) = ?B$ )
proof -
  have *:  $\bigwedge A. (+) (0::'a) \ 'A = A \wedge A. (+) (- (0::'a)) \ 'A = A$ 
    by auto
  show ?thesis
    unfolding affine_hull_insert_span_gen span_substd_basis[OF assms,symmetric]
  * ..
qed

```

```

lemma affine_hull_convex_hull [simp]:  $\text{affine hull } (\text{convex hull } S) = \text{affine hull } S$ 
  by (metis Int_absorb1 Int_absorb2 convex_hull_subset_affine_hull hull_hull hull_mono hull_subset)

```

1.7.14 Moving and scaling convex hulls

```

lemma convex_hull_set_plus:
   $\text{convex hull } (S + T) = \text{convex hull } S + \text{convex hull } T$ 
  unfolding set_plus_image
  apply (subst convex_hull_linear_image [symmetric])
  apply (simp add: linear_iff scaleR_right_distrib)
  apply (simp add: convex_hull_Times)
  done

```

```

lemma translation_eq_singleton_plus:  $(\lambda x. a + x) \ 'T = \{a\} + T$ 
  unfolding set_plus_def by auto

```

```

lemma convex_hull_translation:
   $\text{convex hull } ((\lambda x. a + x) \ 'S) = (\lambda x. a + x) \ '(\text{convex hull } S)$ 
  unfolding translation_eq_singleton_plus
  by (simp only: convex_hull_set_plus convex_hull_singleton)

```

```

lemma convex_hull_scaling:
   $\text{convex hull } ((\lambda x. c *_{\mathbb{R}} x) \ 'S) = (\lambda x. c *_{\mathbb{R}} x) \ '(\text{convex hull } S)$ 
  using linear_scaleR by (rule convex_hull_linear_image [symmetric])

```

```

lemma convex_hull_affinity:
   $\text{convex hull } ((\lambda x. a + c *_{\mathbb{R}} x) \ 'S) = (\lambda x. a + c *_{\mathbb{R}} x) \ '(\text{convex hull } S)$ 
  by (metis convex_hull_scaling convex_hull_translation image_image)

```

1.7.15 Convexity of cone hulls

```

lemma convex_cone_hull:
  assumes convex S
  shows convex (cone hull S)
proof (rule convexI)
  fix x y
  assume xy:  $x \in \text{cone hull } S \ y \in \text{cone hull } S$ 
  then have  $S \neq \{\}$ 
    using cone_hull_empty_iff[of S] by auto
  fix u v :: real
  assume uv:  $u \geq 0 \ v \geq 0 \ u + v = 1$ 
  then have *:  $u *_R x \in \text{cone hull } S \ v *_R y \in \text{cone hull } S$ 
    using cone_cone_hull[of S] xy cone_def[of cone hull S] by auto
  from * obtain cx :: real and xx where  $x: u *_R x = cx *_R xx \ cx \geq 0 \ xx \in S$ 
    using cone_hull_expl[of S] by auto
  from * obtain cy :: real and yy where  $y: v *_R y = cy *_R yy \ cy \geq 0 \ yy \in S$ 
    using cone_hull_expl[of S] by auto
  {
    assume  $cx + cy \leq 0$ 
    then have  $u *_R x = 0$  and  $v *_R y = 0$ 
      using x y by auto
    then have  $u *_R x + v *_R y = 0$ 
      by auto
    then have  $u *_R x + v *_R y \in \text{cone hull } S$ 
      using cone_hull_contains_0[of S]  $\langle S \neq \{\} \rangle$  by auto
  }
  moreover
  {
    assume  $cx + cy > 0$ 
    then have  $(cx / (cx + cy)) *_R xx + (cy / (cx + cy)) *_R yy \in S$ 
      using assms mem_convex_alt[of S xx yy cx cy] x y by auto
    then have  $cx *_R xx + cy *_R yy \in \text{cone hull } S$ 
      using mem_cone_hull[of  $(cx / (cx + cy)) *_R xx + (cy / (cx + cy)) *_R yy \ S \ cx + cy]$ 
 $\langle cx + cy > 0 \rangle$ 
      by (auto simp: scaleR_right_distrib)
    then have  $u *_R x + v *_R y \in \text{cone hull } S$ 
      using x y by auto
  }
  moreover have  $cx + cy \leq 0 \vee cx + cy > 0$  by auto
  ultimately show  $u *_R x + v *_R y \in \text{cone hull } S$  by blast
qed

```

```

lemma cone_convex_hull:
  assumes cone S
  shows cone (convex hull S)
proof (cases  $S = \{\}$ )
  case True
    then show ?thesis by auto
next

```

```

case False
then have *:  $0 \in S \wedge (\forall c. c > 0 \longrightarrow (*_R) c \cdot S = S)$ 
  using cone_iff[of S] assms by auto
{
  fix c :: real
  assume c > 0
  then have  $(*_R) c \cdot (\text{convex hull } S) = \text{convex hull } ((*_R) c \cdot S)$ 
    using convex_hull_scaling[of _ S] by auto
  also have  $\dots = \text{convex hull } S$ 
    using * <math>c > 0</math> by auto
  finally have  $(*_R) c \cdot (\text{convex hull } S) = \text{convex hull } S$ 
    by auto
}
then have  $0 \in \text{convex hull } S \wedge c. c > 0 \implies ((*_R) c \cdot (\text{convex hull } S)) = (\text{convex hull } S)$ 
  using * hull_subset[of S convex] by auto
then show ?thesis
  using <math>S \neq \{ \} > cone_iff[of \text{convex hull } S] by auto
qed

```

1.7.16 Radon's theorem

Formalized by Lars Schewe.

```

lemma Radon_ex_lemma:
  assumes finite c affine_dependent c
  shows  $\exists u. \text{sum } u \ c = 0 \wedge (\exists v \in c. u \ v \neq 0) \wedge \text{sum } (\lambda v. u \ v \ *_R \ v) \ c = 0$ 
proof -
  from assms(2)[unfolded affine_dependent_explicit]
  obtain S u where
    finite S  $S \subseteq c$   $\text{sum } u \ S = 0$   $\exists v \in S. u \ v \neq 0$   $(\sum v \in S. u \ v \ *_R \ v) = 0$ 
  by blast
  then show ?thesis
    apply (rule_tac x= $\lambda v. \text{if } v \in S \text{ then } u \ v \text{ else } 0$  in exI)
    unfolding if_smult scaleR_zero_left
    by (auto simp: Int_absorb1 sum.inter_restrict[OF <math>finite \ c>, symmetric])
qed

```

```

lemma Radon_s_lemma:
  assumes finite S
  and  $\text{sum } f \ S = (0 :: real)$ 
  shows  $\text{sum } f \ \{x \in S. 0 < f \ x\} = - \text{sum } f \ \{x \in S. f \ x < 0\}$ 
proof -
  have *:  $\bigwedge x. (\text{if } f \ x < 0 \text{ then } f \ x \text{ else } 0) + (\text{if } 0 < f \ x \text{ then } f \ x \text{ else } 0) = f \ x$ 
    by auto
  show ?thesis
    unfolding add_eq_0_iff[symmetric] and sum.inter_filter[OF assms(1)]
    and sum.distrib[symmetric] and *
    using assms(2)
    by assumption

```


qed

lemma Radon_v_lemma:

assumes finite S
 and sum f S = 0
 and $\forall x. g\ x = (0::real) \longrightarrow f\ x = (0::'a::euclidean\_space)$
 shows $(\text{sum } f \{x \in S. 0 < g\ x\}) = - \text{sum } f \{x \in S. g\ x < 0\}$
 proof -
 have *: $\bigwedge x. (\text{if } 0 < g\ x \text{ then } f\ x \text{ else } 0) + (\text{if } g\ x < 0 \text{ then } f\ x \text{ else } 0) = f\ x$
 using assms(3) by auto
 show ?thesis
 unfolding eq_neg_iff_add_eq_0 and sum.inter_filter[OF assms(1)]
 and sum.distrib[symmetric] and *
 using assms(2)
 apply assumption
 done

qed

lemma Radon_partition:

assumes finite C affine_dependent C
 shows $\exists m\ p. m \cap p = \{\} \wedge m \cup p = C \wedge (\text{convex hull } m) \cap (\text{convex hull } p) \neq \{\}$
 {}
 proof -
 obtain u v where uv: $\text{sum } u\ C = 0 \wedge v \in C \wedge u\ v \neq 0 \wedge (\sum v \in C. u\ v *_{\mathbb{R}} v) = 0$
 using Radon_ex_lemma[OF assms] by auto
 have fin: $\text{finite } \{x \in C. 0 < u\ x\} \wedge \text{finite } \{x \in C. 0 > u\ x\}$
 using assms(1) by auto
 define z where $z = \text{inverse } (\text{sum } u \{x \in C. u\ x > 0\}) *_{\mathbb{R}} \text{sum } (\lambda x. u\ x *_{\mathbb{R}} x) \{x \in C. u\ x > 0\}$
 have sum_u {x ∈ C. 0 < u x} ≠ 0
 proof (cases u v ≥ 0)
 case False
 then have u v < 0 by auto
 then show ?thesis
 proof (cases ∃ w ∈ {x ∈ C. 0 < u x}. u w > 0)
 case True
 then show ?thesis
 using sum_nonneg_eq_0_iff[of _ u, OF fin(1)] by auto
 next
 case False
 then have $\text{sum } u\ C \leq \text{sum } (\lambda x. \text{if } x=v \text{ then } u\ v \text{ else } 0)\ C$
 by (rule_tac sum_mono, auto)
 then show ?thesis
 unfolding sum.delta[OF assms(1)] using uv(2) and $\langle u\ v < 0 \rangle$ and uv(1)
 by auto
 qed
 qed (insert sum_nonneg_eq_0_iff[of _ u, OF fin(1)] uv(2-3), auto)
 then have *: $\text{sum } u \{x \in C. u\ x > 0\} > 0$

```

    unfolding less_le by (metis (no_types, lifting) mem_Collect_eq sum_nonneg)
  moreover have  $\sum u \{x \in C. 0 < u x\} \cup \{x \in C. u x < 0\} = \sum u C$ 
     $(\sum x \in \{x \in C. 0 < u x\} \cup \{x \in C. u x < 0\}. u x *_R x) = (\sum x \in C. u x *_R x)$ 
    using assms(1)
  by (rule_tac[!] sum_mono_neutral_left, auto)
  then have  $\sum u \{x \in C. 0 < u x\} = - \sum u \{x \in C. 0 > u x\}$ 
     $(\sum x \in \{x \in C. 0 < u x\}. u x *_R x) = - (\sum x \in \{x \in C. 0 > u x\}. u x *_R x)$ 
    unfolding eq_neg_iff_add_eq_0
    using uv(1,4)
  by (auto simp: sum.union_inter_neutral[OF fin, symmetric])
  moreover have  $\forall x \in \{v \in C. u v < 0\}. 0 \leq \text{inverse} (\sum u \{x \in C. 0 < u x\})$ 
* - u x
    using * by (fastforce intro: mult_nonneg_nonneg)
  ultimately have  $z \in \text{convex hull } \{v \in C. u v \leq 0\}$ 
    unfolding convex_hull_explicit mem_Collect_eq
    apply (rule_tac x= $\{v \in C. u v < 0\}$  in exI)
    apply (rule_tac x= $\lambda y. \text{inverse} (\sum u \{x \in C. u x > 0\}) * - u y$  in exI)
    using assms(1) unfolding scaleR_scaleR[symmetric] scaleR_right.sum [symmetric]

    by (auto simp: z_def sum_negf sum_distrib_left[symmetric])
  moreover have  $\forall x \in \{v \in C. 0 < u v\}. 0 \leq \text{inverse} (\sum u \{x \in C. 0 < u x\})$ 
* u x
    using * by (fastforce intro: mult_nonneg_nonneg)
  then have  $z \in \text{convex hull } \{v \in C. u v > 0\}$ 
    unfolding convex_hull_explicit mem_Collect_eq
    apply (rule_tac x= $\{v \in C. 0 < u v\}$  in exI)
    apply (rule_tac x= $\lambda y. \text{inverse} (\sum u \{x \in C. u x > 0\}) * u y$  in exI)
    using assms(1)
    unfolding scaleR_scaleR[symmetric] scaleR_right.sum [symmetric]
    using * by (auto simp: z_def sum_negf sum_distrib_left[symmetric])
  ultimately show ?thesis
    apply (rule_tac x= $\{v \in C. u v \leq 0\}$  in exI)
    apply (rule_tac x= $\{v \in C. u v > 0\}$  in exI, auto)
    done
qed

```

theorem Radon:

```

  assumes affine_dependent c
  obtains  $m \ p$  where  $m \subseteq c \ p \subseteq c \ m \cap p = \{\}$   $(\text{convex hull } m) \cap (\text{convex hull } p) \neq \{\}$ 
proof -
  from assms[unfolded affine_dependent_explicit]
  obtain  $S \ u$  where
     $\text{finite } S \ S \subseteq c \ \sum u \ S = 0 \ \exists v \in S. u v \neq 0 \ (\sum v \in S. u v *_R v) = 0$ 
    by blast
  then have *:  $\text{finite } S \ \text{affine\_dependent } S$  and  $S: S \subseteq c$ 
    unfolding affine_dependent_explicit by auto
  from Radon_partition[OF *]
  obtain  $m \ p$  where  $m \cap p = \{\}$   $m \cup p = S$   $\text{convex hull } m \cap \text{convex hull } p \neq \{\}$ 

```

```

  by blast
with  $S$  show ?thesis
  by (force intro: that[of  $p$   $m$ ])
qed

```

1.7.17 Helly's theorem

```

lemma Helly_induct:
  fixes  $f :: 'a::euclidean\_space \text{ set set}$ 
  assumes  $\text{card } f = n$ 
    and  $n \geq \text{DIM}('a) + 1$ 
    and  $\forall s \in f. \text{convex } s \ \forall t \subseteq f. \text{card } t = \text{DIM}('a) + 1 \longrightarrow \bigcap t \neq \{\}$ 
  shows  $\bigcap f \neq \{\}$ 
  using assms
proof (induction  $n$  arbitrary:  $f$ )
  case 0
  then show ?case by auto
next
  case (Suc  $n$ )
  have finite  $f$ 
    using  $\langle \text{card } f = \text{Suc } n \rangle$  by (auto intro: card_ge_0_finite)
  show  $\bigcap f \neq \{\}$ 
  proof (cases  $n = \text{DIM}('a)$ )
    case True
    then show ?thesis
      by (simp add: Suc.premis(1) Suc.premis(4))
  next
    case False
    have  $\bigcap (f - \{s\}) \neq \{\}$  if  $s \in f$  for  $s$ 
    proof (rule Suc.IH[rule_format])
      show  $\text{card } (f - \{s\}) = n$ 
        by (simp add: Suc.premis(1)  $\langle \text{finite } f \rangle$  that)
      show  $\text{DIM}('a) + 1 \leq n$ 
        using False Suc.premis(2) by linarith
      show  $\bigwedge t. [t \subseteq f - \{s\}; \text{card } t = \text{DIM}('a) + 1] \Longrightarrow \bigcap t \neq \{\}$ 
        by (simp add: Suc.premis(4) subset_Diff_insert)
    qed (use Suc in auto)
    then have  $\forall s \in f. \exists x. x \in \bigcap (f - \{s\})$ 
      by blast
    then obtain  $X$  where  $X: \bigwedge s. s \in f \Longrightarrow X \ s \in \bigcap (f - \{s\})$ 
      by metis
    show ?thesis
    proof (cases  $\text{inj\_on } X \ f$ )
      case False
      then obtain  $s \ t$  where  $s \neq t$  and  $st: s \in f \ t \in f \ X \ s = X \ t$ 
        unfolding inj_on_def by auto
      then have  $*$ :  $\bigcap f = \bigcap (f - \{s\}) \cap \bigcap (f - \{t\})$  by auto
      show ?thesis
        by (metis  $*$   $X$  disjoint_iff_not_equal  $st$ )
    qed
  next

```

```

next
case True
then obtain m p where mp: m ∩ p = {} m ∪ p = X ' f convex hull m ∩
convex hull p ≠ {}
using Radon_partition[of X ' f] and affine_dependent_biggerset[of X ' f]
unfolding card_image[OF True] and ⟨card f = Suc n⟩
using Suc(3) ⟨finite f⟩ and False
by auto
have m ⊆ X ' f p ⊆ X ' f
using mp(2) by auto
then obtain g h where gh:m = X ' g p = X ' h g ⊆ f h ⊆ f
unfolding subset_image_iff by auto
then have f ∪ (g ∪ h) = f by auto
then have f: f = g ∪ h
using inj_on_Un_image_eq_iff[of X f g ∪ h] and True
unfolding mp(2)[unfolded image_Un[symmetric] gh]
by auto
have *: g ∩ h = {}
using gh(1) gh(2) local.mp(1) by blast
have convex_hull (X ' h) ⊆ ⋂ g convex_hull (X ' g) ⊆ ⋂ h
by (rule hull_minimal; use X * f in ⟨auto simp: Suc.prem(3) con-
vex_Inter⟩)+
then show ?thesis
unfolding f using mp(3)[unfolded gh] by blast
qed
qed
qed

```

theorem Helly:

```

fixes f :: 'a::euclidean_space set set
assumes card f ≥ DIM('a) + 1 ∀ s∈f. convex s
and ⋀ t. [t ⊆ f; card t = DIM('a) + 1] ⇒ ⋂ t ≠ {}
shows ⋂ f ≠ {}
using Helly_induct assms by blast

```

1.7.18 Epigraphs of convex functions

definition *epigraph* S ($f :: \_ \Rightarrow \text{real}$) = $\{xy. \text{fst } xy \in S \wedge f(\text{fst } xy) \leq \text{snd } xy\}$

lemma *mem_epigraph*: $(x, y) \in \text{epigraph } S f \longleftrightarrow x \in S \wedge f x \leq y$
unfolding *epigraph_def* **by** *auto*

lemma *convex_epigraph*: $\text{convex } (\text{epigraph } S f) \longleftrightarrow \text{convex\_on } S f \wedge \text{convex } S$
proof *safe*

```

assume L: convex (epigraph S f)
then show convex_on S f
by (auto simp: convex_def convex_on_def epigraph_def)
show convex S
using L by (fastforce simp: convex_def convex_on_def epigraph_def)

```

```

next
  assume convex_on S f convex S
  then show convex (epigraph S f)
    unfolding convex_def convex_on_def epigraph_def
    apply safe
    apply (rule_tac [2] y=u * f a + v * f aa in order_trans)
    apply (auto intro!:mult_left_mono add_mono)
    done
qed

lemma convex_epigraphI: convex_on S f  $\implies$  convex S  $\implies$  convex (epigraph S f)
  unfolding convex_epigraph by auto

lemma convex_epigraph_convex: convex S  $\implies$  convex_on S f  $\longleftrightarrow$  convex(epigraph S f)
  by (simp add: convex_epigraph)

Use this to derive general bound property of convex function

lemma convex_on:
  assumes convex S
  shows convex_on S f  $\longleftrightarrow$ 
    ( $\forall k\ u\ x. (\forall i \in \{1..k::nat\}. 0 \leq u\ i \wedge x\ i \in S) \wedge \text{sum } u\ \{1..k\} = 1 \longrightarrow$ 
       $f(\text{sum } (\lambda i. u\ i * _R x\ i)\ \{1..k\}) \leq \text{sum } (\lambda i. u\ i * f(x\ i))\ \{1..k\})$ 
    (is ?lhs = ( $\forall k\ u\ x. ?rhs\ k\ u\ x$ ))
proof
  assume ?lhs
  then have §: convex {xy. fst xy  $\in$  S  $\wedge$  f (fst xy)  $\leq$  snd xy}
    by (metis assms convex_epigraph epigraph_def)
  show  $\forall k\ u\ x. ?rhs\ k\ u\ x$ 
proof (intro allI)
  fix k u x
  show ?rhs k u x
  using §
  unfolding convex mem_Collect_eq fst_sum snd_sum
  apply safe
  apply (drule_tac x=k in spec)
  apply (drule_tac x=u in spec)
  apply (drule_tac x= $\lambda i. (x\ i, f(x\ i))$  in spec)
  apply simp
  done
qed
next
  assume  $\forall k\ u\ x. ?rhs\ k\ u\ x$ 
  then show ?lhs
    unfolding convex_epigraph_convex[OF assms] convex epigraph_def Ball_def
    mem_Collect_eq fst_sum snd_sum
    using assms[unfolded convex] apply clarsimp

```

apply (*rule_tac* $y = \sum i = 1..k. u\ i * f\ (fst\ (x\ i))$ **in** *order_trans*)
by (*auto simp add: mult_left_mono intro: sum_mono*)
qed

1.7.19 A bound within a convex hull

lemma *convex_on_convex_hull_bound*:
assumes *convex_on* (*convex hull* S) f
and $\forall x \in S. f\ x \leq b$
shows $\forall x \in \text{convex hull } S. f\ x \leq b$
proof
fix x
assume $x \in \text{convex hull } S$
then obtain $k\ u\ v$ **where**
 $u: \forall i \in \{1..k::nat\}. 0 \leq u\ i \wedge v\ i \in S \text{ sum } u\ \{1..k\} = 1\ (\sum i = 1..k. u\ i *_R v$
 $i) = x$
unfolding *convex_hull_indexed mem_Collect_eq* **by** *auto*
have $(\sum i = 1..k. u\ i * f\ (v\ i)) \leq b$
using *sum_mono[of {1..k} $\lambda i. u\ i * f\ (v\ i) \lambda i. u\ i * b$]*
unfolding *sum_distrib_right[symmetric]* $u(2)$ *mult_1*
using *assms(2) mult_left_mono u(1)* **by** *blast*
then show $f\ x \leq b$
using *assms(1)[unfolded convex_on[OF convex_convex_hull], rule_format, of*
 $k\ u\ v]$
using *hull_inc u* **by** *fastforce*
qed

lemma *inner_sum_Basis[simp]*: $i \in \text{Basis} \implies (\sum \text{Basis}) \cdot i = 1$
by (*simp add: inner_sum_left sum.If_cases inner_Basis*)

lemma *convex_set_plus*:
assumes *convex* S **and** *convex* T **shows** *convex* $(S + T)$
proof –
have *convex* $(\bigcup x \in S. \bigcup y \in T. \{x + y\})$
using *assms* **by** (*rule convex_sums*)
moreover have $(\bigcup x \in S. \bigcup y \in T. \{x + y\}) = S + T$
unfolding *set_plus_def* **by** *auto*
finally show *convex* $(S + T)$.
qed

lemma *convex_set_sum*:
assumes $\bigwedge i. i \in A \implies \text{convex } (B\ i)$
shows *convex* $(\sum i \in A. B\ i)$
proof (*cases finite A*)
case *True* **then show** *?thesis* **using** *assms*
by *induct (auto simp: convex_set_plus)*
qed *auto*

lemma *finite_set_sum*:

```

assumes finite A and  $\forall i \in A. \text{finite } (B\ i)$  shows  $\text{finite } (\sum_{i \in A} B\ i)$ 
using assms by (induct set: finite, simp, simp add: finite_set_plus)

lemma box_eq_set_sum_Basis:
   $\{x. \forall i \in \text{Basis}. x \cdot i \in B\ i\} = (\sum_{i \in \text{Basis}. (\lambda x. x *_{\mathbb{R}} i) \text{ ` } (B\ i)) \text{ (is ?lhs = ?rhs) )$ 
proof –
  have  $\bigwedge x. \forall i \in \text{Basis}. x \cdot i \in B\ i \implies$ 
     $\exists s. x = \text{sum } s\ \text{Basis} \wedge (\forall i \in \text{Basis}. s\ i \in (\lambda x. x *_{\mathbb{R}} i) \text{ ` } B\ i)$ 
    by (metis (mono_tags, lifting) euclidean_representation image_iff)
  moreover
  have  $\text{sum } f\ \text{Basis} \cdot i \in B\ i$  if  $i \in \text{Basis}$  and  $f: \forall i \in \text{Basis}. f\ i \in (\lambda x. x *_{\mathbb{R}} i) \text{ ` } B$ 
i for  $i$  if
  proof –
    have  $(\sum_{x \in \text{Basis} - \{i\}. f\ x \cdot i) = 0$ 
    proof (rule sum.neutral, intro strip)
      show  $f\ x \cdot i = 0$  if  $x \in \text{Basis} - \{i\}$  for  $x$ 
      using that  $f\ \langle i \in \text{Basis} \rangle \text{ inner\_Basis}$  that by fastforce
    qed
    then have  $(\sum_{x \in \text{Basis}. f\ x \cdot i) = f\ i \cdot i$ 
    by (metis (no_types) \langle i \in \text{Basis} \rangle add.right_neutral sum.remove [OF finite_Basis])
    then have  $(\sum_{x \in \text{Basis}. f\ x \cdot i) \in B\ i$ 
    using f that(1) by auto
    then show ?thesis
    by (simp add: inner_sum_left)
  qed
  ultimately show ?thesis
  by (subst set_sum_alt [OF finite_Basis]) auto
qed

lemma convex_hull_set_sum:
   $\text{convex hull } (\sum_{i \in A} B\ i) = (\sum_{i \in A} \text{convex hull } (B\ i))$ 
proof (cases finite A)
  assume finite A then show ?thesis
  by (induct set: finite, simp, simp add: convex_hull_set_plus)
qed simp

```

end

1.8 Definition of Finite Cartesian Product Type

```

theory Finite_Cartesian_Product
imports
  Euclidean_Space
  L2_Norm
  HOL-Library.Numeral_Type
  HOL-Library.Countable_Set
  HOL-Library.FuncSet

```

begin

1.8.1 Finite Cartesian products, with indexing and lambdas

```
typedef ('a, 'b) vec = UNIV :: ('b::finite  $\Rightarrow$  'a) set
  morphisms vec_nth vec_lambda ..
```

```
declare vec_lambda_inject [simplified, simp]
```

```
bundle vec_syntax begin
notation
  vec_nth (infixl $ 90) and
  vec_lambda (binder  $\chi$  10)
end
```

```
bundle no_vec_syntax begin
no_notation
  vec_nth (infixl $ 90) and
  vec_lambda (binder  $\chi$  10)
end
```

```
unbundle vec_syntax
```

Concrete syntax for ('a, 'b) vec:

- 'a[~]'b becomes ('a, 'b::finite) vec
- 'a[~]'b::_ becomes ('a, 'b) vec without extra sort-constraint

```
syntax _vec_type :: type  $\Rightarrow$  type  $\Rightarrow$  type (infixl ^ 15)
parse_translation <
  let
    fun vec t u = Syntax.const type_syntax <vec> $ t $ u;
    fun finite_vec_tr [t, u] =
      (case Term_Position.strip_positions u of
        v as Free (x, _) =>
          if Lexicon.is_tid x then
            vec t (Syntax.const syntax_const <_ofsort> $ v $
              Syntax.const class_syntax <finite>)
          else vec t u
        | _ => vec t u)
  in
    [(syntax_const <_vec_type>, K finite_vec_tr)]
  end
>
```

```
lemma vec_eq_iff: (x = y)  $\longleftrightarrow$  ( $\forall i. x\$i = y\$i$ )
  by (simp add: vec_nth_inject [symmetric] fun_eq_iff)
```


lemma *vec_lambda_beta* [simp]: *vec_lambda g \$ i = g i*
by (*simp add: vec_lambda_inverse*)

lemma *vec_lambda_unique*: $(\forall i. f\$i = g\ i) \longleftrightarrow \text{vec\_lambda } g = f$
by (*auto simp add: vec_eq_iff*)

lemma *vec_lambda_eta* [simp]: $(\chi\ i. (g\$i)) = g$
by (*simp add: vec_eq_iff*)

1.8.2 Cardinality of vectors

instance *vec* :: (*finite*, *finite*) *finite*

proof

show *finite* (*UNIV* :: ('a, 'b) *vec set*)

proof (*subst bij_betw_finite*)

show *bij_betw vec_nth UNIV* (*Pi* (*UNIV* :: 'b *set*) ($\lambda\_.$ *UNIV* :: 'a *set*))

by (*intro bij_betwI[of _ _ _ vec_lambda]*) (*auto simp: vec_eq_iff*)

have *finite* (*PiE* (*UNIV* :: 'b *set*) ($\lambda\_.$ *UNIV* :: 'a *set*))

by (*intro finite_PiE*) *auto*

also have (*PiE* (*UNIV* :: 'b *set*) ($\lambda\_.$ *UNIV* :: 'a *set*)) = *Pi UNIV* ($\lambda\_.$ *UNIV*)

by *auto*

finally show *finite*

qed

qed

lemma *countable_PiE*:

finite I $\implies (\bigwedge i. i \in I \implies \text{countable } (F\ i)) \implies \text{countable } (Pi_E\ I\ F)$

by (*induct I arbitrary: F rule: finite_induct*) (*auto simp: PiE_insert_eq*)

instance *vec* :: (*countable*, *finite*) *countable*

proof

have *countable* (*UNIV* :: ('a, 'b) *vec set*)

proof (*rule countableI_bij2*)

show *bij_betw vec_nth UNIV* (*Pi* (*UNIV* :: 'b *set*) ($\lambda\_.$ *UNIV* :: 'a *set*))

by (*intro bij_betwI[of _ _ _ vec_lambda]*) (*auto simp: vec_eq_iff*)

have *countable* (*PiE* (*UNIV* :: 'b *set*) ($\lambda\_.$ *UNIV* :: 'a *set*))

by (*intro countable_PiE*) *auto*

also have (*PiE* (*UNIV* :: 'b *set*) ($\lambda\_.$ *UNIV* :: 'a *set*)) = *Pi UNIV* ($\lambda\_.$ *UNIV*)

by *auto*

finally show *countable*

qed

thus $\exists t::('a, 'b)\ \text{vec} \Rightarrow \text{nat. inj } t$

by (*auto elim!: countableE*)

qed

lemma *infinite_UNIV_vec*:

assumes *infinite* (*UNIV* :: 'a *set*)

shows *infinite* (*UNIV* :: ('a[^]'b) *set*)

proof (*subst bij_betw_finite*)

```

show bij_betw vec_nth UNIV (Pi (UNIV :: 'b set) (λ_. UNIV :: 'a set))
  by (intro bij_betwI[of _ _ _ vec_lambda]) (auto simp: vec_eq_iff)
have infinite (PiE (UNIV :: 'b set) (λ_. UNIV :: 'a set)) (is infinite ?A)
proof
  assume finite ?A
  hence finite ((λf. f undefined) ' ?A)
  by (rule finite_imageI)
  also have (λf. f undefined) ' ?A = UNIV
  by auto
  finally show False
  using ⟨infinite (UNIV :: 'a set)⟩ by contradiction
qed
also have ?A = Pi UNIV (λ_. UNIV)
  by auto
  finally show infinite (Pi (UNIV :: 'b set) (λ_. UNIV :: 'a set)) .
qed

proposition CARD_vec [simp]:
  CARD('a ^ 'b) = CARD('a) ^ CARD('b)
proof (cases finite (UNIV :: 'a set))
  case True
  show ?thesis
  proof (subst bij_betw_same_card)
    show bij_betw vec_nth UNIV (Pi (UNIV :: 'b set) (λ_. UNIV :: 'a set))
      by (intro bij_betwI[of _ _ _ vec_lambda]) (auto simp: vec_eq_iff)
    have CARD('a) ^ CARD('b) = card (PiE (UNIV :: 'b set) (λ_. UNIV :: 'a
set))
      (is _ = card ?A)
    by (subst card_PiE) (auto)
    also have ?A = Pi UNIV (λ_. UNIV)
    by auto
    finally show card ... = CARD('a) ^ CARD('b) ..
  qed
qed (simp_all add: infinite_UNIV_vec)

lemma countable_vector:
  fixes B:: 'n::finite ⇒ 'a set
  assumes ∧i. countable (B i)
  shows countable {V. ∀ i::'n::finite. V $ i ∈ B i}
proof -
  have f ∈ ($) ' {V. ∀ i. V $ i ∈ B i} if f ∈ PiE UNIV B for f
  proof -
    have ∃ W. (∀ i. W $ i ∈ B i) ∧ ($) W = f
    by (metis that PiE_iff UNIV_I vec_lambda_inverse)
    then show f ∈ ($) ' {v. ∀ i. v $ i ∈ B i}
    by blast
  qed
  then have PiE UNIV B = vec_nth ' {V. ∀ i::'n. V $ i ∈ B i}
  by blast

```

```

then have countable (vec_nth ‘ {V.  $\forall i. V \$ i \in B i$ })
by (metis finite_class.finite_UNIV countable_PiE assms)
then have countable (vec_lambda ‘ vec_nth ‘ {V.  $\forall i. V \$ i \in B i$ })
by auto
then show ?thesis
by (simp add: image_comp o_def vec_nth_inverse)
qed

```

1.8.3 Group operations and class instances

```

instantiation vec :: (zero, finite) zero
begin
  definition 0  $\equiv$  ( $\chi i. 0$ )
  instance ..
end

```

```

instantiation vec :: (plus, finite) plus
begin
  definition (+)  $\equiv$  ( $\lambda x y. (\chi i. x \$ i + y \$ i)$ )
  instance ..
end

```

```

instantiation vec :: (minus, finite) minus
begin
  definition (−)  $\equiv$  ( $\lambda x y. (\chi i. x \$ i - y \$ i)$ )
  instance ..
end

```

```

instantiation vec :: (uminus, finite) uminus
begin
  definition uminus  $\equiv$  ( $\lambda x. (\chi i. - (x \$ i))$ )
  instance ..
end

```

```

lemma zero_index [simp]: 0 $ i = 0
unfolding zero_vec_def by simp

```

```

lemma vector_add_component [simp]: (x + y)$i = x$i + y$i
unfolding plus_vec_def by simp

```

```

lemma vector_minus_component [simp]: (x − y)$i = x$i − y$i
unfolding minus_vec_def by simp

```

```

lemma vector_uminus_component [simp]: (− x)$i = − (x$i)
unfolding uminus_vec_def by simp

```

```

instance vec :: (semigroup_add, finite) semigroup_add
by standard (simp add: vec_eq_iff add.assoc)

```

```

instance vec :: (ab_semigroup_add, finite) ab_semigroup_add
  by standard (simp add: vec_eq_iff add.commute)

instance vec :: (monoid_add, finite) monoid_add
  by standard (simp_all add: vec_eq_iff)

instance vec :: (comm_monoid_add, finite) comm_monoid_add
  by standard (simp add: vec_eq_iff)

instance vec :: (cancel_semigroup_add, finite) cancel_semigroup_add
  by standard (simp_all add: vec_eq_iff)

instance vec :: (cancel_ab_semigroup_add, finite) cancel_ab_semigroup_add
  by standard (simp_all add: vec_eq_iff diff_diff_eq)

instance vec :: (cancel_comm_monoid_add, finite) cancel_comm_monoid_add
  ..

instance vec :: (group_add, finite) group_add
  by standard (simp_all add: vec_eq_iff)

instance vec :: (ab_group_add, finite) ab_group_add
  by standard (simp_all add: vec_eq_iff)

```

1.8.4 Basic componentwise operations on vectors

```

instantiation vec :: (times, finite) times
begin

definition (*) ≡ (λ x y. (χ i. (x$i) * (y$i)))
instance ..

end

instantiation vec :: (one, finite) one
begin

definition 1 ≡ (χ i. 1)
instance ..

end

instantiation vec :: (ord, finite) ord
begin

definition x ≤ y ↔ (∀ i. x$i ≤ y$i)
definition x < (y::'a^'b) ↔ x ≤ y ∧ ¬ y ≤ x
instance ..

```

end

The ordering on one-dimensional vectors is linear.

```

instance vec :: (order, finite) order
  by standard (auto simp: less_eq_vec_def less_vec_def vec_eq_iff
    intro: order.trans order.antisym order.strict_implies_order)

instance vec :: (linorder, CARD_1) linorder
proof
  obtain a :: 'b where all:  $\bigwedge P. (\forall i. P\ i) \longleftrightarrow P\ a$ 
  proof –
    have CARD ('b) = 1 by (rule CARD_1)
    then obtain b :: 'b where UNIV = {b} by (auto iff: card_Suc_eq)
    then have  $\bigwedge P. (\forall i \in UNIV. P\ i) \longleftrightarrow P\ b$  by auto
    then show thesis by (auto intro: that)
  qed
  fix x y :: 'a'b::CARD_1
  note [simp] = less_eq_vec_def less_vec_def all vec_eq_iff field_simps
  show x ≤ y ∨ y ≤ x by auto
qed

```

Constant Vectors

definition $vec\ x = (\chi\ i. x)$

Also the scalar-vector multiplication.

definition $vector\_scalar\_mult :: 'a :: times \Rightarrow 'a \wedge 'n \Rightarrow 'a \wedge 'n$ (**infixl** *s 70)
where $c *s\ x = (\chi\ i. c * (x\ \$\ i))$

scalar product

definition $scalar\_product :: 'a :: semiring_1 \wedge 'n \Rightarrow 'a \wedge 'n \Rightarrow 'a$ **where**
 $scalar\_product\ v\ w = (\sum\ i \in UNIV. v\ \$\ i * w\ \$\ i)$

1.8.5 Real vector space

instantiation $vec :: (real\_vector, finite)\ real\_vector$
begin

definition $scaleR \equiv (\lambda\ r\ x. (\chi\ i. scaleR\ r\ (x\ \$\ i)))$

lemma $vector\_scaleR\_component$ [simp]: $(scaleR\ r\ x)\ \$\ i = scaleR\ r\ (x\ \$\ i)$
unfolding $scaleR\_vec\_def$ **by** simp

instance
by standard (simp_all add: vec_eq_iff scaleR_left_distrib scaleR_right_distrib)

end

1.8.6 Topological space

instantiation *vec* :: (*topological_space*, *finite*) *topological_space*
begin

definition [*code del*]:

$$\text{open } (S :: ('a \wedge 'b) \text{ set}) \longleftrightarrow$$

$$(\forall x \in S. \exists A. (\forall i. \text{open } (A \ i) \wedge x \$ i \in A \ i) \wedge$$

$$(\forall y. (\forall i. y \$ i \in A \ i) \longrightarrow y \in S))$$

instance proof
show *open* (*UNIV* :: ('*a* ^ '*b*) *set*)
unfolding *open_vec_def* **by** *auto*
next
fix *S T* :: ('*a* ^ '*b*) *set*
assume *open S open T* **thus** *open (S ∩ T)*
unfolding *open_vec_def*
apply *clarify*
apply (*drule* (1) *bspec*)
apply (*clarify*, *rename_tac Sa Ta*)
apply (*rule_tac* *x=λi. Sa i ∩ Ta i* **in** *exI*)
apply (*simp add: open_Int*)
done
next
fix *K* :: ('*a* ^ '*b*) *set set*
assume $\forall S \in K. \text{open } S$ **thus** *open (⋃ K)*
unfolding *open_vec_def*
by (*metis Union_iff*)
qed
end

lemma *open_vector_box*: $\forall i. \text{open } (S \ i) \implies \text{open } \{x. \forall i. x \$ i \in S \ i\}$
unfolding *open_vec_def* **by** *auto*

lemma *open_vimage_vec_nth*: *open S* $\implies \text{open } ((\lambda x. x \$ i) - ' S)$
unfolding *open_vec_def*
apply *clarify*
apply (*rule_tac* *x=λk. if k = i then S else UNIV* **in** *exI*, *simp*)
done

lemma *closed_vimage_vec_nth*: *closed S* $\implies \text{closed } ((\lambda x. x \$ i) - ' S)$
unfolding *closed_open vimage_Compl* [*symmetric*]
by (*rule open_vimage_vec_nth*)

lemma *closed_vector_box*: $\forall i. \text{closed } (S \ i) \implies \text{closed } \{x. \forall i. x \$ i \in S \ i\}$
proof –
have $\{x. \forall i. x \$ i \in S \ i\} = (\bigcap i. (\lambda x. x \$ i) - ' S \ i)$ **by** *auto*
thus $\forall i. \text{closed } (S \ i) \implies \text{closed } \{x. \forall i. x \$ i \in S \ i\}$
by (*simp add: closed_INT closed_vimage_vec_nth*)

qed

lemma *tendsto_vec_nth* [*tendsto_intros*]:
 assumes $((\lambda x. f\ x) \longrightarrow a)\ net$
 shows $((\lambda x. f\ x\ \$\ i) \longrightarrow a\ \$\ i)\ net$
proof (rule *topological_tendstoI*)
 fix *S* **assume** *open S a \$ i ∈ S*
 then have *open ((λy. y \$ i) - ' S) a ∈ ((λy. y \$ i) - ' S)*
 by (*simp_all add: open_vimage_vec_nth*)
 with *assms* **have** *eventually (λx. f x ∈ (λy. y \$ i) - ' S) net*
 by (rule *topological_tendstoD*)
 then **show** *eventually (λx. f x \$ i ∈ S) net*
 by *simp*
 qed

lemma *isCont_vec_nth* [*simp*]: *isCont f a* $\implies$ *isCont (λx. f x \$ i) a*
 unfolding *isCont_def* **by** (rule *tendsto_vec_nth*)

lemma *vec_tendstoI*:
 assumes $\bigwedge i. ((\lambda x. f\ x\ \$\ i) \longrightarrow a\ \$\ i)\ net$
 shows $((\lambda x. f\ x) \longrightarrow a)\ net$
proof (rule *topological_tendstoI*)
 fix *S* **assume** *open S and a ∈ S*
 then obtain *A* **where** *A: ∧i. open (A i) ∧i. a \$ i ∈ A i*
 and *S: ∧y. ∀i. y \$ i ∈ A i $\implies$ y ∈ S*
 unfolding *open_vec_def* **by** *metis*
 have $\bigwedge i. eventually (\lambda x. f\ x\ \$\ i \in A\ i)\ net$
 using *assms A* **by** (rule *topological_tendstoD*)
 hence *eventually (λx. ∀i. f x \$ i ∈ A i) net*
 by (rule *eventually_all_finite*)
 thus *eventually (λx. f x ∈ S) net*
 by (rule *eventually_mono, simp add: S*)
 qed

lemma *tendsto_vec_lambda* [*tendsto_intros*]:
 assumes $\bigwedge i. ((\lambda x. f\ x\ i) \longrightarrow a\ i)\ net$
 shows $((\lambda x. \chi\ i. f\ x\ i) \longrightarrow (\chi\ i. a\ i))\ net$
 using *assms* **by** (*simp add: vec_tendstoI*)

lemma *open_image_vec_nth*: **assumes** *open S* **shows** *open ((λx. x \$ i) - ' S)*
proof (rule *openI*)
 fix *a* **assume** *a ∈ (λx. x \$ i) - ' S*
 then obtain *z* **where** *a = z \$ i and z ∈ S ..*
 then obtain *A* **where** *A: ∀i. open (A i) ∧ z \$ i ∈ A i*
 and *S: ∀y. (∀i. y \$ i ∈ A i) $\longrightarrow$ y ∈ S*
 using $\langle open\ S \rangle$ **unfolding** *open_vec_def* **by** *auto*
 hence *A i ⊆ (λx. x \$ i) - ' S*
 by (*clarsimp, rule_tac x=χ j. if j = i then x else z \$ j in image_eqI,*
simp_all)

```

    hence open (A i) ∧ a ∈ A i ∧ A i ⊆ (λx. x $ i) ‘ S
    using A ⟨a = z $ i⟩ by simp
    then show ∃ T. open T ∧ a ∈ T ∧ T ⊆ (λx. x $ i) ‘ S by - (rule exI)
qed

```

```

instance vec :: (perfect_space, finite) perfect_space
proof
  fix x :: 'a ^ 'b show ¬ open {x}
  proof
    assume open {x}
    hence ∀ i. open ((λx. x $ i) ‘ {x}) by (fast intro: open_image_vec_nth)
    hence ∀ i. open {x $ i} by simp
    thus False by (simp add: not_open_singleton)
  qed
qed

```

1.8.7 Metric space

```

instantiation vec :: (metric_space, finite) dist
begin

```

```

definition
  dist x y = L2_set (λi. dist (x $ i) (y $ i)) UNIV

```

```

instance ..
end

```

```

instantiation vec :: (metric_space, finite) uniformity_dist
begin

```

```

definition [code del]:
  (uniformity :: (('a ^ 'b :: _) × ('a ^ 'b :: _)) filter) =
    (INF e ∈ {0 <..}. principal {(x, y). dist x y < e})

```

```

instance
  by standard (rule uniformity_vec_def)
end

```

```

declare uniformity_Abort[where 'a='a :: metric_space ^ 'b :: finite, code]

```

```

instantiation vec :: (metric_space, finite) metric_space
begin

```

```

proposition dist_vec_nth_le: dist (x $ i) (y $ i) ≤ dist x y
  unfolding dist_vec_def by (rule member_le_L2_set) simp_all

```

```

instance proof
  fix x y :: 'a ^ 'b
  show dist x y = 0 ⟷ x = y

```



```

    unfolding dist_vec_def
  by (simp add: L2_set_eq_0_iff vec_eq_iff)
next
fix x y z :: 'a ^ 'b
show dist x y ≤ dist x z + dist y z
  unfolding dist_vec_def
  apply (rule order_trans [OF L2_set_triangle_ineq])
  apply (simp add: L2_set_mono dist_triangle2)
  done
next
fix S :: ('a ^ 'b) set
have *: open S ⟷ (∀ x ∈ S. ∃ e > 0. ∀ y. dist y x < e ⟶ y ∈ S)
proof
  assume open S show ∀ x ∈ S. ∃ e > 0. ∀ y. dist y x < e ⟶ y ∈ S
  proof
    fix x assume x ∈ S
    obtain A where A: ∀ i. open (A i) ∀ i. x $ i ∈ A i
    and S: ∀ y. (∀ i. y $ i ∈ A i) ⟶ y ∈ S
    using ⟨open S⟩ and ⟨x ∈ S⟩ unfolding open_vec_def by metis
    have ∀ i ∈ UNIV. ∃ r > 0. ∀ y. dist y (x $ i) < r ⟶ y ∈ A i
    using A unfolding open_dist by simp
    hence ∃ r. ∀ i ∈ UNIV. 0 < r i ∧ (∀ y. dist y (x $ i) < r i ⟶ y ∈ A i)
    by (rule finite_set_choice [OF finite])
    then obtain r where r1: ∀ i. 0 < r i
    and r2: ∀ i y. dist y (x $ i) < r i ⟶ y ∈ A i by fast
    have 0 < Min (range r) ∧ (∀ y. dist y x < Min (range r) ⟶ y ∈ S)
    by (simp add: r1 r2 S le_less_trans [OF dist_vec_nth_le])
    thus ∃ e > 0. ∀ y. dist y x < e ⟶ y ∈ S ..
  qed
qed
next
assume *: ∀ x ∈ S. ∃ e > 0. ∀ y. dist y x < e ⟶ y ∈ S show open S
proof (unfold open_vec_def, rule)
  fix x assume x ∈ S
  then obtain e where 0 < e and S: ∀ y. dist y x < e ⟶ y ∈ S
  using * by fast
  define r where [abs_def]: r i = e / sqrt (of_nat CARD('b)) for i :: 'b
  from ⟨0 < e⟩ have r: ∀ i. 0 < r i
  unfolding r_def by simp_all
  from ⟨0 < e⟩ have e: e = L2_set r UNIV
  unfolding r_def by (simp add: L2_set_constant)
  define A where A i = {y. dist (x $ i) y < r i} for i
  have ∀ i. open (A i) ∧ x $ i ∈ A i
  unfolding A_def by (simp add: open_ball r)
  moreover have ∀ y. (∀ i. y $ i ∈ A i) ⟶ y ∈ S
  by (simp add: A_def S dist_vec_def e L2_set_strict_mono dist_commute)
  ultimately show ∃ A. (∀ i. open (A i) ∧ x $ i ∈ A i) ∧
    (∀ y. (∀ i. y $ i ∈ A i) ⟶ y ∈ S) by metis
qed
qed

```

```

show open  $S = (\forall x \in S. \forall_F (x', y) \text{ in } \text{uniformity}. x' = x \longrightarrow y \in S)$ 
  unfolding * eventually_uniformity_metric
  by (simp del: split_paired_All add: dist_vec_def dist_commute)
qed

end

```

```

lemma Cauchy_vec_nth:
  Cauchy ( $\lambda n. X\ n$ )  $\implies$  Cauchy ( $\lambda n. X\ n\ \$\ i$ )
  unfolding Cauchy_def by (fast intro: le_less_trans [OF dist_vec_nth_le])

```

```

lemma vec_CauchyI:
  fixes  $X :: \text{nat} \Rightarrow 'a::\text{metric\_space} \wedge 'n$ 
  assumes  $X: \bigwedge i. \text{Cauchy } (\lambda n. X\ n\ \$\ i)$ 
  shows Cauchy ( $\lambda n. X\ n$ )
proof (rule metric_CauchyI)
  fix  $r :: \text{real}$  assume  $0 < r$ 
  hence  $0 < r / \text{of\_nat } \text{CARD}('n)$  (is  $0 < ?s$ ) by simp
  define  $N$  where  $N\ i = (\text{LEAST } N. \forall m \geq N. \forall n \geq N. \text{dist } (X\ m\ \$\ i) (X\ n\ \$\ i) < ?s)$ 
  for  $i$ 
  define  $M$  where  $M = \text{Max } (\text{range } N)$ 
  have  $\bigwedge i. \exists N. \forall m \geq N. \forall n \geq N. \text{dist } (X\ m\ \$\ i) (X\ n\ \$\ i) < ?s$ 
    using  $X\ \langle 0 < ?s \rangle$  by (rule metric_CauchyD)
  hence  $\bigwedge i. \forall m \geq N\ i. \forall n \geq N\ i. \text{dist } (X\ m\ \$\ i) (X\ n\ \$\ i) < ?s$ 
    unfolding  $N\_def$  by (rule LeastI_ex)
  hence  $M: \bigwedge i. \forall m \geq M. \forall n \geq M. \text{dist } (X\ m\ \$\ i) (X\ n\ \$\ i) < ?s$ 
    unfolding  $M\_def$  by simp
  {
    fix  $m\ n :: \text{nat}$ 
    assume  $M \leq m\ M \leq n$ 
    have  $\text{dist } (X\ m) (X\ n) = L2\_set\ (\lambda i. \text{dist } (X\ m\ \$\ i) (X\ n\ \$\ i))\ \text{UNIV}$ 
      unfolding dist_vec_def ..
    also have  $\dots \leq \text{sum } (\lambda i. \text{dist } (X\ m\ \$\ i) (X\ n\ \$\ i))\ \text{UNIV}$ 
      by (rule L2_set_le_sum [OF zero_le_dist])
    also have  $\dots < \text{sum } (\lambda i::'n. ?s)\ \text{UNIV}$ 
      by (rule sum_strict_mono, simp_all add: M < M ≤ m > < M ≤ n >)
    also have  $\dots = r$ 
      by simp
    finally have  $\text{dist } (X\ m) (X\ n) < r$  .
  }
  hence  $\forall m \geq M. \forall n \geq M. \text{dist } (X\ m) (X\ n) < r$ 
    by simp
  then show  $\exists M. \forall m \geq M. \forall n \geq M. \text{dist } (X\ m) (X\ n) < r$  ..
qed

```

```

instance vec :: (complete_space, finite) complete_space
proof
  fix  $X :: \text{nat} \Rightarrow 'a \wedge 'b$  assume Cauchy  $X$ 
  have  $\bigwedge i. (\lambda n. X\ n\ \$\ i) \longrightarrow \lim (\lambda n. X\ n\ \$\ i)$ 

```

```

    using Cauchy_vec_nth [OF ‹Cauchy X›]
  by (simp add: Cauchy_convergent_iff convergent_LIMSEQ_iff)
hence  $X \longrightarrow \text{vec\_lambda } (\lambda i. \lim (\lambda n. X\ n\ \$\ i))$ 
  by (simp add: vec_tendstoI)
then show convergent X
  by (rule convergentI)
qed

```

1.8.8 Normed vector space

instantiation $\text{vec} :: (\text{real\_normed\_vector}, \text{finite}) \text{ real\_normed\_vector}$
begin

definition $\text{norm } x = L2\_set (\lambda i. \text{norm } (x\$i)) \text{ UNIV}$

definition $\text{sgn } (x::'^a\text{ } ^b) = \text{scaleR } (\text{inverse } (\text{norm } x))\ x$

instance proof

```

  fix a :: real and x y :: '^a ^ ^b
  show  $\text{norm } x = 0 \longleftrightarrow x = 0$ 
    unfolding norm_vec_def
    by (simp add: L2_set_eq_0_iff vec_eq_iff)
  show  $\text{norm } (x + y) \leq \text{norm } x + \text{norm } y$ 
    unfolding norm_vec_def
    apply (rule order_trans [OF _ L2_set_triangle_ineq])
    apply (simp add: L2_set_mono norm_triangle_ineq)
    done
  show  $\text{norm } (\text{scaleR } a\ x) = |a| * \text{norm } x$ 
    unfolding norm_vec_def
    by (simp add: L2_set_right_distrib)
  show  $\text{sgn } x = \text{scaleR } (\text{inverse } (\text{norm } x))\ x$ 
    by (rule sgn_vec_def)
  show  $\text{dist } x\ y = \text{norm } (x - y)$ 
    unfolding dist_vec_def norm_vec_def
    by (simp add: dist_norm)
qed

```

end

lemma $\text{norm\_nth\_le}: \text{norm } (x\ \$\ i) \leq \text{norm } x$

```

unfolding norm_vec_def
by (rule member_le_L2_set) simp_all

```

lemma $\text{norm\_le\_componentwise\_cart}$:

```

  fixes x :: '^a::real_normed_vector ^ ^n
  assumes  $\bigwedge i. \text{norm}(x\$i) \leq \text{norm}(y\$i)$ 
  shows  $\text{norm } x \leq \text{norm } y$ 
  unfolding norm_vec_def
  by (rule L2_set_mono) (auto simp: assms)

```

lemma *component_le_norm_cart*: $|x\$i| \leq \text{norm } x$
by (*metis norm_nth_le real_norm_def*)

lemma *norm_bound_component_le_cart*: $\text{norm } x \leq e \implies |x\$i| \leq e$
by (*metis component_le_norm_cart order_trans*)

lemma *norm_bound_component_lt_cart*: $\text{norm } x < e \implies |x\$i| < e$
by (*metis component_le_norm_cart le_less_trans*)

lemma *norm_le_l1_cart*: $\text{norm } x \leq \text{sum}(\lambda i. |x\$i|)$ *UNIV*
by (*simp add: norm_vec_def L2_set_le_sum*)

lemma *bounded_linear_vec_nth*[*intro*]: *bounded_linear* ($\lambda x. x \$ i$)
proof
show $\exists K. \forall x. \text{norm } (x \$ i) \leq \text{norm } x * K$
by (*metis mult.commute mult.left_neutral norm_nth_le*)
qed *auto*

instance *vec* :: (*banach*, *finite*) *banach* ..

1.8.9 Inner product space

instantiation *vec* :: (*real_inner*, *finite*) *real_inner*
begin

definition *inner* $x \ y = \text{sum } (\lambda i. \text{inner } (x\$i) (y\$i))$ *UNIV*

instance **proof**
fix $r :: \text{real}$ **and** $x \ y \ z :: 'a \wedge 'b$
show $\text{inner } x \ y = \text{inner } y \ x$
unfolding *inner_vec_def*
by (*simp add: inner_commute*)
show $\text{inner } (x + y) \ z = \text{inner } x \ z + \text{inner } y \ z$
unfolding *inner_vec_def*
by (*simp add: inner_add_left sum.distrib*)
show $\text{inner } (\text{scaleR } r \ x) \ y = r * \text{inner } x \ y$
unfolding *inner_vec_def*
by (*simp add: sum_distrib_left*)
show $0 \leq \text{inner } x \ x$
unfolding *inner_vec_def*
by (*simp add: sum_nonneg*)
show $\text{inner } x \ x = 0 \longleftrightarrow x = 0$
unfolding *inner_vec_def*
by (*simp add: vec_eq_iff sum_nonneg_eq_0_iff*)
show $\text{norm } x = \text{sqrt } (\text{inner } x \ x)$
unfolding *inner_vec_def norm_vec_def L2_set_def*
by (*simp add: power2_norm_eq_inner*)
qed

end

1.8.10 Euclidean space

Vectors pointing along a single axis.

definition $axis\ k\ x = (\chi\ i.\ \text{if } i = k \text{ then } x \text{ else } 0)$

lemma $axis\_nth\ [simp]:\ axis\ i\ x\ \$\ i = x$
unfolding $axis\_def$ **by** $simp$

lemma $axis\_eq\_axis: axis\ i\ x = axis\ j\ y \longleftrightarrow x = y \wedge i = j \vee x = 0 \wedge y = 0$
unfolding $axis\_def\ vec\_eq\_iff$ **by** $auto$

lemma $inner\_axis\_axis:$
 $inner\ (axis\ i\ x)\ (axis\ j\ y) = (\text{if } i = j \text{ then } inner\ x\ y \text{ else } 0)$
by $(simp\ add: inner\_vec\_def\ axis\_def\ sum.neutral\ sum.remove\ [of\_j])$

lemma $inner\_axis: inner\ x\ (axis\ i\ y) = inner\ (x\ \$\ i)\ y$
by $(simp\ add: inner\_vec\_def\ axis\_def\ sum.remove\ [where\ x=i])$

lemma $inner\_axis': inner(axis\ i\ y)\ x = inner\ y\ (x\ \$\ i)$
by $(simp\ add: inner\_axis\ inner\_commute)$

instantiation $vec :: (euclidean\_space, finite)\ euclidean\_space$
begin

definition $Basis = (\bigcup i.\ \bigcup u \in Basis.\ \{axis\ i\ u\})$

instance **proof**
show $(Basis :: ('a \wedge 'b)\ set) \neq \{\}$
unfolding $Basis\_vec\_def$ **by** $simp$
next
show $finite\ (Basis :: ('a \wedge 'b)\ set)$
unfolding $Basis\_vec\_def$ **by** $simp$
next
fix $u\ v :: 'a \wedge 'b$
assume $u \in Basis$ **and** $v \in Basis$
thus $inner\ u\ v = (\text{if } u = v \text{ then } 1 \text{ else } 0)$
unfolding $Basis\_vec\_def$
by $(auto\ simp\ add: inner\_axis\_axis\ axis\_eq\_axis\ inner\_Basis)$
next
fix $x :: 'a \wedge 'b$
show $(\forall u \in Basis.\ inner\ x\ u = 0) \longleftrightarrow x = 0$
unfolding $Basis\_vec\_def$
by $(simp\ add: inner\_axis\ euclidean\_all\_zero\_iff\ vec\_eq\_iff)$
qed

proposition $DIM\_cart\ [simp]: DIM('a \wedge 'b) = CARD('b) * DIM('a)$

proof –
have $\text{card} (\bigcup i::'b. \bigcup u::'a \in \text{Basis}. \{ \text{axis } i \ u \}) = (\sum i::'b \in \text{UNIV}. \text{card} (\bigcup u::'a \in \text{Basis}. \{ \text{axis } i \ u \}))$
by (rule card_UN_disjoint) (auto simp: axis_eq_axis)
also have $\dots = \text{CARD}('b) * \text{DIM}('a)$
by (subst card_UN_disjoint) (auto simp: axis_eq_axis)
finally show ?thesis
by (simp add: Basis_vec_def)
qed

end

lemma norm_axis_1 [simp]: $\text{norm} (\text{axis } m \ (1::\text{real})) = 1$
by (simp add: inner_axis' norm_eq_1)

lemma sum_norm_allsubsets_bound_cart:
fixes $f:: 'a \Rightarrow \text{real}^n$
assumes fP : finite P **and** fPs : $\bigwedge Q. Q \subseteq P \implies \text{norm} (\text{sum } f \ Q) \leq e$
shows $\text{sum} (\lambda x. \text{norm} (f \ x)) \ P \leq 2 * \text{real } \text{CARD}('n) * e$
using sum_norm_allsubsets_bound[OF assms]
by simp

lemma cart_eq_inner_axis: $a \ \$ \ i = \text{inner } a \ (\text{axis } i \ 1)$
by (simp add: inner_axis)

lemma axis_eq_0_iff [simp]:
shows $\text{axis } m \ x = 0 \longleftrightarrow x = 0$
by (simp add: axis_def vec_eq_iff)

lemma axis_in_Basis_iff [simp]: $\text{axis } i \ a \in \text{Basis} \longleftrightarrow a \in \text{Basis}$
by (auto simp: Basis_vec_def axis_eq_axis)

Mapping each basis element to the corresponding finite index

definition axis_index :: $('a::\text{comm\_ring\_1})^n \Rightarrow 'n$ **where** $\text{axis\_index } v \equiv \text{SOME } i. v = \text{axis } i \ 1$

lemma axis_inverse:
fixes $v:: \text{real}^n$
assumes $v \in \text{Basis}$
shows $\exists i. v = \text{axis } i \ 1$

proof –
have $v \in (\bigcup n. \bigcup r \in \text{Basis}. \{ \text{axis } n \ r \})$
using assms Basis_vec_def **by** blast
then show ?thesis
by (force simp add: vec_eq_iff)
qed

lemma axis_index:
fixes $v:: \text{real}^n$

```

assumes  $v \in \text{Basis}$ 
shows  $v = \text{axis } (\text{axis\_index } v) \ 1$ 
by (metis (mono_tags) assms axis_inverse axis_index_def someI_ex)

```

```

lemma axis_index_axis [simp]:
  fixes  $UU :: \text{real}^n$ 
  shows  $(\text{axis\_index } (\text{axis } u \ 1 :: \text{real}^n)) = (u :: ^n)$ 
  by (simp add: axis_eq_axis axis_index_def)

```

1.8.11 A naive proof procedure to lift really trivial arithmetic stuff from the basis of the vector space

```

lemma sum_cong_aux:
   $(\bigwedge x. x \in A \implies f \ x = g \ x) \implies \text{sum } f \ A = \text{sum } g \ A$ 
  by (auto intro: sum.cong)

```

```

hide_fact (open) sum_cong_aux

```

```

method_setup vector = ⟨
  let
    val ss1 =
      simpset_of (put_simpset HOL_basic_ss context
        addsimps [@{thm sum.distrib} RS sym,
          @{thm sum.subtractf} RS sym, @{thm sum_distrib_left},
          @{thm sum_distrib_right}, @{thm sum_negf} RS sym])
    val ss2 =
      simpset_of (context addsimps
        [@{thm plus_vec_def}, @{thm times_vec_def},
          @{thm minus_vec_def}, @{thm uminus_vec_def},
          @{thm one_vec_def}, @{thm zero_vec_def}, @{thm vec_def},
          @{thm scaleR_vec_def}, @{thm vector_scalar_mult_def}])
    fun vector_arith_tac ctxt ths =
      simp_tac (put_simpset ss1 ctxt)
      THEN' (fn i => resolve_tac ctxt @{thms Finite_Cartesian_Product.sum_cong_aux})
    i
      ORELSE resolve_tac ctxt @{thms sum.neutral} i
      ORELSE simp_tac (put_simpset HOL_basic_ss ctxt addsimps [@{thm
vec_eq_iff}]) i)
    (* THEN' TRY o clarify_tac HOL_cs THEN' (TRY o rtac @{thm iffI}) *)
    THEN' asm_full_simp_tac (put_simpset ss2 ctxt addsimps ths)
  in
    Attrib.thms >> (fn ths => fn ctxt => SIMPLE_METHOD' (vector_arith_tac
ctxt ths))
  end
  ⟨ lift trivial vector statements to real arith statements

```

```

lemma vec_0 [simp]:  $\text{vec } 0 = 0$  by vector
lemma vec_1 [simp]:  $\text{vec } 1 = 1$  by vector

```

lemma *vec_inj[simp]*: $\text{vec } x = \text{vec } y \longleftrightarrow x = y$ **by** *vector*

lemma *vec_in_image_vec*: $\text{vec } x \in (\text{vec } 'S) \longleftrightarrow x \in S$ **by** *auto*

lemma *vec_add*: $\text{vec}(x + y) = \text{vec } x + \text{vec } y$ **by** *vector*

lemma *vec_sub*: $\text{vec}(x - y) = \text{vec } x - \text{vec } y$ **by** *vector*

lemma *vec_cmul*: $\text{vec}(c * x) = c *s \text{vec } x$ **by** *vector*

lemma *vec_neg*: $\text{vec}(-x) = - \text{vec } x$ **by** *vector*

lemma *vec_scaleR*: $\text{vec}(c * x) = c *_R \text{vec } x$
by *vector*

lemma *vec_sum*:
 assumes *finite S*
 shows $\text{vec}(\text{sum } f \ S) = \text{sum } (\text{vec } \circ f) \ S$
 using *assms*
proof *induct*
 case *empty*
 then show *?case* **by** *simp*
next
 case *insert*
 then show *?case* **by** (*auto simp add: vec_add*)
qed

Obvious "component-pushing".

lemma *vec_component [simp]*: $\text{vec } x \$ i = x$
by *vector*

lemma *vector_mult_component [simp]*: $(x * y) \$ i = x \$ i * y \$ i$
by *vector*

lemma *vector_smult_component [simp]*: $(c *s y) \$ i = c * (y \$ i)$
by *vector*

lemma *cond_component*: $(\text{if } b \text{ then } x \text{ else } y) \$ i = (\text{if } b \text{ then } x \$ i \text{ else } y \$ i)$ **by** *vector*

lemmas *vector_component* =
 vec_component vector_add_component vector_mult_component
 vector_smult_component vector_minus_component vector_uminus_component
 vector_scaleR_component cond_component

1.8.12 Some frequently useful arithmetic lemmas over vectors

instance *vec* :: (*semigroup_mult, finite*) *semigroup_mult*
by *standard (vector mult.assoc)*

instance *vec* :: (*monoid_mult, finite*) *monoid_mult*
by *standard vector+*


```

instance vec :: (ab_semigroup_mult, finite) ab_semigroup_mult
  by standard (vector_mult.commute)

instance vec :: (comm_monoid_mult, finite) comm_monoid_mult
  by standard vector

instance vec :: (semiring, finite) semiring
  by standard (vector_field_simps)+

instance vec :: (semiring_0, finite) semiring_0
  by standard (vector_field_simps)+
instance vec :: (semiring_1, finite) semiring_1
  by standard vector
instance vec :: (comm_semiring, finite) comm_semiring
  by standard (vector_field_simps)+

instance vec :: (comm_semiring_0, finite) comm_semiring_0 ..
instance vec :: (semiring_0_cancel, finite) semiring_0_cancel ..
instance vec :: (comm_semiring_0_cancel, finite) comm_semiring_0_cancel ..
instance vec :: (ring, finite) ring ..
instance vec :: (semiring_1_cancel, finite) semiring_1_cancel ..
instance vec :: (comm_semiring_1, finite) comm_semiring_1 ..

instance vec :: (ring_1, finite) ring_1 ..

instance vec :: (real_algebra, finite) real_algebra
  by standard (simp_all add: vec_eq_iff)

instance vec :: (real_algebra_1, finite) real_algebra_1 ..

lemma of_nat_index: (of_nat n :: 'a::semiring_1 ^ 'n) $ i = of_nat n
proof (induct n)
  case 0
  then show ?case by vector
next
  case Suc
  then show ?case by vector
qed

lemma one_index [simp]: (1 :: 'a :: one ^ 'n) $ i = 1
  by vector

lemma neg_one_index [simp]: (- 1 :: 'a :: {one, uminus} ^ 'n) $ i = - 1
  by vector

instance vec :: (semiring_char_0, finite) semiring_char_0
proof
  fix m n :: nat

```

```

show inj (of_nat :: nat  $\Rightarrow$  'a ^ 'b)
  by (auto intro!: injI simp add: vec_eq_iff of_nat_index)
qed

instance vec :: (numeral, finite) numeral ..
instance vec :: (semiring_numeral, finite) semiring_numeral ..

lemma numeral_index [simp]: numeral w $ i = numeral w
  by (induct w) (simp_all only: numeral.simps vector_add_component one_index)

lemma neg_numeral_index [simp]: - numeral w $ i = - numeral w
  by (simp only: vector_uminus_component numeral_index)

instance vec :: (comm_ring_1, finite) comm_ring_1 ..
instance vec :: (ring_char_0, finite) ring_char_0 ..

lemma vector_smult_assoc: a *s (b *s x) = ((a::'a::semigroup_mult) * b) *s x
  by (vector mult.assoc)
lemma vector_sadd_rdistrib: ((a::'a::semiring) + b) *s x = a *s x + b *s x
  by (vector field_simps)
lemma vector_add_ldistrib: (c::'a::semiring) *s (x + y) = c *s x + c *s y
  by (vector field_simps)
lemma vector_smult_lzero[simp]: (0::'a::mult_zero) *s x = 0 by vector
lemma vector_smult_lid[simp]: (1::'a::monoid_mult) *s x = x by vector
lemma vector_ssub_ldistrib: (c::'a::ring) *s (x - y) = c *s x - c *s y
  by (vector field_simps)
lemma vector_smult_rneg: (c::'a::ring) *s -x = -(c *s x) by vector
lemma vector_smult_lneg: -(c::'a::ring) *s x = -(c *s x) by vector
lemma vector_sneg_minus1: -x = (-1::'a::ring_1) *s x by vector
lemma vector_smult_rzero[simp]: c *s 0 = (0::'a::mult_zero ^ 'n) by vector
lemma vector_sub_rdistrib: ((a::'a::ring) - b) *s x = a *s x - b *s x
  by (vector field_simps)

lemma vec_eq[simp]: (vec m = vec n)  $\longleftrightarrow$  (m = n)
  by (simp add: vec_eq_iff)

lemma Vector_Spaces_linear_vec [simp]: Vector_Spaces.linear (*) vector_scalar_mult
  vec
  by unfold_locales (vector algebra_simps)+

lemma vector_mul_eq_0[simp]: (a *s x = 0)  $\longleftrightarrow$  a = (0::'a::idom)  $\vee$  x = 0
  by vector

lemma vector_mul_lcancel[simp]: a *s x = a *s y  $\longleftrightarrow$  a = (0::'a::field)  $\vee$  x = y
  by (metis eq_iff_diff_eq_0 vector_mul_eq_0 vector_ssub_ldistrib)

lemma vector_mul_rcancel[simp]: a *s x = b *s x  $\longleftrightarrow$  (a::'a::field) = b  $\vee$  x = 0
  by (subst eq_iff_diff_eq_0, subst vector_sub_rdistrib [symmetric]) simp

```

```
lemma scalar_mult_eq_scaleR [abs_def]:  $c * s \ x = c *_R x$ 
  unfolding scaleR_vec_def vector_scalar_mult_def by simp
```

```
lemma dist_mul[simp]:  $\text{dist } (c * s \ x) \ (c * s \ y) = |c| * \text{dist } x \ y$ 
  unfolding dist_norm scalar_mult_eq_scaleR
  unfolding scaleR_right_diff_distrib[symmetric] by simp
```

```
lemma sum_component [simp]:
  fixes f:: 'a  $\Rightarrow$  ('b::comm_monoid_add)  $\wedge^n$ 
  shows (sum f S)$i = sum ( $\lambda x. (f \ x) \$ i$ ) S
proof (cases finite S)
  case True
  then show ?thesis by induct simp_all
next
  case False
  then show ?thesis by simp
qed
```

```
lemma sum_eq:  $\text{sum } f \ S = (\chi \ i. \text{sum } (\lambda x. (f \ x) \$ i) \ S)$ 
  by (simp add: vec_eq_iff)
```

```
lemma sum_cmul:
  fixes f:: 'c  $\Rightarrow$  ('a::semiring_1)  $\wedge^n$ 
  shows  $\text{sum } (\lambda x. c * s \ f \ x) \ S = c * s \ \text{sum } f \ S$ 
  by (simp add: vec_eq_iff sum_distrib_left)
```

```
lemma linear_vec [simp]: linear vec
  using Vector_Spaces_linear_vec
  by unfold_locales (vector_algebra_simps)+
```

1.8.13 Matrix operations

Matrix notation. NB: an $M \times N$ matrix is of type $((a, 'n) \text{ vec}, 'm) \text{ vec}$, not $((a, 'm) \text{ vec}, 'n) \text{ vec}$

definition $\text{map\_matrix}::(a \Rightarrow b) \Rightarrow ((a, 'i::\text{finite}) \text{ vec}, 'j::\text{finite}) \text{ vec} \Rightarrow ((b, 'i) \text{ vec}, 'j) \text{ vec}$ **where**
 $\text{map\_matrix } f \ x = (\chi \ i \ j. f \ (x \ \$ \ i \ \$ \ j))$

```
lemma nth_map_matrix[simp]:  $\text{map\_matrix } f \ x \ \$ \ i \ \$ \ j = f \ (x \ \$ \ i \ \$ \ j)$ 
  by (simp add: map_matrix_def)
```

definition $\text{matrix\_matrix\_mult}::(a::\text{semiring\_1}) \wedge^n \wedge^m \Rightarrow a \wedge^p \wedge^n \Rightarrow a \wedge^{p \wedge m}$
 (infixl ** 70)
where $m ** m' == (\chi \ i \ j. \text{sum } (\lambda k. ((m \$ i) \$ k) * ((m' \$ k) \$ j)) \ (UNIV :: 'n \text{ set}))$
 $:: a \wedge^p \wedge^{m'}$

definition $\text{matrix\_vector\_mult}::(a::\text{semiring\_1}) \wedge^n \wedge^m \Rightarrow a \wedge^n \Rightarrow a \wedge^m$
 (infixl *v 70)

where $m * v x \equiv (\chi i. \text{sum } (\lambda j. ((m\$i)\$j) * (x\$j))) (UNIV :: 'n \text{ set})) :: 'a \wedge^m$

definition *vector_matrix_mult* :: $'a \wedge^m \Rightarrow ('a :: \text{semiring\_1}) \wedge^n \wedge^m \Rightarrow 'a \wedge^n$
 $(\text{infixl } v * 70)$

where $v * m == (\chi j. \text{sum } (\lambda i. ((m\$i)\$j) * (v\$i))) (UNIV :: 'm \text{ set})) :: 'a \wedge^n$

definition $(\text{mat} :: 'a :: \text{zero} \Rightarrow 'a \wedge^n \wedge^n) k = (\chi i j. \text{if } i = j \text{ then } k \text{ else } 0)$

definition *transpose* where

$(\text{transpose} :: 'a \wedge^n \wedge^m \Rightarrow 'a \wedge^m \wedge^n) A = (\chi i j. ((A\$j)\$i))$

definition $(\text{row} :: 'm \Rightarrow 'a \wedge^n \wedge^m \Rightarrow 'a \wedge^n) i A = (\chi j. ((A\$i)\$j))$

definition $(\text{column} :: 'n \Rightarrow 'a \wedge^n \wedge^m \Rightarrow 'a \wedge^m) j A = (\chi i. ((A\$i)\$j))$

definition $\text{rows}(A :: 'a \wedge^n \wedge^m) = \{ \text{row } i A \mid i. i \in (UNIV :: 'm \text{ set}) \}$

definition $\text{columns}(A :: 'a \wedge^n \wedge^m) = \{ \text{column } i A \mid i. i \in (UNIV :: 'n \text{ set}) \}$

lemma *times0_left* [simp]: $(0 :: 'a :: \text{semiring\_1}) \wedge^n \wedge^m ** (A :: 'a \wedge^p \wedge^n) = 0$
 by (simp add: *matrix_matrix_mult_def zero_vec_def*)

lemma *times0_right* [simp]: $(A :: 'a :: \text{semiring\_1}) \wedge^n \wedge^m ** (0 :: 'a \wedge^p \wedge^n) = 0$
 by (simp add: *matrix_matrix_mult_def zero_vec_def*)

lemma *mat_0* [simp]: $\text{mat } 0 = 0$ by (*vector mat_def*)

lemma *matrix_add_ldistrib*: $(A ** (B + C)) = (A ** B) + (A ** C)$
 by (*vector matrix_matrix_mult_def sum.distrib[symmetric] field_simps*)

lemma *matrix_mul_lid* [simp]:

fixes $A :: 'a :: \text{semiring\_1} \wedge^m \wedge^n$

shows $\text{mat } 1 ** A = A$

unfolding *matrix_matrix_mult_def mat_def*

by (*auto simp: if_distrib if_distribR sum.delta'[OF finite] cong: if_cong*)

lemma *matrix_mul_rid* [simp]:

fixes $A :: 'a :: \text{semiring\_1} \wedge^m \wedge^n$

shows $A ** \text{mat } 1 = A$

unfolding *matrix_matrix_mult_def mat_def*

by (*auto simp: if_distrib if_distribR sum.delta'[OF finite] cong: if_cong*)

proposition *matrix_mul_assoc*: $A ** (B ** C) = (A ** B) ** C$

apply (*vector matrix_matrix_mult_def sum_distrib_left sum_distrib_right mult.assoc*)

apply (*subst sum.swap*)

apply *simp*

done

proposition *matrix_vector_mul_assoc*: $A * v (B * v x) = (A ** B) * v x$

apply (*vector matrix_matrix_mult_def matrix_vector_mult_def*)

sum_distrib_left sum_distrib_right mult.assoc)

apply (*subst sum.swap*)

apply *simp*

done

proposition *scalar_matrix_assoc*:

fixes $A :: ('a::\text{real\_algebra\_1})^m \times^n$
shows $k *_R (A ** B) = (k *_R A) ** B$
by (*simp add: matrix_matrix_mult_def sum_distrib_left mult_ac vec_eq_iff scaleR_sum_right*)

proposition *matrix_scalar_ac*:

fixes $A :: ('a::\text{real\_algebra\_1})^m \times^n$
shows $A ** (k *_R B) = k *_R A ** B$
by (*simp add: matrix_matrix_mult_def sum_distrib_left mult_ac vec_eq_iff*)

lemma *matrix_vector_mul_lid* [*simp*]: $\text{mat } 1 * v \ x = (x :: 'a::\text{semiring\_1})^1 \times^n$

apply (*vector_matrix_vector_mult_def mat_def*)

apply (*simp add: if_distrib if_distribR cong del: if_weak_cong*)

done

lemma *matrix_transpose_mul*:

$\text{transpose}(A ** B) = \text{transpose } B ** \text{transpose } A :: 'a::\text{comm\_semiring\_1}^{\text{transpose } B} \times^{\text{transpose } A}$
by (*simp add: matrix_matrix_mult_def transpose_def vec_eq_iff mult.commute*)

lemma *matrix_mult_transpose_dot_column*:

shows $\text{transpose } A ** A = (\chi \ i \ j. \text{inner } (\text{column } i \ A) (\text{column } j \ A))$

by (*simp add: matrix_matrix_mult_def vec_eq_iff transpose_def column_def inner_vec_def*)

lemma *matrix_mult_transpose_dot_row*:

shows $A ** \text{transpose } A = (\chi \ i \ j. \text{inner } (\text{row } i \ A) (\text{row } j \ A))$

by (*simp add: matrix_matrix_mult_def vec_eq_iff transpose_def row_def inner_vec_def*)

lemma *matrix_eq*:

fixes $A \ B :: 'a::\text{semiring\_1}^m \times^n$

shows $A = B \longleftrightarrow (\forall x. A * v \ x = B * v \ x) \text{ (is ?lhs } \longleftrightarrow \text{ ?rhs)}$

proof

assume *?rhs*

then show *?lhs*

apply (*subst vec_eq_iff*)

apply (*clarsimp simp add: matrix_vector_mult_def if_distrib if_distribR vec_eq_iff cong: if_cong*)

apply (*erule_tac x=axis ia 1 in allE*)

apply (*erule_tac x=i in allE*)

apply (*auto simp add: if_distrib if_distribR axis_def sum.delta[OF finite] cong del: if_weak_cong*)

done

qed *auto*

lemma *matrix_vector_mul_component*: $(A * v \ x) \$ k = \text{inner } (A \$ k) \ x$

by (*simp add: matrix_vector_mult_def inner_vec_def*)

```

lemma dot_lmul_matrix: inner ((x::real ^) v * A) y = inner x (A * v y)
  apply (simp add: inner_vec_def matrix_vector_mult_def vector_matrix_mult_def
sum_distrib_right sum_distrib_left ac_simps)
  apply (subst sum.swap)
  apply simp
done

```

```

lemma transpose_mat [simp]: transpose (mat n) = mat n
  by (vector transpose_def mat_def)

```

```

lemma transpose_transpose [simp]: transpose(transpose A) = A
  by (vector transpose_def)

```

```

lemma row_transpose [simp]: row i (transpose A) = column i A
  by (simp add: row_def column_def transpose_def vec_eq_iff)

```

```

lemma column_transpose [simp]: column i (transpose A) = row i A
  by (simp add: row_def column_def transpose_def vec_eq_iff)

```

```

lemma rows_transpose [simp]: rows(transpose A) = columns A
  by (auto simp add: rows_def columns_def intro: set_eqI)

```

```

lemma columns_transpose [simp]: columns(transpose A) = rows A
  by (metis transpose_transpose rows_transpose)

```

```

lemma transpose_scalar: transpose (k *R A) = k *R transpose A
  unfolding transpose_def
  by (simp add: vec_eq_iff)

```

```

lemma transpose_iff [iff]: transpose A = transpose B  $\longleftrightarrow$  A = B
  by (metis transpose_transpose)

```

```

lemma matrix_mult_sum:
  (A::'a::comm_semiring_1 ^n ^m) * v x = sum (λi. (x $ i) * s column i A) (UNIV::
'n set)
  by (simp add: matrix_vector_mult_def vec_eq_iff column_def mult.commute)

```

```

lemma vector_componentwise:
  (x::'a::ring_1 ^n) = (χ j. ∑ i ∈ UNIV. (x $ i) * (axis i 1 :: 'a ^n) $ j)
  by (simp add: axis_def if_distrib sum.If_cases vec_eq_iff)

```

```

lemma basis_expansion: sum (λi. (x $ i) * s axis i 1) UNIV = (x::('a::ring_1) ^n)
  by (auto simp add: axis_def vec_eq_iff if_distrib sum.If_cases cong del: if_weak_cong)

```

Correspondence between matrices and linear operators.

```

definition matrix :: ('a::{plus,times, one, zero}) ^m  $\Rightarrow$  'a ^n  $\Rightarrow$  'a ^m ^n
  where matrix f = (χ i j. (f(axis j 1)) $ i)

```

```

lemma matrix_id_mat_1: matrix id = mat 1

```

```

  by (simp add: mat_def matrix_def axis_def)

lemma matrix_scaleR: (matrix ((*R) r)) = mat r
  by (simp add: mat_def matrix_def axis_def if_distrib cong: if_cong)

lemma matrix_vector_mul_linear[intro, simp]: linear (λx. A *v (x::'a::real_algebra_1
  ^_))
  by (simp add: linear_iff matrix_vector_mult_def vec_eq_iff field_simps sum_distrib_left
    sum.distrib scaleR_right.sum)

lemma vector_matrix_left_distrib [algebra_simps]:
  shows (x + y) v * A = x v * A + y v * A
  unfolding vector_matrix_mult_def
  by (simp add: algebra_simps sum.distrib vec_eq_iff)

lemma matrix_vector_right_distrib [algebra_simps]:
  A *v (x + y) = A *v x + A *v y
  by (vector matrix_vector_mult_def sum.distrib distrib_left)

lemma matrix_vector_mult_diff_distrib [algebra_simps]:
  fixes A :: 'a::ring_1 ^n ^m
  shows A *v (x - y) = A *v x - A *v y
  by (vector matrix_vector_mult_def sum_subtractf right_diff_distrib)

lemma matrix_vector_mult_scaleR[algebra_simps]:
  fixes A :: real ^n ^m
  shows A *v (c *R x) = c *R (A *v x)
  using linear_iff matrix_vector_mul_linear by blast

lemma matrix_vector_mult_0_right [simp]: A *v 0 = 0
  by (simp add: matrix_vector_mult_def vec_eq_iff)

lemma matrix_vector_mult_0 [simp]: 0 *v w = 0
  by (simp add: matrix_vector_mult_def vec_eq_iff)

lemma matrix_vector_mult_add_rdistrib [algebra_simps]:
  (A + B) *v x = (A *v x) + (B *v x)
  by (vector matrix_vector_mult_def sum.distrib distrib_right)

lemma matrix_vector_mult_diff_rdistrib [algebra_simps]:
  fixes A :: 'a :: ring_1 ^n ^m
  shows (A - B) *v x = (A *v x) - (B *v x)
  by (vector matrix_vector_mult_def sum_subtractf left_diff_distrib)

lemma matrix_vector_column:
  (A::'a::comm_semiring_1 ^n ^_) *v x = sum (λi. (x $ i) *s ((transpose A) $ i))
  (UNIV:: 'n set)
  by (simp add: matrix_vector_mult_def transpose_def vec_eq_iff mult.commute)

```

1.8.14 Inverse matrices (not necessarily square)

definition

$\text{invertible}(A :: 'a :: \text{semiring\_1}^n{}^m) \longleftrightarrow (\exists A' :: 'a^m{}^n. A ** A' = \text{mat } 1 \wedge A' ** A = \text{mat } 1)$

definition

$\text{matrix\_inv}(A :: 'a :: \text{semiring\_1}^n{}^m) =$
 $(\text{SOME } A' :: 'a^m{}^n. A ** A' = \text{mat } 1 \wedge A' ** A = \text{mat } 1)$

lemma *inj_matrix_vector_mult*:

fixes $A :: 'a :: \text{field}^n{}^m$

assumes *invertible A*

shows *inj ((*v) A)*

by (*metis assms inj_on_inverseI invertible_def matrix_vector_mult_assoc matrix_vector_mult_lid*)

lemma *scalar_invertible*:

fixes $A :: ('a :: \text{real\_algebra\_1})^m{}^n$

assumes $k \neq 0$ **and** *invertible A*

shows *invertible (k *_R A)*

proof –

obtain A' **where** $A ** A' = \text{mat } 1$ **and** $A' ** A = \text{mat } 1$

using *assms unfolding invertible_def* **by** *auto*

with $\langle k \neq 0 \rangle$

have $(k *_{\text{R}} A) ** ((1/k) *_{\text{R}} A') = \text{mat } 1$ $((1/k) *_{\text{R}} A') ** (k *_{\text{R}} A) = \text{mat } 1$

by (*simp_all add: assms matrix_scalar_ac*)

thus *invertible (k *_R A)*

unfolding *invertible_def* **by** *auto*

qed

proposition *scalar_invertible_iff*:

fixes $A :: ('a :: \text{real\_algebra\_1})^m{}^n$

assumes $k \neq 0$ **and** *invertible A*

shows $\text{invertible } (k *_{\text{R}} A) \longleftrightarrow k \neq 0 \wedge \text{invertible } A$

by (*simp add: assms scalar_invertible*)

lemma *vector_transpose_matrix* [*simp*]: $x \text{ v* } \text{transpose } A = A \text{ v* } x$

unfolding *transpose_def vector_matrix_mult_def matrix_vector_mult_def*

by *simp*

lemma *transpose_matrix_vector* [*simp*]: $\text{transpose } A \text{ v* } x = x \text{ v* } A$

unfolding *transpose_def vector_matrix_mult_def matrix_vector_mult_def*

by *simp*

lemma *vector_scalar_commute*:

fixes $A :: 'a :: \{\text{field}\}^m{}^n$

shows $A \text{ v* } (c * s \ x) = c * s \ (A \text{ v* } x)$

by (*simp add: vector_scalar_mult_def matrix_vector_mult_def mult_ac sum_distrib_left*)


```

lemma scalar_vector_matrix_assoc:
  fixes  $k :: 'a::\{field\}$  and  $x :: 'a::\{field\}^n$  and  $A :: 'a^{m \times n}$ 
  shows  $(k * s \ x) \ v * A = k * s \ (x \ v * A)$ 
  by (metis transpose_matrix_vector vector_scalar_commute)

lemma vector_matrix_mult_0 [simp]:  $0 \ v * A = 0$ 
  unfolding vector_matrix_mult_def by (simp add: zero_vec_def)

lemma vector_matrix_mult_0_right [simp]:  $x \ v * 0 = 0$ 
  unfolding vector_matrix_mult_def by (simp add: zero_vec_def)

lemma vector_matrix_mul_rid [simp]:
  fixes  $v :: ('a::semiring_1)^n$ 
  shows  $v \ v * mat \ 1 = v$ 
  by (metis matrix_vector_mul_lid transpose_mat vector_transpose_matrix)

lemma scaleR_vector_matrix_assoc:
  fixes  $k :: real$  and  $x :: real^n$  and  $A :: real^{m \times n}$ 
  shows  $(k *_R \ x) \ v * A = k *_R \ (x \ v * A)$ 
  by (metis matrix_vector_mult_scaleR transpose_matrix_vector)

proposition vector_scaleR_matrix_ac:
  fixes  $k :: real$  and  $x :: real^n$  and  $A :: real^{m \times n}$ 
  shows  $x \ v * (k *_R \ A) = k *_R \ (x \ v * A)$ 
proof –
  have  $x \ v * (k *_R \ A) = (k *_R \ x) \ v * A$ 
    unfolding vector_matrix_mult_def
    by (simp add: algebra_simps)
  with scaleR_vector_matrix_assoc
  show  $x \ v * (k *_R \ A) = k *_R \ (x \ v * A)$ 
    by auto
qed

end

```

1.9 Linear Algebra on Finite Cartesian Products

```

theory Cartesian_Space
imports
  HOL-Combinatorics.Transposition
  Finite_Cartesian_Product
  Linear_Algebra
begin

```

1.9.1 Type $('a, 'n) \text{ vec}$ and fields as vector spaces

```

definition cart_basis =  $\{axis \ i \ 1 \mid i. \ i \in UNIV\}$ 

```

```

lemma finite_cart_basis: finite (cart_basis) unfolding cart_basis_def

```

using *finite_Atleast_Atmost_nat* by *fastforce*

lemma *card_cart_basis*: $\text{card}(\text{cart\_basis}::('a::\text{zero\_neq\_one}^i) \text{ set}) = \text{CARD}(i)$
unfolding *cart_basis_def Setcompr_eq_image*
by (*rule card_image*) (*auto simp: inj_on_def axis_eq_axis*)

interpretation *vec*: *vector_space* (**s*)
by *unfold_locales* (*vector_algebra_simps*) +

lemma *independent_cart_basis*:
vec.independent (*cart_basis*)
proof (*rule vec.independent_if_scalars_zero*)
show *finite* (*cart_basis*) **using** *finite_cart_basis* .
fix *f*::('a, 'b) *vec* $\Rightarrow$ 'a **and** *x*::('a, 'b) *vec*
assume *eq_0*: $(\sum x \in \text{cart\_basis}. f\ x * s\ x) = 0$ **and** *x_in*: $x \in \text{cart\_basis}$
obtain *i* **where** *x*: $x = \text{axis}\ i\ 1$ **using** *x_in* **unfolding** *cart_basis_def* **by** *auto*
have *sum_eq_0*: $(\sum x \in (\text{cart\_basis}) - \{x\}. f\ x * (x\ \$\ i)) = 0$
proof (*rule sum.neutral, rule ballI*)
fix *xa* **assume** *xa*: $xa \in \text{cart\_basis} - \{x\}$
obtain *a* **where** *a*: $xa = \text{axis}\ a\ 1$ **and** *a_not_i*: $a \neq i$
using *xa* *x* **unfolding** *cart_basis_def* **by** *auto*
have $xa\ \$\ i = 0$ **unfolding** *a_axis_def* **using** *a_not_i* **by** *auto*
thus $f\ xa * xa\ \$\ i = 0$ **by** *simp*
qed
have $0 = (\sum x \in \text{cart\_basis}. f\ x * s\ x)\ \$\ i$ **using** *eq_0* **by** *simp*
also have $\dots = (\sum x \in \text{cart\_basis}. (f\ x * s\ x)\ \$\ i)$ **unfolding** *sum_component* ..
also have $\dots = (\sum x \in \text{cart\_basis}. f\ x * (x\ \$\ i))$ **unfolding** *vector_smult_component*
..
also have $\dots = f\ x * (x\ \$\ i) + (\sum x \in (\text{cart\_basis}) - \{x\}. f\ x * (x\ \$\ i))$
by (*rule sum.remove[OF finite_cart_basis x_in]*)
also have $\dots = f\ x * (x\ \$\ i)$ **unfolding** *sum_eq_0* **by** *simp*
also have $\dots = f\ x$ **unfolding** *x_axis_def* **by** *auto*
finally show $f\ x = 0$..
qed

lemma *span_cart_basis*:
vec.span (*cart_basis*) = *UNIV*
proof (*auto*)
fix *x*::('a, 'b) *vec*
let *?f*= $\lambda v. x\ \$\ (THE\ i. v = \text{axis}\ i\ 1)$
show $x \in \text{vec.span}(\text{cart\_basis})$
apply (*unfold vec.span_finite[OF finite_cart_basis]*)
apply (*rule image_eqI[of _ _ ?f]*)
apply (*subst vec_eq_iff*)
apply *clarify*
proof -
fix *i*::'b
let *?w* = $\text{axis}\ i\ (1::'a)$
have *the_eq_i*: $(THE\ a. ?w = \text{axis}\ a\ 1) = i$

```

    by (rule the_equality, auto simp: axis_eq_axis)
  have sum_eq_0:  $(\sum_{v \in \text{cart\_basis}} - \{?w\}. x \$ (\text{THE } i. v = \text{axis } i \ 1) * v \$ i) = 0$ 
  proof (rule sum.neutral, rule ballI)
    fix xa::('a, 'b) vec
    assume xa:  $xa \in \text{cart\_basis} - \{?w\}$ 
    obtain j where j:  $xa = \text{axis } j \ 1$  and i_not_j:  $i \neq j$  using xa unfolding
    cart_basis_def by auto
    have the_eq_j:  $(\text{THE } i. xa = \text{axis } i \ 1) = j$ 
    proof (rule the_equality)
      show  $xa = \text{axis } j \ 1$  using j .
      show  $\bigwedge i. xa = \text{axis } i \ 1 \implies i = j$  by (metis axis_eq_axis j zero_neq_one)
    qed
    show  $x \$ (\text{THE } i. xa = \text{axis } i \ 1) * xa \$ i = 0$ 
    apply (subst (2) j)
    unfolding the_eq_j unfolding axis_def using i_not_j by simp
  qed
  have  $(\sum_{v \in \text{cart\_basis}} x \$ (\text{THE } i. v = \text{axis } i \ 1) * s \ v) \$ i =$ 
 $(\sum_{v \in \text{cart\_basis}} (x \$ (\text{THE } i. v = \text{axis } i \ 1) * s \ v) \$ i)$  unfolding sum_component
  ..
  also have  $\dots = (\sum_{v \in \text{cart\_basis}} x \$ (\text{THE } i. v = \text{axis } i \ 1) * v \$ i)$ 
  unfolding vector_smult_component ..
  also have  $\dots = x \$ (\text{THE } a. ?w = \text{axis } a \ 1) * ?w \$ i + (\sum_{v \in (\text{cart\_basis}) - \{?w\}} x \$ (\text{THE } i. v = \text{axis } i \ 1) * v \$ i)$ 
  by (rule sum.remove[OF finite_cart_basis], auto simp add: cart_basis_def)
  also have  $\dots = x \$ (\text{THE } a. ?w = \text{axis } a \ 1) * ?w \$ i$  unfolding sum_eq_0
  by simp
  also have  $\dots = x \$ i$  unfolding the_eq_i unfolding axis_def by auto
  finally show  $x \$ i = (\sum_{v \in \text{cart\_basis}} x \$ (\text{THE } i. v = \text{axis } i \ 1) * s \ v) \$ i$  by
  simp
  qed simp
  qed

```

interpretation vec: *finite_dimensional_vector_space* (*s) cart_basis

by (unfold_locales, auto simp add: finite_cart_basis independent_cart_basis span_cart_basis)

lemma matrix_vector_mul_linear_gen[*intro, simp*]:

Vector_Spaces.linear (*s) (*s) ((*v) A)

by unfold_locales

(vector_matrix_vector_mult_def sum.distrib algebra_simps)+

lemma span_vec_eq: *vec.span* X = *span* X

and *dim_vec_eq*: *vec.dim* X = *dim* X

and *dependent_vec_eq*: *vec.dependent* X = *dependent* X

and *subspace_vec_eq*: *vec.subspace* X = *subspace* X

for X::(*real*ⁿ) set

unfolding span_raw_def dim_raw_def dependent_raw_def subspace_raw_def

by (auto simp: scalar_mult_eq_scaleR)

lemma *linear_componentwise*:

fixes $f :: 'a :: \text{field} \rightarrow^m \Rightarrow 'a \rightarrow^n$

assumes $lf : \text{Vector\_Spaces.linear } (*s) (*s) f$

shows $(f\ x)\$j = \text{sum } (\lambda i. (x\$i) * (f\ (\text{axis } i\ 1)\$j))\ (\text{UNIV} :: 'm\ \text{set})\ (\text{is } ?lhs = ?rhs)$

proof –

interpret $lf : \text{Vector\_Spaces.linear } (*s) (*s) f$

using lf .

let $?M = (\text{UNIV} :: 'm\ \text{set})$

let $?N = (\text{UNIV} :: 'n\ \text{set})$

have $fM : \text{finite } ?M$ by simp

have $?rhs = (\text{sum } (\lambda i. (x\$i) *s (f\ (\text{axis } i\ 1))))\ ?M)\$j$

unfolding *sum_component* by simp

then show *?thesis*

unfolding $lf.\text{sum}[\text{symmetric}]\ lf.\text{scale}[\text{symmetric}]$

unfolding *basis_expansion* by auto

qed

interpretation *vec*: $\text{Vector\_Spaces.linear } (*s) (*s) (*v)\ A$

using *matrix_vector_mul_linear_gen*.

interpretation *vec*: *finite_dimensional_vector_space_pair* $(*s)\ \text{cart\_basis } (*s)\ \text{cart\_basis } ..$

lemma *matrix_works*:

assumes $lf : \text{Vector\_Spaces.linear } (*s) (*s) f$

shows $\text{matrix } f *v\ x = f\ (x :: 'a :: \text{field} \rightarrow^n)$

apply (simp add: *matrix_def matrix_vector_mult_def vec_eq_iff mult commute*)

apply *clarify*

apply (rule *linear_componentwise*[OF lf , *symmetric*])

done

lemma *matrix_of_matrix_vector_mul*[*simp*]: $\text{matrix}(\lambda x. A *v\ (x :: 'a :: \text{field} \rightarrow^n)) = A$

by (simp add: *matrix_eq matrix_works*)

lemma *matrix_compose_gen*:

assumes $lf : \text{Vector\_Spaces.linear } (*s) (*s) (f :: 'a :: \{\text{field}\} \rightarrow^n \Rightarrow 'a \rightarrow^m)$

and $lg : \text{Vector\_Spaces.linear } (*s) (*s) (g :: 'a \rightarrow^m \Rightarrow 'a \rightarrow^_)$

shows $\text{matrix } (g \circ f) = \text{matrix } g ** \text{matrix } f$

using $lf\ lg\ \text{Vector\_Spaces.linear\_compose}[OF\ lf\ lg]\ \text{matrix\_works}[OF\ \text{Vector\_Spaces.linear\_compose}[OF\ lf\ lg]]$

by (simp add: *matrix_eq matrix_works matrix_vector_mul_assoc*[*symmetric*]
o_def)

lemma *matrix_compose*:

assumes $\text{linear } (f :: \text{real} \rightarrow^n \Rightarrow \text{real} \rightarrow^m)\ \text{linear } (g :: \text{real} \rightarrow^m \Rightarrow \text{real} \rightarrow^_)$

shows $\text{matrix } (g \circ f) = \text{matrix } g ** \text{matrix } f$
using $\text{matrix\_compose\_gen}[of\ f\ g]\ \text{assms}$
by ($\text{simp add: linear\_def scalar\_mult\_eq\_scaleR}$)

lemma $\text{left\_invertible\_transpose}$:
 $(\exists (B). B ** \text{transpose } (A) = \text{mat } (1 :: 'a :: \text{comm\_semiring\_1})) \longleftrightarrow (\exists (B). A ** B = \text{mat } 1)$
by ($\text{metis matrix\_transpose\_mul transpose\_mat transpose\_transpose}$)

lemma $\text{right\_invertible\_transpose}$:
 $(\exists (B). \text{transpose } (A) ** B = \text{mat } (1 :: 'a :: \text{comm\_semiring\_1})) \longleftrightarrow (\exists (B). B ** A = \text{mat } 1)$
by ($\text{metis matrix\_transpose\_mul transpose\_mat transpose\_transpose}$)

lemma $\text{linear\_matrix\_vector\_mul\_eq}$:
 $\text{Vector\_Spaces.linear } (*s) (*s) f \longleftrightarrow \text{linear } (f :: \text{real}^n \Rightarrow \text{real}^m)$
by ($\text{simp add: scalar\_mult\_eq\_scaleR linear\_def}$)

lemma $\text{matrix\_vector\_mul}[simp]$:
 $\text{Vector\_Spaces.linear } (*s) (*s) g \Longrightarrow (\lambda y. \text{matrix } g *v\ y) = g$
 $\text{linear } f \Longrightarrow (\lambda x. \text{matrix } f *v\ x) = f$
 $\text{bounded\_linear } f \Longrightarrow (\lambda x. \text{matrix } f *v\ x) = f$
for $f :: \text{real}^n \Rightarrow \text{real}^m$
by ($\text{simp\_all add: ext matrix\_works linear\_matrix\_vector\_mul\_eq linear\_linear}$)

lemma $\text{matrix\_left\_invertible\_injective}$:
fixes $A :: 'a :: \text{field}^n^m$
shows $(\exists B. B ** A = \text{mat } 1) \longleftrightarrow \text{inj } ((*v) A)$
proof *safe*
fix B
assume $B: B ** A = \text{mat } 1$
show $\text{inj } ((*v) A)$
unfolding inj_on_def
by ($\text{metis } B \text{ matrix\_vector\_mul\_assoc matrix\_vector\_mul\_lid}$)
next
assume $\text{inj } ((*v) A)$
from $\text{vec.linear\_injective\_left\_inverse}[OF \text{matrix\_vector\_mul\_linear\_gen this}]$
obtain g **where** $\text{Vector\_Spaces.linear } (*s) (*s) g$ **and** $g: g \circ (*v) A = \text{id}$
by *blast*
have $\text{matrix } g ** A = \text{mat } 1$
by ($\text{metis matrix\_vector\_mul\_linear\_gen } \langle \text{Vector\_Spaces.linear } (*s) (*s) g \rangle g$
 $\text{matrix\_compose\_gen}$
 $\text{matrix\_eq\_matrix\_id\_mat\_1 matrix\_vector\_mul}(1))$
then show $\exists B. B ** A = \text{mat } 1$
by *metis*
qed

lemma $\text{matrix\_left\_invertible\_ker}$:
 $(\exists B. (B :: 'a :: \{\text{field}\}^m^n) ** (A :: 'a :: \{\text{field}\}^n^m) = \text{mat } 1) \longleftrightarrow (\forall x. A *v\ x = 0)$

```

 $x = 0 \longrightarrow x = 0$ )
  unfolding matrix_left_invertible_injective
  using vec.inj_on_iff_eq_0[OF vec.subspace_UNIV, of A]
  by (simp add: inj_on_def)

lemma matrix_right_invertible_surjective:
  ( $\exists B. (A :: 'a :: \text{field}^n \text{ } ^m) ** (B :: 'a :: \text{field}^m \text{ } ^n) = \text{mat } 1$ )  $\longleftrightarrow$  surj ( $\lambda x. A * v \ x$ )
proof -
  { fix B :: 'a ^m ^n
    assume AB: A ** B = mat 1
    { fix x :: 'a ^m
      have A * v (B * v x) = x
        by (simp add: matrix_vector_mul_assoc AB) }
    hence surj ((*v) A) unfolding surj_def by metis }
  moreover
  { assume sf: surj ((*v) A)
    from vec.linear_surjective_right_inverse[OF _ this]
    obtain g: 'a ^m  $\Rightarrow$  'a ^n where g: Vector_Spaces.linear (*s) (*s) g (*v) A
  }
  o g = id
    by blast

  have A ** (matrix g) = mat 1
    unfolding matrix_eq matrix_vector_mul_lid
      matrix_vector_mul_assoc[symmetric] matrix_works[OF g(1)]
    using g(2) unfolding o_def fun_eq_iff id_def
    .
  hence  $\exists B. A ** (B :: 'a ^m ^n) = \text{mat } 1$  by blast
}
ultimately show ?thesis unfolding surj_def by blast
qed

```

```

lemma matrix_left_invertible_independent_columns:
  fixes A :: 'a :: {field} ^n ^m
  shows ( $\exists (B :: 'a ^m ^n). B ** A = \text{mat } 1$ )  $\longleftrightarrow$ 
    ( $\forall c. \text{sum } (\lambda i. c \ i * s \ \text{column } i \ A) \ (UNIV :: 'n \ \text{set}) = 0 \longrightarrow (\forall i. c \ i = 0)$ )
    (is ?lhs  $\longleftrightarrow$  ?rhs)
proof -
  let ?U = UNIV :: 'n set
  { assume k:  $\forall x. A * v \ x = 0 \longrightarrow x = 0$ 
    { fix c i
      assume c:  $\text{sum } (\lambda i. c \ i * s \ \text{column } i \ A) \ ?U = 0$  and i:  $i \in ?U$ 
      let ?x =  $\chi \ i. c \ i$ 
      have th0: A * v ?x = 0
        using c
        by (vector matrix_mult_sum)
      from k[rule_format, OF th0] i
      have c i = 0 by (vector vec_eq_iff)}
    hence ?rhs by blast }
  moreover

```

```

{ assume H: ?rhs
  { fix x assume x: A *v x = 0
    let ?c =  $\lambda i. ((x\$i):: 'a)$ 
    from H[rule_format, of ?c, unfolded matrix_mult_sum[symmetric], OF x]
    have x = 0 by vector }
  }
ultimately show ?thesis unfolding matrix_left_invertible_ker by auto
qed

lemma matrix_right_invertible_independent_rows:
  fixes A :: 'a::field ^n ^m
  shows ( $\exists (B::'a ^m ^n). A ** B = \text{mat } 1$ )  $\longleftrightarrow$ 
    ( $\forall c. \text{sum } (\lambda i. c\ i *s \text{row } i\ A) (\text{UNIV}::'m \text{ set}) = 0 \longrightarrow (\forall i. c\ i = 0)$ )
  unfolding left_invertible_transpose[symmetric]
    matrix_left_invertible_independent_columns
  by (simp add:)

lemma matrix_right_invertible_span_columns:
  ( $\exists (B::'a::field ^n ^m). (A::'a ^m ^n) ** B = \text{mat } 1$ )  $\longleftrightarrow$ 
    vec.span (columns A) = UNIV (is ?lhs = ?rhs)
proof -
  let ?U = UNIV :: 'm set
  have fU: finite ?U by simp
  have lhseq: ?lhs  $\longleftrightarrow$  ( $\forall y. \exists (x::'a ^m). \text{sum } (\lambda i. (x\$i) *s \text{column } i\ A) ?U = y$ )
    unfolding matrix_right_invertible_surjective matrix_mult_sum surj_def
    by (simp add: eq_commute)
  have rhseq: ?rhs  $\longleftrightarrow$  ( $\forall x. x \in \text{vec.span } (\text{columns } A)$ ) by blast
  { assume h: ?lhs
    { fix x:: 'a ^n
      from h[unfolded lhseq, rule_format, of x] obtain y :: 'a ^m
        where y:  $\text{sum } (\lambda i. (y\$i) *s \text{column } i\ A) ?U = x$  by blast
      have x  $\in \text{vec.span } (\text{columns } A)$ 
        unfolding y[symmetric] scalar_mult_eq_scaleR
      proof (rule vec.span_sum [OF vec.span_scale])
        show column i A  $\in \text{vec.span } (\text{columns } A)$  for i
          using columns_def vec.span_superset by auto
      qed
    }
    then have ?rhs unfolding rhseq by blast }
  moreover
  { assume h: ?rhs
    let ?P =  $\lambda(y::'a ^n). \exists (x::'a ^m). \text{sum } (\lambda i. (x\$i) *s \text{column } i\ A) ?U = y$ 
    { fix y
      have y  $\in \text{vec.span } (\text{columns } A)$ 
        unfolding h by blast
      then have ?P y
        proof (induction rule: vec.span_induct_alt)
          case base
            then show ?case

```

```

    by (metis (full_types) matrix_mult_sum matrix_vector_mult_0_right)
  next
  case (step c y1 y2)
  from step obtain i where i: i ∈ ?U y1 = column i A
    unfolding columns_def by blast
  obtain x: 'a ^m where x: sum (λi. (x$i) * column i A) ?U = y2
    using step by blast
  let ?x = (χ j. if j = i then c + (x$i) else (x$j))::'a ^m
  show ?case
    proof (rule exI[where x= ?x], vector, auto simp add: i x[symmetric]
if_distrib distrib_left if_distribR cong del: if_weak_cong)
      fix j
      have th: ∀ xa ∈ ?U. (if xa = i then (c + (x$i)) * ((column xa A)$j)
        else (x$xa) * ((column xa A)$j)) = (if xa = i then c * ((column i A)$j)
        else 0) + ((x$xa) * ((column xa A)$j))
        using i(1) by (simp add: field_simps)
      have sum (λxa. if xa = i then (c + (x$i)) * ((column xa A)$j)
        else (x$xa) * ((column xa A)$j)) ?U = sum (λxa. (if xa = i then c *
        ((column i A)$j) else 0) + ((x$xa) * ((column xa A)$j))) ?U
        by (rule sum.cong[OF refl]) (use th in blast)
      also have ... = sum (λxa. if xa = i then c * ((column i A)$j) else 0) ?U
        + sum (λxa. ((x$xa) * ((column xa A)$j))) ?U
        by (simp add: sum.distrib)
      also have ... = c * ((column i A)$j) + sum (λxa. ((x$xa) * ((column xa
        A)$j))) ?U
        unfolding sum.delta[OF fU]
        using i(1) by simp
      finally show sum (λxa. if xa = i then (c + (x$i)) * ((column xa A)$j)
        else (x$xa) * ((column xa A)$j)) ?U = c * ((column i A)$j) + sum (λxa.
        ((x$xa) * ((column xa A)$j))) ?U .
    qed
  qed
}
then have ?lhs unfolding lhseq ..
}
ultimately show ?thesis by blast
qed

```

lemma *matrix_left_invertible_span_rows_gen*:
 $(\exists (B::'a ^m ^n). B ** (A::'a::field ^n ^m) = \text{mat } 1) \longleftrightarrow \text{vec.span (rows } A) = \text{UNIV}$

```

  unfolding right_invertible_transpose[symmetric]
  unfolding columns_transpose[symmetric]
  unfolding matrix_right_invertible_span_columns
  ..

```

lemma *matrix_left_invertible_span_rows*:
 $(\exists (B::\text{real} ^m ^n). B ** (A::\text{real} ^n ^m) = \text{mat } 1) \longleftrightarrow \text{span (rows } A) = \text{UNIV}$
 using *matrix_left_invertible_span_rows_gen*[of A] by (simp add: span_vec_eq)


```

lemma matrix_left_right_inverse:
  fixes A A' :: 'a::{field}^n^n
  shows A ** A' = mat 1  $\longleftrightarrow$  A' ** A = mat 1
proof -
  { fix A A' :: 'a ^n^n
    assume AA': A ** A' = mat 1
    have sA: surj ((*v) A)
      using AA' matrix_right_invertible_surjective by auto
    from vec.linear_surjective_isomorphism[OF matrix_vector_mul_linear_gen
sA]
    obtain f' :: 'a ^n  $\Rightarrow$  'a ^n
      where f': Vector_Spaces.linear (*s) (*s) f'  $\forall x. f' (A *v x) = x \forall x. A *v f'
x = x$  by blast
    have th: matrix f' ** A = mat 1
      by (simp add: matrix_eq matrix_works[OF f'(1)]
matrix_vector_mul_assoc[symmetric] f'(2)[rule_format])
    hence (matrix f' ** A) ** A' = mat 1 ** A' by simp
    hence matrix f' = A'
      by (simp add: matrix_mul_assoc[symmetric] AA')
    hence matrix f' ** A = A' ** A by simp
    hence A' ** A = mat 1 by (simp add: th)
  }
  then show ?thesis by blast
qed

```

```

lemma invertible_left_inverse:
  fixes A :: 'a::{field}^n^n
  shows invertible A  $\longleftrightarrow$  ( $\exists (B::'a^n^n). B ** A = mat 1$ )
  by (metis invertible_def matrix_left_right_inverse)

```

```

lemma invertible_right_inverse:
  fixes A :: 'a::{field}^n^n
  shows invertible A  $\longleftrightarrow$  ( $\exists (B::'a^n^n). A ** B = mat 1$ )
  by (metis invertible_def matrix_left_right_inverse)

```

```

lemma invertible_mult:
  assumes inv_A: invertible A
  and inv_B: invertible B
  shows invertible (A**B)
proof -
  obtain A' where AA': A ** A' = mat 1 and A'A: A' ** A = mat 1
    using inv_A unfolding invertible_def by blast
  obtain B' where BB': B ** B' = mat 1 and B'B: B' ** B = mat 1
    using inv_B unfolding invertible_def by blast
  show ?thesis
proof (unfold invertible_def, rule exI[of _ B'**A'], rule conjI)
  have A ** B ** (B' ** A') = A ** (B ** (B' ** A'))
    using matrix_mul_assoc[of A B (B' ** A'), symmetric] .

```

```

    also have ... = A ** (B ** B' ** A') unfolding matrix_mul_assoc[of B B'
A'] ..
    also have ... = A ** (mat 1 ** A') unfolding BB' ..
    also have ... = A ** A' unfolding matrix_mul_lid ..
    also have ... = mat 1 unfolding AA' ..
    finally show A ** B ** (B' ** A') = mat (1::'a) .
    have B' ** A' ** (A ** B) = B' ** (A' ** (A ** B)) using matrix_mul_assoc[of
B' A' (A ** B), symmetric] .
    also have ... = B' ** (A' ** A ** B) unfolding matrix_mul_assoc[of A' A
B] ..
    also have ... = B' ** (mat 1 ** B) unfolding A'A ..
    also have ... = B' ** B unfolding matrix_mul_lid ..
    also have ... = mat 1 unfolding B'B ..
    finally show B' ** A' ** (A ** B) = mat 1 .
qed
qed

```

lemma *transpose_invertible*:

```

  fixes A :: realn
  assumes invertible A
  shows invertible (transpose A)
  by (meson assms invertible_def matrix_left_right_inverse right_invertible_transpose)

```

lemma *vector_matrix_mul_assoc*:

```

  fixes v :: ('a::comm_semiring_1)n
  shows (v v* M) v* N = v v* (M ** N)
proof -
  from matrix_vector_mul_assoc
  have transpose N *v (transpose M *v v) = (transpose N ** transpose M) *v v
  by fast
  thus (v v* M) v* N = v v* (M ** N)
  by (simp add: matrix_transpose_mul [symmetric])
qed

```

lemma *matrix_scaleR_vector_ac*:

```

  fixes A :: real(m::finite)n
  shows A *v (k *R v) = k *R A *v v
  by (metis matrix_vector_mult_scaleR transpose_scalar vector_scaleR_matrix_ac
vector_transpose_matrix)

```

lemma *scaleR_matrix_vector_assoc*:

```

  fixes A :: real(m::finite)n
  shows k *R (A *v v) = k *R A *v v
  by (metis matrix_scaleR_vector_ac matrix_vector_mult_scaleR)

```

locale *linear_first_finite_dimensional_vector_space* =
 l?: *Vector_Spaces.linear scaleB scaleC f* +

```

B?: finite_dimensional_vector_space scaleB BasisB
for scaleB :: ('a::field => 'b::ab_group_add => 'b) (infixr *b 75)
and scaleC :: ('a => 'c::ab_group_add => 'c) (infixr *c 75)
and BasisB :: ('b set)
and f :: ('b=>'c)

lemma vec_dim_card: vec.dim (UNIV::('a::{field}n) set) = CARD ('n)
proof -
  let ?f=λi::'n. axis i (1::'a)
  have vec.dim (UNIV::('a::{field}n) set) = card (cart_basis::('an) set)
    unfolding vec.dim_UNIV ..
  also have ... = card ({i. i ∈ UNIV}::('n) set)
    proof (rule bij_betw_same_card[of ?f, symmetric], unfold bij_betw_def, auto)
      show inj (λi::'n. axis i (1::'a)) by (simp add: inj_on_def axis_eq_axis)
      fix i::'n
      show axis i 1 ∈ cart_basis unfolding cart_basis_def by auto
      fix x::'an
      assume x ∈ cart_basis
      thus x ∈ range (λi. axis i 1) unfolding cart_basis_def by auto
    qed
  also have ... = CARD('n) by auto
  finally show ?thesis .
qed

interpretation vector_space_over_itself: vector_space (*) :: 'a::field ⇒ 'a ⇒ 'a
by unfold_locales (simp_all add: algebra_simps)

lemmas [simp del] = vector_space_over_itself.scale_scale

interpretation vector_space_over_itself: finite_dimensional_vector_space
  (*) :: 'a::field => 'a => 'a {1}
by unfold_locales (auto simp: vector_space_over_itself.span_singleton)

lemma dimension_eq_1[code_unfold]: vector_space_over_itself.dimension TYPE('a::field)=
1
  unfolding vector_space_over_itself.dimension_def by simp

lemma dim_subset_UNIV_cart_gen:
  fixes S :: ('a::fieldn) set
  shows vec.dim S ≤ CARD('n)
by (metis vec.dim_eq_full vec.dim_subset_UNIV vec.span_UNIV vec.dim_card)

lemma dim_subset_UNIV_cart:
  fixes S :: (realn) set
  shows dim S ≤ CARD('n)
  using dim_subset_UNIV_cart_gen[of S] by (simp add: dim_vec_eq)

```

Two sometimes fruitful ways of looking at matrix-vector multiplication.

```
lemma matrix_mult_dot:  $A * v \ x = (\chi \ i. \text{inner } (A\$i) \ x)$ 
  by (simp add: matrix_vector_mult_def inner_vec_def)
```

```
lemma adjoint_matrix:  $\text{adjoint}(\lambda x. (A :: \text{real}^n \times \text{real}^m) * v \ x) = (\lambda x. \text{transpose } A * v \ x)$ 
  apply (rule adjoint_unique)
  apply (simp add: transpose_def inner_vec_def matrix_vector_mult_def
    sum_distrib_right sum_distrib_left)
  apply (subst sum.swap)
  apply (simp add: ac_simps)
  done
```

```
lemma matrix_adjoint: assumes lf: linear (f ::  $\text{real}^n \Rightarrow \text{real}^m$ )
  shows matrix(adjoint f) = transpose(matrix f)
proof -
  have matrix(adjoint f) = matrix(adjoint ((*v) (matrix f)))
    by (simp add: lf)
  also have ... = transpose(matrix f)
    unfolding adjoint_matrix matrix_of_matrix_vector_mul
    apply rule
    done
  finally show ?thesis .
qed
```

1.9.2 Rank of a matrix

Equivalence of row and column rank is taken from George Mackiw's paper, Mathematics Magazine 1995, p. 285.

```
lemma matrix_vector_mult_in_columnspace_gen:
  fixes  $A :: 'a :: \text{field}^n \times \text{real}^m$ 
  shows  $(A * v \ x) \in \text{vec.span}(\text{columns } A)$ 
  apply (simp add: matrix_vector_column columns_def transpose_def column_def)
  apply (intro vec.span_sum vec.span_scale)
  apply (force intro: vec.span_base)
  done
```

```
lemma matrix_vector_mult_in_columnspace:
  fixes  $A :: \text{real}^n \times \text{real}^m$ 
  shows  $(A * v \ x) \in \text{span}(\text{columns } A)$ 
  using matrix_vector_mult_in_columnspace_gen[of A x] by (simp add: span_vec_eq)
```

```
lemma subspace_orthogonal_to_vector: subspace {y. orthogonal x y}
  by (simp add: subspace_def orthogonal_clauses)
```

```
lemma orthogonal_nullspace_rowspace:
  fixes  $A :: \text{real}^n \times \text{real}^m$ 
  assumes 0:  $A * v \ x = 0$  and y:  $y \in \text{span}(\text{rows } A)$ 
  shows orthogonal x y
  using y
```

```

proof (induction rule: span_induct)
  case base
  then show ?case
    by (simp add: subspace_orthogonal_to_vector)
next
  case (step v)
  then obtain i where v = row i A
    by (auto simp: rows_def)
  with 0 show ?case
    unfolding orthogonal_def inner_vec_def matrix_vector_mult_def row_def
    by (simp add: mult.commute) (metis (no_types) vec_lambda_beta zero_index)
qed

```

```

lemma nullspace_inter_rowspace:
  fixes A :: realnm
  shows A * v x = 0 ∧ x ∈ span(rows A) ⟷ x = 0
  using orthogonal_nullspace_rowspace orthogonal_self span_zero matrix_vector_mult_0_right
  by blast

```

```

lemma matrix_vector_mul_injective_on_rowpace:
  fixes A :: realnm
  shows ⟦A * v x = A * v y; x ∈ span(rows A); y ∈ span(rows A)⟧ ⟹ x = y
  using nullspace_inter_rowspace [of A x y]
  by (metis diff_eq_diff_eq diff_self matrix_vector_mult_diff_distrib span_diff)

```

```

definition rank :: 'a::fieldnm ⇒ nat
  where row_rank_def_gen: rank A ≡ vec.dim(rows A)

```

```

lemma row_rank_def: rank A = dim (rows A) for A::realnm
  by (auto simp: row_rank_def_gen dim_vec_eq)

```

```

lemma dim_rows_le_dim_columns:
  fixes A :: realnm
  shows dim(rows A) ≤ dim(columns A)

```

```

proof -
  have dim (span (rows A)) ≤ dim (span (columns A))
  proof -
    obtain B where independent B span(rows A) ⊆ span B
      and B: B ⊆ span(rows A) card B = dim (span(rows A))
    using basis_exists [of span(rows A)] by metis
    with span_subspace have eq: span B = span(rows A)
    by auto
    then have inj: inj_on ((*v) A) (span B)
    by (simp add: inj_on_def matrix_vector_mul_injective_on_rowpace)
    then have ind: independent ((*v) A ' B)
    by (rule linear_independent_injective_image [OF Finite_Cartesian_Product.matrix_vector_mul_linear
      independent B])
    have dim (span (rows A)) ≤ card ((*v) A ' B)
    unfolding B(2)[symmetric]

```

```

    using inj
    by (auto simp: card_image inj_on_subset span_superset)
  also have ... ≤ dim (span (columns A))
    using _ ind
    by (rule independent_card_le_dim) (auto intro!: matrix_vector_mult_in_columnspace)
  finally show ?thesis .
qed
then show ?thesis
  by (simp)
qed

```

```

lemma column_rank_def:
  fixes A :: realnm
  shows rank A = dim(columns A)
  unfolding row_rank_def
  by (metis columns_transpose dim_rows_le_dim_columns le_antisym rows_transpose)

```

```

lemma rank_transpose:
  fixes A :: realnm
  shows rank(transpose A) = rank A
  by (metis column_rank_def row_rank_def rows_transpose)

```

```

lemma matrix_vector_mult_basis:
  fixes A :: realnm
  shows A * v (axis k 1) = column k A
  by (simp add: cart_eq_inner_axis column_def matrix_mult_dot)

```

```

lemma columns_image_basis:
  fixes A :: realnm
  shows columns A = (*v) A ‘ (range (λi. axis i 1))
  by (force simp: columns_def matrix_vector_mult_basis [symmetric])

```

```

lemma rank_dim_range:
  fixes A :: realnm
  shows rank A = dim(range (λx. A * v x))
  unfolding column_rank_def
proof (rule span_eq_dim)
  have span (columns A) ⊆ span (range ((*v) A)) (is ?l ⊆ ?r)
    by (simp add: columns_image_basis image_subsetI span_mono)
  then show ?l = ?r
    by (metis (no_types, lifting) image_subset_iff matrix_vector_mult_in_columnspace
      span_eq span_span)
qed

```

```

lemma rank_bound:
  fixes A :: realnm
  shows rank A ≤ min CARD('m) (CARD('n))
  by (metis (mono_tags, lifting) dim_subset_UNIV_cart min.bounded_iff
    column_rank_def row_rank_def)

```

```

lemma full_rank_injective:
  fixes A :: real'n'm
  shows rank A = CARD('n)  $\longleftrightarrow$  inj ((*v) A)
  by (simp add: matrix_left_invertible_injective [symmetric] matrix_left_invertible_span_rows
row_rank_def
dim_eq_full [symmetric] card_cart_basis vec.dimension_def)

```

```

lemma full_rank_surjective:
  fixes A :: real'n'm
  shows rank A = CARD('m)  $\longleftrightarrow$  surj ((*v) A)
  by (simp add: matrix_right_invertible_surjective [symmetric] left_invertible_transpose
[symmetric]
matrix_left_invertible_injective full_rank_injective [symmetric]
rank_transpose)

```

```

lemma rank_I: rank(mat 1::real'n'n) = CARD('n)
  by (simp add: full_rank_injective inj_on_def)

```

```

lemma less_rank_noninjective:
  fixes A :: real'n'm
  shows rank A < CARD('n)  $\longleftrightarrow$   $\neg$  inj ((*v) A)
using less_le rank_bound by (auto simp: full_rank_injective [symmetric])

```

```

lemma matrix_nonfull_linear_equations_eq:
  fixes A :: real'n'm
  shows ( $\exists x. (x \neq 0) \wedge A * v x = 0$ )  $\longleftrightarrow$  rank A  $\neq$  CARD('n)
  by (meson matrix_left_invertible_injective full_rank_injective matrix_left_invertible_ker)

```

```

lemma rank_eq_0: rank A = 0  $\longleftrightarrow$  A = 0 and rank_0 [simp]: rank (0::real'n'm)
= 0
  for A :: real'n'm
  by (auto simp: rank_dim_range matrix_eq)

```

```

lemma rank_mul_le_right:
  fixes A :: real'n'm and B :: real'p'n
  shows rank(A ** B)  $\leq$  rank B
proof -
  have rank(A ** B)  $\leq$  dim ((*v) A ' range ((*v) B))
  by (auto simp: rank_dim_range image_comp o_def matrix_vector_mul_assoc)
  also have ...  $\leq$  rank B
  by (simp add: rank_dim_range dim_image_le)
  finally show ?thesis .
qed

```

```

lemma rank_mul_le_left:
  fixes A :: real'n'm and B :: real'p'n
  shows rank(A ** B)  $\leq$  rank A
  by (metis matrix_transpose_mul rank_mul_le_right rank_transpose)

```

1.9.3 Lemmas for working on $\text{real}^1/2/3/4$

lemma *exhaust_2*:

fixes $x :: 2$

shows $x = 1 \vee x = 2$

proof (*induct* x)

case (*of_int* z)

then have $0 \leq z$ and $z < 2$ by *simp_all*

then have $z = 0 \mid z = 1$ by *arith*

then show *?case* by *auto*

qed

lemma *forall_2*: $(\forall i::2. P\ i) \longleftrightarrow P\ 1 \wedge P\ 2$

by (*metis exhaust_2*)

lemma *exhaust_3*:

fixes $x :: 3$

shows $x = 1 \vee x = 2 \vee x = 3$

proof (*induct* x)

case (*of_int* z)

then have $0 \leq z$ and $z < 3$ by *simp_all*

then have $z = 0 \vee z = 1 \vee z = 2$ by *arith*

then show *?case* by *auto*

qed

lemma *forall_3*: $(\forall i::3. P\ i) \longleftrightarrow P\ 1 \wedge P\ 2 \wedge P\ 3$

by (*metis exhaust_3*)

lemma *exhaust_4*:

fixes $x :: 4$

shows $x = 1 \vee x = 2 \vee x = 3 \vee x = 4$

proof (*induct* x)

case (*of_int* z)

then have $0 \leq z$ and $z < 4$ by *simp_all*

then have $z = 0 \vee z = 1 \vee z = 2 \vee z = 3$ by *arith*

then show *?case* by *auto*

qed

lemma *forall_4*: $(\forall i::4. P\ i) \longleftrightarrow P\ 1 \wedge P\ 2 \wedge P\ 3 \wedge P\ 4$

by (*metis exhaust_4*)

lemma *UNIV_1* [*simp*]: $UNIV = \{1::1\}$

by (*auto simp add: num1_eq_iff*)

lemma *UNIV_2*: $UNIV = \{1::2, 2::2\}$

using *exhaust_2* by *auto*

lemma *UNIV_3*: $UNIV = \{1::3, 2::3, 3::3\}$

using *exhaust_3* by *auto*


```
lemma UNIV_4: UNIV = {1::4, 2::4, 3::4, 4::4}
  using exhaust_4 by auto
```

```
lemma sum_1: sum f (UNIV::1 set) = f 1
  unfolding UNIV_1 by simp
```

```
lemma sum_2: sum f (UNIV::2 set) = f 1 + f 2
  unfolding UNIV_2 by simp
```

```
lemma sum_3: sum f (UNIV::3 set) = f 1 + f 2 + f 3
  unfolding UNIV_3 by (simp add: ac_simps)
```

```
lemma sum_4: sum f (UNIV::4 set) = f 1 + f 2 + f 3 + f 4
  unfolding UNIV_4 by (simp add: ac_simps)
```

1.9.4 The collapse of the general concepts to dimension one

```
lemma vector_one: (x::'a ^1) = ( $\chi$  i. (x$1))
  by (simp add: vec_eq_iff)
```

```
lemma forall_one: ( $\forall$  (x::'a ^1). P x)  $\longleftrightarrow$  ( $\forall$  x. P( $\chi$  i. x))
  apply auto
  apply (erule_tac x= x$1 in allE)
  apply (simp only: vector_one[symmetric])
  done
```

```
lemma norm_vector_1: norm (x :: _ ^1) = norm (x$1)
  by (simp add: norm_vec_def)
```

```
lemma dist_vector_1:
  fixes x :: 'a::real_normed_vector ^1
  shows dist x y = dist (x$1) (y$1)
  by (simp add: dist_norm norm_vector_1)
```

```
lemma norm_real: norm(x::real ^1) = |x$1|
  by (simp add: norm_vector_1)
```

```
lemma dist_real: dist(x::real ^1) y = |(x$1) - (y$1)|
  by (auto simp add: norm_real dist_norm)
```

1.9.5 Routine results connecting the types (real, 1) vec and real

```
lemma vector_one_nth [simp]:
  fixes x :: 'a ^1 shows vec (x $ 1) = x
  by (metis vec_def vector_one)
```

```
lemma tendsto_at_within_vector_1:
  fixes S :: 'a :: metric_space set
```

```

    assumes (f ⟶ fx) (at x within S)
    shows ((λy::'a^1. χ i. f (y $ 1)) ⟶ (vec fx::'a^1)) (at (vec x) within vec '
S)
proof (rule topological_tendstoI)
  fix T :: ('a^1) set
  assume open T vec fx ∈ T
  have ∀F x in at x within S. f x ∈ (λx. x $ 1) ' T
    using ⟨open T⟩ ⟨vec fx ∈ T⟩ assms open_image_vec_nth tendsto_def by
fastforce
  then show ∀F x::'a^1 in at (vec x) within vec ' S. (χ i. f (x $ 1)) ∈ T
    unfolding eventually_at dist_norm [symmetric]
    by (rule ex_forward)
      (use ⟨open T⟩ in
        ⟨fastforce simp: dist_norm dist_vec_def L2_set_def image_iff vector_one
open_vec_def⟩)
qed

```

```

lemma has_derivative_vector_1:
  assumes der_g: (g has_derivative (λx. x * g' a)) (at a within S)
  shows ((λx. vec (g (x $ 1))) has_derivative (*R) (g' a))
    (at ((vec a)::real^1) within vec ' S)
  using der_g
  apply (auto simp: Deriv.has_derivative_within bounded_linear_scaleR_right
norm_vector_1)
  apply (drule tendsto_at_within_vector_1, vector)
  apply (auto simp: algebra_simps eventually_at tendsto_def)
  done

```

1.9.6 Explicit vector construction from lists

```

definition vector l = (χ i. foldr (λx f n. fun_upd (f (n+1)) n x) l (λn x. 0) 1 i)

```

```

lemma vector_1 [simp]: (vector[x]) $1 = x
  unfolding vector_def by simp

```

```

lemma vector_2 [simp]: (vector[x,y]) $1 = x (vector[x,y] :: 'a^2)$2 = (y::'a::zero)
  unfolding vector_def by simp_all

```

```

lemma vector_3 [simp]:
  (vector [x,y,z] :: ('a::zero)^3)$1 = x
  (vector [x,y,z] :: ('a::zero)^3)$2 = y
  (vector [x,y,z] :: ('a::zero)^3)$3 = z
  unfolding vector_def by simp_all

```

```

lemma forall_vector_1: (∀ v::'a::zero^1. P v) ⟷ (∀ x. P(vector[x]))
  by (metis vector_1 vector_one)

```

```

lemma forall_vector_2: (∀ v::'a::zero^2. P v) ⟷ (∀ x y. P(vector[x, y]))
  apply auto

```

```

apply (erule_tac x=v$1 in allE)
apply (erule_tac x=v$2 in allE)
apply (subgoal_tac vector [v$1, v$2] = v)
apply simp
apply (vector vector_def)
apply (simp add: forall_2)
done

```

```

lemma forall_vector_3: ( $\forall v::'a::zero^3. P\ v$ )  $\longleftrightarrow$  ( $\forall x\ y\ z. P(\text{vector}[x, y, z])$ )
apply auto
apply (erule_tac x=v$1 in allE)
apply (erule_tac x=v$2 in allE)
apply (erule_tac x=v$3 in allE)
apply (subgoal_tac vector [v$1, v$2, v$3] = v)
apply simp
apply (vector vector_def)
apply (simp add: forall_3)
done

```

1.9.7 lambda skolemization on cartesian products

```

lemma lambda_skolem: ( $\forall i. \exists x. P\ i\ x$ )  $\longleftrightarrow$ 
  ( $\exists x::'a \wedge 'n. \forall i. P\ i\ (x\ \$\ i)$ ) (is ?lhs  $\longleftrightarrow$  ?rhs)
proof -
  let ?S = (UNIV :: 'n set)
  { assume H: ?rhs
    then have ?lhs by auto }
  moreover
  { assume H: ?lhs
    then obtain f where f:  $\forall i. P\ i\ (f\ i)$  unfolding choice_iff by metis
    let ?x = ( $\chi\ i. (f\ i)$ ) :: 'a  $\wedge$  'n
    { fix i
      from f have P i (f i) by metis
      then have P i (?x $ i) by auto
    }
    hence  $\forall i. P\ i\ (?x\ \$\ i)$  by metis
    hence ?rhs by metis }
  ultimately show ?thesis by metis
qed

```

The same result in terms of square matrices.

Considering an n-element vector as an n-by-1 or 1-by-n matrix.

definition rowvector $v = (\chi\ i\ j. (v\$j))$

definition columnvector $v = (\chi\ i\ j. (v\$i))$

```

lemma transpose_columnvector: transpose(columnvector v) = rowvector v
  by (simp add: transpose_def rowvector_def columnvector_def vec_eq_iff)

```

```

lemma transpose_rowvector: transpose(rowvector v) = columnvector v
  by (simp add: transpose_def columnvector_def rowvector_def vec_eq_iff)

lemma dot_rowvector_columnvector: columnvector (A *v v) = A ** columnvector
v
  by (vector columnvector_def matrix_matrix_mult_def matrix_vector_mult_def)

lemma dot_matrix_product:
  (x::realn) • y = (((rowvector x :: realn1) ** (columnvector y :: real1n))$1)$1
  by (vector matrix_matrix_mult_def rowvector_def columnvector_def inner_vec_def)

lemma dot_matrix_vector_mul:
  fixes A B :: realn n and x y :: realn
  shows (A *v x) • (B *v y) =
    (((rowvector x :: realn1) ** ((transpose A ** B) ** (columnvector y :: real1n)))$1)$1
  unfolding dot_matrix_product transpose_columnvector[symmetric]
    dot_rowvector_columnvector matrix_transpose_mul matrix_mul_assoc ..

lemma dim_substandard_cart: vec.dim {x::'a::fieldn. ∀ i. i ∉ d ⟶ x $ i = 0} =
card d
  (is vec.dim ?A = _)
proof (rule vec.dim_unique)
  let ?B = ((λx. axis x 1) ' d)
  have subset_basis: ?B ⊆ cart_basis
    by (auto simp: cart_basis_def)
  show ?B ⊆ ?A
    by (auto simp: axis_def)
  show vec.independent ((λx. axis x 1) ' d)
    using subset_basis
    by (rule vec.independent_mono[OF vec.independent_Basis])
  have x ∈ vec.span ?B if ∀ i. i ∉ d ⟶ x $ i = 0 for x::'an
  proof -
    have finite ?B
      using subset_basis finite_cart_basis
      by (rule finite_subset)
    have x = (∑ i ∈ UNIV. x $ i *s axis i 1)
      by (rule basis_expansion[symmetric])
    also have ... = (∑ i ∈ d. (x $ i) *s axis i 1)
      by (rule sum.mono_neutral_cong_right) (auto simp: that)
    also have ... ∈ vec.span ?B
      by (simp add: vec.span_sum vec.span_clauses)
    finally show x ∈ vec.span ?B .
  qed
  then show ?A ⊆ vec.span ?B by auto
qed (simp add: card_image inj_on_def axis_eq_axis)

lemma affinity_inverses:

```

```

assumes m0:  $m \neq (0::'a::field)$ 
shows  $(\lambda x. m * x + c) \circ (\lambda x. inverse(m) * x + -(inverse(m) * c)) = id$ 
 $(\lambda x. inverse(m) * x + -(inverse(m) * c)) \circ (\lambda x. m * x + c) = id$ 
using m0
by (auto simp add: fun_eq_iff vector_add_ldistrib diff_conv_add_uminus simp
del: add_uminus_conv_diff)

```

```

lemma vector_affinity_eq:
assumes m0:  $(m::'a::field) \neq 0$ 
shows  $m * x + c = y \longleftrightarrow x = inverse\ m * y + -(inverse\ m * c)$ 
proof
assume h:  $m * x + c = y$ 
hence  $m * x = y - c$  by (simp add: field_simps)
hence  $inverse\ m * (m * x) = inverse\ m * (y - c)$  by simp
then show  $x = inverse\ m * y + -(inverse\ m * c)$ 
using m0 by (simp add: vector_smult_assoc vector_ssub_ldistrib)
next
assume h:  $x = inverse\ m * y + -(inverse\ m * c)$ 
show  $m * x + c = y$  unfolding h
using m0 by (simp add: vector_smult_assoc vector_ssub_ldistrib)
qed

```

```

lemma vector_eq_affinity:
 $(m::'a::field) \neq 0 \implies (y = m * x + c \longleftrightarrow inverse(m) * y + -(inverse(m) * c) = x)$ 
using vector_affinity_eq[where  $m=m$  and  $x=x$  and  $y=y$  and  $c=c$ ]
by metis

```

```

lemma vector_cart:
fixes f ::  $real^n \Rightarrow real$ 
shows  $(\chi\ i. f\ (axis\ i\ 1)) = (\sum\ i \in Basis. f\ i *_{\mathbb{R}} i)$ 
unfolding euclidean_eq_iff[where  $'a=real^n$ ]
by simp (simp add: Basis_vec_def inner_axis)

```

```

lemma const_vector_cart:  $((\chi\ i. d)::real^n) = (\sum\ i \in Basis. d *_{\mathbb{R}} i)$ 
by (rule vector_cart)

```

1.9.8 Explicit formulas for low dimensions

```

lemma prod_neutral_const:  $prod\ f\ \{(1::nat)..1\} = f\ 1$ 
by simp

```

```

lemma prod_2:  $prod\ f\ \{(1::nat)..2\} = f\ 1 * f\ 2$ 
by (simp add: eval_nat_numeral atLeastAtMostSuc_conv mult.commute)

```

```

lemma prod_3:  $prod\ f\ \{(1::nat)..3\} = f\ 1 * f\ 2 * f\ 3$ 
by (simp add: eval_nat_numeral atLeastAtMostSuc_conv mult.commute)

```

1.9.9 Orthogonality of a matrix

definition *orthogonal_matrix* ($Q :: 'a :: \text{semiring}_1 \wedge 'n \wedge 'n$) $\longleftrightarrow$
 $\text{transpose } Q ** Q = \text{mat } 1 \wedge Q ** \text{transpose } Q = \text{mat } 1$

lemma *orthogonal_matrix*: *orthogonal_matrix* ($Q :: \text{real } \wedge 'n \wedge 'n$) $\longleftrightarrow$ *transpose* $Q ** Q = \text{mat } 1$
by (*metis matrix_left_right_inverse orthogonal_matrix_def*)

lemma *orthogonal_matrix_id*: *orthogonal_matrix* ($\text{mat } 1 :: \wedge 'n \wedge 'n$)
by (*simp add: orthogonal_matrix_def*)

proposition *orthogonal_matrix_mul*:
fixes $A :: \text{real } \wedge 'n \wedge 'n$
assumes *orthogonal_matrix* A *orthogonal_matrix* B
shows *orthogonal_matrix*($A ** B$)
using *assms*
by (*simp add: orthogonal_matrix matrix_transpose_mul matrix_left_right_inverse matrix_mul_assoc*)

proposition *orthogonal_transformation_matrix*:
fixes $f :: \text{real } \wedge 'n \Rightarrow \text{real } \wedge 'n$
shows *orthogonal_transformation* $f \longleftrightarrow$ *linear* $f \wedge$ *orthogonal_matrix*(*matrix* f)
(is ?lhs $\longleftrightarrow$?rhs)

proof –
let $?mf = \text{matrix } f$
let $?ot = \text{orthogonal_transformation } f$
let $?U = \text{UNIV} :: 'n \text{ set}$
have fU : *finite* $?U$ **by** *simp*
let $?m1 = \text{mat } 1 :: \text{real } \wedge 'n \wedge 'n$
{
assume ot : $?ot$
from ot **have** lf : *Vector_Spaces.linear* $(*) (s) f$ **and** fd : $\bigwedge v w. f v \cdot f w = v \cdot w$
unfolding *orthogonal_transformation_def orthogonal_matrix linear_def scalar_mult_eq_scaleR*
by *blast+*
{
fix $i j$
let $?A = \text{transpose } ?mf ** ?mf$
have $th0$: $\bigwedge (x :: 'a :: \text{comm\_ring}_1). (if\ b\ \text{then}\ 1\ \text{else}\ 0) * x = (if\ b\ \text{then}\ x\ \text{else}\ 0)$
 $\bigwedge (x :: 'a :: \text{comm\_ring}_1). x * (if\ b\ \text{then}\ 1\ \text{else}\ 0) = (if\ b\ \text{then}\ x\ \text{else}\ 0)$
by *simp_all*
from fd [of *axis* i 1 *axis* j 1,
 simplified matrix_works [OF lf , *symmetric*] *dot_matrix_vector_mul*]
have $?A \$ i \$ j = ?m1 \$ i \$ j$
by (*simp add: inner_vec_def matrix_matrix_mult_def columnvector_def rowvector_def*
 $th0$ *sum.delta* [OF fU] *mat_def axis_def*)
}
}

```

    }
    then have orthogonal_matrix ?mf
      unfolding orthogonal_matrix
      by vector
    with lf have ?rhs
      unfolding linear_def scalar_mult_eq_scaleR
      by blast
  }
  moreover
  {
    assume lf: Vector_Spaces.linear (*s) (*s) f and om: orthogonal_matrix ?mf
    from lf om have ?lhs
      unfolding orthogonal_matrix_def norm_eq orthogonal_transformation
      apply (simp only: matrix_works[OF lf, symmetric] dot_matrix_vector_mul)
      apply (simp add: dot_matrix_product linear_def scalar_mult_eq_scaleR)
      done
  }
  ultimately show ?thesis
    by (auto simp: linear_def scalar_mult_eq_scaleR)
qed

```

1.9.10 Finding an Orthogonal Matrix

We can find an orthogonal matrix taking any unit vector to any other.

lemma *orthogonal_matrix_transpose* [simp]:
 $\text{orthogonal\_matrix}(\text{transpose } A) \longleftrightarrow \text{orthogonal\_matrix } A$
 by (auto simp: orthogonal_matrix_def)

lemma *orthogonal_matrix_orthonormal_columns*:
 fixes $A :: \text{real}^n \times \text{real}^n$
 shows $\text{orthogonal\_matrix } A \longleftrightarrow$
 $(\forall i. \text{norm}(\text{column } i \ A) = 1) \wedge$
 $(\forall i \ j. i \neq j \longrightarrow \text{orthogonal}(\text{column } i \ A) (\text{column } j \ A))$
 by (auto simp: orthogonal_matrix_def matrix_mult_transpose_dot_column vec_eq_iff
 mat_def norm_eq_1 orthogonal_def)

lemma *orthogonal_matrix_orthonormal_rows*:
 fixes $A :: \text{real}^n \times \text{real}^n$
 shows $\text{orthogonal\_matrix } A \longleftrightarrow$
 $(\forall i. \text{norm}(\text{row } i \ A) = 1) \wedge$
 $(\forall i \ j. i \neq j \longrightarrow \text{orthogonal}(\text{row } i \ A) (\text{row } j \ A))$
 using orthogonal_matrix_orthonormal_columns [of transpose A] by simp

proposition *orthogonal_matrix_exists_basis*:
 fixes $a :: \text{real}^n$
 assumes $\text{norm } a = 1$
 obtains A where $\text{orthogonal\_matrix } A$ $A * v$ (axis k 1) $= a$
 proof –
 obtain S where $a \in S$ pairwise orthogonal S and $\text{noS}: \bigwedge x. x \in S \implies \text{norm } x$

```

= 1
  and independent S card S = CARD('n) span S = UNIV
  using vector_in_orthonormal_basis assms by force
  then obtain f0 where bij_betw f0 (UNIV::'n set) S
  by (metis finite_class.finite_UNIV finite_same_card_bij finiteI_independent)
  then obtain f where f: bij_betw f (UNIV::'n set) S and a: a = f k
  using bij_swap_iff [of f0 k inv f0 a]
  by (metis UNIV_I 'a ∈ S' bij_betw_inv_into_right bij_betw_swap_iff swap_apply(1))
  show thesis
  proof
    have [simp]:  $\bigwedge i. \text{norm } (f\ i) = 1$ 
    using bij_betwE [OF 'bij_betw f UNIV S'] by (blast intro: noS)
    have [simp]:  $\bigwedge i\ j. i \neq j \implies \text{orthogonal } (f\ i) (f\ j)$ 
    using 'pairwise orthogonal S' 'bij_betw f UNIV S'
    by (auto simp: pairwise_def bij_betw_def inj_on_def)
    show orthogonal_matrix ( $\chi\ i\ j. f\ j\ \$\ i$ )
    by (simp add: orthogonal_matrix_orthonormal_columns column_def)
    show ( $\chi\ i\ j. f\ j\ \$\ i$ ) *v axis k 1 = a
    by (simp add: matrix_vector_mult_def axis_def a_if_distrib cong: if_cong)
  qed
qed

```

```

lemma orthogonal_transformation_exists_1:
  fixes a b :: real^'n
  assumes norm a = 1 norm b = 1
  obtains f where orthogonal_transformation f f a = b
  proof -
    obtain k::'n where True
    by simp
    obtain A B where AB: orthogonal_matrix A orthogonal_matrix B and eq: A
    *v (axis k 1) = a B *v (axis k 1) = b
    using orthogonal_matrix_exists_basis assms by metis
    let ?f =  $\lambda x. (B ** \text{transpose } A) *v\ x$ 
    show thesis
    proof
      show orthogonal_transformation ?f
      by (subst orthogonal_transformation_matrix)
      (auto simp: AB orthogonal_matrix_mul)
    next
      show ?f a = b
      using 'orthogonal_matrix A' unfolding orthogonal_matrix_def
      by (metis eq matrix_mul_rid matrix_vector_mul_assoc)
    qed
  qed

```

```

proposition orthogonal_transformation_exists:
  fixes a b :: real^'n
  assumes norm a = norm b
  obtains f where orthogonal_transformation f f a = b

```



```

proof (cases  $a = 0 \vee b = 0$ )
  case True
    with assms show ?thesis
    using that by force
next
  case False
    then obtain f where f: orthogonal_transformation f and eq:  $f (a /_R \text{norm } a)$ 
     $= (b /_R \text{norm } b)$ 
    by (auto intro: orthogonal_transformation_exists_1 [of  $a /_R \text{norm } a$   $b /_R \text{norm } b$ ])
    show ?thesis
    proof
      interpret linear f
      using f by (simp add: orthogonal_transformation_linear)
      have  $f a /_R \text{norm } a = f (a /_R \text{norm } a)$ 
      by (simp add: scale)
      also have  $\dots = b /_R \text{norm } a$ 
      by (simp add: eq assms [symmetric])
      finally show  $f a = b$ 
      using False by auto
    qed (use f in auto)
  qed

```

1.9.11 Scaling and isometry

proposition *scaling_linear*:

fixes $f :: 'a::\text{real\_inner} \Rightarrow 'a::\text{real\_inner}$
assumes $f0: f\ 0 = 0$
and $fd: \forall x\ y. \text{dist } (f\ x) (f\ y) = c * \text{dist } x\ y$
shows *linear* *f*

proof –

```

{
  fix v w
  have  $\text{norm } (f\ x) = c * \text{norm } x$  for x
  by (metis dist_0_norm f0 fd)
  then have  $f\ v \cdot f\ w = c^2 * (v \cdot w)$ 
  unfolding dot_norm_neg dist_norm [symmetric]
  by (simp add: fd power2_eq_square field_simps)
}
then show ?thesis
unfolding linear_iff vector_eq [where 'a'=a] scalar_mult_eq_scaleR
by (simp add: inner_add field_simps)

```

qed

lemma *isometry_linear*:

$f (0::'a::\text{real\_inner}) = (0::'a) \implies \forall x\ y. \text{dist}(f\ x) (f\ y) = \text{dist } x\ y \implies \text{linear } f$
by (*rule* *scaling_linear* [**where** $c=1$]) *simp_all*

Hence another formulation of orthogonal transformation

proposition *orthogonal_transformation_isometry:*

orthogonal_transformation $f \longleftrightarrow f(0::'a::\text{real\_inner}) = (0::'a) \wedge (\forall x\ y. \text{dist}(f\ x)\ (f\ y) = \text{dist}\ x\ y)$
unfolding *orthogonal_transformation*
apply (*auto simp: linear_0 isometry_linear*)
apply (*metis (no_types, opaque_lifting) dist_norm linear_diff*)
by (*metis dist_0_norm*)

Can extend an isometry from unit sphere:

lemma *isometry_sphere_extend:*

fixes $f::'a::\text{real\_inner} \Rightarrow 'a$
assumes $f1: \bigwedge x. \text{norm}\ x = 1 \implies \text{norm}\ (f\ x) = 1$
and $fd1: \bigwedge x\ y. \llbracket \text{norm}\ x = 1; \text{norm}\ y = 1 \rrbracket \implies \text{dist}\ (f\ x)\ (f\ y) = \text{dist}\ x\ y$
shows $\exists g. \text{orthogonal\_transformation}\ g \wedge (\forall x. \text{norm}\ x = 1 \longrightarrow g\ x = f\ x)$
proof –
{
fix $x\ y\ x'\ y'\ u\ v\ u'\ v'::'a$
assume $H: x = \text{norm}\ x *_R u\ y = \text{norm}\ y *_R v$
 $x' = \text{norm}\ x *_R u'\ y' = \text{norm}\ y *_R v'$
and $J: \text{norm}\ u = 1\ \text{norm}\ u' = 1\ \text{norm}\ v = 1\ \text{norm}\ v' = 1\ \text{norm}(u' - v') = \text{norm}(u - v)$
then have $*$: $u \cdot v = u' \cdot v' + v' \cdot u' - v \cdot u$
by (*simp add: norm_eq norm_eq_1 inner_add inner_diff*)
have $\text{norm}\ (\text{norm}\ x *_R u' - \text{norm}\ y *_R v') = \text{norm}\ (\text{norm}\ x *_R u - \text{norm}\ y *_R v)$
using J **by** (*simp add: norm_eq norm_eq_1 inner_diff * field_simps*)
then have $\text{norm}(x' - y') = \text{norm}(x - y)$
using H **by** *metis*
}
note $\text{norm\_eq} = \text{this}$
let $?g = \lambda x. \text{if } x = 0 \text{ then } 0 \text{ else } \text{norm}\ x *_R f\ (x /_R \text{norm}\ x)$
have *thfg*: $?g\ x = f\ x$ **if** $\text{norm}\ x = 1$ **for** x
using *that* **by** *auto*
have *thd*: $\text{dist}\ (?g\ x)\ (?g\ y) = \text{dist}\ x\ y$ **for** $x\ y$
proof (*cases* $x=0 \vee y=0$)
case *False*
show $\text{dist}\ (?g\ x)\ (?g\ y) = \text{dist}\ x\ y$
unfolding *dist_norm*
proof (*rule norm_eq*)
show $x = \text{norm}\ x *_R (x /_R \text{norm}\ x)\ y = \text{norm}\ y *_R (y /_R \text{norm}\ y)$
 $\text{norm}\ (f\ (x /_R \text{norm}\ x)) = 1\ \text{norm}\ (f\ (y /_R \text{norm}\ y)) = 1$
using *False f1* **by** *auto*
qed (*use False in <auto simp: field_simps intro: f1 fd1[unfolded dist_norm]>*)
qed (*auto simp: f1*)
show *?thesis*
unfolding *orthogonal_transformation_isometry*
by (*rule exI[where* $x = ?g$ *]) (metis thfg thd)*
qed

1.9.12 Induction on matrix row operations

lemma *induct_matrix_row_operations*:

fixes $P :: \text{real}^n \Rightarrow \text{bool}$

assumes *zero_row*: $\bigwedge A \ i. \text{row } i \ A = 0 \implies P \ A$

and *diagonal*: $\bigwedge A. (\bigwedge i \ j. i \neq j \implies A[i][j] = 0) \implies P \ A$

and *swap_cols*: $\bigwedge A \ m \ n. \llbracket P \ A; \ m \neq n \rrbracket \implies P(\chi \ i \ j. A \ \$ \ i \ \$ \ \text{Transposition.transpose } m \ n \ j)$

and *row_op*: $\bigwedge A \ m \ n \ c. \llbracket P \ A; \ m \neq n \rrbracket$

$\implies P(\chi \ i. \text{if } i = m \text{ then row } m \ A + c *_R \text{ row } n \ A \text{ else row } i \ A)$

shows $P \ A$

proof –

have $P \ A$ **if** $(\bigwedge i \ j. \llbracket j \in -K; \ i \neq j \rrbracket \implies A[i][j] = 0)$ **for** $A \ K$

proof –

have *finite K*

by *simp*

then show *?thesis* **using** *that*

proof (*induction arbitrary: A rule: finite_induct*)

case *empty*

with *diagonal* **show** *?case*

by *simp*

next

case (*insert k K*)

note *insertK = insert*

have $P \ A$ **if** $kk: A[k][k] \neq 0$

and $0: \bigwedge i \ j. \llbracket j \in - \text{insert } k \ K; \ i \neq j \rrbracket \implies A[i][j] = 0$

$\bigwedge i. \llbracket i \in -L; \ i \neq k \rrbracket \implies A[i][k] = 0$ **for** $A \ L$

proof –

have *finite L*

by *simp*

then show *?thesis* **using** $0 \ kk$

proof (*induction arbitrary: A rule: finite_induct*)

case (*empty B*)

show *?case*

proof (*rule insertK*)

fix $i \ j$

assume $i \in - \ K \ j \neq i$

show $B \ \$ \ j \ \$ \ i = 0$

using $\langle j \neq i \rangle \langle i \in - \ K \rangle$ *empty*

by (*metis ComplD ComplI Compl_eq_Diff_UNIV Diff_empty UNIV_I*

insert_iff)

qed

next

case (*insert l L B*)

show *?case*

proof (*cases k = l*)

case *True*

with *insert* **show** *?thesis*

by *auto*

next

```

      case False
      let ?C =  $\chi$  i. if  $i = l$  then  $\text{row } l \ B - (B \$ l \$ k / B \$ k \$ k) *_R \text{row } k$ 
B else row i B
      have 1:  $\llbracket j \in - \text{insert } k \ K; i \neq j \rrbracket \implies ?C \$ i \$ j = 0$  for  $j \ i$ 
      by (auto simp: insert.prem(1) row_def)
      have 2:  $?C \$ i \$ k = 0$ 
      if  $i \in - L \ i \neq k$  for  $i$ 
      proof (cases  $i=l$ )
      case True
      with that insert.prem show ?thesis
      by (simp add: row_def)
      next
      case False
      with that show ?thesis
      by (simp add: insert.prem(2) row_def)
      qed
      have 3:  $?C \$ k \$ k \neq 0$ 
      by (auto simp: insert.prem row_def  $\langle k \neq l \rangle$ )
      have PC:  $P \ ?C$ 
      using insert.IH [OF 1 2 3] by auto
      have eqB:  $(\chi \ i. \text{if } i = l \text{ then } \text{row } l \ ?C + (B \$ l \$ k / B \$ k \$ k) *_R \text{row } k \ ?C \text{ else } \text{row } i \ ?C) = B$ 
      using  $\langle k \neq l \rangle$  by (simp add: vec_eq_iff row_def)
      show ?thesis
      using row_op [OF PC, of  $l \ k$ , where  $c = B \$ l \$ k / B \$ k \$ k$ ] eqB  $\langle k \neq l \rangle$ 
      by (simp add: cong: if_cong)
      qed
    qed
  qed
  then have nonzero_hyp:  $P \ A$ 
  if  $kk: A \$ k \$ k \neq 0$  and zeroes:  $\bigwedge i \ j. j \in - \text{insert } k \ K \wedge i \neq j \implies A \$ i \$ j = 0$ 
for A
  by (auto simp: intro!: kk zeroes)
  show ?case
  proof (cases  $\text{row } k \ A = 0$ )
  case True
  with zero_row show ?thesis by auto
  next
  case False
  then obtain  $l$  where  $l: A \$ k \$ l \neq 0$ 
  by (auto simp: row_def zero_vec_def vec_eq_iff)
  show ?thesis
  proof (cases  $k = l$ )
  case True
  with  $l$  nonzero_hyp insert.prem show ?thesis
  by blast
  next
  case False
  have *:  $A \$ i \$ \text{Transposition.transpose } k \ l \ j = 0$  if  $j \neq k \ j \notin K \ i \neq j$  for

```

```

i j
  using False l insert.prems that
  by (auto simp add: Transposition.transpose_def)
  have P ( $\chi$  i j. ( $\chi$  i j. A $ i $ Transposition.transpose k l j) $ i $
Transposition.transpose k l j)
  by (rule swap_cols [OF nonzero_hyp False]) (auto simp: l *)
  moreover
  have ( $\chi$  i j. ( $\chi$  i j. A $ i $ Transposition.transpose k l j) $ i $ Transposi-
tion.transpose k l j) = A
  by simp
  ultimately show ?thesis
  by simp
qed
qed
qed
qed
then show ?thesis
  by blast
qed

```

lemma *induct_matrix_elementary*:

```

fixes P ::  $\text{real}^n \Rightarrow \text{bool}$ 
assumes mult:  $\bigwedge A B. \llbracket P A; P B \rrbracket \implies P(A ** B)$ 
  and zero_row:  $\bigwedge A i. \text{row } i A = 0 \implies P A$ 
  and diagonal:  $\bigwedge A. (\bigwedge i j. i \neq j \implies A\$i\$j = 0) \implies P A$ 
  and swap1:  $\bigwedge m n. m \neq n \implies P(\chi i j. \text{mat } 1 \$ i \$ \text{Transposition.transpose } m$ 
n j)
  and idplus:  $\bigwedge m n c. m \neq n \implies P(\chi i j. \text{if } i = m \wedge j = n \text{ then } c \text{ else of\_bool}$ 
(i = j))
shows P A
proof -
  have swap:  $P(\chi i j. A \$ i \$ \text{Transposition.transpose } m n j)$  (is P ?C)
  if P A m ≠ n for A m n
  proof -
    have  $A ** (\chi i j. \text{mat } 1 \$ i \$ \text{Transposition.transpose } m n j) = ?C$ 
    by (simp add: matrix_matrix_mult_def mat_def vec_eq_iff if_distrib sum.delta_remove)
    then show ?thesis
    using mult swap1 that by metis
  qed
  have row:  $P(\chi i. \text{if } i = m \text{ then row } m A + c *_R \text{row } n A \text{ else row } i A)$  (is P
?C)
  if P A m ≠ n for A m n c
  proof -
    let ?B =  $\chi i j. \text{if } i = m \wedge j = n \text{ then } c \text{ else of\_bool } (i = j)$ 
    have ?B ** A = ?C
    using  $\langle m \neq n \rangle$  unfolding matrix_matrix_mult_def row_def of_bool_def
    by (auto simp: vec_eq_iff if_distrib [of  $\lambda x. x * y$  for y] sum.remove cong
if_cong)
    then show ?thesis

```

```

    by (rule subst) (auto simp: that mult idplus)
  qed
  show ?thesis
    by (rule induct_matrix_row_operations [OF zero_row diagonal swap row])
  qed

lemma induct_matrix_elementary_alt:
  fixes P :: realn ⇒ bool
  assumes mult:  $\bigwedge A B. \llbracket P A; P B \rrbracket \implies P(A ** B)$ 
    and zero_row:  $\bigwedge A i. \text{row } i A = 0 \implies P A$ 
    and diagonal:  $\bigwedge A. (\bigwedge i j. i \neq j \implies A\$i\$j = 0) \implies P A$ 
    and swap1:  $\bigwedge m n. m \neq n \implies P(\chi i j. \text{mat } 1 \$ i \$ \text{Transposition.transpose } m$ 
n j)
    and idplus:  $\bigwedge m n. m \neq n \implies P(\chi i j. \text{of\_bool } (i = m \wedge j = n \vee i = j))$ 
  shows P A
proof -
  have *:  $P(\chi i j. \text{if } i = m \wedge j = n \text{ then } c \text{ else of\_bool } (i = j))$ 
    if  $m \neq n$  for m n c
  proof (cases c = 0)
    case True
      with diagonal show ?thesis by auto
    next
      case False
        then have eq:  $(\chi i j. \text{if } i = m \wedge j = n \text{ then } c \text{ else of\_bool } (i = j)) =$ 
 $(\chi i j. \text{if } i = j \text{ then } (\text{if } j = n \text{ then inverse } c \text{ else } 1) \text{ else } 0) **$ 
 $(\chi i j. \text{of\_bool } (i = m \wedge j = n \vee i = j)) **$ 
 $(\chi i j. \text{if } i = j \text{ then if } j = n \text{ then } c \text{ else } 1 \text{ else } 0)$ 
          using ‹m ≠ n›
        apply (simp add: matrix_matrix_mult_def vec_eq_iff of_bool_def if_distrib
[of  $\lambda x. y * x$  for y] cong: if_cong)
        apply (simp add: if_if_eq_conj sum.neutral conj_commute cong: conj_cong)
        done
      show ?thesis
        apply (subst eq)
        apply (intro mult idplus that)
        apply (auto intro: diagonal)
        done
    qed
  show ?thesis
    by (rule induct_matrix_elementary) (auto intro: assms *)
  qed

```

```

lemma matrix_vector_mult_matrix_matrix_mult_compose:
   $(*v) (A ** B) = (*v) A \circ (*v) B$ 
  by (auto simp: matrix_vector_mul_assoc)

```

```

lemma induct_linear_elementary:
  fixes f :: realn ⇒ realn
  assumes linear f

```

```

and comp:  $\bigwedge f g. \llbracket \text{linear } f; \text{ linear } g; P f; P g \rrbracket \implies P(f \circ g)$ 
and zeroes:  $\bigwedge f i. \llbracket \text{linear } f; \bigwedge x. (f x) \$ i = 0 \rrbracket \implies P f$ 
and const:  $\bigwedge c. P(\lambda x. \chi i. c i * x \$ i)$ 
and swap:  $\bigwedge m n :: 'n. m \neq n \implies P(\lambda x. \chi i. x \$ \text{Transposition.transpose } m n i)$ 
and idplus:  $\bigwedge m n :: 'n. m \neq n \implies P(\lambda x. \chi i. \text{if } i = m \text{ then } x \$ m + x \$ n \text{ else } x \$ i)$ 
shows  $P f$ 
proof -
  have  $P ((*v) A)$  for  $A$ 
  proof (rule induct_matrix_elementary_alt)
    fix  $A B$ 
    assume  $P ((*v) A)$  and  $P ((*v) B)$ 
    then show  $P ((*v) (A ** B))$ 
    by (auto simp add: matrix_vector_mult_matrix_matrix_mult_compose intro!: comp)
  next
    fix  $A :: \text{real}^{n' n}$  and  $i$ 
    assume  $\text{row } i A = 0$ 
    show  $P ((*v) A)$ 
    using matrix_vector_mul_linear
    by (rule zeroes[where  $i=i$ ])
      (metis  $\langle \text{row } i A = 0 \rangle$  inner_zero_left matrix_vector_mul_component row_def vec_lambda_eta)
  next
    fix  $A :: \text{real}^{n' n}$ 
    assume  $0: \bigwedge i j. i \neq j \implies A \$ i \$ j = 0$ 
    have  $A \$ i \$ i * x \$ i = (\sum_{j \in \text{UNIV}} A \$ i \$ j * x \$ j)$  for  $x$  and  $i :: 'n$ 
    by (simp add: 0 comm_monoid_add_class.sum_remove [where  $x=i$ ])
    then have  $(\lambda x. \chi i. A \$ i \$ i * x \$ i) = ((*v) A)$ 
    by (auto simp: 0 matrix_vector_mult_def)
    then show  $P ((*v) A)$ 
    using const [of  $\lambda i. A \$ i \$ i$ ] by simp
  next
    fix  $m n :: 'n$ 
    assume  $m \neq n$ 
    have  $\text{eq}: (\sum_{j \in \text{UNIV}} \text{if } i = \text{Transposition.transpose } m n j \text{ then } x \$ j \text{ else } 0) =$ 
       $(\sum_{j \in \text{UNIV}} \text{if } j = \text{Transposition.transpose } m n i \text{ then } x \$ j \text{ else } 0)$ 
    for  $i$  and  $x :: \text{real}^{n' n}$ 
    by (rule sum.cong) (auto simp add: swap_id_eq)
    have  $(\lambda x :: \text{real}^{n' n}. \chi i. x \$ \text{Transposition.transpose } m n i) = ((*v) (\chi i j. \text{if } i = \text{Transposition.transpose } m n j \text{ then } 1 \text{ else } 0))$ 
    by (auto simp: mat_def matrix_vector_mult_def eq_if_distrib [of  $\lambda x. x * y$  for  $y$ ] cong: if_cong)
    with swap [OF  $\langle m \neq n \rangle$ ] show  $P ((*v) (\chi i j. \text{mat } 1 \$ i \$ \text{Transposition.transpose } m n j))$ 
    by (simp add: mat_def matrix_vector_mult_def)
  next
    fix  $m n :: 'n$ 
    assume  $m \neq n$ 

```

```

    then have  $x \$ m + x \$ n = (\sum_{j \in UNIV}. of\_bool (j = n \vee m = j) * x \$ j)$ 
  for  $x :: real^{n^{\wedge}}$ 
    by (auto simp: of_bool_def if_distrib [of  $\lambda x. x * y$  for  $y$ ] sum.remove cong:
if_cong)
    then have  $(\lambda x :: real^{n^{\wedge}}. \chi i. if i = m then x \$ m + x \$ n else x \$ i) =$ 
       $((*v) (\chi i j. of\_bool (i = m \wedge j = n \vee i = j)))$ 
    unfolding matrix_vector_mult_def of_bool_def
    by (auto simp: vec_eq_iff if_distrib [of  $\lambda x. x * y$  for  $y$ ] cong: if_cong)
    then show  $P ((*v) (\chi i j. of\_bool (i = m \wedge j = n \vee i = j)))$ 
      using idplus [OF  $\langle m \neq n \rangle$ ] by simp
  qed
  then show ?thesis
    by (metis  $\langle linear f \rangle$  matrix_vector_mul(2))
  qed
end

```

1.10 Traces and Determinants of Square Matrices

```

theory Determinants
imports
  HOL-Combinatorics.Permutations
  Cartesian_Space
begin

```

1.10.1 Trace

```

definition trace :: ' $a :: semiring_1^{n^{\wedge} n} \Rightarrow 'a$ '
  where trace  $A = sum (\lambda i. ((A \$ i) \$ i)) (UNIV :: 'n set)$ 

```

```

lemma trace_0: trace (mat 0) = 0
  by (simp add: trace_def mat_def)

```

```

lemma trace_I: trace (mat 1 :: ' $a :: semiring_1^{n^{\wedge} n}$ ') = of_nat (CARD('n))
  by (simp add: trace_def mat_def)

```

```

lemma trace_add: trace (( $A :: 'a :: comm\_semiring_1^{n^{\wedge} n}$ ) + B) = trace A +
  trace B
  by (simp add: trace_def sum.distrib)

```

```

lemma trace_sub: trace (( $A :: 'a :: comm\_ring_1^{n^{\wedge} n}$ ) - B) = trace A - trace B
  by (simp add: trace_def sum_subtractf)

```

```

lemma trace_mul_sym: trace (( $A :: 'a :: comm\_semiring_1^{n^{\wedge} m}$ ) ** B) = trace
  (B ** A)
  apply (simp add: trace_def matrix_matrix_mult_def)
  apply (subst sum.swap)
  apply (simp add: mult.commute)
  done

```


Definition of determinant

definition $\text{det} :: 'a :: \text{comm\_ring\_1} \Rightarrow 'a$ **where**

$\text{det } A =$
 $\text{sum } (\lambda p. \text{of\_int } (\text{sign } p) * \text{prod } (\lambda i. A \$ i \$ p \ i) \ (\text{UNIV} :: 'n \text{ set}))$
 $\{p. p \text{ permutes } (\text{UNIV} :: 'n \text{ set})\}$

Basic determinant properties

lemma $\text{det\_transpose} [\text{simp}]: \text{det } (\text{transpose } A) = \text{det } (A :: 'a :: \text{comm\_ring\_1} \Rightarrow 'a)$

proof –

let $?di = \lambda A \ i \ j. A \$ i \$ j$
let $?U = (\text{UNIV} :: 'n \text{ set})$
have $fU: \text{finite } ?U$ **by** simp
{
 fix p
 assume $p: p \in \{p. p \text{ permutes } ?U\}$
 from p **have** $pU: p \text{ permutes } ?U$
 by blast
 have $\text{sth}: \text{sign } (\text{inv } p) = \text{sign } p$
 by $(\text{metis } \text{sign\_inverse } fU \ p \ \text{mem\_Collect\_eq } \text{permutation\_permutes})$
 from $\text{permutes\_inj} [OF \ pU]$
 have $pi: \text{inj\_on } p \ ?U$
 by $(\text{blast } \text{intro: subset\_inj\_on})$
 from $\text{permutes\_image} [OF \ pU]$
 have $\text{prod } (\lambda i. ?di \ (\text{transpose } A) \ i \ (\text{inv } p \ i)) \ ?U =$
 $\text{prod } (\lambda i. ?di \ (\text{transpose } A) \ i \ (\text{inv } p \ i)) \ (p \text{ ` } ?U)$
 by simp
 also have $\dots = \text{prod } ((\lambda i. ?di \ (\text{transpose } A) \ i \ (\text{inv } p \ i)) \circ p) \ ?U$
 unfolding $\text{prod.reindex} [OF \ pi] \dots$
 also have $\dots = \text{prod } (\lambda i. ?di \ A \ i \ (p \ i)) \ ?U$
 proof –
 have $((\lambda i. ?di \ (\text{transpose } A) \ i \ (\text{inv } p \ i)) \circ p) \ i = ?di \ A \ i \ (p \ i)$ **if** $i \in ?U$ **for** i
 using $\text{that } \text{permutes\_inv\_o} [OF \ pU] \ \text{permutes\_in\_image} [OF \ pU]$
 unfolding transpose_def **by** $(\text{simp } \text{add: fun\_eq\_iff})$
 then show $\text{prod } ((\lambda i. ?di \ (\text{transpose } A) \ i \ (\text{inv } p \ i)) \circ p) \ ?U = \text{prod } (\lambda i. ?di \ A \ i \ (p \ i)) \ ?U$
 by $(\text{auto } \text{intro: prod.cong})$
 qed
 finally have $\text{of\_int } (\text{sign } (\text{inv } p)) * (\text{prod } (\lambda i. ?di \ (\text{transpose } A) \ i \ (\text{inv } p \ i)) \ ?U) =$
 $\text{of\_int } (\text{sign } p) * (\text{prod } (\lambda i. ?di \ A \ i \ (p \ i)) \ ?U)$
 using sth **by** simp
}
then show $?thesis$
 unfolding det_def
 by $(\text{subst } \text{sum\_permutations\_inverse}) \ (\text{blast } \text{intro: sum.cong})$
qed

lemma $\text{det\_lowerdiagonal}:$

fixes $A :: 'a :: \text{comm\_ring\_1} \Rightarrow ('n :: \{\text{finite}, \text{wellorder}\}) \Rightarrow ('n :: \{\text{finite}, \text{wellorder}\})$

```

    assumes  $ld: \bigwedge i j. i < j \implies A\$i\$j = 0$ 
    shows  $\det A = \text{prod } (\lambda i. A\$i\$i) \text{ (UNIV:: 'n set)}$ 
  proof -
    let ?U = UNIV:: 'n set
    let ?PU = {p. p permutes ?U}
    let ?pp =  $\lambda p. \text{of\_int } (\text{sign } p) * \text{prod } (\lambda i. A\$i\$p \ i) \text{ (UNIV :: 'n set)}$ 
    have fU: finite ?U
    by simp
    have id0: {id}  $\subseteq$  ?PU
    by (auto simp: permutes_id)
    have p0:  $\forall p \in ?PU - \{id\}. ?pp \ p = 0$ 
    proof
      fix p
      assume  $p \in ?PU - \{id\}$ 
      then obtain i where  $i: p \ i > i$ 
      by clarify (meson leI permutes_natset_le)
      from ld[OF i] have  $\exists i \in ?U. A\$i\$p \ i = 0$ 
      by blast
      with prod_zero[OF fU] show ?pp p = 0
      by force
    qed
    from sum.mono_neutral_cong_left[OF finite_permutations[OF fU] id0 p0] show
    ?thesis
    unfolding det_def by (simp add: sign_id)
  qed

lemma det_upperdiagonal:
  fixes A :: 'a::comm_ring_1 ^ n::{finite, wellorder} ^ n::{finite, wellorder}
  assumes  $ld: \bigwedge i j. i > j \implies A\$i\$j = 0$ 
  shows  $\det A = \text{prod } (\lambda i. A\$i\$i) \text{ (UNIV:: 'n set)}$ 
  proof -
    let ?U = UNIV:: 'n set
    let ?PU = {p. p permutes ?U}
    let ?pp =  $(\lambda p. \text{of\_int } (\text{sign } p) * \text{prod } (\lambda i. A\$i\$p \ i) \text{ (UNIV :: 'n set)})$ 
    have fU: finite ?U
    by simp
    have id0: {id}  $\subseteq$  ?PU
    by (auto simp: permutes_id)
    have p0:  $\forall p \in ?PU - \{id\}. ?pp \ p = 0$ 
    proof
      fix p
      assume  $p: p \in ?PU - \{id\}$ 
      then obtain i where  $i: p \ i < i$ 
      by clarify (meson leI permutes_natset_ge)
      from ld[OF i] have  $\exists i \in ?U. A\$i\$p \ i = 0$ 
      by blast
      with prod_zero[OF fU] show ?pp p = 0
      by force
    qed
  qed

```

```

from sum.mono_neutral_cong_left[OF finite_permutations[OF fU] id0 p0] show
?thesis

```

```

  unfolding det_def by (simp add: sign_id)
qed

```

proposition *det_diagonal*:

```

fixes A :: 'a::comm_ring_1'^n'^n
assumes ld:  $\bigwedge i j. i \neq j \implies A\$i\$j = 0$ 
shows det A = prod ( $\lambda i. A\$i\$i$ ) (UNIV::'n set)

```

proof –

```

let ?U = UNIV::'n set
let ?PU = {p. p permutes ?U}
let ?pp =  $\lambda p. \text{of\_int } (\text{sign } p) * \text{prod } (\lambda i. A\$i\$p \ i) \ (\text{UNIV} :: 'n \text{ set})$ 
have fU: finite ?U by simp
from finite_permutations[OF fU] have fPU: finite ?PU .
have id0: {id}  $\subseteq$  ?PU

```

```

  by (auto simp: permutes_id)
have p0:  $\forall p \in ?PU - \{id\}. ?pp \ p = 0$ 

```

proof

```

  fix p
  assume p: p  $\in ?PU - \{id\}$ 
  then obtain i where i: p i  $\neq i$ 
    by fastforce
  with ld have  $\exists i \in ?U. A\$i\$p \ i = 0$ 
    by (metis UNIV_I)
  with prod_zero [OF fU] show ?pp p = 0
    by force

```

qed

```

from sum.mono_neutral_cong_left[OF fPU id0 p0] show ?thesis
  unfolding det_def by (simp add: sign_id)

```

qed

```

lemma det_I [simp]: det (mat 1 :: 'a::comm_ring_1'^n'^n) = 1
  by (simp add: det_diagonal mat_def)

```

```

lemma det_0 [simp]: det (mat 0 :: 'a::comm_ring_1'^n'^n) = 0
  by (simp add: det_def prod_zero power_0_left)

```

lemma *det_permute_rows*:

```

fixes A :: 'a::comm_ring_1'^n'^n
assumes p: p permutes (UNIV::'n::finite set)
shows det ( $\chi \ i. A\$p \ i :: 'a'^n'^n$ ) = of_int (sign p) * det A

```

proof –

```

let ?U = UNIV::'n set
let ?PU = {p. p permutes ?U}
have *: ( $\sum q \in ?PU. \text{of\_int } (\text{sign } (q \circ p)) * (\prod i \in ?U. A \$ p \ i \$ (q \circ p) \ i)$ ) =
  ( $\sum n \in ?PU. \text{of\_int } (\text{sign } p) * \text{of\_int } (\text{sign } n) * (\prod i \in ?U. A \$ i \$ n \ i)$ )
proof (rule sum.cong)
  fix q

```

```

    assume qPU: q ∈ ?PU
    have fU: finite ?U
    by simp
    from qPU have q: q permutes ?U
    by blast
    have prod (λi. A$p i$ (q ∘ p) i) ?U = prod ((λi. A$p i$(q ∘ p) i) ∘ inv p) ?U
    by (simp only: prod.permute[OF permutes_inv[OF p], symmetric])
    also have ... = prod (λi. A $ (p ∘ inv p) i $ (q ∘ (p ∘ inv p)) i) ?U
    by (simp only: o_def)
    also have ... = prod (λi. A$i$q i) ?U
    by (simp only: o_def permutes_inverses[OF p])
    finally have thp: prod (λi. A$p i$ (q ∘ p) i) ?U = prod (λi. A$i$q i) ?U
    by blast
    from p q have pp: permutation p and qp: permutation q
    by (metis fU permutation_permutes)+
    show of_int (sign (q ∘ p)) * prod (λi. A$ p i$ (q ∘ p) i) ?U =
      of_int (sign p) * of_int (sign q) * prod (λi. A$i$q i) ?U
    by (simp only: thp sign_compose[OF qp pp] mult.commute of_int_mult)
qed auto
show ?thesis
  apply (simp add: det_def sum_distrib_left mult.assoc[symmetric])
  apply (subst sum_permutations_compose_right[OF p])
  apply (rule *)
  done
qed

lemma det_permute_columns:
  fixes A :: 'a::comm_ring_1^~n^~n
  assumes p: p permutes (UNIV :: 'n set)
  shows det(χ i j. A$i$ p j :: 'a^~n^~n) = of_int (sign p) * det A
proof -
  let ?Ap = χ i j. A$i$ p j :: 'a^~n^~n
  let ?At = transpose A
  have of_int (sign p) * det A = det (transpose (χ i. transpose A $ p i))
    unfolding det_permute_rows[OF p, of ?At] det_transpose ..
  moreover
  have ?Ap = transpose (χ i. transpose A $ p i)
    by (simp add: transpose_def vec_eq_iff)
  ultimately show ?thesis
    by simp
qed

lemma det_identical_columns:
  fixes A :: 'a::comm_ring_1^~n^~n
  assumes jk: j ≠ k
    and r: column j A = column k A
  shows det A = 0
proof -
  let ?U = UNIV :: 'n set

```

```

let ?t_jk = Transposition.transpose j k
let ?PU = {p. p permutes ?U}
let ?S1 = {p. p ∈ ?PU ∧ evenperm p}
let ?S2 = {(?t_jk ∘ p) | p. p ∈ ?S1}
let ?f = λp. of_int (sign p) * (∏ i ∈ UNIV. A $ i $ p i)
let ?g = λp. ?t_jk ∘ p
have g_S1: ?S2 = ?g' ?S1 by auto
have inj_g: inj_on ?g ?S1
proof (unfold inj_on_def, auto)
  fix x y assume x: x permutes ?U and even_x: evenperm x
  and y: y permutes ?U and even_y: evenperm y and eq: ?t_jk ∘ x = ?t_jk
  ∘ y
  show x = y by (metis (opaque_lifting, no_types) comp_assoc eq_id_comp
  swap_id_idempotent)
qed
have tjk_permutes: ?t_jk permutes ?U
  by (auto simp add: permutes_def dest: transpose_eq_imp_eq) (meson trans-
  pose_involutory)
have tjk_eq: ∀ i l. A $ i $ ?t_jk l = A $ i $ l
  using r_jk
  unfolding column_def vec_eq_iff by (simp add: Transposition.transpose_def)

have sign_tjk: sign ?t_jk = -1 using sign_swap_id[of j k] jk by auto
{fix x
  assume x: x ∈ ?S1
  have sign (?t_jk ∘ x) = sign (?t_jk) * sign x
    by (metis (lifting) finite_class.finite_UNIV mem_Collect_eq
    permutation_permutes permutation_swap_id sign_compose x)
  also have ... = - sign x using sign_tjk by simp
  also have ... ≠ sign x unfolding sign_def by simp
  finally have sign (?t_jk ∘ x) ≠ sign x and (?t_jk ∘ x) ∈ ?S2
    using x by force+
}
hence disjoint: ?S1 ∩ ?S2 = {}
  by (force simp: sign_def)
have PU_decomposition: ?PU = ?S1 ∪ ?S2
proof (auto)
  fix x
  assume x: x permutes ?U and ∀ p. p permutes ?U ⟶ x = Transposi-
  tion.transpose j k ∘ p ⟶ ¬ evenperm p
  then obtain p where p: p permutes UNIV and x_eq: x = Transposition.transpose
  j k ∘ p
  and odd_p: ¬ evenperm p
  by (metis (mono_tags) id_o_o_assoc permutes_compose swap_id_idempotent
  tjk_permutes)
  thus evenperm x
  by (meson evenperm_comp evenperm_swap finite_class.finite_UNIV
  jk_permutation_permutes permutation_swap_id)
next

```

```

    fix p assume p: p permutes ?U
    show Transposition.transpose j k ∘ p permutes UNIV by (metis p permutes_compose
tjk_permutes)
  qed
  have sum ?f ?S2 = sum ((λp. of_int (sign p) * (∏ i∈UNIV. A $ i $ p i))
    ∘ (∘) (Transposition.transpose j k)) {p ∈ {p. p permutes UNIV}. evenperm p}
    unfolding g_S1 by (rule sum.reindex[OF inj_g])
  also have ... = sum (λp. of_int (sign (?t_jk ∘ p)) * (∏ i∈UNIV. A $ i $ p i))
    ?S1
    unfolding o_def by (rule sum.cong, auto simp: tjk_eq)
  also have ... = sum (λp. - ?f p) ?S1
  proof (rule sum.cong, auto)
    fix x assume x: x permutes ?U
    and even_x: evenperm x
    hence perm_x: permutation x and perm_tjk: permutation ?t_jk
    using permutation_permutes[of x] permutation_permutes[of ?t_jk] permuta-
tion_swap_id
    by (metis finite_code)+
    have (sign (?t_jk ∘ x)) = - (sign x)
    unfolding sign_compose[OF perm_tjk perm_x] sign_tjk by auto
    thus of_int (sign (?t_jk ∘ x)) * (∏ i∈UNIV. A $ i $ x i)
      = - (of_int (sign x) * (∏ i∈UNIV. A $ i $ x i))
    by auto
  qed
  also have ... = - sum ?f ?S1 unfolding sum_negf ..
  finally have *: sum ?f ?S2 = - sum ?f ?S1 .
  have det A = (∑ p | p permutes UNIV. of_int (sign p) * (∏ i∈UNIV. A $ i $
p i))
    unfolding det_def ..
  also have ... = sum ?f ?S1 + sum ?f ?S2
    by (subst PU_decomposition, rule sum.union_disjoint[OF __ disjoint], auto)
  also have ... = sum ?f ?S1 - sum ?f ?S1 unfolding * by auto
  also have ... = 0 by simp
  finally show det A = 0 by simp
qed

lemma det_identical_rows:
  fixes A :: 'a::comm_ring_1 ^ n ^ n
  assumes ij: i ≠ j and r: row i A = row j A
  shows det A = 0
  by (metis column_transpose det_identical_columns det_transpose ij r)

lemma det_zero_row:
  fixes A :: 'a::{idom, ring_char_0} ^ n ^ n and F :: 'b::{field} ^ m ^ m
  shows row i A = 0 ⟹ det A = 0 and row j F = 0 ⟹ det F = 0
  by (force simp: row_def det_def vec_eq_iff sign_nz intro!: sum.neutral)+

lemma det_zero_column:
  fixes A :: 'a::{idom, ring_char_0} ^ n ^ n and F :: 'b::{field} ^ m ^ m

```

shows $\text{column } i \ A = 0 \implies \det A = 0$ **and** $\text{column } j \ F = 0 \implies \det F = 0$
unfolding *atomize_conj atomize_imp*
by (*metis det_transpose det_zero_row row_transpose*)

lemma *det_row_add*:

fixes $a \ b \ c :: 'n::\text{finite} \Rightarrow \_ \wedge 'n$

shows $\det((\lambda i. \text{if } i = k \text{ then } a \ i + b \ i \text{ else } c \ i) :: 'a::\text{comm\_ring}_1 \wedge 'n) =$
 $\det((\lambda i. \text{if } i = k \text{ then } a \ i \text{ else } c \ i) :: 'a::\text{comm\_ring}_1 \wedge 'n) +$
 $\det((\lambda i. \text{if } i = k \text{ then } b \ i \text{ else } c \ i) :: 'a::\text{comm\_ring}_1 \wedge 'n)$

unfolding *det_def vec_lambda_beta sum.distrib[symmetric]*

proof (*rule sum.cong*)

let $?U = \text{UNIV} :: 'n \text{ set}$

let $?pU = \{p. p \text{ permutes } ?U\}$

let $?f = (\lambda i. \text{if } i = k \text{ then } a \ i + b \ i \text{ else } c \ i) :: 'n \Rightarrow 'a::\text{comm\_ring}_1 \wedge 'n$

let $?g = (\lambda i. \text{if } i = k \text{ then } a \ i \text{ else } c \ i) :: 'n \Rightarrow 'a::\text{comm\_ring}_1 \wedge 'n$

let $?h = (\lambda i. \text{if } i = k \text{ then } b \ i \text{ else } c \ i) :: 'n \Rightarrow 'a::\text{comm\_ring}_1 \wedge 'n$

fix p

assume $p: p \in ?pU$

let $?Uk = ?U - \{k\}$

from p **have** $pU: p \text{ permutes } ?U$

by *blast*

have $kU: ?U = \text{insert } k \ ?Uk$

by *blast*

have $\text{eq}: \text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?Uk = \text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ ?Uk$

$\text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?Uk = \text{prod } (\lambda i. ?h \ i \ \$ \ p \ i) \ ?Uk$

by *auto*

have $Uk: \text{finite } ?Uk \ k \notin ?Uk$

by *auto*

have $\text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?U = \text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ (\text{insert } k \ ?Uk)$

unfolding $kU[\text{symmetric}]$ **..**

also have $\dots = ?f \ k \ \$ \ p \ k * \text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?Uk$

by (*rule prod.insert*) *auto*

also have $\dots = (a \ k \ \$ \ p \ k * \text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?Uk) + (b \ k \ \$ \ p \ k * \text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?Uk)$

by (*simp add: field_simps*)

also have $\dots = (a \ k \ \$ \ p \ k * \text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ ?Uk) + (b \ k \ \$ \ p \ k * \text{prod } (\lambda i. ?h \ i \ \$ \ p \ i) \ ?Uk)$

by (*metis eq*)

also have $\dots = \text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ (\text{insert } k \ ?Uk) + \text{prod } (\lambda i. ?h \ i \ \$ \ p \ i) \ (\text{insert } k \ ?Uk)$

unfolding *prod.insert[OF Uk]* **by** *simp*

finally have $\text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?U = \text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ ?U + \text{prod } (\lambda i. ?h \ i \ \$ \ p \ i) \ ?U$

unfolding $kU[\text{symmetric}]$ **.**

then show $\text{of\_int } (\text{sign } p) * \text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?U =$

$\text{of\_int } (\text{sign } p) * \text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ ?U + \text{of\_int } (\text{sign } p) * \text{prod } (\lambda i. ?h \ i \ \$ \ p \ i) \ ?U$

by (*simp add: field_simps*)

qed *auto*

```

lemma det_row_mul:
  fixes a b :: 'n::finite  $\Rightarrow$   $\_ \wedge 'n$ 
  shows  $\det((\chi \ i. \text{if } i = k \text{ then } c * s \ a \ i \text{ else } b \ i) :: 'a :: \text{comm\_ring\_1}^{\wedge 'n \wedge 'n}) =$ 
     $c * \det((\chi \ i. \text{if } i = k \text{ then } a \ i \text{ else } b \ i) :: 'a :: \text{comm\_ring\_1}^{\wedge 'n \wedge 'n})$ 
  unfolding det_def vec_lambda_beta sum_distrib_left
proof (rule sum.cong)
  let ?U = UNIV :: 'n set
  let ?pU = {p. p permutes ?U}
  let ?f = ( $\lambda i. \text{if } i = k \text{ then } c * s \ a \ i \text{ else } b \ i$ ) :: 'n  $\Rightarrow$  'a :: comm_ring_1^'n
  let ?g = ( $\lambda i. \text{if } i = k \text{ then } a \ i \text{ else } b \ i$ ) :: 'n  $\Rightarrow$  'a :: comm_ring_1^'n
  fix p
  assume p: p  $\in$  ?pU
  let ?Uk = ?U - {k}
  from p have pU: p permutes ?U
  by blast
  have kU: ?U = insert k ?Uk
  by blast
  have eq:  $\text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?Uk = \text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ ?Uk$ 
  by auto
  have Uk: finite ?Uk k  $\notin$  ?Uk
  by auto
  have  $\text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?U = \text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ (\text{insert } k \ ?Uk)$ 
  unfolding kU[symmetric] ..
  also have  $\dots = ?f \ k \ \$ \ p \ k * \text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?Uk$ 
  by (rule prod.insert) auto
  also have  $\dots = (c * s \ a \ k) \ \$ \ p \ k * \text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?Uk$ 
  by (simp add: field_simps)
  also have  $\dots = c * (a \ k \ \$ \ p \ k * \text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ ?Uk)$ 
  unfolding eq by (simp add: ac_simps)
  also have  $\dots = c * (\text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ (\text{insert } k \ ?Uk))$ 
  unfolding prod.insert[OF Uk] by simp
  finally have  $\text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?U = c * (\text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ ?U)$ 
  unfolding kU[symmetric] .
  then show  $\text{of\_int } (\text{sign } p) * \text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?U = c * (\text{of\_int } (\text{sign } p) * \text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ ?U)$ 
  by (simp add: field_simps)
qed auto

lemma det_row_0:
  fixes b :: 'n::finite  $\Rightarrow$   $\_ \wedge 'n$ 
  shows  $\det((\chi \ i. \text{if } i = k \text{ then } 0 \text{ else } b \ i) :: 'a :: \text{comm\_ring\_1}^{\wedge 'n \wedge 'n}) = 0$ 
  using det_row_mul[of k 0  $\lambda i. 1 \ b$ ]
  apply simp
  apply (simp only: vector_smult_lzero)
  done

lemma det_row_operation:
  fixes A :: 'a :: {comm_ring_1}^{\wedge 'n \wedge 'n}

```



```

    assumes ij:  $i \neq j$ 
    shows  $\det (\chi \ k. \text{if } k = i \text{ then row } i \ A + c * s \text{ row } j \ A \text{ else row } k \ A) = \det A$ 
  proof -
    let ?Z =  $(\chi \ k. \text{if } k = i \text{ then row } j \ A \text{ else row } k \ A) :: 'a^{n \times n}$ 
    have th:  $\text{row } i \ ?Z = \text{row } j \ A$  by (vector row_def)
    have th2:  $(\chi \ k. \text{if } k = i \text{ then row } i \ A \text{ else row } k \ A) :: 'a^{n \times n} = A$ 
      by (vector row_def)
    show ?thesis
      unfolding det_row_add [of i] det_row_mul [of i] det_identical_rows [OF ij th]
    th2
      by simp
  qed

```

```

lemma det_row_span:
  fixes  $A :: 'a :: \text{field}^{n \times n}$ 
  assumes x:  $x \in \text{vec.span } \{\text{row } j \ A \mid j. j \neq i\}$ 
  shows  $\det (\chi \ k. \text{if } k = i \text{ then row } i \ A + x \text{ else row } k \ A) = \det A$ 
  using x
proof (induction rule: vec.span_induct_alt)
  case base
    have  $(\text{if } k = i \text{ then row } i \ A + 0 \text{ else row } k \ A) = \text{row } k \ A$  for k
      by simp
    then show ?case
      by (simp add: row_def)
  next
    case (step c z y)
    then obtain j where  $z = \text{row } j \ A$   $i \neq j$ 
      by blast
    let ?w =  $\text{row } i \ A + y$ 
    have th0:  $\text{row } i \ A + (c * s \ z + y) = ?w + c * s \ z$ 
      by vector
    let ?d =  $\lambda x. \det (\chi \ k. \text{if } k = i \text{ then } x \text{ else row } k \ A)$ 
    have thz:  $?d \ z = 0$ 
      apply (rule det_identical_rows [OF j(2)])
      using j
      apply (vector row_def)
      done
    have  $?d (\text{row } i \ A + (c * s \ z + y)) = ?d (?w + c * s \ z)$ 
      unfolding th0 ..
    then have  $?d (\text{row } i \ A + (c * s \ z + y)) = \det A$ 
      unfolding thz step.IH det_row_mul [of i] det_row_add [of i] by simp
    then show ?case
      unfolding scalar_mult_eq_scaleR .
  qed

```

```

lemma matrix_id [simp]:  $\det (\text{matrix id}) = 1$ 
  by (simp add: matrix_id_mat_1)

```

```

proposition det_matrix_scaleR [simp]:  $\det (\text{matrix } ((*_R) \ r)) :: \text{real}^{n \times n} = r$ 

```

```

 $\wedge$  CARD('n::finite)
  apply (subst det_diagonal)
  apply (auto simp: matrix_def mat_def)
  apply (simp add: cart_eq_inner_axis inner_axis_axis)
done

```

May as well do this, though it's a bit unsatisfactory since it ignores exact duplicates by considering the rows/columns as a set.

```

lemma det_dependent_rows:
  fixes A:: 'a::{'field'}nn
  assumes d: vec.dependent (rows A)
  shows det A = 0
proof -
  let ?U = UNIV :: 'n set
  from d obtain i where i: row i A ∈ vec.span (rows A - {row i A})
    unfolding vec.dependent_def rows_def by blast
  show ?thesis
proof (cases  $\forall i j. i \neq j \longrightarrow \text{row } i A \neq \text{row } j A$ )
  case True
  with i have vec.span (rows A - {row i A})  $\subseteq$  vec.span {row j A | j. j  $\neq$  i}
    by (auto simp: rows_def intro!: vec.span_mono)
  then have - row i A ∈ vec.span {row j A | j. j  $\neq$  i}
    by (meson i subsetCE vec.span_neg)
  from det_row_span[OF this]
  have det A = det ( $\chi$  k. if k = i then 0 *s 1 else row k A)
    unfolding right_minus vector_smult_lzero ..
  with det_row_mul[of i 0  $\lambda i. 1$ ]
  show ?thesis by simp
next
  case False
  then obtain j k where jk: j  $\neq$  k row j A = row k A
    by auto
  from det_identical_rows[OF jk] show ?thesis .
qed
qed

```

```

lemma det_dependent_columns:
  assumes d: vec.dependent (columns (A::realnn))
  shows det A = 0
  by (metis d det_dependent_rows rows_transpose det_transpose)

```

Multilinearity and the multiplication formula

```

lemma Cart_lambda_cong: ( $\bigwedge x. f x = g x$ )  $\implies$  (vec_lambda f::'an) = (vec_lambda g::'an)
  by auto

```

```

lemma det_linear_row_sum:
  assumes fS: finite S
  shows det ( $\chi i. \text{if } i = k \text{ then sum } (a \ i) \ S \text{ else } c \ i$ ::'a::comm_ring_1nn) =

```

```

    sum ( $\lambda j. \det ((\chi i. \text{if } i = k \text{ then } a \ i \ j \text{ else } c \ i)::'a'^n'^n)) \ S$ 
  using fS by (induct rule: finite_induct; simp add: det_row_0 det_row_add
cong: if_cong)

```

lemma *finite_bounded_functions:*

```

  assumes fS: finite S
  shows finite {f. ( $\forall i \in \{1..(k::nat)\}. f \ i \in S$ )  $\wedge$  ( $\forall i. i \notin \{1..k\} \longrightarrow f \ i = i$ )}
proof (induct k)
  case 0
  have *: {f.  $\forall i. f \ i = i$ } = {id}
  by auto
  show ?case
  by (auto simp: *)
next
  case (Suc k)
  let ?f =  $\lambda(y::nat,g) \ i. \text{if } i = \text{Suc } k \text{ then } y \text{ else } g \ i$ 
  let ?S = ?f ' ( $S \times \{f. (\forall i \in \{1..k\}. f \ i \in S) \wedge (\forall i. i \notin \{1..k\} \longrightarrow f \ i = i)\}$ )
  have ?S = {f. ( $\forall i \in \{1.. \text{Suc } k\}. f \ i \in S$ )  $\wedge$  ( $\forall i. i \notin \{1.. \text{Suc } k\} \longrightarrow f \ i = i$ )}
  apply (auto simp: image_iff)
  apply (rename_tac f)
  apply (rule_tac x=f (Suc k) in bexI)
  apply (rule_tac x =  $\lambda i. \text{if } i = \text{Suc } k \text{ then } i \text{ else } f \ i$  in exI, auto)
  done
  with finite_imageI[OF finite_cartesian_product[OF fS Suc.hyps(1)], of ?f]
  show ?case
  by metis
qed

```

lemma *det_linear_rows_sum_lemma:*

```

  assumes fS: finite S
  and fT: finite T
  shows det (( $\chi i. \text{if } i \in T \text{ then } \text{sum } (a \ i) \ S \text{ else } c \ i$ )::' $a::\text{comm\_ring\_1}$ ' $^n'^n$ ) =
    sum ( $\lambda f. \det((\chi i. \text{if } i \in T \text{ then } a \ i \ (f \ i) \text{ else } c \ i)::'a'^n'^n)$ )
      {f. ( $\forall i \in T. f \ i \in S$ )  $\wedge$  ( $\forall i. i \notin T \longrightarrow f \ i = i$ )}
  using fT
proof (induct T arbitrary: a c set: finite)
  case empty
  have th0:  $\bigwedge x \ y. (\chi i. \text{if } i \in \{\} \text{ then } x \ i \text{ else } y \ i) = (\chi i. y \ i)$ 
  by vector
  from empty.premis show ?case
  unfolding th0 by (simp add: eq_id_iff)
next
  case (insert z T a c)
  let ?F =  $\lambda T. \{f. (\forall i \in T. f \ i \in S) \wedge (\forall i. i \notin T \longrightarrow f \ i = i)\}$ 
  let ?h =  $\lambda(y,g) \ i. \text{if } i = z \text{ then } y \text{ else } g \ i$ 
  let ?k =  $\lambda h. (h(z), (\lambda i. \text{if } i = z \text{ then } i \text{ else } h \ i))$ 
  let ?s =  $\lambda k \ a \ c \ f. \det((\chi i. \text{if } i \in T \text{ then } a \ i \ (f \ i) \text{ else } c \ i)::'a'^n'^n)$ 
  let ?c =  $\lambda j \ i. \text{if } i = z \text{ then } a \ i \ j \text{ else } c \ i$ 

```

```

have thif:  $\bigwedge a\ b\ c\ d. (if\ a\ \vee\ b\ then\ c\ else\ d) = (if\ a\ then\ c\ else\ if\ b\ then\ c\ else\ d)$ 
by simp
have thif2:  $\bigwedge a\ b\ c\ d\ e. (if\ a\ then\ b\ else\ if\ c\ then\ d\ else\ e) =$ 
   $(if\ c\ then\ (if\ a\ then\ b\ else\ d)\ else\ (if\ a\ then\ b\ else\ e))$ 
by simp
from  $\langle z \notin T \rangle$  have nz:  $\bigwedge i. i \in T \implies i \neq z$ 
by auto
have det  $(\chi\ i. if\ i \in insert\ z\ T\ then\ sum\ (a\ i)\ S\ else\ c\ i) =$ 
   $det\ (\chi\ i. if\ i = z\ then\ sum\ (a\ i)\ S\ else\ if\ i \in T\ then\ sum\ (a\ i)\ S\ else\ c\ i)$ 
unfolding insert_iff thif ..
also have ...  $= (\sum_{j \in S}. det\ (\chi\ i. if\ i \in T\ then\ sum\ (a\ i)\ S\ else\ if\ i = z\ then\ a\ i\ j\ else\ c\ i))$ 
unfolding det_linear_row_sum[OF fS]
by (subst thif2) (simp add: nz cong: if_cong)
finally have tha:
   $det\ (\chi\ i. if\ i \in insert\ z\ T\ then\ sum\ (a\ i)\ S\ else\ c\ i) =$ 
   $(\sum_{(j, f) \in S \times ?F\ T. det\ (\chi\ i. if\ i \in T\ then\ a\ i\ (f\ i)\ else\ if\ i = z\ then\ a\ i\ j\ else\ c\ i))$ 
unfolding insert.hyps unfolding sum.cartesian_product by blast
show ?case unfolding tha
using  $\langle z \notin T \rangle$ 
by (intro sum.reindex_bij_witness[where i=?k and j=?h])
  (auto intro!: cong[OF refl[of det]] simp: vec_eq_iff)
qed

```

```

lemma det_linear_rows_sum:
  fixes  $S :: 'n::finite\ set$ 
  assumes fS:  $finite\ S$ 
  shows  $det\ (\chi\ i. sum\ (a\ i)\ S) =$ 
   $sum\ (\lambda f. det\ (\chi\ i. a\ i\ (f\ i) :: 'a::comm\_ring\_1 \wedge 'n \wedge 'n))\ \{f. \forall i. f\ i \in S\}$ 
proof -
  have th0:  $\bigwedge x\ y. ((\chi\ i. if\ i \in (UNIV :: 'n\ set)\ then\ x\ i\ else\ y\ i) :: 'a \wedge 'n \wedge 'n) = (\chi\ i. x\ i)$ 
by vector
  from det_linear_rows_sum_lemma[OF fS, of UNIV :: 'n set a, unfolded th0, OF finite]
  show ?thesis by simp
qed

```

```

lemma matrix_mul_sum_alt:
  fixes  $A\ B :: 'a::comm\_ring\_1 \wedge 'n \wedge 'n$ 
  shows  $A ** B = (\chi\ i. sum\ (\lambda k. A\ \$i\ \$k * B\ \$\ k)\ (UNIV :: 'n\ set))$ 
by (vector matrix_matrix_mult_def sum_component)

```

```

lemma det_rows_mul:
   $det((\chi\ i. c\ i * s\ a\ i) :: 'a::comm\_ring\_1 \wedge 'n \wedge 'n) =$ 
   $prod\ (\lambda i. c\ i)\ (UNIV :: 'n\ set) * det((\chi\ i. a\ i) :: 'a \wedge 'n \wedge 'n)$ 
proof (simp add: det_def sum_distrib_left cong add: prod.cong, rule sum.cong)

```

```

let ?U = UNIV :: 'n set
let ?PU = {p. p permutes ?U}
fix p
assume pU: p ∈ ?PU
let ?s = of_int (sign p)
from pU have p: p permutes ?U
  by blast
have prod (λi. c i * a i $ p i) ?U = prod c ?U * prod (λi. a i $ p i) ?U
  unfolding prod.distrib ..
then show ?s * (∏ xa∈?U. c xa * a xa $ p xa) =
  prod c ?U * (?s * (∏ xa∈?U. a xa $ p xa))
  by (simp add: field_simps)
qed rule

```

proposition det_mul:

```

fixes A B :: 'a::comm_ring_1'^n'^n
shows det (A ** B) = det A * det B
proof -
let ?U = UNIV :: 'n set
let ?F = {f. (∀ i ∈ ?U. f i ∈ ?U) ∧ (∀ i. i ∉ ?U ⟶ f i = i)}
let ?PU = {p. p permutes ?U}
have p ∈ ?F if p permutes ?U for p
  by simp
then have PUF: ?PU ⊆ ?F by blast
{
  fix f
  assume fPU: f ∈ ?F - ?PU
  have fUU: f ' ?U ⊆ ?U
  using fPU by auto
  from fPU have f: ∀ i ∈ ?U. f i ∈ ?U ∧ i ∉ ?U ⟶ f i = i ∧ (∀ y. ∃! x. f x = y)
  unfolding permutes_def by auto

  let ?A = (χ i. A $ i $ f i * s B $ f i) :: 'a'^n'^n
  let ?B = (χ i. B $ f i) :: 'a'^n'^n
  {
    assume fni: ¬ inj_on f ?U
    then obtain i j where ij: f i = f j ∧ i ≠ j
    unfolding inj_on_def by blast
    then have row i ?B = row j ?B
    by (vector row_def)
    with det_identical_rows[OF ij(2)]
    have det (χ i. A $ i $ f i * s B $ f i) = 0
    unfolding det_rows_mul by force
  }
  moreover
  {
    assume fi: inj_on f ?U
    from f fi have fith: ∧ i j. f i = f j ⟹ i = j

```

```

      unfolding inj_on_def by metis
      note fs = fi[unfolded surjective_iff_injective_gen[OF finite finite refl fUU,
symmetric]]
      have  $\exists!x. f\ x = y$  for  $y$ 
      using fith fs by blast
      with  $f(\beta)$  have  $\det(\chi\ i.\ A\$i\$f\ i\ *s\ B\$f\ i) = 0$ 
      by blast
    }
    ultimately have  $\det(\chi\ i.\ A\$i\$f\ i\ *s\ B\$f\ i) = 0$ 
    by blast
  }
  then have zth:  $\forall f \in ?F - ?PU. \det(\chi\ i.\ A\$i\$f\ i\ *s\ B\$f\ i) = 0$ 
  by simp
  {
    fix p
    assume pU:  $p \in ?PU$ 
    from pU have p:  $p$  permutes  $?U$ 
    by blast
    let ?s =  $\lambda p. \text{of\_int}(\text{sign } p)$ 
    let ?f =  $\lambda q. ?s\ p * (\prod i \in ?U. A\ \$\ i\ \$\ p\ i) * (?s\ q * (\prod i \in ?U. B\ \$\ i\ \$\ q\ i))$ 
    have (sum ( $\lambda q. ?s\ q *$ 
      ( $\prod i \in ?U. (\chi\ i.\ A\ \$\ i\ \$\ p\ i\ *s\ B\ \$\ p\ i :: 'a^{n \sim n})\ \$\ i\ \$\ q\ i$ )) ?PU) =
      (sum ( $\lambda q. ?s\ p * (\prod i \in ?U. A\ \$\ i\ \$\ p\ i) * (?s\ q * (\prod i \in ?U. B\ \$\ i\ \$\ q\ i))$ ))
    ?PU)
    unfolding sum_permutations_compose_right[OF permutes_inv[OF p], of ?f]
    proof (rule sum.cong)
      fix q
      assume qU:  $q \in ?PU$ 
      then have q:  $q$  permutes  $?U$ 
      by blast
      from p q have pp: permutation  $p$  and pq: permutation  $q$ 
      unfolding permutation_permutes by auto
      have th00:  $\text{of\_int}(\text{sign } p) * \text{of\_int}(\text{sign } p) = (1::'a)$ 
       $\wedge a. \text{of\_int}(\text{sign } p) * (\text{of\_int}(\text{sign } p) * a) = a$ 
      unfolding mult.assoc[symmetric]
      unfolding of_int_mult[symmetric]
      by (simp_all add: sign_idempotent)
      have ths:  $?s\ q = ?s\ p * ?s\ (q \circ \text{inv } p)$ 
      using pp pq permutation_inverse[OF pp] sign_inverse[OF pp]
      by (simp add: th00 ac_simps sign_idempotent sign_compose)
      have th001:  $\text{prod}(\lambda i. B\$i\$ q (\text{inv } p\ i))\ ?U = \text{prod}((\lambda i. B\$i\$ q (\text{inv } p\ i)) \circ$ 
       $p)\ ?U$ 
      by (rule prod.permute[OF p])
      have thp:  $\text{prod}(\lambda i. (\chi\ i.\ A\$i\$p\ i\ *s\ B\$p\ i :: 'a^{n \sim n})\ \$\ i\ \$\ q\ i)\ ?U =$ 
       $\text{prod}(\lambda i. A\$i\$p\ i)\ ?U * \text{prod}(\lambda i. B\$i\$ q (\text{inv } p\ i))\ ?U$ 
      unfolding th001 prod.distrib[symmetric] o_def permutes_inverses[OF p]
      apply (rule prod.cong[OF refl])
      using permutes_in_image[OF q]
      apply vector

```

```

    done
  show ?s q * prod (λi. (((χ i. A$i$p i *s B$p i) :: 'a^n^n)$i$q i)) ?U =
    ?s p * (prod (λi. A$i$p i) ?U) * (?s (q ∘ inv p) * prod (λi. B$i$(q ∘ inv p)
i) ?U)
    using ths thp pp pq permutation_inverse[OF pp] sign_inverse[OF pp]
    by (simp add: sign_nz th00 field_simps sign_idempotent sign_compose)
  qed rule
}
then have th2: sum (λf. det (χ i. A$i$f i *s B$f i)) ?PU = det A * det B
  unfolding det_def sum_product
  by (rule sum.cong [OF refl])
have det (A**B) = sum (λf. det (χ i. A $ i $ f i *s B $ f i)) ?F
  unfolding matrix_mul_sum_alt det_linear_rows_sum[OF finite]
  by simp
also have ... = sum (λf. det (χ i. A$i$f i *s B$f i)) ?PU
  using sum.mono_neutral_cong_left[OF finite PUF zth, symmetric]
  unfolding det_rows_mul by auto
finally show ?thesis unfolding th2 .
qed

```

1.10.2 Relation to invertibility

```

proposition invertible_det_nz:
  fixes A::'a::{field}^n^n
  shows invertible A ⟷ det A ≠ 0
proof (cases invertible A)
  case True
  then obtain B :: 'a^n^n where B: A ** B = mat 1
    unfolding invertible_right_inverse by blast
  then have det (A ** B) = det (mat 1 :: 'a^n^n)
    by simp
  then show ?thesis
    by (metis True det_I det_mul mult_zero_left one_neq_zero)
next
  case False
  let ?U = UNIV :: 'n set
  have fU: finite ?U
    by simp
  from False obtain c i where c: sum (λi. c i *s row i A) ?U = 0 and iU: i ∈
?U and ci: c i ≠ 0
    unfolding invertible_right_inverse matrix_right_invertible_independent_rows
    by blast
  have thr0: - row i A = sum (λj. (1 / c i) *s (c j *s row j A)) (?U - {i})
    unfolding sum_cmul using c ci
    by (auto simp: sum.remove[OF fU iU] eq_vector_fraction_iff add_eq_0_iff)
  have thr: - row i A ∈ vec.span {row j A | j. j ≠ i}
    unfolding thr0 by (auto intro: vec.span_base vec.span_scale vec.span_sum)
  let ?B = (χ k. if k = i then 0 else row k A) :: 'a^n^n
  have thrb: row i ?B = 0 using iU by (vector row_def)

```

```

have det A = 0
  unfolding det_row_span[OF thr, symmetric] right_minus
  unfolding det_zero_row(2)[OF thrb] ..
then show ?thesis
  by (simp add: False)
qed

```

```

lemma det_nz_iff_inj_gen:
  fixes f :: 'a::fieldn ⇒ 'a::fieldn
  assumes Vector_Spaces.linear (*s) (*s) f
  shows det (matrix f) ≠ 0 ⟷ inj f
proof
  assume det (matrix f) ≠ 0
  then show inj f
    using assms invertible_det_nz inj_matrix_vector_mult by force
next
  assume inj f
  show det (matrix f) ≠ 0
    using vec.linear_injective_left_inverse [OF assms ⟨inj f⟩]
    by (metis assms invertible_det_nz invertible_left_inverse matrix_compose_gen
matrix_id_mat_1)
qed

```

```

lemma det_nz_iff_inj:
  fixes f :: realn ⇒ realn
  assumes linear f
  shows det (matrix f) ≠ 0 ⟷ inj f
  using det_nz_iff_inj_gen[of f] assms
  unfolding linear_matrix_vector_mul_eq .

```

```

lemma det_eq_0_rank:
  fixes A :: realnn
  shows det A = 0 ⟷ rank A < CARD('n)
  using invertible_det_nz [of A]
  by (auto simp: matrix_left_invertible_injective invertible_left_inverse less_rank_noninjective)

```

Invertibility of matrices and corresponding linear functions

```

lemma matrix_left_invertible_gen:
  fixes f :: 'a::fieldm ⇒ 'a::fieldn
  assumes Vector_Spaces.linear (*s) (*s) f
  shows ((∃ B. B ** matrix f = mat 1) ⟷ (∃ g. Vector_Spaces.linear (*s) (*s)
g ∧ g ∘ f = id))
proof safe
  fix B
  assume 1: B ** matrix f = mat 1
  show ∃ g. Vector_Spaces.linear (*s) (*s) g ∧ g ∘ f = id
  proof (intro exI conjI)

```



```

show Vector_Spaces.linear (*s) (*s) ( $\lambda y. B * v y$ )
by simp
show (( $*v$ ) B)  $\circ f = id$ 
unfolding o_def
by (metis assms 1 eq_id_iff matrix_vector_mul(1) matrix_vector_mul_assoc
matrix_vector_mul_lid)
qed
next
fix g
assume Vector_Spaces.linear (*s) (*s) g  $g \circ f = id$ 
then have matrix g ** matrix f = mat 1
by (metis assms matrix_compose_gen matrix_id_mat_1)
then show  $\exists B. B ** matrix f = mat 1 ..$ 
qed

```

```

lemma matrix_left_invertible:
  linear f  $\implies ((\exists B. B ** matrix f = mat 1) \longleftrightarrow (\exists g. linear g \wedge g \circ f = id))$  for
  f::realm  $\Rightarrow$  realn
using matrix_left_invertible_gen[of f]
by (auto simp: linear_matrix_vector_mul_eq)

```

```

lemma matrix_right_invertible_gen:
  fixes f :: 'a::fieldm  $\Rightarrow$  'an
assumes Vector_Spaces.linear (*s) (*s) f
shows  $((\exists B. matrix f ** B = mat 1) \longleftrightarrow (\exists g. Vector_Spaces.linear (*s) (*s) g \wedge f \circ g = id))$ 
proof safe
fix B
assume 1: matrix f ** B = mat 1
show  $\exists g. Vector_Spaces.linear (*s) (*s) g \wedge f \circ g = id$ 
proof (intro exI conjI)
show Vector_Spaces.linear (*s) (*s) (( $*v$ ) B)
by simp
show f  $\circ (*v) B = id$ 
using 1 assms comp_apply eq_id_iff vec.linear_id matrix_id_mat_1 matrix_vector_mul_assoc matrix_works
by (metis (no_types, opaque_lifting))
qed
next
fix g
assume Vector_Spaces.linear (*s) (*s) g and f  $\circ g = id$ 
then have matrix f ** matrix g = mat 1
by (metis assms matrix_compose_gen matrix_id_mat_1)
then show  $\exists B. matrix f ** B = mat 1 ..$ 
qed

```

```

lemma matrix_right_invertible:
  linear f  $\implies ((\exists B. matrix f ** B = mat 1) \longleftrightarrow (\exists g. linear g \wedge f \circ g = id))$  for
  f::realm  $\Rightarrow$  realn

```

```

using matrix_right_invertible_gen[of f]
by (auto simp: linear_matrix_vector_mul_eq)

```

```

lemma matrix_invertible_gen:
  fixes f :: 'a::field^m ⇒ 'a::field^n
  assumes Vector_Spaces.linear (*s) (*s) f
  shows invertible (matrix f) ⟷ (∃ g. Vector_Spaces.linear (*s) (*s) g ∧ f ∘ g
= id ∧ g ∘ f = id)
    (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
    by (metis assms invertible_def left_right_inverse_eq matrix_left_invertible_gen
matrix_right_invertible_gen)
  next
    assume ?rhs then show ?lhs
      by (metis assms invertible_def matrix_compose_gen matrix_id_mat_1)
qed

```

```

lemma matrix_invertible:
  linear f ⟹ invertible (matrix f) ⟷ (∃ g. linear g ∧ f ∘ g = id ∧ g ∘ f = id)
  for f::real^m ⇒ real^n
  using matrix_invertible_gen[of f]
  by (auto simp: linear_matrix_vector_mul_eq)

```

```

lemma invertible_eq_bij:
  fixes m :: 'a::field^m^n
  shows invertible m ⟷ bij ((*v) m)
  using matrix_invertible_gen[OF matrix_vector_mul_linear_gen, of m, simpli-
fied matrix_of_matrix_vector_mul]
  by (metis bij_betw_def left_right_inverse_eq matrix_vector_mul_linear_gen
o_bij
    vec.linear_injective_left_inverse vec.linear_surjective_right_inverse)

```

1.10.3 Cramer's rule

```

lemma cramer_lemma_transpose:
  fixes A:: 'a::{field}^n^n
  and x :: 'a::{field}^n
  shows det ((χ i. if i = k then sum (λi. x$i * s row i A) (UNIV::'n set)
    else row i A)::'a::{field}^n^n) = x$k * det A
    (is ?lhs = ?rhs)
proof -
  let ?U = UNIV :: 'n set
  let ?Uk = ?U - {k}
  have U: ?U = insert k ?Uk
    by blast
  have kUk: k ∉ ?Uk
    by simp
  have th00: ∧ k s. x$k * s row k A + s = (x$k - 1) * s row k A + row k A + s

```

```

  by (vector field_simps)
  have th001:  $\bigwedge f k . (\lambda x. \text{if } x = k \text{ then } f k \text{ else } f x) = f$ 
  by auto
  have  $(\chi i. \text{row } i A) = A$  by (vector row_def)
  then have thd1:  $\det (\chi i. \text{row } i A) = \det A$ 
  by simp
  have thd0:  $\det (\chi i. \text{if } i = k \text{ then row } k A + (\sum i \in ?Uk. x \$ i * s \text{ row } i A) \text{ else row } i A) = \det A$ 
  by (force intro: det_row_span vec.span_sum vec.span_scale vec.span_base)
  show ?lhs =  $x \$ k * \det A$ 
  apply (subst U)
  unfolding sum.insert[OF finite kUk]
  apply (subst th001)
  unfolding add.assoc
  apply (subst det_row_add)
  unfolding thd0
  unfolding det_row_mul
  unfolding th001[of k  $\lambda i. \text{row } i A$ ]
  unfolding thd1
  apply (simp add: field_simps)
  done
qed

```

proposition *cramer_lemma:*

```

  fixes A :: 'a::fieldn
  shows  $\det((\chi i j. \text{if } j = k \text{ then } (A * v x) \$ i \text{ else } A \$ i \$ j) :: 'a::field^{n \times n}) = x \$ k * \det A$ 
  *  $\det A$ 
  proof -
    let ?U = UNIV :: 'n set
    have *:  $\bigwedge c. \text{sum } (\lambda i. c i * s \text{ row } i (\text{transpose } A)) ?U = \text{sum } (\lambda i. c i * s \text{ column } i A) ?U$ 
    by (auto intro: sum.cong)
    show ?thesis
    unfolding matrix_mult_sum
    unfolding cramer_lemma_transpose[of k x transpose A, unfolded det_transpose, symmetric]
    unfolding *[of  $\lambda i. x \$ i$ ]
    apply (subst det_transpose[symmetric])
    apply (rule cong[OF refl[of det]])
    apply (vector transpose_def column_def row_def)
    done
  qed

```

proposition *cramer:*

```

  fixes A :: 'a::fieldn
  assumes d0:  $\det A \neq 0$ 
  shows  $A * v x = b \iff x = (\chi k. \det(\chi i j. \text{if } j=k \text{ then } b \$ i \text{ else } A \$ i \$ j) / \det A)$ 
  proof -
    from d0 obtain B where B:  $A ** B = \text{mat } 1 \text{ } B ** A = \text{mat } 1$ 

```

```

    unfolding invertible_det_nz[symmetric] invertible_def
  by blast
have (A ** B) *v b = b
  by (simp add: B)
then have A *v (B *v b) = b
  by (simp add: matrix_vector_mul_assoc)
then have xe:  $\exists x. A *v x = b$ 
  by blast
{
  fix x
  assume x:  $A *v x = b$ 
  have  $x = (\chi k. \det(\chi i j. \text{if } j=k \text{ then } b\$i \text{ else } A\$i\$j) / \det A)$ 
    unfolding x[symmetric]
    using d0 by (simp add: vec_eq_iff cramer_lemma field_simps)
}
with xe show ?thesis
  by auto
qed

lemma det_1:  $\det (A::'a::comm\_ring\_1^{1^1}) = A\$1\$1$ 
  by (simp add: det_def sign_id)

lemma det_2:  $\det (A::'a::comm\_ring\_1^{2^2}) = A\$1\$1 * A\$2\$2 - A\$1\$2 * A\$2\$1$ 
proof -
  have f12:  $\text{finite } \{2::2\} \ 1 \notin \{2::2\}$  by auto
  show ?thesis
    unfolding det_def UNIV_2
    unfolding sum_over_permutations_insert[OF f12]
    unfolding permutes_sing
    by (simp add: sign_swap_id sign_id swap_id_eq)
qed

lemma det_3:
 $\det (A::'a::comm\_ring\_1^{3^3}) =$ 
 $A\$1\$1 * A\$2\$2 * A\$3\$3 +$ 
 $A\$1\$2 * A\$2\$3 * A\$3\$1 +$ 
 $A\$1\$3 * A\$2\$1 * A\$3\$2 -$ 
 $A\$1\$1 * A\$2\$3 * A\$3\$2 -$ 
 $A\$1\$2 * A\$2\$1 * A\$3\$3 -$ 
 $A\$1\$3 * A\$2\$2 * A\$3\$1$ 
proof -
  have f123:  $\text{finite } \{2::3, 3\} \ 1 \notin \{2::3, 3\}$ 
    by auto
  have f23:  $\text{finite } \{3::3\} \ 2 \notin \{3::3\}$ 
    by auto

  show ?thesis
    unfolding det_def UNIV_3

```

```

    unfolding sum_over_permutations_insert[OF f123]
    unfolding sum_over_permutations_insert[OF f23]
    unfolding permutes_sing
    by (simp add: sign_swap_id permutation_swap_id sign_compose sign_id
swap_id_eq)
qed

```

```

proposition det_orthogonal_matrix:
  fixes Q:: 'a::linordered_idomn
  assumes oQ: orthogonal_matrix Q
  shows det Q = 1  $\vee$  det Q = - 1
proof -
  have Q ** transpose Q = mat 1
    by (metis oQ orthogonal_matrix_def)
  then have det (Q ** transpose Q) = det (mat 1:: 'an)
    by simp
  then have det Q * det Q = 1
    by (simp add: det_mul)
  then show ?thesis
    by (simp add: square_eq_1_iff)
qed

```

```

proposition orthogonal_transformation_det [simp]:
  fixes f :: realn  $\Rightarrow$  realn
  shows orthogonal_transformation f  $\implies$  |det (matrix f)| = 1
  using det_orthogonal_matrix orthogonal_transformation_matrix by fastforce

```

1.10.4 Rotation, reflection, rotoinversion

```

definition rotation_matrix Q  $\longleftrightarrow$  orthogonal_matrix Q  $\wedge$  det Q = 1
definition rotoinversion_matrix Q  $\longleftrightarrow$  orthogonal_matrix Q  $\wedge$  det Q = - 1

```

```

lemma orthogonal_rotation_or_rotoinversion:
  fixes Q :: 'a::linordered_idomn
  shows orthogonal_matrix Q  $\longleftrightarrow$  rotation_matrix Q  $\vee$  rotoinversion_matrix Q
  by (metis rotoinversion_matrix_def rotation_matrix_def det_orthogonal_matrix)

```

Slightly stronger results giving rotation, but only in two or more dimensions

```

lemma rotation_matrix_exists_basis:
  fixes a :: realn
  assumes 2: 2  $\leq$  CARD('n) and norm a = 1
  obtains A where rotation_matrix A  $\wedge$  A * v (axis k 1) = a
proof -
  obtain A where orthogonal_matrix A and A: A * v (axis k 1) = a
    using orthogonal_matrix_exists_basis assms by metis
  with orthogonal_rotation_or_rotoinversion
  consider rotation_matrix A | rotoinversion_matrix A
    by metis
  then show thesis

```

```

proof cases
  assume rotation_matrix A
  then show ?thesis
    using ⟨A * v axis k 1 = a⟩ that by auto
next
  from ex_card[OF 2] obtain h i::'n where h ≠ i
    by (auto simp add: eval_nat_numeral card_Suc_eq)
  then obtain j where j ≠ k
    by (metis (full_types))
  let ?TA = transpose A
  let ?A =  $\chi$  i. if i = j then - 1 *R (?TA $ i) else ?TA $ i
  assume rotoinversion_matrix A
  then have [simp]: det A = -1
    by (simp add: rotoinversion_matrix_def)
  show ?thesis
proof
  have [simp]: row i ( $\chi$  i. if i = j then - 1 *R ?TA $ i else ?TA $ i) = (if i =
j then - row i ?TA else row i ?TA) for i
    by (auto simp: row_def)
  have orthogonal_matrix ?A
    unfolding orthogonal_matrix_orthonormal_rows
  using ⟨orthogonal_matrix A⟩ by (auto simp: orthogonal_matrix_orthonormal_columns
orthogonal_clauses)
  then show rotation_matrix (transpose ?A)
    unfolding rotation_matrix_def
  by (simp add: det_row_mul[of j _  $\lambda$ i. ?TA $ i, unfolded scalar_mult_eq_scaleR])
  show transpose ?A * v axis k 1 = a
    using ⟨j ≠ k⟩ A by (simp add: matrix_vector_column_axis_def scalar_mult_eq_scaleR
if_distrib [of  $\lambda$ z. z *R c for c] cong: if_cong)
  qed
qed
qed

lemma rotation_exists_1:
  fixes a :: real'n
  assumes 2 ≤ CARD('n) norm a = 1 norm b = 1
  obtains f where orthogonal_transformation f det(matrix f) = 1 f a = b
proof -
  obtain k::'n where True
    by simp
  obtain A B where AB: rotation_matrix A rotation_matrix B
    and eq: A * v (axis k 1) = a B * v (axis k 1) = b
    using rotation_matrix_exists_basis assms by metis
  let ?f =  $\lambda$ x. (B ** transpose A) * v x
  show thesis
proof
  show orthogonal_transformation ?f
    using AB orthogonal_matrix_mul orthogonal_transformation_matrix rotation_matrix_def matrix_vector_mul_linear by force

```

```

  show  $\det(\text{matrix } ?f) = 1$ 
    using AB by (auto simp: det_mul rotation_matrix_def)
  show  $?f a = b$ 
    using AB unfolding orthogonal_matrix_def rotation_matrix_def
    by (metis eq_matrix_mul_rid matrix_vector_mul_assoc)
qed
qed

lemma rotation_exists:
  fixes  $a :: \text{real}^n$ 
  assumes  $2: 2 \leq \text{CARD}(n)$  and  $\text{eq: norm } a = \text{norm } b$ 
  obtains  $f$  where  $\text{orthogonal\_transformation } f \ \det(\text{matrix } f) = 1 \ f a = b$ 
proof (cases  $a = 0 \vee b = 0$ )
  case True
    with assms have  $a = 0 \ b = 0$ 
      by auto
    then show ?thesis
      by (metis eq_id_iff matrix_id orthogonal_transformation_id that)
  next
  case False
    then obtain  $f$  where  $f: \text{orthogonal\_transformation } f \ \det(\text{matrix } f) = 1$ 
      and  $f': f(a /_{\mathbb{R}} \text{norm } a) = b /_{\mathbb{R}} \text{norm } b$ 
      using rotation_exists_1 [of  $a /_{\mathbb{R}} \text{norm } a \ b /_{\mathbb{R}} \text{norm } b, OF 2$ ] by auto
    then interpret linear  $f$  by (simp add: orthogonal_transformation)
    have  $f a = b$ 
      using  $f'$  False
      by (simp add: eq scale)
    with  $f$  show thesis ..
qed

lemma rotation_rightward_line:
  fixes  $a :: \text{real}^n$ 
  obtains  $f$  where  $\text{orthogonal\_transformation } f \ 2 \leq \text{CARD}(n) \implies \det(\text{matrix } f) = 1$ 
    and  $f(\text{norm } a *_{\mathbb{R}} \text{axis } k \ 1) = a$ 
proof (cases  $\text{CARD}(n) = 1$ )
  case True
    obtain  $f$  where  $\text{orthogonal\_transformation } f \ f(\text{norm } a *_{\mathbb{R}} \text{axis } k \ (1::\text{real})) = a$ 
    proof (rule orthogonal_transformation_exists)
      show  $\text{norm } (\text{norm } a *_{\mathbb{R}} \text{axis } k \ (1::\text{real})) = \text{norm } a$ 
        by simp
    qed auto
    then show thesis
      using True that by auto
  next
  case False
    obtain  $f$  where  $\text{orthogonal\_transformation } f \ \det(\text{matrix } f) = 1 \ f(\text{norm } a *_{\mathbb{R}} \text{axis } k \ 1) = a$ 
    proof (rule rotation_exists)

```

```

show  $2 \leq \text{CARD}(n)$ 
  using False one_le_card_finite [where  $'a = n$ ] by linarith
show  $\text{norm} (\text{norm } a *_{\mathbb{R}} \text{axis } k (1 :: \text{real})) = \text{norm } a$ 
  by simp
qed auto
then show thesis
  using that by blast
qed

end

```


Chapter 2

Topology

```
theory Elementary_Topology
imports
  HOL-Library.Set_Idioms
  HOL-Library.Disjoint_Sets
  Product_Vector
begin
```

2.1 Elementary Topology

Affine transformations of intervals

```
lemma real_affinity_le:  $0 < m \implies m * x + c \leq y \longleftrightarrow x \leq \text{inverse } m * y + -$   
 $(c / m)$   
  for  $m :: 'a::\text{linordered\_field}$   
  by (simp add: field_simps)
```

```
lemma real_le_affinity:  $0 < m \implies y \leq m * x + c \longleftrightarrow \text{inverse } m * y + - (c /$   
 $m) \leq x$   
  for  $m :: 'a::\text{linordered\_field}$   
  by (simp add: field_simps)
```

```
lemma real_affinity_lt:  $0 < m \implies m * x + c < y \longleftrightarrow x < \text{inverse } m * y + -$   
 $(c / m)$   
  for  $m :: 'a::\text{linordered\_field}$   
  by (simp add: field_simps)
```

```
lemma real_lt_affinity:  $0 < m \implies y < m * x + c \longleftrightarrow \text{inverse } m * y + - (c /$   
 $m) < x$   
  for  $m :: 'a::\text{linordered\_field}$   
  by (simp add: field_simps)
```

```
lemma real_affinity_eq:  $m \neq 0 \implies m * x + c = y \longleftrightarrow x = \text{inverse } m * y + -$   
 $(c / m)$   
  for  $m :: 'a::\text{linordered\_field}$ 
```

by (simp add: field_simps)

lemma *real_eq_affinity*: $m \neq 0 \implies y = m * x + c \longleftrightarrow \text{inverse } m * y + - (c / m) = x$
 for $m :: 'a::\text{linordered\_field}$
 by (simp add: field_simps)

2.1.1 Topological Basis

context *topological_space*
begin

definition *topological_basis* $B \longleftrightarrow$
 $(\forall b \in B. \text{open } b) \wedge (\forall x. \text{open } x \longrightarrow (\exists B'. B' \subseteq B \wedge \bigcup B' = x))$

lemma *topological_basis*:
topological_basis $B \longleftrightarrow (\forall x. \text{open } x \longleftrightarrow (\exists B'. B' \subseteq B \wedge \bigcup B' = x))$
unfolding *topological_basis_def*
apply *safe*
apply *fastforce*
apply *fastforce*
apply (erule_tac $x=x$ in *allE*, *simp*)
apply (rule_tac $x=\{x\}$ in *exI*, *auto*)
done

lemma *topological_basis_iff*:
assumes $\bigwedge B'. B' \in B \implies \text{open } B'$
shows *topological_basis* $B \longleftrightarrow (\forall O'. \text{open } O' \longrightarrow (\forall x \in O'. \exists B' \in B. x \in B' \wedge B' \subseteq O'))$
 (is $\_ \longleftrightarrow ?rhs$)

proof *safe*
fix O' **and** $x::'a$
assume H : *topological_basis* B *open* O' $x \in O'$
then have $(\exists B' \subseteq B. \bigcup B' = O')$ **by** (simp add: *topological_basis_def*)
then obtain B' **where** $B' \subseteq B$ $O' = \bigcup B'$ **by** *auto*
then show $\exists B' \in B. x \in B' \wedge B' \subseteq O'$ **using** H **by** *auto*

next
assume H : *?rhs*
show *topological_basis* B
using *assms* **unfolding** *topological_basis_def*
proof *safe*
fix $O' :: 'a$ *set*
assume *open* O'
with H **obtain** f **where** $\forall x \in O'. f\ x \in B \wedge x \in f\ x \wedge f\ x \subseteq O'$
by (force *intro*: *bchoice* *simp*: *Bex_def*)
then show $\exists B' \subseteq B. \bigcup B' = O'$
by (*auto* *intro*: *exI* [**where** $x=\{f\ x \mid x. x \in O'\}$])
qed
qed

```

lemma topological_basisI:
  assumes  $\bigwedge B'. B' \in B \implies \text{open } B'$ 
  and  $\bigwedge O' x. \text{open } O' \implies x \in O' \implies \exists B' \in B. x \in B' \wedge B' \subseteq O'$ 
  shows topological_basis B
  using assms by (subst topological_basis_iff) auto

lemma topological_basisE:
  fixes O'
  assumes topological_basis B
  and open O'
  and  $x \in O'$ 
  obtains B' where  $B' \in B$   $x \in B'$   $B' \subseteq O'$ 
proof atomize_elim
  from assms have  $\bigwedge B'. B' \in B \implies \text{open } B'$ 
  by (simp add: topological_basis_def)
  with topological_basis_iff assms
  show  $\exists B'. B' \in B \wedge x \in B' \wedge B' \subseteq O'$ 
  using assms by (simp add: Bex_def)
qed

lemma topological_basis_open:
  assumes topological_basis B
  and  $X \in B$ 
  shows open X
  using assms by (simp add: topological_basis_def)

lemma topological_basis_imp_subbasis:
  assumes B: topological_basis B
  shows open = generate_topology B
proof (intro ext iffI)
  fix S :: 'a set
  assume open S
  with B obtain B' where  $B' \subseteq B$   $S = \bigcup B'$ 
  unfolding topological_basis_def by blast
  then show generate_topology B S
  by (auto intro: generate_topology.intros dest: topological_basis_open)
next
  fix S :: 'a set
  assume generate_topology B S
  then show open S
  by induct (auto dest: topological_basis_open[OF B])
qed

lemma basis_dense:
  fixes B :: 'a set set
  and f :: 'a set  $\Rightarrow$  'a
  assumes topological_basis B
  and choosefrom_basis:  $\bigwedge B'. B' \neq \{\} \implies f B' \in B'$ 

```

```

    shows  $\forall X. \text{open } X \longrightarrow X \neq \{\} \longrightarrow (\exists B' \in B. f B' \in X)$ 
  proof (intro allI impI)
    fix X :: 'a set
    assume open X and X  $\neq \{\}$ 
    from topological_basisE[OF  $\langle \text{topological\_basis } B \rangle \langle \text{open } X \rangle \text{choosefrom\_basis}[OF$ 
 $\langle X \neq \{\} \rangle]$ ]
    obtain B' where B'  $\in B$  f X  $\in B'$  B'  $\subseteq X$  .
    then show  $\exists B' \in B. f B' \in X$ 
      by (auto intro!: choosefrom_basis)
  qed

end

lemma topological_basis_prod:
  assumes A: topological_basis A
    and B: topological_basis B
  shows topological_basis  $((\lambda(a, b). a \times b) \text{ ` } (A \times B))$ 
  unfolding topological_basis_def
proof (safe, simp_all del: ex_simps add: subset_image_iff ex_simps(1)[symmetric])
  fix S :: ('a  $\times$  'b) set
  assume open S
  then show  $\exists X \subseteq A \times B. (\bigcup (a, b) \in X. a \times b) = S$ 
  proof (safe intro!: exI[of _  $\{x \in A \times B. \text{fst } x \times \text{snd } x \subseteq S\}$ ])
    fix x y
    assume  $(x, y) \in S$ 
    from open_prod_elim[OF  $\langle \text{open } S \rangle$  this]
    obtain a b where a: open a  $x \in a$  and b: open b  $y \in b$  and  $a \times b \subseteq S$ 
      by (metis mem_Sigma_iff)
    moreover
    from A a obtain A0 where A0  $\in A$   $x \in A0$  A0  $\subseteq a$ 
      by (rule topological_basisE)
    moreover
    from B b obtain B0 where B0  $\in B$   $y \in B0$  B0  $\subseteq b$ 
      by (rule topological_basisE)
    ultimately show  $(x, y) \in (\bigcup (a, b) \in \{X \in A \times B. \text{fst } X \times \text{snd } X \subseteq S\}. a \times b)$ 
      by (intro UN_I[of (A0, B0)]) auto
  qed auto
qed (metis A B topological_basis_open open_Times)

```

2.1.2 Countable Basis

```

locale countable_basis = topological_space p for p::'a set  $\Rightarrow$  bool +
  fixes B :: 'a set set
  assumes is_basis: topological_basis B
    and countable_basis: countable B
begin

lemma open_countable_basis_ex:

```

```

assumes  $p\ X$ 
shows  $\exists B' \subseteq B. X = \bigcup B'$ 
using assms countable_basis is_basis
unfolding topological_basis_def by blast

```

```

lemma open_countable_basisE:
  assumes  $p\ X$ 
  obtains  $B'$  where  $B' \subseteq B\ X = \bigcup B'$ 
  using assms open_countable_basis_ex
  by atomize_elim simp

```

```

lemma countable_dense_exists:
   $\exists D :: 'a\ set. countable\ D \wedge (\forall X. p\ X \longrightarrow X \neq \{\} \longrightarrow (\exists d \in D. d \in X))$ 
proof -
  let  $?f = (\lambda B'. SOME\ x. x \in B')$ 
  have countable ( $?f\ 'B$ ) using countable_basis by simp
  with basis_dense[OF is_basis, of ?f] show ?thesis
    by (intro exI[where x=?f 'B]) (metis (mono_tags) all_not_in_conv imageI someI)
qed

```

```

lemma countable_dense_setE:
  obtains  $D :: 'a\ set$ 
  where countable  $D \wedge X. p\ X \Longrightarrow X \neq \{\} \Longrightarrow \exists d \in D. d \in X$ 
  using countable_dense_exists by blast

```

end

```

lemma countable_basis_openI: countable_basis open B
  if countable B topological_basis B
  using that
  by unfold_locales
  (simp_all add: topological_basis topological_space.topological_basis topological_space_axioms)

```

```

lemma (in first_countable_topology) first_countable_basisE:
  fixes  $x :: 'a$ 
  obtains  $\mathcal{A}$  where countable  $\mathcal{A} \wedge A. A \in \mathcal{A} \Longrightarrow x \in A \wedge A. A \in \mathcal{A} \Longrightarrow open\ A$ 
   $\wedge S. open\ S \Longrightarrow x \in S \Longrightarrow (\exists A \in \mathcal{A}. A \subseteq S)$ 
proof -
  obtain  $\mathcal{A}$  where  $\mathcal{A}: (\forall i :: nat. x \in \mathcal{A}\ i \wedge open\ (\mathcal{A}\ i))\ (\forall S. open\ S \wedge x \in S \longrightarrow (\exists i. \mathcal{A}\ i \subseteq S))$ 
  using first_countable_basis[of x] by metis
  show thesis
  proof
    show countable (range  $\mathcal{A}$ )
    by simp
  qed (use  $\mathcal{A}$  in auto)
qed

```

```

lemma (in first_countable_topology) first_countable_basis_Int_stableE:
  obtains  $\mathcal{A}$  where countable  $\mathcal{A} \wedge A. A \in \mathcal{A} \implies x \in A \wedge A. A \in \mathcal{A} \implies \text{open } A$ 
     $\wedge S. \text{open } S \implies x \in S \implies (\exists A \in \mathcal{A}. A \subseteq S)$ 
     $\wedge A \ B. A \in \mathcal{A} \implies B \in \mathcal{A} \implies A \cap B \in \mathcal{A}$ 
proof atomize_elim
  obtain  $\mathcal{B}$  where  $\mathcal{B}$ :
    countable  $\mathcal{B}$ 
     $\wedge B. B \in \mathcal{B} \implies x \in B$ 
     $\wedge B. B \in \mathcal{B} \implies \text{open } B$ 
     $\wedge S. \text{open } S \implies x \in S \implies \exists B \in \mathcal{B}. B \subseteq S$ 
  by (rule first_countable_basisE) blast
  define  $\mathcal{A}$  where [abs_def]:
     $\mathcal{A} = (\lambda N. \bigcap ((\lambda n. \text{from\_nat\_into } \mathcal{B} \ n) \ ' N)) \ ' (\text{Collect finite::nat set set})$ 
  then show  $\exists \mathcal{A}. \text{countable } \mathcal{A} \wedge (\forall A. A \in \mathcal{A} \longrightarrow x \in A) \wedge (\forall A. A \in \mathcal{A} \longrightarrow \text{open } A) \wedge$ 
     $(\forall S. \text{open } S \longrightarrow x \in S \longrightarrow (\exists A \in \mathcal{A}. A \subseteq S)) \wedge (\forall A \ B. A \in \mathcal{A} \longrightarrow B \in \mathcal{A} \longrightarrow A \cap B \in \mathcal{A})$ 
  proof (safe intro!: exI[where x= $\mathcal{A}$ ])
    show countable  $\mathcal{A}$ 
      unfolding  $\mathcal{A}$ _def by (intro countable_image countable_Collect_finite)
    fix  $A$ 
    assume  $A \in \mathcal{A}$ 
    then show  $x \in A$  open  $A$ 
      using  $\mathcal{B}(4)[OF \text{ open\_UNIV}]$  by (auto simp:  $\mathcal{A}$ _def intro:  $\mathcal{B}$  from_nat_into)
  next
    let ?int =  $\lambda N. \bigcap (\text{from\_nat\_into } \mathcal{B} \ ' N)$ 
    fix  $A \ B$ 
    assume  $A \in \mathcal{A} \ B \in \mathcal{A}$ 
    then obtain  $N \ M$  where  $A = ?int \ N \ B = ?int \ M$  finite  $(N \cup M)$ 
      by (auto simp:  $\mathcal{A}$ _def)
    then show  $A \cap B \in \mathcal{A}$ 
      by (auto simp:  $\mathcal{A}$ _def intro!: image_eqI[where x= $N \cup M$ ])
  next
    fix  $S$ 
    assume open  $S \ x \in S$ 
    then obtain  $a$  where  $a: a \in \mathcal{B} \ a \subseteq S$  using  $\mathcal{B}$  by blast
    then show  $\exists a \in \mathcal{A}. a \subseteq S$  using  $a \ \mathcal{B}$ 
      by (intro bexI[where x= $a$ ]) (auto simp:  $\mathcal{A}$ _def intro: image_eqI[where
 $x = \{\text{to\_nat\_on } \mathcal{B} \ a\}$ ])
  qed
qed

```

```

lemma (in topological_space) first_countableI:
  assumes countable  $\mathcal{A}$ 
    and 1:  $\wedge A. A \in \mathcal{A} \implies x \in A \wedge A. A \in \mathcal{A} \implies \text{open } A$ 
    and 2:  $\wedge S. \text{open } S \implies x \in S \implies \exists A \in \mathcal{A}. A \subseteq S$ 
  shows  $\exists \mathcal{A}::\text{nat} \Rightarrow 'a \text{ set}. (\forall i. x \in \mathcal{A} \ i \wedge \text{open } (\mathcal{A} \ i)) \wedge (\forall S. \text{open } S \wedge x \in S \longrightarrow (\exists i. \mathcal{A} \ i \subseteq S))$ 

```

```

proof (safe intro!: exI[of _ from_nat_into  $\mathcal{A}$ ])
  fix  $i$ 
  have  $\mathcal{A} \neq \{\}$  using 2[of UNIV] by auto
  show  $x \in \text{from\_nat\_into } \mathcal{A} \ i \ \text{open } (\text{from\_nat\_into } \mathcal{A} \ i)$ 
    using range_from_nat_into_subset[OF  $\langle \mathcal{A} \neq \{\} \rangle$ ] 1 by auto
next
  fix  $S$ 
  assume  $\text{open } S \ x \in S$  from 2[OF this]
  show  $\exists i. \text{from\_nat\_into } \mathcal{A} \ i \subseteq S$ 
    using subset_range_from_nat_into[OF  $\langle \text{countable } \mathcal{A} \rangle$ ] by auto
qed

instance prod :: (first_countable_topology, first_countable_topology) first_countable_topology
proof
  fix  $x :: 'a \times 'b$ 
  obtain  $\mathcal{A}$  where  $\mathcal{A}$ :
    countable  $\mathcal{A}$ 
     $\bigwedge a. a \in \mathcal{A} \implies \text{fst } x \in a$ 
     $\bigwedge a. a \in \mathcal{A} \implies \text{open } a$ 
     $\bigwedge S. \text{open } S \implies \text{fst } x \in S \implies \exists a \in \mathcal{A}. a \subseteq S$ 
  by (rule first_countable_basisE[of fst  $x$ ]) blast
  obtain  $B$  where  $B$ :
    countable  $B$ 
     $\bigwedge a. a \in B \implies \text{snd } x \in a$ 
     $\bigwedge a. a \in B \implies \text{open } a$ 
     $\bigwedge S. \text{open } S \implies \text{snd } x \in S \implies \exists a \in B. a \subseteq S$ 
  by (rule first_countable_basisE[of snd  $x$ ]) blast
  show  $\exists \mathcal{A} :: \text{nat} \Rightarrow ('a \times 'b) \ \text{set.}$ 
     $(\forall i. x \in \mathcal{A} \ i \wedge \text{open } (\mathcal{A} \ i)) \wedge (\forall S. \text{open } S \wedge x \in S \longrightarrow (\exists i. \mathcal{A} \ i \subseteq S))$ 
  proof (rule first_countableI[of  $(\lambda(a, b). a \times b) \ '(\mathcal{A} \times B)$ ], safe)
    fix  $a \ b$ 
    assume  $x: a \in \mathcal{A} \ b \in B$ 
    show  $x \in a \times b$ 
      by (simp add:  $\mathcal{A}(2) \ B(2) \ \text{mem\_Times\_iff } x$ )
    show  $\text{open } (a \times b)$ 
      by (simp add:  $\mathcal{A}(3) \ B(3) \ \text{open\_Times } x$ )
  next
  fix  $S$ 
  assume  $\text{open } S \ x \in S$ 
  then obtain  $a' \ b'$  where  $a' b'$ :  $\text{open } a' \ \text{open } b' \ x \in a' \times b' \ a' \times b' \subseteq S$ 
    by (rule open_prod_elim)
  moreover
  from  $a' b'$   $\mathcal{A}(4)[\text{of } a'] \ B(4)[\text{of } b']$ 
  obtain  $a \ b$  where  $a \in \mathcal{A} \ a \subseteq a' \ b \in B \ b \subseteq b'$ 
    by auto
  ultimately
  show  $\exists a \in (\lambda(a, b). a \times b) \ '(\mathcal{A} \times B). a \subseteq S$ 
    by (auto intro!: bexI[of _  $a \times b$ ] bexI[of _  $a$ ] bexI[of _  $b$ ])
qed (simp add:  $\mathcal{A} \ B$ )

```

qed

```

class second_countable_topology = topological_space +
  assumes ex_countable_subbasis:
     $\exists B::'a \text{ set set. countable } B \wedge \text{open} = \text{generate\_topology } B$ 
begin

lemma ex_countable_basis:  $\exists B::'a \text{ set set. countable } B \wedge \text{topological\_basis } B$ 
proof -
  from ex_countable_subbasis obtain B where B: countable B open = generate_topology B
  by blast
  let ?B = Inter ' {b. finite b  $\wedge$  b  $\subseteq$  B }

  show ?thesis
  proof (intro exI conjI)
    show countable ?B
    by (intro countable_image countable_Collect_finite_subset B)
  {
    fix S
    assume open S
    then have  $\exists B' \subseteq \{b. \text{finite } b \wedge b \subseteq B\}. (\bigcup b \in B'. \bigcap b) = S$ 
      unfolding B
    proof induct
      case UNIV
      show ?case by (intro exI[of _ {}]) simp
    next
      case (Int a b)
      then obtain x y where x:  $a = \bigcup (\text{Inter ' } x) \wedge i. i \in x \implies \text{finite } i \wedge i \subseteq B$ 
        and y:  $b = \bigcup (\text{Inter ' } y) \wedge i. i \in y \implies \text{finite } i \wedge i \subseteq B$ 
        by blast
      show ?case
      unfolding x y Int_UN_distrib2
      by (intro exI[of _ {i  $\cup$  j | i j. i  $\in$  x  $\wedge$  j  $\in$  y}]) (auto dest: x(2) y(2))
    next
      case (UN K)
      then have  $\forall k \in K. \exists B' \subseteq \{b. \text{finite } b \wedge b \subseteq B\}. \bigcup (\text{Inter ' } B') = k$  by auto
      then obtain k where
         $\forall ka \in K. k \text{ ka} \subseteq \{b. \text{finite } b \wedge b \subseteq B\} \wedge \bigcup (\text{Inter ' } (k \text{ ka})) = k \text{ ka}$ 
        unfolding bchoice_iff ..
      then show  $\exists B' \subseteq \{b. \text{finite } b \wedge b \subseteq B\}. \bigcup (\text{Inter ' } B') = \bigcup K$ 
        by (intro exI[of _  $\bigcup (k \text{ ' } K)$ ]) auto
    next
      case (Basis S)
      then show ?case
      by (intro exI[of _ {{S}}]) auto
  }
qed
then have  $(\exists B' \subseteq \text{Inter ' } \{b. \text{finite } b \wedge b \subseteq B\}. \bigcup B' = S)$ 
  unfolding subset_image_iff by blast }

```



```

    then show topological_basis ?B
      unfolding topological_basis_def
      by (safe intro!: open_Inter)
        (simp_all add: B generate_topology.Basis subset_eq)
  qed
qed

end

lemma univ_second_countable:
  obtains  $\mathcal{B} :: 'a::second\_countable\_topology\ set\ set$ 
  where countable  $\mathcal{B} \wedge C. C \in \mathcal{B} \implies open\ C$ 
     $\wedge S. open\ S \implies \exists U. U \subseteq \mathcal{B} \wedge S = \bigcup U$ 
  by (metis ex_countable_basis topological_basis_def)

proposition Lindelof:
  fixes  $\mathcal{F} :: 'a::second\_countable\_topology\ set\ set$ 
  assumes  $\mathcal{F}: \wedge S. S \in \mathcal{F} \implies open\ S$ 
  obtains  $\mathcal{F}'$  where  $\mathcal{F}' \subseteq \mathcal{F}$  countable  $\mathcal{F}' \cup \mathcal{F}' = \bigcup \mathcal{F}$ 
  proof -
    obtain  $\mathcal{B} :: 'a\ set\ set$ 
      where countable  $\mathcal{B} \wedge C. C \in \mathcal{B} \implies open\ C$ 
        and  $\mathcal{B}: \wedge S. open\ S \implies \exists U. U \subseteq \mathcal{B} \wedge S = \bigcup U$ 
      using univ_second_countable by blast
    define  $\mathcal{D}$  where  $\mathcal{D} \equiv \{S. S \in \mathcal{B} \wedge (\exists U. U \in \mathcal{F} \wedge S \subseteq U)\}$ 
    have countable  $\mathcal{D}$ 
      apply (rule countable_subset [OF _ ⟨countable  $\mathcal{B}$ ⟩])
      apply (force simp:  $\mathcal{D}$ _def)
      done
    have  $\wedge S. \exists U. S \in \mathcal{D} \longrightarrow U \in \mathcal{F} \wedge S \subseteq U$ 
      by (simp add:  $\mathcal{D}$ _def)
    then obtain  $G$  where  $G: \wedge S. S \in \mathcal{D} \longrightarrow G\ S \in \mathcal{F} \wedge S \subseteq G\ S$ 
      by metis
    have  $\bigcup \mathcal{F} \subseteq \bigcup \mathcal{D}$ 
      unfolding  $\mathcal{D}$ _def by (blast dest:  $\mathcal{F}\ \mathcal{B}$ )
    moreover have  $\bigcup \mathcal{D} \subseteq \bigcup \mathcal{F}$ 
      using  $\mathcal{D}$ _def by blast
    ultimately have eq1:  $\bigcup \mathcal{F} = \bigcup \mathcal{D} ..$ 
    have eq2:  $\bigcup \mathcal{D} = \bigcup (G\ \mathcal{D})$ 
      using  $G$  eq1 by auto
    show ?thesis
      apply (rule_tac  $\mathcal{F}' = G\ \mathcal{D}$  in that)
      using  $G$  ⟨countable  $\mathcal{D}$ ⟩
      by (auto simp: eq1 eq2)
  qed

```

```

lemma countable_disjoint_open_subsets:
  fixes  $\mathcal{F} :: 'a::second\_countable\_topology\ set\ set$ 

```

```

    assumes  $\bigwedge S. S \in \mathcal{F} \implies \text{open } S$  and pw: pairwise disjoint  $\mathcal{F}$ 
    shows countable  $\mathcal{F}$ 
  proof -
    obtain  $\mathcal{F}'$  where  $\mathcal{F}' \subseteq \mathcal{F}$  countable  $\mathcal{F}' \cup \mathcal{F}' = \bigcup \mathcal{F}$ 
    by (meson assms Lindelof)
    with pw have  $\mathcal{F} \subseteq \text{insert } \{\} \mathcal{F}'$ 
    by (fastforce simp add: pairwise_def disjoint_iff)
    then show ?thesis
    by (simp add: countable_F' countable_subset)
  qed

```

```

sublocale second_countable_topology <
  countable_basis open SOME B. countable B  $\wedge$  topological_basis B
  using someI_ex[OF ex_countable_basis]
  by unfold_locales safe

```

```

instance prod :: (second_countable_topology, second_countable_topology) second_countable_topology
proof
  obtain A :: 'a set set where countable A topological_basis A
  using ex_countable_basis by auto
  moreover
  obtain B :: 'b set set where countable B topological_basis B
  using ex_countable_basis by auto
  ultimately show  $\exists B::('a \times 'b)$  set set. countable B  $\wedge$  open = generate_topology
  B
  by (auto intro!: exI[of _  $(\lambda(a, b). a \times b)$  '  $(A \times B)$ ] topological_basis_prod
    topological_basis_imp_subbasis)
  qed

```

```

instance second_countable_topology  $\subseteq$  first_countable_topology
proof
  fix x :: 'a
  define B :: 'a set set where B = (SOME B. countable B  $\wedge$  topological_basis B)
  then have B: countable B topological_basis B
  using countable_basis is_basis
  by (auto simp: countable_basis is_basis)
  then show  $\exists A::\text{nat} \Rightarrow 'a$  set.
    ( $\forall i. x \in A\ i \wedge \text{open } (A\ i)$ )  $\wedge$  ( $\forall S. \text{open } S \wedge x \in S \longrightarrow (\exists i. A\ i \subseteq S)$ )
  by (intro first_countableI[of {b  $\in$  B. x  $\in$  b}])
    (fastforce simp: topological_space_class.topological_basis_def)+
  qed

```

```

instance nat :: second_countable_topology
proof
  show  $\exists B::\text{nat}$  set set. countable B  $\wedge$  open = generate_topology B
  by (intro exI[of _ range lessThan  $\cup$  range greaterThan]) (auto simp: open_nat_def)
  qed

```

```

lemma countable_separating_set_linorder1:
  shows  $\exists B::('a::\{linorder\_topology, second\_countable\_topology\} set). countable$ 
 $B \wedge (\forall x y. x < y \longrightarrow (\exists b \in B. x < b \wedge b \leq y))$ 
proof -
  obtain A::'a set set where countable A topological_basis A using ex_countable_basis
  by auto
  define B1 where  $B1 = \{(LEAST x. x \in U) \mid U. U \in A\}$ 
  then have countable B1 using countable A by (simp add: Setcompr_eq_image)
  define B2 where  $B2 = \{(SOME x. x \in U) \mid U. U \in A\}$ 
  then have countable B2 using countable A by (simp add: Setcompr_eq_image)
  have  $\exists b \in B1 \cup B2. x < b \wedge b \leq y$  if  $x < y$  for  $x y$ 
  proof (cases)
    assume  $\exists z. x < z \wedge z < y$ 
    then obtain z where  $z: x < z \wedge z < y$  by auto
    define U where  $U = \{x <..<y\}$ 
    then have open U by simp
    moreover have  $z \in U$  using z U_def by simp
    ultimately obtain V where  $V \in A z \in V V \subseteq U$ 
    using topological_basisE[OF topological_basis A] by auto
    define w where  $w = (SOME x. x \in V)$ 
    then have  $w \in V$  using z V by (metis someI2)
    then have  $x < w \wedge w \leq y$  using w V V U_def by fastforce
    moreover have  $w \in B1 \cup B2$  using w_def B2_def V V A by auto
    ultimately show ?thesis by auto
  next
    assume  $\neg(\exists z. x < z \wedge z < y)$ 
    then have *:  $\bigwedge z. z > x \implies z \geq y$  by auto
    define U where  $U = \{x <..<y\}$ 
    then have open U by simp
    moreover have  $y \in U$  using x y U_def by simp
    ultimately obtain V where  $V \in A y \in V V \subseteq U$ 
    using topological_basisE[OF topological_basis A] by auto
    have  $U = \{y..<y\}$  unfolding U_def using * x y by auto
    then have  $V \subseteq \{y..<y\}$  using V V U by simp
    then have  $(LEAST w. w \in V) = y$  using y V by (meson Least_equality
atLeast_iff subsetCE)
    then have  $y \in B1 \cup B2$  using V V A B1_def by auto
    moreover have  $x < y \wedge y \leq y$  using x y by simp
    ultimately show ?thesis by auto
  qed
  moreover have countable (B1  $\cup$  B2) using countable B1 countable B2 by
simp
  ultimately show ?thesis by auto
qed

```

```

lemma countable_separating_set_linorder2:
  shows  $\exists B::('a::\{linorder\_topology, second\_countable\_topology\} set). countable$ 
 $B \wedge (\forall x y. x < y \longrightarrow (\exists b \in B. x \leq b \wedge b < y))$ 
proof -

```

```

obtain A::'a set set where countable A topological_basis A using ex_countable_basis
by auto
define B1 where B1 = {(GREATEST x. x ∈ U) | U. U ∈ A}
then have countable B1 using ⟨countable A⟩ by (simp add: Setcompr_eq_image)
define B2 where B2 = {(SOME x. x ∈ U) | U. U ∈ A}
then have countable B2 using ⟨countable A⟩ by (simp add: Setcompr_eq_image)
have ∃ b ∈ B1 ∪ B2. x ≤ b ∧ b < y if x < y for x y
proof (cases)
  assume ∃ z. x < z ∧ z < y
  then obtain z where z: x < z ∧ z < y by auto
  define U where U = {x <..then have open U by simp
  moreover have z ∈ U using z U_def by simp
  ultimately obtain V where V ∈ A z ∈ V V ⊆ U
    using topological_basisE[OF ⟨topological_basis A⟩] by auto
  define w where w = (SOME x. x ∈ V)
  then have w ∈ V using ⟨z ∈ V⟩ by (metis someI2)
  then have x ≤ w ∧ w < y using ⟨w ∈ V⟩ ⟨V ⊆ U⟩ U_def by fastforce
  moreover have w ∈ B1 ∪ B2 using w_def B2_def ⟨V ∈ A⟩ by auto
  ultimately show ?thesis by auto
next
  assume ¬(∃ z. x < z ∧ z < y)
  then have *: ∧ z. z < y ⇒ z ≤ x using leI by blast
  define U where U = {..then have open U by simp
  moreover have x ∈ U using ⟨x < y⟩ U_def by simp
  ultimately obtain V where V ∈ A x ∈ V V ⊆ U
    using topological_basisE[OF ⟨topological_basis A⟩] by auto
  have U = {..x} unfolding U_def using * ⟨x < y⟩ by auto
  then have V ⊆ {..x} using ⟨V ⊆ U⟩ by simp
  then have (GREATEST x. x ∈ V) = x using ⟨x ∈ V⟩ by (meson Greatest_equality atMost_iff subsetCE)
  then have x ∈ B1 ∪ B2 using ⟨V ∈ A⟩ B1_def by auto
  moreover have x ≤ x ∧ x < y using ⟨x < y⟩ by simp
  ultimately show ?thesis by auto
qed
moreover have countable (B1 ∪ B2) using ⟨countable B1⟩ ⟨countable B2⟩ by
simp
ultimately show ?thesis by auto
qed

```

lemma countable_separating_set_dense_linorder:

shows ∃ B::('a::{linorder_topology, dense_linorder, second_countable_topology} set). countable B ∧ (∀ x y. x < y ⇒ (∃ b ∈ B. x < b ∧ b < y))

proof –

obtain B::'a set **where** B: countable B ∧ x y. x < y ⇒ (∃ b ∈ B. x < b ∧ b ≤ y)

using countable_separating_set_linorder1 **by** auto

have ∃ b ∈ B. x < b ∧ b < y **if** x < y **for** x y

```

proof -
  obtain  $z$  where  $x < z$   $z < y$  using  $\langle x < y \rangle$  dense by blast
  then obtain  $b$  where  $b \in B$   $x < b \wedge b \leq z$  using  $B(2)$  by auto
  then have  $x < b \wedge b < y$  using  $\langle z < y \rangle$  by auto
  then show ?thesis using  $\langle b \in B \rangle$  by auto
qed
then show ?thesis using  $B(1)$  by auto
qed

```

2.1.3 Polish spaces

Textbooks define Polish spaces as completely metrizable. We assume the topology to be complete for a given metric.

```
class polish_space = complete_space + second_countable_topology
```

2.1.4 Limit Points

```

definition (in topological_space) islimpt:: 'a  $\Rightarrow$  'a set  $\Rightarrow$  bool (infixr islimpt 60)
  where  $x$  islimpt  $S \iff (\forall T. x \in T \longrightarrow \text{open } T \longrightarrow (\exists y \in S. y \in T \wedge y \neq x))$ 

```

```

lemma islimptI:
  assumes  $\bigwedge T. x \in T \implies \text{open } T \implies \exists y \in S. y \in T \wedge y \neq x$ 
  shows  $x$  islimpt  $S$ 
  using assms unfolding islimpt_def by auto

```

```

lemma islimptE:
  assumes  $x$  islimpt  $S$  and  $x \in T$  and open  $T$ 
  obtains  $y$  where  $y \in S$  and  $y \in T$  and  $y \neq x$ 
  using assms unfolding islimpt_def by auto

```

```

lemma islimpt_iff_eventually:  $x$  islimpt  $S \iff \neg \text{eventually } (\lambda y. y \notin S) \text{ (at } x)$ 
  unfolding islimpt_def eventually_at_topological by auto

```

```

lemma islimpt_subset:  $x$  islimpt  $S \implies S \subseteq T \implies x$  islimpt  $T$ 
  unfolding islimpt_def by fast

```

```

lemma islimpt_UNIV_iff:  $x$  islimpt UNIV  $\iff \neg \text{open } \{x\}$ 
  unfolding islimpt_def by (safe, fast, case_tac  $T = \{x\}$ , fast, fast)

```

```

lemma islimpt_punctured:  $x$  islimpt  $S = x$  islimpt  $(S - \{x\})$ 
  unfolding islimpt_def by blast

```

A perfect space has no isolated points.

```

lemma islimpt_UNIV [simp, intro]:  $x$  islimpt UNIV
  for  $x :: 'a::\text{perfect\_space}$ 
  unfolding islimpt_UNIV_iff by (rule not_open_singleton)

```

```

lemma closed_limpt: closed  $S \iff (\forall x. x$  islimpt  $S \longrightarrow x \in S)$ 
  unfolding closed_def

```

```

apply (subst open_subopen)
apply (simp add: islimpt_def subset_eq)
apply (metis ComplE ComplI)
done

```

```

lemma islimpt_EMPTY[simp]:  $\neg x \text{ islimpt } \{\}$ 
  by (auto simp: islimpt_def)

```

```

lemma islimpt_Un:  $x \text{ islimpt } (S \cup T) \longleftrightarrow x \text{ islimpt } S \vee x \text{ islimpt } T$ 
  by (simp add: islimpt_iff_eventually_eventually_conj_iff)

```

```

lemma islimpt_insert:
  fixes  $x :: 'a::t1\_space$ 
  shows  $x \text{ islimpt } (\text{insert } a \ s) \longleftrightarrow x \text{ islimpt } s$ 
proof
  assume  $*$ :  $x \text{ islimpt } (\text{insert } a \ s)$ 
  show  $x \text{ islimpt } s$ 
  proof (rule islimptI)
    fix  $t$ 
    assume  $t$ :  $x \in t \text{ open } t$ 
    show  $\exists y \in s. y \in t \wedge y \neq x$ 
    proof (cases  $x = a$ )
      case True
        obtain  $y$  where  $y \in \text{insert } a \ s \ y \in t \ y \neq x$ 
        using  $*$  by (rule islimptE)
        with  $\langle x = a \rangle$  show ?thesis by auto
      case False
        with  $t$  have  $t'$ :  $x \in t - \{a\} \text{ open } (t - \{a\})$ 
        by (simp_all add: open_Diff)
        obtain  $y$  where  $y \in \text{insert } a \ s \ y \in t - \{a\} \ y \neq x$ 
        using  $*$   $t'$  by (rule islimptE)
        then show ?thesis by auto
    qed
  qed
next
  assume  $x \text{ islimpt } s$ 
  then show  $x \text{ islimpt } (\text{insert } a \ s)$ 
    by (rule islimpt_subset) auto
qed

```

```

lemma islimpt_finite:
  fixes  $x :: 'a::t1\_space$ 
  shows finite  $s \implies \neg x \text{ islimpt } s$ 
  by (induct set: finite) (simp_all add: islimpt_insert)

```

```

lemma islimpt_Un_finite:
  fixes  $x :: 'a::t1\_space$ 

```

shows $\text{finite } s \implies x \text{ islimpt } (s \cup t) \longleftrightarrow x \text{ islimpt } t$
by (*simp add: islimpt_Un islimpt_finite*)

lemma *islimpt_eq_acc_point*:

fixes $l :: 'a :: t1\_space$

shows $l \text{ islimpt } S \longleftrightarrow (\forall U. l \in U \longrightarrow \text{open } U \longrightarrow \text{infinite } (U \cap S))$

proof (*safe intro!: islimptI*)

fix U

assume $l \text{ islimpt } S \ l \in U \ \text{open } U \ \text{finite } (U \cap S)$

then have $l \text{ islimpt } S \ l \in (U - (U \cap S - \{l\})) \ \text{open } (U - (U \cap S - \{l\}))$

by (*auto intro: finite_imp_closed*)

then show *False*

by (*rule islimptE*) *auto*

next

fix T

assume $l \in T \ \forall U. l \in U \longrightarrow \text{open } U \longrightarrow \text{infinite } (U \cap S) \ l \in T \ \text{open } T$

then have $\text{infinite } (T \cap S - \{l\})$

by *auto*

then have $\exists x. x \in (T \cap S - \{l\})$

unfolding *ex_in_conv* **by** (*intro notI*) *simp*

then show $\exists y \in S. y \in T \wedge y \neq l$

by *auto*

qed

lemma *acc_point_range_imp_convergent_subsequence*:

fixes $l :: 'a :: \text{first\_countable\_topology}$

assumes $l: \forall U. l \in U \longrightarrow \text{open } U \longrightarrow \text{infinite } (U \cap \text{range } f)$

shows $\exists r :: \text{nat} \Rightarrow \text{nat}. \text{strict\_mono } r \wedge (f \circ r) \longrightarrow l$

proof –

from *countable_basis_at_decseq*[*of l*]

obtain A **where** A :

$\bigwedge i. \text{open } (A \ i)$

$\bigwedge i. l \in A \ i$

$\bigwedge S. \text{open } S \implies l \in S \implies \text{eventually } (\lambda i. A \ i \subseteq S) \text{ sequentially}$

by *blast*

define s **where** $s \ n \ i = (\text{SOME } j. i < j \wedge f \ j \in A \ (Suc \ n))$ **for** $n \ i$

{

fix $n \ i$

have $\text{infinite } (A \ (Suc \ n) \cap \text{range } f - f' \{.. \ i\})$

using $l \ A$ **by** *auto*

then have $\exists x. x \in A \ (Suc \ n) \cap \text{range } f - f' \{.. \ i\}$

unfolding *ex_in_conv* **by** (*intro notI*) *simp*

then have $\exists j. f \ j \in A \ (Suc \ n) \wedge j \notin \{.. \ i\}$

by *auto*

then have $\exists a. i < a \wedge f \ a \in A \ (Suc \ n)$

by (*auto simp: not_le*)

then have $i < s \ n \ i \ f \ (s \ n \ i) \in A \ (Suc \ n)$

unfolding *s_def* **by** (*auto intro: someI2_ex*)

}

```

note  $s = \text{this}$ 
define  $r$  where  $r = \text{rec\_nat } (s \ 0 \ 0) \ s$ 
have  $\text{strict\_mono } r$ 
  by ( $\text{auto simp: } r\_def \ s \ \text{strict\_mono\_Suc\_iff}$ )
moreover
have  $(\lambda n. f \ (r \ n)) \longrightarrow l$ 
proof ( $\text{rule topological\_tendstoI}$ )
  fix  $S$ 
  assume  $\text{open } S \ l \in S$ 
  with  $A(3)$  have  $\text{eventually } (\lambda i. A \ i \subseteq S) \text{ sequentially}$ 
    by  $\text{auto}$ 
  moreover
  {
    fix  $i$ 
    assume  $\text{Suc } 0 \leq i$ 
    then have  $f \ (r \ i) \in A \ i$ 
      by ( $\text{cases } i$ ) ( $\text{simp\_all add: } r\_def \ s$ )
  }
  then have  $\text{eventually } (\lambda i. f \ (r \ i) \in A \ i) \text{ sequentially}$ 
    by ( $\text{auto simp: eventually\_sequentially}$ )
  ultimately show  $\text{eventually } (\lambda i. f \ (r \ i) \in S) \text{ sequentially}$ 
    by  $\text{eventually\_elim auto}$ 
qed
ultimately show  $\exists r::\text{nat} \Rightarrow \text{nat}. \text{strict\_mono } r \wedge (f \circ r) \longrightarrow l$ 
  by ( $\text{auto simp: convergent\_def comp\_def}$ )
qed

```

```

lemma  $\text{islimpt\_range\_imp\_convergent\_subsequence}$ :
  fixes  $l :: 'a :: \{t1\_space, \text{first\_countable\_topology}\}$ 
  assumes  $l: l \text{ islimpt } (\text{range } f)$ 
  shows  $\exists r::\text{nat} \Rightarrow \text{nat}. \text{strict\_mono } r \wedge (f \circ r) \longrightarrow l$ 
  using  $l$  unfolding  $\text{islimpt\_eq\_acc\_point}$ 
  by ( $\text{rule acc\_point\_range\_imp\_convergent\_subsequence}$ )

```

```

lemma  $\text{sequence\_unique\_limpt}$ :
  fixes  $f :: \text{nat} \Rightarrow 'a::t2\_space$ 
  assumes  $(f \longrightarrow l) \text{ sequentially}$ 
    and  $l' \text{ islimpt } (\text{range } f)$ 
  shows  $l' = l$ 
proof ( $\text{rule ccontr}$ )
  assume  $l' \neq l$ 
  obtain  $s \ t$  where  $\text{open } s \ \text{open } t \ l' \in s \ l \in t \ s \cap t = \{\}$ 
    using  $\text{hausdorff } [OF \ \langle l' \neq l \rangle]$  by  $\text{auto}$ 
  have  $\text{eventually } (\lambda n. f \ n \in t) \text{ sequentially}$ 
    using  $\text{assms}(1) \ \langle \text{open } t \rangle \ \langle l \in t \rangle$  by ( $\text{rule topological\_tendstoD}$ )
  then obtain  $N$  where  $\forall n \geq N. f \ n \in t$ 
    unfolding  $\text{eventually\_sequentially}$  by  $\text{auto}$ 

  have  $UNIV = \{..<N\} \cup \{N..\}$ 

```



```

    by auto
  then have  $l' \text{ islimpt } (f \text{ ' } (\{..<N\} \cup \{N..\}))$ 
    using  $\text{assms}(2)$  by simp
  then have  $l' \text{ islimpt } (f \text{ ' } \{..<N\} \cup f \text{ ' } \{N..\})$ 
    by (simp add: image_Un)
  then have  $l' \text{ islimpt } (f \text{ ' } \{N..\})$ 
    by (simp add: islimpt_Un_finite)
  then obtain  $y$  where  $y \in f \text{ ' } \{N..\}$   $y \in s$   $y \neq l'$ 
    using  $\langle l' \in s \rangle \langle \text{open } s \rangle$  by (rule islimptE)
  then obtain  $n$  where  $N \leq n$   $f \ n \in s$   $f \ n \neq l'$ 
    by auto
  with  $\langle \forall n \geq N. f \ n \in t \rangle$  have  $f \ n \in s \cap t$ 
    by simp
  with  $\langle s \cap t = \{ \} \rangle$  show False
    by simp
qed

```

2.1.5 Interior of a Set

definition $\text{interior} :: ('a::\text{topological\_space}) \text{ set} \Rightarrow 'a \text{ set}$ **where**
 $\text{interior } S = \bigcup \{T. \text{open } T \wedge T \subseteq S\}$

lemma interiorI [intro?]:
 assumes $\text{open } T$ and $x \in T$ and $T \subseteq S$
 shows $x \in \text{interior } S$
 using assms **unfolding** interior_def by fast

lemma interiorE [elim?]:
 assumes $x \in \text{interior } S$
 obtains T where $\text{open } T$ and $x \in T$ and $T \subseteq S$
 using assms **unfolding** interior_def by fast

lemma open_interior [simp, intro]: $\text{open } (\text{interior } S)$
 by (simp add: interior_def open_Union)

lemma interior_subset : $\text{interior } S \subseteq S$
 by (auto simp: interior_def)

lemma interior_maximal : $T \subseteq S \implies \text{open } T \implies T \subseteq \text{interior } S$
 by (auto simp: interior_def)

lemma interior_open : $\text{open } S \implies \text{interior } S = S$
 by (intro equalityI interior_subset interior_maximal subset_refl)

lemma interior_eq : $\text{interior } S = S \longleftrightarrow \text{open } S$
 by (metis open_interior interior_open)

lemma $\text{open\_subset\_interior}$: $\text{open } S \implies S \subseteq \text{interior } T \longleftrightarrow S \subseteq T$
 by (metis interior_maximal interior_subset subset_trans)

```

lemma interior_empty [simp]: interior {} = {}
  using open_empty by (rule interior_open)

lemma interior_UNIV [simp]: interior UNIV = UNIV
  using open_UNIV by (rule interior_open)

lemma interior_interior [simp]: interior (interior S) = interior S
  using open_interior by (rule interior_open)

lemma interior_mono:  $S \subseteq T \implies \text{interior } S \subseteq \text{interior } T$ 
  by (auto simp: interior_def)

lemma interior_unique:
  assumes  $T \subseteq S$  and open T
  assumes  $\bigwedge T'. T' \subseteq S \implies \text{open } T' \implies T' \subseteq T$ 
  shows interior S = T
  by (intro equalityI assms interior_subset open_interior interior_maximal)

lemma interior_singleton [simp]: interior {a} = {}
  for a :: 'a::perfect_space
  by (meson interior_eq interior_subset not_open_singleton subset_singletonD)

lemma interior_Int [simp]: interior (S  $\cap$  T) = interior S  $\cap$  interior T
  by (meson Int_mono Int_subset_iff antisym_conv interior_maximal interior_subset
open_Int open_interior)

lemma eventually_nhds_in_nhd:  $x \in \text{interior } s \implies \text{eventually } (\lambda y. y \in s) (\text{nhds } x)$ 
  using interior_subset[of s] by (subst eventually_nhds) blast

lemma interior_limit_point [intro]:
  fixes x :: 'a::perfect_space
  assumes x:  $x \in \text{interior } S$ 
  shows x islimpt S
  using x islimpt_UNIV [of x]
  unfolding interior_def islimpt_def
  apply (clarsimp, rename_tac T T')
  apply (drule_tac x=T  $\cap$  T' in spec)
  apply (auto simp: open_Int)
  done

lemma interior_closed_Un_empty_interior:
  assumes cS: closed S
  and iT: interior T = {}
  shows interior (S  $\cup$  T) = interior S
proof
  show interior S  $\subseteq$  interior (S  $\cup$  T)
  by (rule interior_mono) (rule Un_upper1)

```

```

show  $\text{interior } (S \cup T) \subseteq \text{interior } S$ 
proof
  fix  $x$ 
  assume  $x \in \text{interior } (S \cup T)$ 
  then obtain  $R$  where  $\text{open } R$   $x \in R$   $R \subseteq S \cup T$  ..
  show  $x \in \text{interior } S$ 
  proof (rule ccontr)
    assume  $x \notin \text{interior } S$ 
    with  $\langle x \in R \rangle$   $\langle \text{open } R \rangle$  obtain  $y$  where  $y \in R - S$ 
      unfolding interior_def by fast
    from  $\langle \text{open } R \rangle$   $\langle \text{closed } S \rangle$  have  $\text{open } (R - S)$ 
      by (rule open_Diff)
    from  $\langle R \subseteq S \cup T \rangle$  have  $R - S \subseteq T$ 
      by fast
    from  $\langle y \in R - S \rangle$   $\langle \text{open } (R - S) \rangle$   $\langle R - S \subseteq T \rangle$   $\langle \text{interior } T = \{\} \rangle$  show
      False
    unfolding interior_def by fast
  qed
qed
qed

lemma interior_Times:  $\text{interior } (A \times B) = \text{interior } A \times \text{interior } B$ 
proof (rule interior_unique)
  show  $\text{interior } A \times \text{interior } B \subseteq A \times B$ 
    by (intro Sigma_mono interior_subset)
  show  $\text{open } (\text{interior } A \times \text{interior } B)$ 
    by (intro open_Times open_interior)
  fix  $T$ 
  assume  $T \subseteq A \times B$  and  $\text{open } T$ 
  then show  $T \subseteq \text{interior } A \times \text{interior } B$ 
  proof safe
    fix  $x$   $y$ 
    assume  $(x, y) \in T$ 
    then obtain  $C$   $D$  where  $\text{open } C$   $\text{open } D$   $C \times D \subseteq T$   $x \in C$   $y \in D$ 
      using  $\langle \text{open } T \rangle$  unfolding open_prod_def by fast
    then have  $\text{open } C$   $\text{open } D$   $C \subseteq A$   $D \subseteq B$   $x \in C$   $y \in D$ 
      using  $\langle T \subseteq A \times B \rangle$  by auto
    then show  $x \in \text{interior } A$  and  $y \in \text{interior } B$ 
      by (auto intro: interiorI)
  qed
qed

lemma interior_Ici:
  fixes  $x :: 'a :: \{\text{dense\_linorder}, \text{linorder\_topology}\}$ 
  assumes  $b < x$ 
  shows  $\text{interior } \{x ..\} = \{x <..\}$ 
proof (rule interior_unique)
  fix  $T$ 
  assume  $T \subseteq \{x ..\}$   $\text{open } T$ 

```

```

moreover have  $x \notin T$ 
proof
  assume  $x \in T$ 
  obtain  $y$  where  $y < x$   $\{y <.. x\} \subseteq T$ 
    using  $\text{open\_left}[OF \langle \text{open } T \rangle \langle x \in T \rangle \langle b < x \rangle]$  by auto
  with  $\text{dense}[OF \langle y < x \rangle]$  obtain  $z$  where  $z \in T$   $z < x$ 
    by (auto simp: subset_eq Ball_def)
  with  $\langle T \subseteq \{x ..\} \rangle$  show False by auto
qed
ultimately show  $T \subseteq \{x <.. \}$ 
  by (auto simp: subset_eq less_le)
qed auto

```

```

lemma interior_Iic:
  fixes  $x :: 'a :: \{\text{dense\_linorder}, \text{linorder\_topology}\}$ 
  assumes  $x < b$ 
  shows  $\text{interior } \{.. x\} = \{.. < x\}$ 
proof (rule interior_unique)
  fix  $T$ 
  assume  $T \subseteq \{.. x\}$  open  $T$ 
  moreover have  $x \notin T$ 
  proof
    assume  $x \in T$ 
    obtain  $y$  where  $x < y$   $\{x .. < y\} \subseteq T$ 
      using  $\text{open\_right}[OF \langle \text{open } T \rangle \langle x \in T \rangle \langle x < b \rangle]$  by auto
    with  $\text{dense}[OF \langle x < y \rangle]$  obtain  $z$  where  $z \in T$   $x < z$ 
      by (auto simp: subset_eq Ball_def less_le)
    with  $\langle T \subseteq \{.. x\} \rangle$  show False by auto
  qed
  ultimately show  $T \subseteq \{.. < x\}$ 
    by (auto simp: subset_eq less_le)
qed auto

```

```

lemma countable_disjoint_nonempty_interior_subsets:
  fixes  $\mathcal{F} :: 'a :: \text{second\_countable\_topology}$  set set
  assumes  $\text{pw: pairwise disjoint } \mathcal{F}$  and  $\text{int: } \bigwedge S. \llbracket S \in \mathcal{F}; \text{interior } S = \{\} \rrbracket \implies S = \{\}$ 
  shows countable  $\mathcal{F}$ 
proof (rule countable_image_inj_on)
  have disjoint (interior ‘  $\mathcal{F}$ )
    using  $\text{pw}$  by (simp add: disjoint_image_subset interior_subset)
  then show countable (interior ‘  $\mathcal{F}$ )
    by (auto intro: countable_disjoint_open_subsets)
  show inj_on interior  $\mathcal{F}$ 
    using  $\text{pw}$  apply (clarsimp simp: inj_on_def pairwise_def)
    apply (metis disjoint_def disjoint_subset1 inf.orderE int interior_subset)
    done
qed

```

2.1.6 Closure of a Set

definition *closure* :: $('a::\text{topological\_space}) \text{ set} \Rightarrow 'a \text{ set}$ **where**
closure $S = S \cup \{x . x \text{ islimpt } S\}$

lemma *interior_closure*: $\text{interior } S = - (\text{closure } (- S))$
by (*auto simp: interior_def closure_def islimpt_def*)

lemma *closure_interior*: $\text{closure } S = - \text{interior } (- S)$
by (*simp add: interior_closure*)

lemma *closed_closure*[*simp, intro*]: $\text{closed } (\text{closure } S)$
by (*simp add: closure_interior closed_Compl*)

lemma *closure_subset*: $S \subseteq \text{closure } S$
by (*simp add: closure_def*)

lemma *closure_hull*: $\text{closure } S = \text{closed hull } S$
by (*auto simp: hull_def closure_interior interior_def*)

lemma *closure_eq*: $\text{closure } S = S \longleftrightarrow \text{closed } S$
unfolding *closure_hull* **using** *closed_Inter* **by** (*rule hull_eq*)

lemma *closure_closed* [*simp*]: $\text{closed } S \Longrightarrow \text{closure } S = S$
by (*simp only: closure_eq*)

lemma *closure_closure* [*simp*]: $\text{closure } (\text{closure } S) = \text{closure } S$
unfolding *closure_hull* **by** (*rule hull_hull*)

lemma *closure_mono*: $S \subseteq T \Longrightarrow \text{closure } S \subseteq \text{closure } T$
unfolding *closure_hull* **by** (*rule hull_mono*)

lemma *closure_minimal*: $S \subseteq T \Longrightarrow \text{closed } T \Longrightarrow \text{closure } S \subseteq T$
unfolding *closure_hull* **by** (*rule hull_minimal*)

lemma *closure_unique*:
assumes $S \subseteq T$
and $\text{closed } T$
and $\bigwedge T'. S \subseteq T' \Longrightarrow \text{closed } T' \Longrightarrow T \subseteq T'$
shows $\text{closure } S = T$
using *assms* **unfolding** *closure_hull* **by** (*rule hull_unique*)

lemma *closure_empty* [*simp*]: $\text{closure } \{\} = \{\}$
using *closed_empty* **by** (*rule closure_closed*)

lemma *closure_UNIV* [*simp*]: $\text{closure } \text{UNIV} = \text{UNIV}$
using *closed_UNIV* **by** (*rule closure_closed*)

lemma *closure_Un* [*simp*]: $\text{closure } (S \cup T) = \text{closure } S \cup \text{closure } T$
by (*simp add: closure_interior*)

lemma *closure_eq_empty* [iff]: $\text{closure } S = \{\} \longleftrightarrow S = \{\}$
using *closure_empty closure_subset[of S]* **by** *blast*

lemma *closure_subset_eq*: $\text{closure } S \subseteq S \longleftrightarrow \text{closed } S$
using *closure_eq[of S] closure_subset[of S]* **by** *simp*

lemma *open_Int_closure_eq_empty*: $\text{open } S \implies (S \cap \text{closure } T) = \{\} \longleftrightarrow S \cap T = \{\}$
using *open_subset_interior[of S - T]*
using *interior_subset[of - T]*
by (*auto simp: closure_interior*)

lemma *open_Int_closure_subset*: $\text{open } S \implies S \cap \text{closure } T \subseteq \text{closure } (S \cap T)$

proof
fix x
assume *: $\text{open } S \ x \in S \cap \text{closure } T$
have $x \text{ islimpt } (S \cap T)$ **if** **: $x \text{ islimpt } T$
proof (*rule islimptI*)
fix A
assume $x \in A \text{ open } A$
with * **have** $x \in A \cap S \text{ open } (A \cap S)$
by (*simp_all add: open_Int*)
with ** **obtain** y **where** $y \in T \ y \in A \cap S \ y \neq x$
by (*rule islimptE*)
then have $y \in S \cap T \ y \in A \wedge y \neq x$
by *simp_all*
then show $\exists y \in (S \cap T). y \in A \wedge y \neq x$..
qed
with * **show** $x \in \text{closure } (S \cap T)$
unfolding *closure_def* **by** *blast*
qed

lemma *closure_complement*: $\text{closure } (- S) = - \text{interior } S$
by (*simp add: closure_interior*)

lemma *interior_complement*: $\text{interior } (- S) = - \text{closure } S$
by (*simp add: closure_interior*)

lemma *interior_diff*: $\text{interior } (S - T) = \text{interior } S - \text{closure } T$
by (*simp add: Diff_eq interior_complement*)

lemma *closure_Times*: $\text{closure } (A \times B) = \text{closure } A \times \text{closure } B$

proof (*rule closure_unique*)
show $A \times B \subseteq \text{closure } A \times \text{closure } B$
by (*intro Sigma_mono closure_subset*)
show $\text{closed } (\text{closure } A \times \text{closure } B)$
by (*intro closed_Times closed_closure*)
fix T

```

assume  $A \times B \subseteq T$  and closed  $T$ 
then show  $\text{closure } A \times \text{closure } B \subseteq T$ 
  apply (simp add: closed_def open_prod_def, clarify)
  apply (rule ccontr)
  apply (drule_tac x=(a, b) in bspec, simp, clarify, rename_tac C D)
  apply (simp add: closure_interior interior_def)
  apply (drule_tac x=C in spec)
  apply (drule_tac x=D in spec, auto)
  done
qed

lemma closure_open_Int_superset:
  assumes  $\text{open } S \subseteq \text{closure } T$ 
  shows  $\text{closure}(S \cap T) = \text{closure } S$ 
proof –
  have  $\text{closure } S \subseteq \text{closure}(S \cap T)$ 
  by (metis assms closed_closure closure_minimal inf.orderE open_Int_closure_subset)
  then show ?thesis
  by (simp add: closure_mono dual_order.antisym)
qed

lemma closure_Int:  $\text{closure}(\bigcap I) \leq \bigcap \{\text{closure } S \mid S. S \in I\}$ 
proof –
  {
    fix  $y$ 
    assume  $y \in \bigcap I$ 
    then have  $y: \forall S \in I. y \in S$  by auto
    {
      fix  $S$ 
      assume  $S \in I$ 
      then have  $y \in \text{closure } S$ 
      using closure_subset y by auto
    }
    then have  $y \in \bigcap \{\text{closure } S \mid S. S \in I\}$ 
    by auto
  }
  then have  $\bigcap I \subseteq \bigcap \{\text{closure } S \mid S. S \in I\}$ 
  by auto
  moreover have  $\text{closed}(\bigcap \{\text{closure } S \mid S. S \in I\})$ 
  unfolding closed_Inter closed_closure by auto
  ultimately show ?thesis using closure_hull[of  $\bigcap I$ ]
  hull_minimal[of  $\bigcap I \cap \{\text{closure } S \mid S. S \in I\}$  closed] by auto
qed

lemma islimpt_in_closure:  $(x \text{ islimpt } S) = (x \in \text{closure}(S - \{x\}))$ 
  unfolding closure_def using islimpt_punctured by blast

lemma connected_imp_connected_closure:  $\text{connected } S \implies \text{connected}(\text{closure } S)$ 
  by (rule connectedI) (meson closure_subset open_Int open_Int_closure_eq_empty)

```

subset_trans connectedD)

lemma *bdd_below_closure*:
 fixes $A :: \text{real set}$
 assumes *bdd_below A*
 shows *bdd_below (closure A)*
proof –
 from *assms* obtain m where $\bigwedge x. x \in A \implies m \leq x$
 by (*auto simp: bdd_below_def*)
 then have $A \subseteq \{m..\}$ by *auto*
 then have $\text{closure } A \subseteq \{m..\}$
 using *closed_real_atLeast* by (*rule closure_minimal*)
 then show *?thesis*
 by (*auto simp: bdd_below_def*)
qed

2.1.7 Frontier (also known as boundary)

definition *frontier* :: $('a::\text{topological\_space}) \text{ set} \Rightarrow 'a \text{ set}$ **where**
frontier S = closure S - interior S

lemma *frontier_closed [iff]: closed (frontier S)*
 by (*simp add: frontier_def closed_Diff*)

lemma *frontier_closures: frontier S = closure S $\cap$ closure (- S)*
 by (*auto simp: frontier_def interior_closure*)

lemma *frontier_Int: frontier(S $\cap$ T) = closure(S $\cap$ T) $\cap$ (frontier S $\cup$ frontier T)*

proof –
 have $\text{closure } (S \cap T) \subseteq \text{closure } S \text{ closure } (S \cap T) \subseteq \text{closure } T$
 by (*simp_all add: closure_mono*)
 then show *?thesis*
 by (*auto simp: frontier_closures*)
qed

lemma *frontier_Int_subset: frontier(S $\cap$ T) $\subseteq$ frontier S $\cup$ frontier T*
 by (*auto simp: frontier_Int*)

lemma *frontier_Int_closed*:
 assumes *closed S closed T*
 shows *frontier(S $\cap$ T) = (frontier S $\cap$ T) $\cup$ (S $\cap$ frontier T)*

proof –
 have $\text{closure } (S \cap T) = T \cap S$
 using *assms* by (*simp add: Int_commute closed_Int*)
 moreover have $T \cap (\text{closure } S \cap \text{closure } (- S)) = \text{frontier } S \cap T$
 by (*simp add: Int_commute frontier_closures*)
 ultimately show *?thesis*
 by (*simp add: Int_Un_distrib Int_assoc Int_left_commute assms frontier_closures*)

qed

lemma *frontier_subset_closed*: $\text{closed } S \implies \text{frontier } S \subseteq S$
by (*metis frontier_def closure_closed Diff_subset*)

lemma *frontier_empty* [*simp*]: $\text{frontier } \{\} = \{\}$
by (*simp add: frontier_def*)

lemma *frontier_subset_eq*: $\text{frontier } S \subseteq S \longleftrightarrow \text{closed } S$
proof –
 {
 assume $\text{frontier } S \subseteq S$
 then have $\text{closure } S \subseteq S$
 using *interior_subset unfolding frontier_def* **by** *auto*
 then have $\text{closed } S$
 using *closure_subset_eq* **by** *auto*
 }
then show *?thesis* **using** *frontier_subset_closed*[*of S*] **..**
 qed

lemma *frontier_complement* [*simp*]: $\text{frontier } (-S) = \text{frontier } S$
by (*auto simp: frontier_def closure_complement interior_complement*)

lemma *frontier_Un_subset*: $\text{frontier}(S \cup T) \subseteq \text{frontier } S \cup \text{frontier } T$
by (*metis compl_sup frontier_Int_subset frontier_complement*)

lemma *frontier_disjoint_eq*: $\text{frontier } S \cap S = \{\} \longleftrightarrow \text{open } S$
using *frontier_complement frontier_subset_eq*[*of - S*]
unfolding *open_closed* **by** *auto*

lemma *frontier_UNIV* [*simp*]: $\text{frontier } \text{UNIV} = \{\}$
using *frontier_complement frontier_empty* **by** *fastforce*

lemma *frontier_interiors*: $\text{frontier } s = - \text{interior}(s) - \text{interior}(-s)$
by (*simp add: Int_commute frontier_def interior_closure*)

lemma *frontier_interior_subset*: $\text{frontier}(\text{interior } S) \subseteq \text{frontier } S$
by (*simp add: Diff_mono frontier_interiors interior_mono interior_subset*)

lemma *closure_Un_frontier*: $\text{closure } S = S \cup \text{frontier } S$
proof –
have $S \cup \text{interior } S = S$
 using *interior_subset* **by** *auto*
then show *?thesis*
 using *closure_subset* **by** (*auto simp: frontier_def*)
 qed

2.1.8 Filters and the “eventually true” quantifier

Identify Trivial limits, where we can’t approach arbitrarily closely.

lemma *trivial_limit_within*: $\text{trivial\_limit } (at\ a\ \text{within } S) \longleftrightarrow \neg\ a\ \text{islimpt } S$

proof

```

  assume trivial_limit (at a within S)
  then show  $\neg\ a\ \text{islimpt } S$ 
    unfolding trivial_limit_def
    unfolding eventually_at_topological
    unfolding islimpt_def
    apply (clarsimp simp add: set_eq_iff)
    apply (rename_tac T, rule_tac x=T in exI)
    apply (clarsimp, drule_tac x=y in bspec, simp_all)
  done

```

next

```

  assume  $\neg\ a\ \text{islimpt } S$ 
  then show trivial_limit (at a within S)
    unfolding trivial_limit_def eventually_at_topological islimpt_def
    by metis

```

qed

lemma *trivial_limit_at_iff*: $\text{trivial\_limit } (at\ a) \longleftrightarrow \neg\ a\ \text{islimpt } UNIV$

using *trivial_limit_within* [of *a UNIV*] **by** *simp*

lemma *trivial_limit_at*: $\neg\ \text{trivial\_limit } (at\ a)$

for *a* :: '*a*::perfect_space
by (rule *at_neq_bot*)

lemma *not_trivial_limit_within*: $\neg\ \text{trivial\_limit } (at\ x\ \text{within } S) = (x \in \text{closure } (S - \{x\}))$

using *islimpt_in_closure* **by** (metis *trivial_limit_within*)

lemma *not_in_closure_trivial_limitI*:

```

 $x \notin \text{closure } s \implies \text{trivial\_limit } (at\ x\ \text{within } s)$ 
using not_trivial_limit_within[of x s]
by safe (metis Diff_empty Diff_insert0 closure_subset contra_subsetD)

```

lemma *filterlim_at_within_closure_implies_filterlim*: $\text{filterlim } f\ l\ (at\ x\ \text{within } s)$

if $x \in \text{closure } s \implies \text{filterlim } f\ l\ (at\ x\ \text{within } s)$

by (metis *bot.extremum filterlim_filtercomap filterlim_mono not_in_closure_trivial_limitI that*)

lemma *at_within_eq_bot_iff*: $at\ c\ \text{within } A = \text{bot} \longleftrightarrow c \notin \text{closure } (A - \{c\})$

using *not_trivial_limit_within*[of *c A*] **by** *blast*

Some property holds "sufficiently close" to the limit point.

lemma *trivial_limit_eventually*: $\text{trivial\_limit } net \implies \text{eventually } P\ net$

by *simp*

lemma *trivial_limit_eq*: $\text{trivial\_limit } net \longleftrightarrow (\forall P. \text{eventually } P \text{ } net)$
by (*simp add: filter_eq_iff*)

lemma *Lim_topological*:
 $(f \longrightarrow l) \text{ } net \longleftrightarrow$
 $\text{trivial\_limit } net \vee (\forall S. \text{open } S \longrightarrow l \in S \longrightarrow \text{eventually } (\lambda x. f \ x \in S) \text{ } net)$
unfolding *tendsto_def trivial_limit_eq* **by** *auto*

lemma *eventually_within_Un*:
 $\text{eventually } P \text{ } (at \ x \text{ within } (s \cup t)) \longleftrightarrow$
 $\text{eventually } P \text{ } (at \ x \text{ within } s) \wedge \text{eventually } P \text{ } (at \ x \text{ within } t)$
unfolding *eventually_at_filter*
by (*auto elim!: eventually_rev_mp*)

lemma *Lim_within_union*:
 $(f \longrightarrow l) \text{ } (at \ x \text{ within } (s \cup t)) \longleftrightarrow$
 $(f \longrightarrow l) \text{ } (at \ x \text{ within } s) \wedge (f \longrightarrow l) \text{ } (at \ x \text{ within } t)$
unfolding *tendsto_def*
by (*auto simp: eventually_within_Un*)

2.1.9 Limits

The expected monotonicity property.

lemma *Lim_Un*:
assumes $(f \longrightarrow l) \text{ } (at \ x \text{ within } S) \ (f \longrightarrow l) \text{ } (at \ x \text{ within } T)$
shows $(f \longrightarrow l) \text{ } (at \ x \text{ within } (S \cup T))$
using *assms* **unfolding** *at_within_union* **by** (*rule filterlim_sup*)

lemma *Lim_Un_univ*:
 $(f \longrightarrow l) \text{ } (at \ x \text{ within } S) \implies (f \longrightarrow l) \text{ } (at \ x \text{ within } T) \implies$
 $S \cup T = \text{UNIV} \implies (f \longrightarrow l) \text{ } (at \ x)$
by (*metis Lim_Un*)

Interrelations between restricted and unrestricted limits.

lemma *Lim_at_imp_Lim_at_within*: $(f \longrightarrow l) \text{ } (at \ x) \implies (f \longrightarrow l) \text{ } (at \ x \text{ within } S)$
by (*metis order_refl filterlim_mono subset_UNIV at_le*)

lemma *eventually_within_interior*:
assumes $x \in \text{interior } S$
shows $\text{eventually } P \text{ } (at \ x \text{ within } S) \longleftrightarrow \text{eventually } P \text{ } (at \ x)$
(is ?lhs = ?rhs)
proof
from *assms* **obtain** *T* **where** $T: \text{open } T \ x \in T \ T \subseteq S \ ..$
{
assume *?lhs*
then obtain *A* **where** $\text{open } A \text{ and } x \in A \text{ and } \forall y \in A. y \neq x \longrightarrow y \in S \longrightarrow$
 $P \ y$
by (*auto simp: eventually_at_topological*)

```

with T have open (A ∩ T) and x ∈ A ∩ T and ∀ y ∈ A ∩ T. y ≠ x → P y
  by auto
then show ?rhs
  by (auto simp: eventually_at_topological)
next
  assume ?rhs
  then show ?lhs
    by (auto elim: eventually_mono simp: eventually_at_filter)
}
qed

```

lemma *at_within_interior*: $x \in \text{interior } S \implies \text{at } x \text{ within } S = \text{at } x$
unfolding *filter_eq_iff* **by** (intro allI eventually_within_interior)

lemma *Lim_within_LIMSEQ*:
fixes $a :: 'a :: \text{first\_countable\_topology}$
assumes $\forall S. (\forall n. S\ n \neq a \wedge S\ n \in T) \wedge S \longrightarrow a \longrightarrow (\lambda n. X\ (S\ n)) \longrightarrow L$
shows $(X \longrightarrow L) \text{ (at } a \text{ within } T)$
using *assms* **unfolding** *tendsto_def* [**where** $l=L$]
by (*simp add: sequentially_imp_eventually_within*)

lemma *Lim_right_bound*:
fixes $f :: 'a :: \{\text{linorder\_topology, conditionally\_complete\_linorder, no\_top}\} \Rightarrow$
 $'b :: \{\text{linorder\_topology, conditionally\_complete\_linorder}\}$
assumes *mono*: $\bigwedge a\ b. a \in I \implies b \in I \implies x < a \implies a \leq b \implies f\ a \leq f\ b$
and *bnd*: $\bigwedge a. a \in I \implies x < a \implies K \leq f\ a$
shows $(f \longrightarrow \text{Inf } (f\ ' (\{x<..\} \cap I))) \text{ (at } x \text{ within } (\{x<..\} \cap I))$
proof (*cases* $\{x<..\} \cap I = \{\}$)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
show *?thesis*
proof (*rule order_tendstoI*)
fix a
assume $a: a < \text{Inf } (f\ ' (\{x<..\} \cap I))$
 {
fix y
assume $y \in \{x<..\} \cap I$
with *False bnd* **have** $\text{Inf } (f\ ' (\{x<..\} \cap I)) \leq f\ y$
by (*auto intro!: cInf_lower bdd_belowI2*)
with a **have** $a < f\ y$
by (*blast intro: less_le_trans*)
 }
then show *eventually* $(\lambda x. a < f\ x) \text{ (at } x \text{ within } (\{x<..\} \cap I))$
by (*auto simp: eventually_at_filter intro: exI[of _ 1] zero_less_one*)
next
fix a

```

    assume  $\text{Inf } (f \text{ ` } (\{x < ..\} \cap I)) < a$ 
    from  $c\text{Inf\_lessD}[OF \text{ _this}] \text{ False}$  obtain  $y$  where  $y: x < y \ y \in I \ f \ y < a$ 
    by auto
    then have  $\text{eventually } (\lambda x. x \in I \longrightarrow f \ x < a) \ (\text{at\_right } x)$ 
    unfolding  $\text{eventually\_at\_right}[OF \text{ ` } x < y]$  by  $(\text{metis less\_imp\_le le\_less\_trans mono})$ 
    then show  $\text{eventually } (\lambda x. f \ x < a) \ (\text{at } x \text{ within } (\{x < ..\} \cap I))$ 
    unfolding  $\text{eventually\_at\_filter}$  by  $\text{eventually\_elim simp}$ 
  qed
qed

```

lemma *islimpt_sequential:*

```

  fixes  $x :: 'a::\text{first\_countable\_topology}$ 
  shows  $x \text{ islimpt } S \longleftrightarrow (\exists f. (\forall n::\text{nat}. f \ n \in S - \{x\}) \wedge (f \longrightarrow x) \text{ sequentially})$ 
    (is  $?lhs = ?rhs$ )
proof
  assume  $?lhs$ 
  from  $\text{countable\_basis\_at\_decseq}[of \ x]$  obtain  $A$  where  $A$ :
     $\bigwedge i. \text{open } (A \ i)$ 
     $\bigwedge i. x \in A \ i$ 
     $\bigwedge S. \text{open } S \implies x \in S \implies \text{eventually } (\lambda i. A \ i \subseteq S) \text{ sequentially}$ 
  by blast
  define  $f$  where  $f \ n = (\text{SOME } y. y \in S \wedge y \in A \ n \wedge x \neq y)$  for  $n$ 
  {
    fix  $n$ 
    from  $\text{` } ?lhs$  have  $\exists y. y \in S \wedge y \in A \ n \wedge x \neq y$ 
    unfolding  $\text{islimpt\_def}$  using  $A(1,2)[of \ n]$  by auto
    then have  $f \ n \in S \wedge f \ n \in A \ n \wedge x \neq f \ n$ 
    unfolding  $f\_def$  by  $(\text{rule someI\_ex})$ 
    then have  $f \ n \in S \wedge f \ n \in A \ n \wedge x \neq f \ n$  by auto
  }
  then have  $\forall n. f \ n \in S - \{x\}$  by auto
  moreover have  $(\lambda n. f \ n) \longrightarrow x$ 
  proof  $(\text{rule topological\_tendstoI})$ 
    fix  $S$ 
    assume  $\text{open } S \ x \in S$ 
    from  $A(3)[OF \text{ _this}] \text{ ` } \bigwedge n. f \ n \in A \ n$ 
    show  $\text{eventually } (\lambda x. f \ x \in S) \text{ sequentially}$ 
    by  $(\text{auto elim!: eventually\_mono})$ 
  qed
  ultimately show  $?rhs$  by fast
next
  assume  $?rhs$ 
  then obtain  $f :: \text{nat} \Rightarrow 'a$  where  $f: \bigwedge n. f \ n \in S - \{x\}$  and  $\text{lim: } f \longrightarrow x$ 
  by auto
  show  $?lhs$ 
    unfolding  $\text{islimpt\_def}$ 
  proof safe

```

```

fix  $T$ 
assume  $\text{open } T \ x \in T$ 
from  $\text{lim}[THEN \text{topological\_tendsto}D, \text{OF this}] f$ 
show  $\exists y \in S. y \in T \wedge y \neq x$ 
unfolding  $\text{eventually\_sequentially}$  by  $\text{auto}$ 
qed
qed

```

These are special for limits out of the same topological space.

```

lemma  $\text{Lim\_within\_id}: (id \longrightarrow a) \text{ (at } a \text{ within } s)$ 
unfolding  $\text{id\_def}$  by  $(\text{rule tendsto\_ident\_at})$ 

```

```

lemma  $\text{Lim\_at\_id}: (id \longrightarrow a) \text{ (at } a)$ 
unfolding  $\text{id\_def}$  by  $(\text{rule tendsto\_ident\_at})$ 

```

It's also sometimes useful to extract the limit point from the filter.

```

abbreviation  $\text{netlimit} :: 'a::t2\_space \text{ filter} \Rightarrow 'a$ 
where  $\text{netlimit } F \equiv \text{Lim } F \ (\lambda x. x)$ 

```

```

lemma  $\text{netlimit\_at} [\text{simp}]$ :
fixes  $a :: 'a::\{\text{perfect\_space}, t2\_space\}$ 
shows  $\text{netlimit} \text{ (at } a) = a$ 
using  $\text{Lim\_ident\_at}$   $[\text{of } a \text{ UNIV}]$  by  $\text{simp}$ 

```

```

lemma  $\text{lim\_within\_interior}$ :
 $x \in \text{interior } S \implies (f \longrightarrow l) \text{ (at } x \text{ within } S) \longleftrightarrow (f \longrightarrow l) \text{ (at } x)$ 
by  $(\text{metis at\_within\_interior})$ 

```

```

lemma  $\text{netlimit\_within\_interior}$ :
fixes  $x :: 'a::\{t2\_space, \text{perfect\_space}\}$ 
assumes  $x \in \text{interior } S$ 
shows  $\text{netlimit} \text{ (at } x \text{ within } S) = x$ 
using  $\text{assms}$  by  $(\text{metis at\_within\_interior netlimit\_at})$ 

```

Useful lemmas on closure and set of possible sequential limits.

```

lemma  $\text{closure\_sequential}$ :
fixes  $l :: 'a::\text{first\_countable\_topology}$ 
shows  $l \in \text{closure } S \longleftrightarrow (\exists x. (\forall n. x \ n \in S) \wedge (x \longrightarrow l) \text{ sequentially})$ 
(is ?lhs = ?rhs)
proof
assume  $?lhs$ 
moreover
{
assume  $l \in S$ 
then have  $?rhs$  using  $\text{tendsto\_const}$   $[\text{of } l \text{ sequentially}]$  by  $\text{auto}$ 
}
moreover
{
assume  $l \text{ islimpt } S$ 

```

```

    then have ?rhs unfolding islimpt_sequential by auto
  }
  ultimately show ?rhs
    unfolding closure_def by auto
next
  assume ?rhs
  then show ?lhs unfolding closure_def islimpt_sequential by auto
qed

lemma closed_sequential_limits:
  fixes S :: 'a::first_countable_topology set
  shows closed S  $\longleftrightarrow$  ( $\forall x l. (\forall n. x n \in S) \wedge (x \longrightarrow l)$  sequentially  $\longrightarrow l \in S$ )
  by (metis closure_sequential closure_subset_eq subset_iff)

lemma tendsto_If_within_closures:
  assumes f:  $x \in s \cup (\text{closure } s \cap \text{closure } t) \implies$ 
    ( $f \longrightarrow l x$ ) (at x within s  $\cup$  ( $\text{closure } s \cap \text{closure } t$ ))
  assumes g:  $x \in t \cup (\text{closure } s \cap \text{closure } t) \implies$ 
    ( $g \longrightarrow l x$ ) (at x within t  $\cup$  ( $\text{closure } s \cap \text{closure } t$ ))
  assumes  $x \in s \cup t$ 
  shows (( $\lambda x. \text{if } x \in s \text{ then } f x \text{ else } g x$ )  $\longrightarrow l x$ ) (at x within s  $\cup$  t)
proof -
  have *:  $(s \cup t) \cap \{x. x \in s\} = s$   $(s \cup t) \cap \{x. x \notin s\} = t - s$ 
  by auto
  have ( $f \longrightarrow l x$ ) (at x within s)
  by (rule filterlim_at_within_closure_implies_filterlim)
    (use  $\langle x \in \_ \rangle$  in  $\langle \text{auto simp: inf_commute closure_def intro: tendsto_within_subset[OF f] \rangle$ )
  moreover
  have ( $g \longrightarrow l x$ ) (at x within t - s)
  by (rule filterlim_at_within_closure_implies_filterlim)
    (use  $\langle x \in \_ \rangle$  in  $\langle \text{auto intro!: tendsto_within_subset[OF g] simp: closure_def intro: islimpt_subset} \rangle$ )
  ultimately show ?thesis
  by (intro filterlim_at_within_If) (simp_all only: *)
qed

```

2.1.10 Compactness

```

lemma brouwer_compactness_lemma:
  fixes f :: 'a::topological_space  $\Rightarrow$  'b::real_normed_vector
  assumes compact s
    and continuous_on s f
    and  $\neg (\exists x \in s. f x = 0)$ 
  obtains d where  $0 < d$  and  $\forall x \in s. d \leq \text{norm } (f x)$ 
proof (cases  $s = \{\}$ )
  case True
  show thesis

```

```

    by (rule that [of 1]) (auto simp: True)
next
case False
have continuous_on s (norm ∘ f)
  by (rule continuous_intros continuous_on_norm assms(2))+
with False obtain x where x: x ∈ s ∀ y ∈ s. (norm ∘ f) x ≤ (norm ∘ f) y
  using continuous_attains_inf[OF assms(1), of norm ∘ f]
  unfolding o_def
  by auto
have (norm ∘ f) x > 0
  using assms(3) and x(1)
  by auto
then show ?thesis
  by (rule that) (insert x(2), auto simp: o_def)
qed

```

Bolzano-Weierstrass property

proposition *Heine_Borel_imp_Bolzano_Weierstrass:*

```

  assumes compact s
    and infinite t
    and t ⊆ s
  shows ∃ x ∈ s. x islimpt t
proof (rule ccontr)
  assume ¬ (∃ x ∈ s. x islimpt t)
  then obtain f where f: ∀ x ∈ s. x ∈ f x ∧ open (f x) ∧ (∀ y ∈ t. y ∈ f x ⟶ y =
x)
    unfolding islimpt_def
    using bchoice[of s λ x T. x ∈ T ∧ open T ∧ (∀ y ∈ t. y ∈ T ⟶ y = x)]
    by auto
  obtain g where g: g ⊆ {t. ∃ x. x ∈ s ∧ t = f x} finite g s ⊆ ⋃ g
    using assms(1)[unfolded compact_eq_Heine_Borel, THEN spec[where x={t.
∃ x. x ∈ s ∧ t = f x}]]
    using f by auto
  from g(1,3) have g': ∀ x ∈ g. ∃ xa ∈ s. x = f xa
    by auto
  {
    fix x y
    assume x ∈ t y ∈ t f x = f y
    then have x ∈ f x y ∈ f x ⟶ y = x
      using f[THEN bspec[where x=x]] and ⟨t ⊆ s⟩ by auto
    then have x = y
      using ⟨f x = f y⟩ and f[THEN bspec[where x=y]] and ⟨y ∈ t⟩ and ⟨t ⊆ s⟩
      by auto
  }
  then have inj_on f t
    unfolding inj_on_def by simp
  then have infinite (f ` t)
    using assms(2) using finite_imageD by auto

```



```

moreover
{
  fix x
  assume  $x \in t$   $f x \notin g$ 
  from  $g(\mathcal{B})$   $assms(\mathcal{B}) \langle x \in t \rangle$  obtain  $h$  where  $h \in g$  and  $x \in h$ 
  by auto
  then obtain  $y$  where  $y \in s$   $h = f y$ 
  using  $g'[THEN bspec[where x=h]]$  by auto
  then have  $y = x$ 
  using  $f[THEN bspec[where x=y]]$  and  $\langle x \in t \rangle$  and  $\langle x \in h \rangle [unfolded \langle h = f y \rangle]$ 
  by auto
  then have False
  using  $\langle f x \notin g \rangle \langle h \in g \rangle$  unfolding  $\langle h = f y \rangle$ 
  by auto
}
then have  $f' t \subseteq g$  by auto
ultimately show False
using  $g(2)$  using finite_subset by auto
qed

```

```

lemma sequence_infinite_lemma:
  fixes  $f :: nat \Rightarrow 'a::t1\_space$ 
  assumes  $\forall n. f n \neq l$ 
  and  $(f \longrightarrow l)$  sequentially
  shows infinite (range  $f$ )
proof
  assume finite (range  $f$ )
  then have  $l \notin \text{range } f \wedge \text{closed } (\text{range } f)$ 
  using  $\langle \text{finite } (\text{range } f) \rangle$   $assms(1)$  finite_imp_closed by blast
  then have eventually  $(\lambda n. f n \in - \text{range } f)$  sequentially
  by (metis Compl_iff  $assms(2)$  open_Compl topological_tendstoD)
  then show False
  unfolding eventually_sequentially by auto
qed

```

```

lemma Bolzano_Weierstrass_imp_closed:
  fixes  $s :: 'a::\{first\_countable\_topology, t2\_space\}$  set
  assumes  $\forall t. \text{infinite } t \wedge t \subseteq s \longrightarrow (\exists x \in s. x \text{ islimpt } t)$ 
  shows closed  $s$ 
proof -
  {
    fix  $x l$ 
    assume  $as: \forall n::nat. x n \in s \ (x \longrightarrow l)$  sequentially
    then have  $l \in s$ 
    proof (cases  $\forall n. x n \neq l$ )
    case False
    then show  $l \in s$  using  $as(1)$  by auto
    next
    case True note  $cas = this$ 

```

```

with as(2) have infinite (range x)
  using sequence_infinite_lemma[of x l] by auto
then obtain l' where l' ∈ s l' islimpt (range x)
  using assms[THEN spec[where x=range x]] as(1) by auto
then show l ∈ s using sequence_unique_limpt[of x l l']
  using as cas by auto
qed
}
then show ?thesis
  unfolding closed_sequential_limits by fast
qed

```

```

lemma closure_insert:
  fixes x :: 'a::t1_space
  shows closure (insert x s) = insert x (closure s)
  by (meson closed_closure closed_insert closure_minimal closure_subset dual_order.antisym
insert_mono insert_subset)

```

```

lemma finite_not_islimpt_in_compact:
  assumes compact A ∧ z. z ∈ A ⇒ ¬z islimpt B
  shows finite (A ∩ B)
proof (rule ccontr)
  assume infinite (A ∩ B)
  have ∃ z ∈ A. z islimpt A ∩ B
    by (rule Heine_Borel_imp_Bolzano_Weierstrass) (use assms ⟨infinite _⟩ in
auto)
  hence ∃ z ∈ A. z islimpt B
    using islimpt_subset by blast
  thus False using assms(2)
    by simp
qed

```

In particular, some common special cases.

```

lemma compact_Un [intro]:
  assumes compact s
  and compact t
  shows compact (s ∪ t)
proof (rule compactI)
  fix f
  assume *: Ball f open s ∪ t ⊆ ⋃ f
  from * ⟨compact s⟩ obtain s' where s' ⊆ f ∧ finite s' ∧ s ⊆ ⋃ s'
    unfolding compact_eq_Heine_Borel by (auto elim!: allE[of _ f])
  moreover
  from * ⟨compact t⟩ obtain t' where t' ⊆ f ∧ finite t' ∧ t ⊆ ⋃ t'
    unfolding compact_eq_Heine_Borel by (auto elim!: allE[of _ f])
  ultimately show ∃ f' ⊆ f. finite f' ∧ s ∪ t ⊆ ⋃ f'
    by (auto intro!: exI[of _ s' ∪ t'])
qed

```

lemma *compact_Union* [intro]: *finite S* $\implies (\bigwedge T. T \in S \implies \text{compact } T) \implies \text{compact } (\bigcup S)$

by (*induct set: finite*) *auto*

lemma *compact_UN* [intro]:

finite A $\implies (\bigwedge x. x \in A \implies \text{compact } (B\ x)) \implies \text{compact } (\bigcup_{x \in A} B\ x)$

by (*rule compact_Union*) *auto*

lemma *closed_Int_compact* [intro]:

assumes *closed s*

and *compact t*

shows *compact (s $\cap$ t)*

using *compact_Int_closed* [of *t s*] *assms*

by (*simp add: Int_commute*)

lemma *compact_Int* [intro]:

fixes *s t :: 'a :: t2_space set*

assumes *compact s*

and *compact t*

shows *compact (s $\cap$ t)*

using *assms* **by** (*intro compact_Int_closed compact_imp_closed*)

lemma *compact_sing* [simp]: *compact {a}*

unfolding *compact_eq_Heine_Borel* **by** *auto*

lemma *compact_insert* [simp]:

assumes *compact s*

shows *compact (insert x s)*

proof –

have *compact ({x} $\cup$ s)*

using *compact_sing* *assms* **by** (*rule compact_Un*)

then show *?thesis* **by** *simp*

qed

lemma *finite_imp_compact*: *finite s* $\implies \text{compact } s$

by (*induct set: finite*) *simp_all*

lemma *open_delete*:

fixes *s :: 'a::t1_space set*

shows *open s* $\implies \text{open } (s - \{x\})$

by (*simp add: open_Diff*)

Compactness expressed with filters

lemma *closure_iff_nhds_not_empty*:

$x \in \text{closure } X \iff (\forall A. \forall S \subseteq A. \text{open } S \longrightarrow x \in S \longrightarrow X \cap A \neq \{\})$

proof *safe*

assume *x: x $\in$ closure X*

fix *S A*

assume *open S x $\in$ S X $\cap$ A = {} S $\subseteq$ A*

```

then have  $x \notin \text{closure } (-S)$ 
  by (auto simp: closure_complement subset_eq[symmetric] intro: interiorI)
with  $x$  have  $x \in \text{closure } X - \text{closure } (-S)$ 
  by auto
also have  $\dots \subseteq \text{closure } (X \cap S)$ 
  using  $\langle \text{open } S \rangle \text{ open\_Int\_closure\_subset[of } S \text{ } X \rangle$  by (simp add: closed_Compl
ac_simps)
finally have  $X \cap S \neq \{\}$  by auto
then show False using  $\langle X \cap A = \{\} \rangle \langle S \subseteq A \rangle$  by auto
next
assume  $\forall A \ S. S \subseteq A \longrightarrow \text{open } S \longrightarrow x \in S \longrightarrow X \cap A \neq \{\}$ 
from this[THEN spec, of  $-X$ , THEN spec, of  $-\text{closure } X$ ]
show  $x \in \text{closure } X$ 
  by (simp add: closure_subset open_Compl)
qed

```

lemma *compact_filter*:

$\text{compact } U \longleftrightarrow (\forall F. F \neq \text{bot} \longrightarrow \text{eventually } (\lambda x. x \in U) F \longrightarrow (\exists x \in U. \inf$
 $(\text{nhds } x) F \neq \text{bot}))$

proof (intro allI iffI impI compact_fip[THEN iffD2] notI)

fix F

assume *compact* U

assume $F: F \neq \text{bot}$ eventually $(\lambda x. x \in U) F$

then have $U \neq \{\}$

by (auto simp: eventually_False)

define Z **where** $Z = \text{closure } \{A. \text{eventually } (\lambda x. x \in A) F\}$

then have $\forall z \in Z. \text{closed } z$

by auto

moreover

have $\text{ev\_}Z: \bigwedge z. z \in Z \implies \text{eventually } (\lambda x. x \in z) F$

unfolding Z_def by (auto elim: eventually_mono intro: subsetD[OF closure_subset])

have $(\forall B \subseteq Z. \text{finite } B \longrightarrow U \cap \bigcap B \neq \{\})$

proof (intro allI impI)

fix B assume *finite* $B \ B \subseteq Z$

with $\langle \text{finite } B \rangle \text{ ev\_}Z \ F(2)$ have eventually $(\lambda x. x \in U \cap (\bigcap B)) F$

by (auto simp: eventually_ball_finite_distrib eventually_conj_iff)

with F show $U \cap \bigcap B \neq \{\}$

by (intro notI) (simp add: eventually_False)

qed

ultimately have $U \cap \bigcap Z \neq \{\}$

using $\langle \text{compact } U \rangle$ unfolding compact_fip by blast

then obtain x **where** $x \in U$ **and** $x: \bigwedge z. z \in Z \implies x \in z$

by auto

have $\bigwedge P. \text{eventually } P (\inf (\text{nhds } x) F) \implies P \neq \text{bot}$

unfolding eventually_inf eventually_nhds

proof safe

```

    fix P Q R S
    assume eventually R F open S x ∈ S
    with open_Int_closure_eq_empty[of S {x. R x}] x[of closure {x. R x}]
    have S ∩ {x. R x} ≠ {} by (auto simp: Z_def)
    moreover assume Ball S Q ∀ x. Q x ∧ R x ⟶ bot x
    ultimately show False by (auto simp: set_eq_iff)
  qed
  with ⟨x ∈ U⟩ show ∃ x ∈ U. inf (nhds x) F ≠ bot
    by (metis eventually_bot)
next
fix A
assume A: ∀ a ∈ A. closed a ∀ B ⊆ A. finite B ⟶ U ∩ ⋂ B ≠ {} U ∩ ⋂ A = {}
define F where F = (INF a ∈ insert U A. principal a)
have F ≠ bot
  unfolding F_def
proof (rule INF_filter_not_bot)
  fix X
  assume X: X ⊆ insert U A finite X
  with A(2)[THEN spec, of X - {U}] have U ∩ ⋂ (X - {U}) ≠ {}
    by auto
  with X show (INF a ∈ X. principal a) ≠ bot
    by (auto simp: INF_principal_finite principal_eq_bot_iff)
qed
moreover
have F ≤ principal U
  unfolding F_def by auto
then have eventually (λx. x ∈ U) F
  by (auto simp: le_filter_def eventually_principal)
moreover
assume ∀ F. F ≠ bot ⟶ eventually (λx. x ∈ U) F ⟶ (∃ x ∈ U. inf (nhds x)
F ≠ bot)
ultimately obtain x where x ∈ U and x: inf (nhds x) F ≠ bot
  by auto

{ fix V assume V ∈ A
  then have F ≤ principal V
    unfolding F_def by (intro INF_lower2[of V]) auto
  then have V: eventually (λx. x ∈ V) F
    by (auto simp: le_filter_def eventually_principal)
  have x ∈ closure V
    unfolding closure_iff_nhds_not_empty
  proof (intro impI allI)
    fix S A
    assume open S x ∈ S S ⊆ A
    then have eventually (λx. x ∈ A) (nhds x)
      by (auto simp: eventually_nhds)
    with V have eventually (λx. x ∈ V ∩ A) (inf (nhds x) F)
      by (auto simp: eventually_inf)
    with x show V ∩ A ≠ {}
  }

```

```

    by (auto simp del: Int_iff simp add: trivial_limit_def)
  qed
  then have  $x \in V$ 
    using  $\langle V \in A \rangle A(1)$  by simp
}
with  $\langle x \in U \rangle$  have  $x \in U \cap \bigcap A$  by auto
with  $\langle U \cap \bigcap A = \{\} \rangle$  show False by auto
qed

```

definition *countably_compact* :: $('a::\text{topological\_space}) \text{ set} \Rightarrow \text{bool}$ **where**
countably_compact $U \iff$
 $(\forall A. \text{countable } A \longrightarrow (\forall a \in A. \text{open } a) \longrightarrow U \subseteq \bigcup A$
 $\longrightarrow (\exists T \subseteq A. \text{finite } T \wedge U \subseteq \bigcup T))$

lemma *countably_compactE*:
assumes *countably_compact* s **and** $\forall t \in C. \text{open } t$ **and** $s \subseteq \bigcup C$ *countable* C
obtains C' **where** $C' \subseteq C$ **and** *finite* C' **and** $s \subseteq \bigcup C'$
using *assms* **unfolding** *countably_compact_def* **by** *metis*

lemma *countably_compactI*:
assumes $\bigwedge C. \forall t \in C. \text{open } t \implies s \subseteq \bigcup C \implies \text{countable } C \implies (\exists C' \subseteq C. \text{finite } C' \wedge s \subseteq \bigcup C')$
shows *countably_compact* s
using *assms* **unfolding** *countably_compact_def* **by** *metis*

lemma *compact_imp_countably_compact*: *compact* $U \implies \text{countably\_compact } U$
by (auto simp: *compact_eq_Heine_Borel* *countably_compact_def*)

lemma *countably_compact_imp_compact*:
assumes *countably_compact* U
and *ccover*: *countable* $B \forall b \in B. \text{open } b$
and *basis*: $\bigwedge T x. \text{open } T \implies x \in T \implies x \in U \implies \exists b \in B. x \in b \wedge b \cap U \subseteq T$
shows *compact* U
using $\langle \text{countably\_compact } U \rangle$
unfolding *compact_eq_Heine_Borel* *countably_compact_def*

proof *safe*
fix A
assume $A: \forall a \in A. \text{open } a \implies U \subseteq \bigcup A$
assume *: $\forall A. \text{countable } A \longrightarrow (\forall a \in A. \text{open } a) \longrightarrow U \subseteq \bigcup A \longrightarrow (\exists T \subseteq A. \text{finite } T \wedge U \subseteq \bigcup T)$
moreover **define** C **where** $C = \{b \in B. \exists a \in A. b \cap U \subseteq a\}$
ultimately **have** *countable* $C \forall a \in C. \text{open } a$
unfolding C_def **using** *ccover* **by** *auto*
moreover
have $\bigcup A \cap U \subseteq \bigcup C$
proof *safe*
fix $x \in a$
assume $x \in U \implies x \in a \implies a \in A$

```

with basis[of a x] A obtain b where b ∈ B x ∈ b b ∩ U ⊆ a
  by blast
with ⟨a ∈ A⟩ show x ∈ ⋃ C
  unfolding C_def by auto
qed
then have U ⊆ ⋃ C using ⟨U ⊆ ⋃ A⟩ by auto
ultimately obtain T where T: T ⊆ C finite T U ⊆ ⋃ T
  using * by metis
then have ∀ t ∈ T. ∃ a ∈ A. t ∩ U ⊆ a
  by (auto simp: C_def)
then obtain f where ∀ t ∈ T. f t ∈ A ∧ t ∩ U ⊆ f t
  unfolding bchoice_iff Bex_def ..
with T show ∃ T ⊆ A. finite T ∧ U ⊆ ⋃ T
  unfolding C_def by (intro exI[of _ f`T]) fastforce
qed

```

```

proposition countably_compact_imp_compact_second_countable:
  countably_compact U ⟹ compact (U :: 'a :: second_countable_topology set)
proof (rule countably_compact_imp_compact)
  fix T and x :: 'a
  assume open T x ∈ T
  from topological_basisE[OF is_basis this] obtain b where
    b ∈ (SOME B. countable B ∧ topological_basis B) x ∈ b b ⊆ T .
  then show ∃ b ∈ SOME B. countable B ∧ topological_basis B. x ∈ b ∧ b ∩ U ⊆
    T
    by blast
qed (insert countable_basis topological_basis_open[OF is_basis], auto)

```

```

lemma countably_compact_eq_compact:
  countably_compact U ⟷ compact (U :: 'a :: second_countable_topology set)
  using countably_compact_imp_compact_second_countable compact_imp_countably_compact
  by blast

```

Sequential compactness

```

definition seq_compact :: 'a::topological_space set ⇒ bool where
  seq_compact S ⟷
    (∀ f. (∀ n. f n ∈ S)
      ⟶ (∃ l ∈ S. ∃ r::nat⇒nat. strict_mono r ∧ ((f ∘ r) ⟶ l) sequentially))

```

```

lemma seq_compactI:
  assumes ⋀ f. ∀ n. f n ∈ S ⟹ ∃ l ∈ S. ∃ r::nat⇒nat. strict_mono r ∧ ((f ∘ r)
    ⟶ l) sequentially
  shows seq_compact S
  unfolding seq_compact_def using assms by fast

```

```

lemma seq_compactE:
  assumes seq_compact S ∀ n. f n ∈ S
  obtains l r where l ∈ S strict_mono (r :: nat ⇒ nat) ((f ∘ r) ⟶ l) sequen-

```

tially

using *assms* **unfolding** *seq_compact_def* **by** *fast*

lemma *closed_sequentially*:

assumes *closed s* **and** $\forall n. f\ n \in s$ **and** $f \longrightarrow l$

shows $l \in s$

proof (*rule ccontr*)

assume $l \notin s$

with $\langle \text{closed } s \rangle$ **and** $\langle f \longrightarrow l \rangle$ **have** *eventually* $(\lambda n. f\ n \in -\ s)$ *sequentially*

by (*fast intro: topological_tendstoD*)

with $\langle \forall n. f\ n \in s \rangle$ **show** *False*

by *simp*

qed

lemma *seq_compact_Int_closed*:

assumes *seq_compact s* **and** *closed t*

shows *seq_compact* $(s \cap t)$

proof (*rule seq_compactI*)

fix *f* **assume** $\forall n::\text{nat}. f\ n \in s \cap t$

hence $\forall n. f\ n \in s$ **and** $\forall n. f\ n \in t$

by *simp_all*

from $\langle \text{seq\_compact } s \rangle$ **and** $\langle \forall n. f\ n \in s \rangle$

obtain *l r* **where** $l \in s$ **and** *r*: *strict_mono r* **and** *l*: $(f \circ r) \longrightarrow l$

by (*rule seq_compactE*)

from $\langle \forall n. f\ n \in t \rangle$ **have** $\forall n. (f \circ r)\ n \in t$

by *simp*

from $\langle \text{closed } t \rangle$ **and** *this* **and** *l* **have** $l \in t$

by (*rule closed_sequentially*)

with $\langle l \in s \rangle$ **and** *r* **and** *l* **show** $\exists l \in s \cap t. \exists r. \text{strict\_mono } r \wedge (f \circ r) \longrightarrow l$

by *fast*

qed

lemma *seq_compact_closed_subset*:

assumes *closed s* **and** $s \subseteq t$ **and** *seq_compact t*

shows *seq_compact s*

using *assms* *seq_compact_Int_closed* [*of t s*] **by** (*simp add: Int_absorb1*)

lemma *seq_compact_imp_countably_compact*:

fixes *U* :: *'a* :: *first_countable_topology set*

assumes *seq_compact U*

shows *countably_compact U*

proof (*safe intro!: countably_compactI*)

fix *A*

assume *A*: $\forall a \in A. \text{open } a\ U \subseteq \bigcup A$ *countable A*

have *subseq*: $\bigwedge X. \text{range } X \subseteq U \implies \exists r\ x. x \in U \wedge \text{strict\_mono } (r :: \text{nat} \Rightarrow \text{nat}) \wedge (X \circ r) \longrightarrow x$

using $\langle \text{seq\_compact } U \rangle$ **by** (*fastforce simp: seq_compact_def subset_eq*)

show $\exists T \subseteq A. \text{finite } T \wedge U \subseteq \bigcup T$

proof *cases*


```

    assume finite A
    with A show ?thesis by auto
next
    assume infinite A
    then have  $A \neq \{\}$  by auto
    show ?thesis
    proof (rule ccontr)
      assume  $\neg (\exists T \subseteq A. \text{finite } T \wedge U \subseteq \bigcup T)$ 
      then have  $\forall T. \exists x. T \subseteq A \wedge \text{finite } T \longrightarrow (x \in U - \bigcup T)$ 
        by auto
      then obtain  $X'$  where  $T: \bigwedge T. T \subseteq A \implies \text{finite } T \implies X' T \in U - \bigcup T$ 
        by metis
      define X where  $X n = X' (\text{from\_nat\_into } A \text{ ' } \{.. n\})$  for n
      have  $X: \bigwedge n. X n \in U - (\bigcup i \leq n. \text{from\_nat\_into } A i)$ 
        using  $\langle A \neq \{\} \rangle$  unfolding X_def by (intro T) (auto intro: from_nat_into)
      then have  $\text{range } X \subseteq U$ 
        by auto
      with subseq[of X] obtain r x where  $x \in U$  and  $r: \text{strict\_mono } r \ (X \circ r)$ 
         $\longrightarrow x$ 
        by auto
      from  $\langle x \in U \rangle \langle U \subseteq \bigcup A \rangle$  from_nat_into_surj[OF  $\langle \text{countable } A \rangle$ ]
      obtain n where  $x \in \text{from\_nat\_into } A n$  by auto
      with  $r(2) \ A(1)$  from_nat_into[OF  $\langle A \neq \{\} \rangle$ , of n]
      have eventually  $(\lambda i. X (r i) \in \text{from\_nat\_into } A n)$  sequentially
        unfolding tendsto_def by (auto simp: comp_def)
      then obtain N where  $\bigwedge i. N \leq i \implies X (r i) \in \text{from\_nat\_into } A n$ 
        by (auto simp: eventually_sequentially)
      moreover from X have  $\bigwedge i. n \leq r i \implies X (r i) \notin \text{from\_nat\_into } A n$ 
        by auto
      moreover from  $\langle \text{strict\_mono } r \rangle$  [THEN seq_suble, of max n N] have  $\exists i. n$ 
 $\leq r i \wedge N \leq i$ 
        by (auto intro!: exI[of _ max n N])
      ultimately show False
        by auto
    qed
  qed
qed

```

```

lemma compact_imp_seq_compact:
  fixes U :: 'a :: first_countable_topology set
  assumes compact U
  shows seq_compact U
  unfolding seq_compact_def

```

```

proof safe
  fix X :: nat  $\Rightarrow$  'a
  assume  $\forall n. X n \in U$ 
  then have eventually  $(\lambda x. x \in U)$  (filtermap X sequentially)
    by (auto simp: eventually_filtermap)
  moreover

```

```

have filtermap X sequentially ≠ bot
  by (simp add: trivial_limit_def eventually_filtermap)
ultimately
obtain x where x ∈ U and x: inf (nhds x) (filtermap X sequentially) ≠ bot (is
?F ≠ _)
  using ⟨compact U⟩ by (auto simp: compact_filter)

from countable_basis_at_decseq[of x]
obtain A where A:
  ∧i. open (A i)
  ∧i. x ∈ A i
  ∧S. open S ⇒ x ∈ S ⇒ eventually (λi. A i ⊆ S) sequentially
  by blast
define s where s n i = (SOME j. i < j ∧ X j ∈ A (Suc n)) for n i
{
  fix n i
  have ∃ a. i < a ∧ X a ∈ A (Suc n)
  proof (rule ccontr)
    assume ¬ (∃ a > i. X a ∈ A (Suc n))
    then have ∧a. Suc i ≤ a ⇒ X a ∉ A (Suc n)
      by auto
    then have eventually (λx. x ∉ A (Suc n)) (filtermap X sequentially)
      by (auto simp: eventually_filtermap eventually_sequentially)
    moreover have eventually (λx. x ∈ A (Suc n)) (nhds x)
      using A(1,2)[of Suc n] by (auto simp: eventually_nhds)
    ultimately have eventually (λx. False) ?F
      by (auto simp: eventually_inf)
    with x show False
      by (simp add: eventually_False)
  qed
  then have i < s n i ∧ X (s n i) ∈ A (Suc n)
    unfolding s_def by (auto intro: someI2_ex)
}
note s = this
define r where r = rec_nat (s 0 0) s
have strict_mono r
  by (auto simp: r_def s strict_mono_Suc_iff)
moreover
have (λn. X (r n)) ⟶ x
proof (rule topological_tendstoI)
  fix S
  assume open S x ∈ S
  with A(3) have eventually (λi. A i ⊆ S) sequentially
    by auto
  moreover
  {
    fix i
    assume Suc 0 ≤ i
    then have X (r i) ∈ A i

```

```

      by (cases i) (simp_all add: r_def s)
    }
  then have eventually ( $\lambda i. X (r i) \in A$  i) sequentially
    by (auto simp: eventually_sequentially)
  ultimately show eventually ( $\lambda i. X (r i) \in S$ ) sequentially
    by eventually_elim auto
qed
ultimately show  $\exists x \in U. \exists r. \text{strict\_mono } r \wedge (X \circ r) \longrightarrow x$ 
  using  $\langle x \in U \rangle$  by (auto simp: convergent_def comp_def)
qed

lemma countably_compact_imp_acc_point:
  assumes countably_compact s
    and countable t
    and infinite t
    and  $t \subseteq s$ 
  shows  $\exists x \in s. \forall U. x \in U \wedge \text{open } U \longrightarrow \text{infinite } (U \cap t)$ 
proof (rule ccontr)
  define C where  $C = (\lambda F. \text{interior } (F \cup (- t))) \text{ ` } \{F. \text{finite } F \wedge F \subseteq t\}$ 
  note  $\langle \text{countably\_compact } s \rangle$ 
  moreover have  $\forall t \in C. \text{open } t$ 
    by (auto simp: C_def)
  moreover
  assume  $\neg (\exists x \in s. \forall U. x \in U \wedge \text{open } U \longrightarrow \text{infinite } (U \cap t))$ 
  then have  $s: \bigwedge x. x \in s \implies \exists U. x \in U \wedge \text{open } U \wedge \text{finite } (U \cap t)$  by metis
  have  $s \subseteq \bigcup C$ 
    using  $\langle t \subseteq s \rangle$ 
    unfolding C_def
    apply (safe dest!: s)
    apply (rule_tac a =  $U \cap t$  in UN_I)
    apply (auto intro!: interiorI simp add: finite_subset)
    done
  moreover
  from  $\langle \text{countable } t \rangle$  have countable C
    unfolding C_def by (auto intro: countable_Collect_finite_subset)
  ultimately
  obtain D where  $D \subseteq C$  finite D  $s \subseteq \bigcup D$ 
    by (rule countably_compactE)
  then obtain E where  $E: E \subseteq \{F. \text{finite } F \wedge F \subseteq t\}$  finite E
    and  $s: s \subseteq (\bigcup F \in E. \text{interior } (F \cup (- t)))$ 
    by (metis (lifting) finite_subset_image C_def)
  from  $s \langle t \subseteq s \rangle$  have  $t \subseteq \bigcup E$ 
    using interior_subset by blast
  moreover have finite ( $\bigcup E$ )
    using E by auto
  ultimately show False using  $\langle \text{infinite } t \rangle$ 
    by (auto simp: finite_subset)
qed

```

```

lemma countable_acc_point_imp_seq_compact:
  fixes s :: 'a::first_countable_topology set
  assumes  $\forall t. \text{infinite } t \wedge \text{countable } t \wedge t \subseteq s \longrightarrow$ 
    ( $\exists x \in s. \forall U. x \in U \wedge \text{open } U \longrightarrow \text{infinite } (U \cap t)$ )
  shows seq_compact s
proof -
  {
    fix f :: nat  $\Rightarrow$  'a
    assume f:  $\forall n. f\ n \in s$ 
    have  $\exists l \in s. \exists r. \text{strict\_mono } r \wedge ((f \circ r) \longrightarrow l) \text{ sequentially}$ 
    proof (cases finite (range f))
      case True
        obtain l where infinite {n. f n = f l}
          using pigeonhole_infinite[OF _ True] by auto
        then obtain r :: nat  $\Rightarrow$  nat where strict_mono r and fr:  $\forall n. f\ (r\ n) = f\ l$ 
          using infinite_enumerate by blast
        then have strict_mono r  $\wedge (f \circ r) \longrightarrow f\ l$ 
          by (simp add: fr o_def)
        with f show  $\exists l \in s. \exists r. \text{strict\_mono } r \wedge (f \circ r) \longrightarrow l$ 
          by auto
      case False
        with f assms have  $\exists x \in s. \forall U. x \in U \wedge \text{open } U \longrightarrow \text{infinite } (U \cap \text{range } f)$ 
          by auto
        then obtain l where  $l \in s \wedge \forall U. l \in U \wedge \text{open } U \longrightarrow \text{infinite } (U \cap \text{range } f) ..$ 
          from this(2) have  $\exists r. \text{strict\_mono } r \wedge ((f \circ r) \longrightarrow l) \text{ sequentially}$ 
            using acc_point_range_imp_convergent_subsequence[of l f] by auto
        with  $\langle l \in s \rangle$  show  $\exists l \in s. \exists r. \text{strict\_mono } r \wedge ((f \circ r) \longrightarrow l) \text{ sequentially}$ 
      ..
    qed
  }
  then show ?thesis
    unfolding seq_compact_def by auto
qed

```

```

lemma seq_compact_eq_countably_compact:
  fixes U :: 'a::first_countable_topology set
  shows seq_compact U  $\longleftrightarrow$  countably_compact U
  using
    countable_acc_point_imp_seq_compact
    countably_compact_imp_acc_point
    seq_compact_imp_countably_compact
  by metis

```

```

lemma seq_compact_eq_acc_point:
  fixes s :: 'a::first_countable_topology set
  shows seq_compact s  $\longleftrightarrow$ 
    ( $\forall t. \text{infinite } t \wedge \text{countable } t \wedge t \subseteq s \longrightarrow (\exists x \in s. \forall U. x \in U \wedge \text{open } U \longrightarrow$ 
      infinite (U  $\cap$  t)))

```

```

using
  countable_acc_point_imp_seq_compact[of s]
  countably_compact_imp_acc_point[of s]
  seq_compact_imp_countably_compact[of s]
by metis

```

```

lemma seq_compact_eq_compact:
  fixes U :: 'a :: second_countable_topology set
  shows seq_compact U  $\longleftrightarrow$  compact U
  using seq_compact_eq_countably_compact countably_compact_eq_compact by
blast

```

```

proposition Bolzano_Weierstrass_imp_seq_compact:
  fixes s :: 'a::{t1_space, first_countable_topology} set
  shows  $\forall t. \text{infinite } t \wedge t \subseteq s \longrightarrow (\exists x \in s. x \text{ islimpt } t) \implies \text{seq\_compact } s$ 
  by (rule countable_acc_point_imp_seq_compact) (metis islimpt_eq_acc_point)

```

2.1.11 Cartesian products

```

lemma seq_compact_Times: seq_compact s  $\implies$  seq_compact t  $\implies$  seq_compact
(s  $\times$  t)
  unfolding seq_compact_def
  apply clarify
  apply (drule_tac x=fst  $\circ$  f in spec)
  apply (drule mp, simp add: mem_Times_iff)
  apply (clarify, rename_tac l1 r1)
  apply (drule_tac x=snd  $\circ$  f  $\circ$  r1 in spec)
  apply (drule mp, simp add: mem_Times_iff)
  apply (clarify, rename_tac l2 r2)
  apply (rule_tac x=(l1, l2) in rev_bexI, simp)
  apply (rule_tac x=r1  $\circ$  r2 in exI)
  apply (rule conjI, simp add: strict_mono_def)
  apply (drule_tac f=r2 in LIMSEQ_subseq_LIMSEQ, assumption)
  apply (drule (1) tendsto_Pair) back
  apply (simp add: o_def)
done

```

```

lemma compact_Times:
  assumes compact s compact t
  shows compact (s  $\times$  t)
proof (rule compactI)
  fix C
  assume C:  $\forall t \in C. \text{open } t \wedge s \times t \subseteq \bigcup C$ 
  have  $\forall x \in s. \exists a. \text{open } a \wedge x \in a \wedge (\exists d \subseteq C. \text{finite } d \wedge a \times t \subseteq \bigcup d)$ 
  proof
    fix x
    assume x  $\in$  s
    have  $\forall y \in t. \exists a \ b \ c. c \in C \wedge \text{open } a \wedge \text{open } b \wedge x \in a \wedge y \in b \wedge a \times b \subseteq c$ 
    (is  $\forall y \in t. ?P \ y$ )

```

```

proof
  fix  $y$ 
  assume  $y \in t$ 
  with  $\langle x \in s \rangle C$  obtain  $c$  where  $c \in C$   $(x, y) \in c$  open  $c$  by auto
  then show  $?P\ y$  by (auto elim!: open_prod_elim)
qed
then obtain  $a\ b\ c$  where  $b: \bigwedge y. y \in t \implies \text{open } (b\ y)$ 
  and  $c: \bigwedge y. y \in t \implies c\ y \in C \wedge \text{open } (a\ y) \wedge \text{open } (b\ y) \wedge x \in a\ y \wedge y \in b$ 
 $y \wedge a\ y \times b\ y \subseteq c\ y$ 
  by metis
  then have  $\forall y \in t. \text{open } (b\ y) \ t \subseteq (\bigcup y \in t. b\ y)$  by auto
  with compactE_image[OF  $\langle \text{compact } t \rangle$ ] obtain  $D$  where  $D: D \subseteq t$  finite  $D\ t$ 
 $\subseteq (\bigcup y \in D. b\ y)$ 
  by metis
  moreover from  $D\ c$  have  $(\bigcap y \in D. a\ y) \times t \subseteq (\bigcup y \in D. c\ y)$ 
  by (fastforce simp: subset_eq)
  ultimately show  $\exists a. \text{open } a \wedge x \in a \wedge (\exists d \subseteq C. \text{finite } d \wedge a \times t \subseteq \bigcup d)$ 
  using  $c$  by (intro exI[of  $\_ c'D$ ] exI[of  $\_ \bigcap(a'D)$ ] conjI) (auto intro!:
open_INT)
qed
then obtain  $a\ d$  where  $a: \bigwedge x. x \in s \implies \text{open } (a\ x) \ s \subseteq (\bigcup x \in s. a\ x)$ 
  and  $d: \bigwedge x. x \in s \implies d\ x \subseteq C \wedge \text{finite } (d\ x) \wedge a\ x \times t \subseteq \bigcup (d\ x)$ 
  unfolding subset_eq UN_iff by metis
moreover
from compactE_image[OF  $\langle \text{compact } s \rangle a$ ]
obtain  $e$  where  $e: e \subseteq s$  finite  $e$  and  $s: s \subseteq (\bigcup x \in e. a\ x)$ 
  by auto
moreover
{
  from  $s$  have  $s \times t \subseteq (\bigcup x \in e. a\ x \times t)$ 
  by auto
  also have  $\dots \subseteq (\bigcup x \in e. \bigcup (d\ x))$ 
  using  $d\ \langle e \subseteq s \rangle$  by (intro UN_mono) auto
  finally have  $s \times t \subseteq (\bigcup x \in e. \bigcup (d\ x))$  .
}
ultimately show  $\exists C' \subseteq C. \text{finite } C' \wedge s \times t \subseteq \bigcup C'$ 
  by (intro exI[of  $\_ (\bigcup x \in e. d\ x)$ ]) (auto simp: subset_eq)
qed

```

lemma *tube_lemma*:

```

assumes compact  $K$ 
assumes open  $W$ 
assumes  $\{x0\} \times K \subseteq W$ 
shows  $\exists X0. x0 \in X0 \wedge \text{open } X0 \wedge X0 \times K \subseteq W$ 

```

proof –

```

{
  fix  $y$  assume  $y \in K$ 
  then have  $(x0, y) \in W$  using assms by auto

```

```

  with ⟨open W⟩
  have  $\exists X0\ Y. \text{open } X0 \wedge \text{open } Y \wedge x0 \in X0 \wedge y \in Y \wedge X0 \times Y \subseteq W$ 
    by (rule open_prod_elim) blast
}
then obtain X0 Y where
  *:  $\forall y \in K. \text{open } (X0\ y) \wedge \text{open } (Y\ y) \wedge x0 \in X0\ y \wedge y \in Y\ y \wedge X0\ y \times Y\ y$ 
 $\subseteq W$ 
  by metis
from * have  $\forall t \in Y\ 'K. \text{open } t\ K \subseteq \bigcup (Y\ 'K)$  by auto
with ⟨compact K⟩ obtain CC where CC:  $CC \subseteq Y\ 'K$  finite CC  $K \subseteq \bigcup CC$ 
  by (meson compactE)
then obtain c where c:  $\bigwedge C. C \in CC \implies c\ C \in K \wedge C = Y\ (c\ C)$ 
  by (force intro!: choice)
with * CC show ?thesis
  by (force intro!: exI [where x =  $\bigcap C \in CC. X0\ (c\ C)$ ])
qed

```

lemma continuous_on_prod_compactE:

fixes fx::'a::topological_space $\times$ 'b::topological_space $\Rightarrow$ 'c::metric_space
and e::real

assumes cont_fx: continuous_on (U $\times$ C) fx

assumes compact C

assumes [intro]: $x0 \in U$

notes [continuous_intros] = continuous_on_compose2[OF cont_fx]

assumes $e > 0$

obtains X0 where $x0 \in X0$ open X0

$\forall x \in X0 \cap U. \forall t \in C. \text{dist } (fx\ (x, t))\ (fx\ (x0, t)) \leq e$

proof –

define psi where $psi = (\lambda(x, t). \text{dist } (fx\ (x, t))\ (fx\ (x0, t)))$

define W0 where $W0 = \{(x, t) \in U \times C. psi\ (x, t) < e\}$

have W0_eq: $W0 = psi - \{..<e\} \cap U \times C$

by (auto simp: vimage_def W0_def)

have open {.. e } by simp

have continuous_on (U $\times$ C) psi

by (auto intro!: continuous_intros simp: psi_def split_beta')

from this[unfolded continuous_on_open_invariant, rule_format, OF ⟨open {.. e }⟩]

obtain W where W: open W $W \cap U \times C = W0 \cap U \times C$

unfolding W0_eq by blast

have $\{x0\} \times C \subseteq W \cap U \times C$

unfolding W

by (auto simp: W0_def psi_def ⟨0 < e⟩)

then have $\{x0\} \times C \subseteq W$ by blast

from tube_lemma[OF ⟨compact C⟩ ⟨open W⟩ this]

obtain X0 where X0: $x0 \in X0$ open X0 $X0 \times C \subseteq W$

by blast

have $\forall x \in X0 \cap U. \forall t \in C. \text{dist } (fx\ (x, t))\ (fx\ (x0, t)) \leq e$

proof safe

fix x assume x: $x \in X0$ $x \in U$

```

fix t assume t: t ∈ C
have dist (fx (x, t)) (fx (x0, t)) = psi (x, t)
  by (auto simp: psi_def)
also
{
  have (x, t) ∈ X0 × C
    using t x
    by auto
  also note ⟨... ⊆ W⟩
  finally have (x, t) ∈ W .
  with t x have (x, t) ∈ W ∩ U × C
    by blast
  also note ⟨W ∩ U × C = W0 ∩ U × C⟩
  finally have psi (x, t) < e
    by (auto simp: W0_def)
}
finally show dist (fx (x, t)) (fx (x0, t)) ≤ e by simp
qed
from X0(1,2) this show ?thesis ..
qed

```

2.1.12 Continuity

lemma *continuous_at_imp_continuous_within*:
 $\text{continuous (at } x) f \implies \text{continuous (at } x \text{ within } s) f$
unfolding *continuous_within* *continuous_at* **using** *Lim_at_imp_Lim_at_within*
by *auto*

lemma *Lim_trivial_limit*: $\text{trivial\_limit } net \implies (f \longrightarrow l) \text{ } net$
by *simp*

lemmas *continuous_on = continuous_on_def* — legacy theorem name

lemma *continuous_within_subset*:
 $\text{continuous (at } x \text{ within } s) f \implies t \subseteq s \implies \text{continuous (at } x \text{ within } t) f$
unfolding *continuous_within* **by**(*metis tendsto_within_subset*)

lemma *continuous_on_interior*:
 $\text{continuous\_on } s f \implies x \in \text{interior } s \implies \text{continuous (at } x) f$
by (*metis continuous_on_eq_continuous_at continuous_on_subset interiorE*)

lemma *continuous_on_eq*:
 $\llbracket \text{continuous\_on } s f; \bigwedge x. x \in s \implies f x = g x \rrbracket \implies \text{continuous\_on } s g$
unfolding *continuous_on_def* *tendsto_def* *eventually_at_topological*
by *simp*

Characterization of various kinds of continuity in terms of sequences.

lemma *continuous_within_sequentiallyI*:
fixes $f :: 'a :: \{\text{first\_countable\_topology}, \text{t2\_space}\} \Rightarrow 'b :: \text{topological\_space}$


```

  assumes  $\bigwedge u::nat \Rightarrow 'a. u \longrightarrow a \Longrightarrow (\forall n. u\ n \in s) \Longrightarrow (\lambda n. f\ (u\ n)) \longrightarrow$ 
 $f\ a$ 
  shows continuous (at a within s) f
  using assms unfolding continuous_within_tendsto_def [where  $l = f\ a$ ]
  by (auto intro!: sequentially_imp_eventually_within)

```

```

lemma continuous_within_tendsto_compose:
  fixes  $f::'a::t2\_space \Rightarrow 'b::topological\_space$ 
  assumes continuous (at a within s) f
           eventually  $(\lambda n. x\ n \in s)\ F$ 
            $(x \longrightarrow a)\ F$ 
  shows  $((\lambda n. f\ (x\ n)) \longrightarrow f\ a)\ F$ 
proof -
  have *: filterlim x (inf (nhds a) (principal s)) F
  using assms(2) assms(3) unfolding at_within_def filterlim_inf by (auto
simp: filterlim_principal eventually_mono)
  show ?thesis
  by (auto simp: assms(1) continuous_within[symmetric] tendsto_at_within_iff_tendsto_nhds[symmetric]
intro!: filterlim_compose[OF _ *])
qed

```

```

lemma continuous_within_tendsto_compose':
  fixes  $f::'a::t2\_space \Rightarrow 'b::topological\_space$ 
  assumes continuous (at a within s) f
            $\bigwedge n. x\ n \in s$ 
            $(x \longrightarrow a)\ F$ 
  shows  $((\lambda n. f\ (x\ n)) \longrightarrow f\ a)\ F$ 
  by (auto intro!: continuous_within_tendsto_compose[OF assms(1)] simp add:
assms)

```

```

lemma continuous_within_sequentially:
  fixes  $f::'a::\{first\_countable\_topology, t2\_space\} \Rightarrow 'b::topological\_space$ 
  shows continuous (at a within s)  $f \longleftrightarrow$ 
 $(\forall x. (\forall n::nat. x\ n \in s) \wedge (x \longrightarrow a)\ sequentially$ 
 $\longrightarrow ((f \circ x) \longrightarrow f\ a)\ sequentially)$ 
  using continuous_within_tendsto_compose'[of a s f sequentially]
        continuous_within_sequentiallyI[of a s f]
  by (auto simp: o_def)

```

```

lemma continuous_at_sequentiallyI:
  fixes  $f::'a::\{first\_countable\_topology, t2\_space\} \Rightarrow 'b::topological\_space$ 
  assumes  $\bigwedge u. u \longrightarrow a \Longrightarrow (\lambda n. f\ (u\ n)) \longrightarrow f\ a$ 
  shows continuous (at a) f
  using continuous_within_sequentiallyI[of a UNIV f] assms by auto

```

```

lemma continuous_at_sequentially:
  fixes  $f::'a::metric\_space \Rightarrow 'b::topological\_space$ 
  shows continuous (at a)  $f \longleftrightarrow$ 
 $(\forall x. (x \longrightarrow a)\ sequentially \dashrightarrow ((f \circ x) \longrightarrow f\ a)\ sequentially)$ 

```

using *continuous_within_sequentially*[of *a UNIV f*] by *simp*

lemma *continuous_on_sequentiallyI*:

fixes *f* :: 'a::{*first_countable_topology*, *t2_space*} $\Rightarrow$ 'b::*topological_space*
assumes $\bigwedge u a. (\forall n. u\ n \in s) \implies a \in s \implies u \longrightarrow a \implies (\lambda n. f\ (u\ n))$
 $\longrightarrow f\ a$

shows *continuous_on* *s f*

using *assms unfolding continuous_on_eq_continuous_within*

using *continuous_within_sequentiallyI*[of *s f*] by *auto*

lemma *continuous_on_sequentially*:

fixes *f* :: 'a::{*first_countable_topology*, *t2_space*} $\Rightarrow$ 'b::*topological_space*

shows *continuous_on* *s f* $\longleftrightarrow$

$(\forall x. \forall a \in s. (\forall n. x(n) \in s) \wedge (x \longrightarrow a) \text{ sequentially}$

$\longrightarrow ((f \circ x) \longrightarrow f\ a) \text{ sequentially})$

(**is** *?lhs* = *?rhs*)

proof

assume *?rhs*

then show *?lhs*

using *continuous_within_sequentially*[of *s f*]

unfolding *continuous_on_eq_continuous_within*

by *auto*

next

assume *?lhs*

then show *?rhs*

unfolding *continuous_on_eq_continuous_within*

using *continuous_within_sequentially*[of *s f*]

by *auto*

qed

Continuity in terms of open preimages.

lemma *continuous_at_open*:

continuous (at *x*) *f* $\longleftrightarrow (\forall t. \text{open } t \wedge f\ x \in t \longrightarrow (\exists s. \text{open } s \wedge x \in s \wedge (\forall x' \in s. (f\ x') \in t)))$

unfolding *continuous_within_topological* [of *x UNIV f*]

unfolding *imp_conjL*

by (*intro all_cong imp_cong ex_cong conj_cong refl*) *auto*

lemma *continuous_imp_tendsto*:

assumes *continuous* (at *x0*) *f*

and $x \longrightarrow x0$

shows $(f \circ x) \longrightarrow (f\ x0)$

proof (*rule topological_tendstoI*)

fix *S*

assume *open* *S* $f\ x0 \in S$

then obtain *T* **where** *T_def*: *open* *T* $x0 \in T \ \forall x \in T. f\ x \in S$

using *assms continuous_at_open* by *metis*

then have *eventually* $(\lambda n. x\ n \in T)$ *sequentially*

using *assms T_def* by (*auto simp: tendsto_def*)

then show *eventually* $(\lambda n. (f \circ x) \ n \in S)$ *sequentially*
using *T_def* **by** (*auto elim! eventually_mono*)
qed

2.1.13 Homeomorphisms

definition *homeomorphism* $s \ t \ f \ g \longleftrightarrow$
 $(\forall x \in s. (g(f \ x) = x)) \wedge (f \ ' \ s = t) \wedge \text{continuous\_on } s \ f \wedge$
 $(\forall y \in t. (f(g \ y) = y)) \wedge (g \ ' \ t = s) \wedge \text{continuous\_on } t \ g$

lemma *homeomorphismI* [*intro?*]:
assumes *continuous_on S f continuous_on T g*
 $f \ ' \ S \subseteq T \ g \ ' \ T \subseteq S \wedge x. x \in S \implies g(f \ x) = x \wedge y. y \in T \implies f(g \ y) = y$
shows *homeomorphism S T f g*
using *assms* **by** (*force simp: homeomorphism_def*)

lemma *homeomorphism_translation*:
fixes $a :: 'a :: \text{real\_normed\_vector}$
shows *homeomorphism* $((+) \ a \ ' \ S) \ S \ ((+) \ (- \ a)) \ ((+) \ a)$
unfolding *homeomorphism_def* **by** (*auto simp: algebra_simps continuous_intros*)

lemma *homeomorphism_ident*: *homeomorphism T T* $(\lambda a. a) \ (\lambda a. a)$
by (*rule homeomorphismI auto*)

lemma *homeomorphism_compose*:
assumes *homeomorphism S T f g homeomorphism T U h k*
shows *homeomorphism S U* $(h \ o \ f) \ (g \ o \ k)$
using *assms*
unfolding *homeomorphism_def*
by (*intro conjI ballI continuous_on_compose*) (*auto simp: image_iff*)

lemma *homeomorphism_cong*:
homeomorphism X' Y' f' g'
if *homeomorphism X Y f g* $X' = X \ Y' = Y \wedge x. x \in X \implies f' \ x = f \ x \wedge y. y \in Y \implies g' \ y = g \ y$
using *that* **by** (*auto simp add: homeomorphism_def*)

lemma *homeomorphism_empty* [*simp*]:
homeomorphism $\{\} \ \{\} \ f \ g$
unfolding *homeomorphism_def* **by** *auto*

lemma *homeomorphism_symD*: *homeomorphism S t f g* $\implies$ *homeomorphism t S* $g \ f$
by (*simp add: homeomorphism_def*)

lemma *homeomorphism_sym*: *homeomorphism S t f g* = *homeomorphism t S g f*
by (*force simp: homeomorphism_def*)

definition *homeomorphic* $:: 'a::\text{topological\_space} \ \text{set} \Rightarrow 'b::\text{topological\_space} \ \text{set}$

```

⇒ bool
  (infixr homeomorphic 60)
  where s homeomorphic t ≡ (∃ f g. homeomorphism s t f g)

lemma homeomorphic_empty [iff]:
  S homeomorphic {} ⟷ S = {} {} homeomorphic S ⟷ S = {}
  by (auto simp: homeomorphic_def homeomorphism_def)

lemma homeomorphic_refl: s homeomorphic s
  unfolding homeomorphic_def homeomorphism_def
  using continuous_on_id
  apply (rule_tac x = (λx. x) in exI)
  apply (rule_tac x = (λx. x) in exI)
  apply blast
  done

lemma homeomorphic_sym: s homeomorphic t ⟷ t homeomorphic s
  unfolding homeomorphic_def homeomorphism_def
  by blast

lemma homeomorphic_trans [trans]:
  assumes S homeomorphic T
  and T homeomorphic U
  shows S homeomorphic U
  using assms
  unfolding homeomorphic_def
  by (metis homeomorphism_compose)

lemma homeomorphic_minimal:
  s homeomorphic t ⟷
    (∃ f g. (∀ x ∈ s. f(x) ∈ t ∧ (g(f(x)) = x)) ∧
      (∀ y ∈ t. g(y) ∈ s ∧ (f(g(y)) = y)) ∧
      continuous_on s f ∧ continuous_on t g)
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (fastforce simp: homeomorphic_def homeomorphism_def)
next
  assume ?rhs
  then show ?lhs
    apply clarify
    unfolding homeomorphic_def homeomorphism_def
    by (metis equalityI image_subset_iff subsetI)
qed

lemma homeomorphicI [intro?]:
  ⟦f ‘ S = T; g ‘ T = S;
  continuous_on S f; continuous_on T g;

```

$\bigwedge x. x \in S \implies g(f(x)) = x;$
 $\bigwedge y. y \in T \implies f(g(y)) = y \implies S \text{ homeomorphic } T$
unfolding *homeomorphic_def homeomorphism_def* **by** *metis*

lemma *homeomorphism_of_subsets*:
 $\llbracket \text{homeomorphism } S \ T \ f \ g; S' \subseteq S; T'' \subseteq T; f \restriction S' = T' \rrbracket$
 $\implies \text{homeomorphism } S' \ T' \ f \ g$
apply (*auto simp: homeomorphism_def elim!: continuous_on_subset*)
by (*metis subsetD imageI*)

lemma *homeomorphism_apply1*: $\llbracket \text{homeomorphism } S \ T \ f \ g; x \in S \rrbracket \implies g(f \ x) = x$
by (*simp add: homeomorphism_def*)

lemma *homeomorphism_apply2*: $\llbracket \text{homeomorphism } S \ T \ f \ g; x \in T \rrbracket \implies f(g \ x) = x$
by (*simp add: homeomorphism_def*)

lemma *homeomorphism_image1*: $\text{homeomorphism } S \ T \ f \ g \implies f \restriction S = T$
by (*simp add: homeomorphism_def*)

lemma *homeomorphism_image2*: $\text{homeomorphism } S \ T \ f \ g \implies g \restriction T = S$
by (*simp add: homeomorphism_def*)

lemma *homeomorphism_cont1*: $\text{homeomorphism } S \ T \ f \ g \implies \text{continuous\_on } S \ f$
by (*simp add: homeomorphism_def*)

lemma *homeomorphism_cont2*: $\text{homeomorphism } S \ T \ f \ g \implies \text{continuous\_on } T \ g$
by (*simp add: homeomorphism_def*)

lemma *continuous_on_no_limpt*:
 $(\bigwedge x. \neg x \text{ islimpt } S) \implies \text{continuous\_on } S \ f$
unfolding *continuous_on_def*
by (*metis UNIV_I empty_iff eventually_at_topological islimptE open_UNIV tendsto_def trivial_limit_within*)

lemma *continuous_on_finite*:
fixes $S :: 'a::t1\_space \text{ set}$
shows $\text{finite } S \implies \text{continuous\_on } S \ f$
by (*metis continuous_on_no_limpt islimpt_finite*)

lemma *homeomorphic_finite*:
fixes $S :: 'a::t1\_space \text{ set}$ **and** $T :: 'b::t1\_space \text{ set}$
assumes *finite T*
shows $S \text{ homeomorphic } T \longleftrightarrow \text{finite } S \wedge \text{finite } T \wedge \text{card } S = \text{card } T$ (**is** *?lhs = ?rhs*)
proof
assume $S \text{ homeomorphic } T$
with *assms* **show** *?rhs*
apply (*auto simp: homeomorphic_def homeomorphism_def*)

```

    apply (metis finite_imageI)
  by (metis card_image_le finite_imageI le_antisym)
next
  assume R: ?rhs
  with finite_same_card_bij obtain h where bij_betw h S T
  by auto
  with R show ?lhs
  apply (auto simp: homeomorphic_def homeomorphism_def continuous_on_finite)
  apply (rule_tac x=h in exI)
  apply (rule_tac x=inv_into S h in exI)
  apply (auto simp: bij_betw_inv_into_left bij_betw_inv_into_right bij_betw_imp_surj_on
inv_into_inv bij_betwE)
  apply (metis bij_betw_def bij_betw_inv_into)
  done
qed

```

Relatively weak hypotheses if a set is compact.

```

lemma homeomorphism_compact:
  fixes f :: 'a::topological_space  $\Rightarrow$  'b::t2_space
  assumes compact s continuous_on s f f' s = t inj_on f s
  shows  $\exists g. \text{homeomorphism } s \ t \ f \ g$ 
proof -
  define g where g x = (SOME y. y  $\in$  s  $\wedge$  f y = x) for x
  have g:  $\forall x \in s. g (f x) = x$ 
    using assms(3) assms(4)[unfolded inj_on_def] unfolding g_def by auto
  {
    fix y
    assume y  $\in$  t
    then obtain x where x: f x = y x  $\in$  s
      using assms(3) by auto
    then have g (f x) = x using g by auto
    then have f (g y) = y unfolding x(1)[symmetric] by auto
  }
  then have g':  $\forall x \in t. f (g x) = x$  by auto
  moreover
  {
    fix x
    have x  $\in$  s  $\implies$  x  $\in$  g' t
      using g[THEN bspec][where x=x]
      unfolding image_iff
      using assms(3)
      by (auto intro!: bexI[where x=f x])
    moreover
    {
      assume x  $\in$  g' t
      then obtain y where y: y  $\in$  t g y = x by auto
      then obtain x' where x': x'  $\in$  s f x' = y
        using assms(3) by auto
      then have x  $\in$  s
    }
  }

```

```

    unfolding g_def
    using someI2[of  $\lambda b. b \in s \wedge f b = y \ x' \ \lambda x. x \in s$ ]
    unfolding y(2)[symmetric] and g_def
    by auto
  }
  ultimately have  $x \in s \longleftrightarrow x \in g \text{ ` } t \dots$ 
}
then have  $g \text{ ` } t = s$  by auto
ultimately show ?thesis
  unfolding homeomorphism_def homeomorphic_def
  using assms continuous_on_inv by fastforce
qed

```

```

lemma homeomorphic_compact:
  fixes f :: 'a::topological_space  $\Rightarrow$  'b::t2_space
  shows compact s  $\implies$  continuous_on s f  $\implies$  (f ` s = t)  $\implies$  inj_on f s  $\implies$  s
  homeomorphic t
  unfolding homeomorphic_def by (metis homeomorphism_compact)

```

Preservation of topological properties.

```

lemma homeomorphic_compactness: s homeomorphic t  $\implies$  (compact s  $\longleftrightarrow$  compact t)
  unfolding homeomorphic_def homeomorphism_def
  by (metis compact_continuous_image)

```

2.1.14 On Linorder Topologies

```

lemma islimpt_greaterThanLessThan1:
  fixes a b :: 'a::{\linorder_topology, dense_order}
  assumes a < b
  shows a islimpt {a <..<b}
proof (rule islimptI)
  fix T
  assume open T a  $\in$  T
  from open_right[OF this <a < b>]
  obtain c where c: a < c {a <..<c}  $\subseteq$  T by auto
  with assms dense[of a min c b]
  show  $\exists y \in \{a <..<b\}. y \in T \wedge y \neq a$ 
  by (metis atLeastLessThan_iff greaterThanLessThan_iff min_less_iff_conj
    not_le order.strict_implies_order subset_eq)
qed

```

```

lemma islimpt_greaterThanLessThan2:
  fixes a b :: 'a::{\linorder_topology, dense_order}
  assumes a < b
  shows b islimpt {a <..<b}
proof (rule islimptI)
  fix T
  assume open T b  $\in$  T

```

```

from open_left[OF this ‹a < b›]
obtain c where c: c < b {c<..b} ⊆ T by auto
with assms dense[of max a c b]
show ∃ y∈{a<..b}. y ∈ T ∧ y ≠ b
  by (metis greaterThanAtMost_iff greaterThanLessThan_iff max_less_iff conj
    not_le order.strict_implies_order subset_eq)
qed

```

```

lemma closure_greaterThanLessThan[simp]:
  fixes a b::'a::{linorder_topology, dense_order}
  shows a < b ⇒ closure {a <..b} = {a .. b} (is _ ⇒ ?l = ?r)
proof
  have ?l ⊆ closure ?r
    by (rule closure_mono) auto
  thus closure {a<..b} ⊆ {a..b} by simp
qed (auto simp: closure_def order.order_iff_strict islimpt_greaterThanLessThan1
  islimpt_greaterThanLessThan2)

```

```

lemma closure_greaterThan[simp]:
  fixes a b::'a::{no_top, linorder_topology, dense_order}
  shows closure {a<..b} = {a..b}
proof -
  from gt_ex obtain b where a < b by auto
  hence {a<..b} = {a<..b} ∪ {b..b} by auto
  also have closure ... = {a..b} using ‹a < b› unfolding closure_Un
    by auto
  finally show ?thesis .
qed

```

```

lemma closure_lessThan[simp]:
  fixes b::'a::{no_bot, linorder_topology, dense_order}
  shows closure {..b} = {..b}
proof -
  from lt_ex obtain a where a < b by auto
  hence {..b} = {a<..b} ∪ {..a} by auto
  also have closure ... = {..b} using ‹a < b› unfolding closure_Un
    by auto
  finally show ?thesis .
qed

```

```

lemma closure_atLeastLessThan[simp]:
  fixes a b::'a::{linorder_topology, dense_order}
  assumes a < b
  shows closure {a ..< b} = {a .. b}
proof -
  from assms have {a ..< b} = {a} ∪ {a <..b} by auto
  also have closure ... = {a .. b} unfolding closure_Un
    by (auto simp: assms less_imp_le)
  finally show ?thesis .

```


qed

```

lemma closure_greaterThanAtMost[simp]:
  fixes a b::'a::{linorder_topology, dense_order}
  assumes a < b
  shows closure {a <.. b} = {a .. b}
proof -
  from assms have {a <.. b} = {b} ∪ {a <..< b} by auto
  also have closure ... = {a .. b} unfolding closure_Un
    by (auto simp: assms less_imp_le)
  finally show ?thesis .
qed

```

end

2.2 Operators involving abstract topology

```

theory Abstract_Topology
imports
  Complex_Main
  HOL-Library.Set_Idioms
  HOL-Library.FuncSet
begin

```

2.2.1 General notion of a topology as a value

```

definition istopology :: ('a set  $\Rightarrow$  bool)  $\Rightarrow$  bool where
  istopology L  $\equiv$  ( $\forall S T. L S \longrightarrow L T \longrightarrow L (S \cap T)$ )  $\wedge$  ( $\forall \mathcal{K}. (\forall K \in \mathcal{K}. L K) \longrightarrow L (\bigcup \mathcal{K})$ )

```

```

typedef 'a topology = {L::('a set)  $\Rightarrow$  bool. istopology L}
morphisms openin topology
unfolding istopology_def by blast

```

```

lemma istopology_openin[intro]: istopology(openin U)
  using openin[of U] by blast

```

```

lemma istopology_open: istopology open
  by (auto simp: istopology_def)

```

```

lemma topology_inverse': istopology U  $\Longrightarrow$  openin (topology U) = U
  using topology_inverse[unfolded mem_Collect_eq] .

```

```

lemma topology_inverse_iff: istopology U  $\longleftrightarrow$  openin (topology U) = U
  using topology_inverse[of U] istopology_openin[of topology U] by auto

```

```

lemma topology_eq: T1 = T2  $\longleftrightarrow$  ( $\forall S. openin T1 S \longleftrightarrow openin T2 S$ )
proof
  assume T1 = T2

```

```

    then show  $\forall S. \text{openin } T1 \ S \longleftrightarrow \text{openin } T2 \ S$  by simp
next
  assume  $H: \forall S. \text{openin } T1 \ S \longleftrightarrow \text{openin } T2 \ S$ 
  then have  $\text{openin } T1 = \text{openin } T2$  by (simp add: fun_eq_iff)
  then have  $\text{topology } (\text{openin } T1) = \text{topology } (\text{openin } T2)$  by simp
  then show  $T1 = T2$  unfolding openin_inverse .
qed

```

The "universe": the union of all sets in the topology.

definition $\text{topspace } T = \bigcup \{S. \text{openin } T \ S\}$

Main properties of open sets

proposition *openin_clauses*:

fixes $U :: 'a \text{ topology}$

shows

$\text{openin } U \ \{\}$

$\bigwedge S \ T. \text{openin } U \ S \implies \text{openin } U \ T \implies \text{openin } U \ (S \cap T)$

$\bigwedge K. (\forall S \in K. \text{openin } U \ S) \implies \text{openin } U \ (\bigcup K)$

using openin[of U] unfolding istopology_def by auto

lemma *openin_subset*: $\text{openin } U \ S \implies S \subseteq \text{topspace } U$

unfolding topspace_def by blast

lemma *openin_empty*[simp]: $\text{openin } U \ \{\}$

by (rule openin_clauses)

lemma *openin_Int*[intro]: $\text{openin } U \ S \implies \text{openin } U \ T \implies \text{openin } U \ (S \cap T)$

by (rule openin_clauses)

lemma *openin_Union*[intro]: $(\bigwedge S. S \in K \implies \text{openin } U \ S) \implies \text{openin } U \ (\bigcup K)$

using openin_clauses by blast

lemma *openin_Un*[intro]: $\text{openin } U \ S \implies \text{openin } U \ T \implies \text{openin } U \ (S \cup T)$

using openin_Union[of {S,T} U] by auto

lemma *openin_topspace*[intro, simp]: $\text{openin } U \ (\text{topspace } U)$

by (force simp: openin_Union topspace_def)

lemma *openin_subopen*: $\text{openin } U \ S \longleftrightarrow (\forall x \in S. \exists T. \text{openin } U \ T \wedge x \in T \wedge T \subseteq S)$

(is ?lhs $\longleftrightarrow$?rhs)

proof

assume ?lhs

then show ?rhs by auto

next

assume $H: ?rhs$

let $?t = \bigcup \{T. \text{openin } U \ T \wedge T \subseteq S\}$

have $\text{openin } U \ ?t$ by (force simp: openin_Union)

also have $?t = S$ using H by *auto*
 finally show $\text{openin } U \ S$.
 qed

lemma *openin_INT* [intro]:
 assumes *finite I*
 $\bigwedge i. i \in I \implies \text{openin } T \ (U \ i)$
 shows $\text{openin } T \ ((\bigcap i \in I. U \ i) \cap \text{topspace } T)$
 using *assms* by (induct, auto simp: *inf_sup_aci*(2) *openin_Int*)

lemma *openin_INT2* [intro]:
 assumes *finite I I* $\neq \{\}$
 $\bigwedge i. i \in I \implies \text{openin } T \ (U \ i)$
 shows $\text{openin } T \ (\bigcap i \in I. U \ i)$
proof –
 have $(\bigcap i \in I. U \ i) \subseteq \text{topspace } T$
 using $\langle I \neq \{\} \rangle$ *openin_subset*[OF *assms*(3)] by *auto*
 then show *?thesis*
 using *openin_INT*[of _ _ U , OF *assms*(1) *assms*(3)] by (simp add: *inf_absorb2*
inf_commute)
 qed

lemma *openin_Inter* [intro]:
 assumes *finite F* $F \neq \{\}$ $\bigwedge X. X \in F \implies \text{openin } T \ X$ shows $\text{openin } T \ (\bigcap F)$
 by (metis (*full_types*) *assms openin_INT2 image_ident*)

lemma *openin_Int_Inter*:
 assumes *finite F* $\text{openin } T \ U$ $\bigwedge X. X \in F \implies \text{openin } T \ X$ shows $\text{openin } T \ (U \cap \bigcap F)$
 using *openin_Inter* [of *insert U F*] *assms* by *auto*

Closed sets

definition *closedin* :: '*a topology* $\Rightarrow$ '*a set* $\Rightarrow$ bool **where**
closedin U S $\longleftrightarrow S \subseteq \text{topspace } U \wedge \text{openin } U \ (\text{topspace } U - S)$

lemma *closedin_subset*: $\text{closedin } U \ S \implies S \subseteq \text{topspace } U$
 by (metis *closedin_def*)

lemma *closedin_empty*[simp]: $\text{closedin } U \ \{\}$
 by (simp add: *closedin_def*)

lemma *closedin_topspace*[intro, simp]: $\text{closedin } U \ (\text{topspace } U)$
 by (simp add: *closedin_def*)

lemma *closedin_Un*[intro]: $\text{closedin } U \ S \implies \text{closedin } U \ T \implies \text{closedin } U \ (S \cup T)$
 by (auto simp: *Diff_Un closedin_def*)

lemma *Diff_Inter*[intro]: $A - \bigcap S = \bigcup \{A - s \mid s. s \in S\}$
by *auto*

lemma *closedin_Union*:
assumes *finite* $S \wedge T. T \in S \implies \text{closedin } U \ T$
shows $\text{closedin } U \ (\bigcup S)$
using *assms* **by** *induction auto*

lemma *closedin_Inter*[intro]:
assumes $Ke: K \neq \{\}$
and $Kc: \bigwedge S. S \in K \implies \text{closedin } U \ S$
shows $\text{closedin } U \ (\bigcap K)$
using $Ke \ Kc$ **unfolding** *closedin_def Diff_Inter* **by** *auto*

lemma *closedin_INT*[intro]:
assumes $A \neq \{\} \wedge x. x \in A \implies \text{closedin } U \ (B \ x)$
shows $\text{closedin } U \ (\bigcap_{x \in A. B \ x})$
using *assms* **by** *blast*

lemma *closedin_Int*[intro]: $\text{closedin } U \ S \implies \text{closedin } U \ T \implies \text{closedin } U \ (S \cap T)$
using *closedin_Inter*[of $\{S, T\} \ U$] **by** *auto*

lemma *openin_closedin_eq*: $\text{openin } U \ S \longleftrightarrow S \subseteq \text{topspace } U \wedge \text{closedin } U \ (\text{topspace } U - S)$
by (*metis Diff_subset closedin_def double_diff equalityD1 openin_subset*)

lemma *topology_finer_closedin*:
 $\text{topspace } X = \text{topspace } Y \implies (\forall S. \text{openin } Y \ S \longrightarrow \text{openin } X \ S) \longleftrightarrow (\forall S. \text{closedin } Y \ S \longrightarrow \text{closedin } X \ S)$
by (*metis closedin_def openin_closedin_eq*)

lemma *openin_closedin*: $S \subseteq \text{topspace } U \implies (\text{openin } U \ S \longleftrightarrow \text{closedin } U \ (\text{topspace } U - S))$
by (*simp add: openin_closedin_eq*)

lemma *openin_diff*[intro]:
assumes $oS: \text{openin } U \ S$
and $cT: \text{closedin } U \ T$
shows $\text{openin } U \ (S - T)$

proof –
have $S - T = S \cap (\text{topspace } U - T)$ **using** *openin_subset*[of $U \ S$] $oS \ cT$
by (*auto simp: topspace_def openin_subset*)
then show *?thesis* **using** $oS \ cT$
by (*auto simp: closedin_def*)

qed

lemma *closedin_diff*[intro]:
assumes $oS: \text{closedin } U \ S$

```

    and cT: openin U T
  shows closedin U (S - T)
proof -
  have S - T = S ∩ (topspace U - T)
    using closedin_subset[of U S] oS cT by (auto simp: topspace_def)
  then show ?thesis
    using oS cT by (auto simp: openin_closedin_eq)
qed

```

2.2.2 The discrete topology

definition *discrete_topology* where $\text{discrete\_topology } U \equiv \text{topology } (\lambda S. S \subseteq U)$

lemma *openin_discrete_topology* [simp]: $\text{openin } (\text{discrete\_topology } U) S \longleftrightarrow S \subseteq U$

```

proof -
  have istopology ( $\lambda S. S \subseteq U$ )
    by (auto simp: istopology_def)
  then show ?thesis
    by (simp add: discrete_topology_def topology_inverse')
qed

```

lemma *topspace_discrete_topology* [simp]: $\text{topspace}(\text{discrete\_topology } U) = U$
 by (meson openin_discrete_topology openin_subset openin_topspace order_refl subset_antisym)

lemma *closedin_discrete_topology* [simp]: $\text{closedin } (\text{discrete\_topology } U) S \longleftrightarrow S \subseteq U$
 by (simp add: closedin_def)

lemma *discrete_topology_unique*:

$\text{discrete\_topology } U = X \longleftrightarrow \text{topspace } X = U \wedge (\forall x \in U. \text{openin } X \{x\})$ (is ?lhs = ?rhs)

```

proof
  assume R: ?rhs
  then have openin X S if S ⊆ U for S
    using openin_subopen subsetD that by fastforce
  moreover have x ∈ topspace X if openin X S and x ∈ S for x S
    using openin_subset that by blast
  ultimately
  show ?lhs
    using R by (auto simp: topology_eq)
qed auto

```

lemma *discrete_topology_unique_alt*:

$\text{discrete\_topology } U = X \longleftrightarrow \text{topspace } X \subseteq U \wedge (\forall x \in U. \text{openin } X \{x\})$
 using openin_subset
 by (auto simp: discrete_topology_unique)

lemma *subtopology_eq_discrete_topology_empty*:
 $X = \text{discrete\_topology } \{\} \longleftrightarrow \text{topspace } X = \{\}$
using *discrete_topology_unique* [of $\{\}$ X] **by** *auto*

lemma *subtopology_eq_discrete_topology_sing*:
 $X = \text{discrete\_topology } \{a\} \longleftrightarrow \text{topspace } X = \{a\}$
by (*metis discrete_topology_unique openin_topspace singletonD*)

2.2.3 Subspace topology

definition *subtopology* :: 'a topology $\Rightarrow$ 'a set $\Rightarrow$ 'a topology **where**
subtopology $U\ V = \text{topology } (\lambda T. \exists S. T = S \cap V \wedge \text{openin } U\ S)$

lemma *istopology_subtopology*: *istopology* $(\lambda T. \exists S. T = S \cap V \wedge \text{openin } U\ S)$
(is *istopology* ? L)

proof –

have ? $L\ \{\}$ **by** *blast*
{
 fix $A\ B$
 assume $A: ?L\ A$ **and** $B: ?L\ B$
 from $A\ B$ **obtain** Sa **and** Sb **where** $Sa: \text{openin } U\ Sa$ $A = Sa \cap V$ **and** $Sb:$
 $\text{openin } U\ Sb$ $B = Sb \cap V$
 by *blast*
 have $A \cap B = (Sa \cap Sb) \cap V$ $\text{openin } U\ (Sa \cap Sb)$
 using $Sa\ Sb$ **by** *blast+*
 then have ? $L\ (A \cap B)$ **by** *blast*
}
moreover
{
 fix K
 assume $K: K \subseteq \text{Collect } ?L$
 have $th0: \text{Collect } ?L = (\lambda S. S \cap V) \text{ ` Collect } (\text{openin } U)$
 by *blast*
 from K [*unfolded th0 subset_image_iff*]
 obtain Sk **where** $Sk: Sk \subseteq \text{Collect } (\text{openin } U)$ $K = (\lambda S. S \cap V) \text{ ` } Sk$
 by *blast*
 have $\bigcup K = (\bigcup Sk) \cap V$
 using Sk **by** *auto*
 moreover have $\text{openin } U\ (\bigcup Sk)$
 using Sk **by** (*auto simp: subset_eq*)
 ultimately have ? $L\ (\bigcup K)$ **by** *blast*
}
ultimately show ?thesis
 unfolding *subset_eq mem_Collect_eq istopology_def* **by** *auto*
qed

lemma *openin_subtopology*: *openin* (*subtopology* $U\ V$) $S \longleftrightarrow (\exists T. \text{openin } U\ T \wedge S = T \cap V)$
unfolding *subtopology_def topology_inverse'* [*OF istopology_subtopology*]

by *auto*

lemma *openin_subtopology_Int*:
 $\text{openin } X \ S \implies \text{openin } (\text{subtopology } X \ T) \ (S \cap T)$
using *openin_subtopology* **by** *auto*

lemma *openin_subtopology_Int2*:
 $\text{openin } X \ T \implies \text{openin } (\text{subtopology } X \ S) \ (S \cap T)$
using *openin_subtopology* **by** *auto*

lemma *openin_subtopology_diff_closed*:
 $\llbracket S \subseteq \text{topspace } X; \text{closedin } X \ T \rrbracket \implies \text{openin } (\text{subtopology } X \ S) \ (S - T)$
unfolding *closedin_def openin_subtopology*
by (*rule_tac* $x = \text{topspace } X - T$ **in** *exI*) *auto*

lemma *openin_relative_to*: $(\text{openin } X \ \text{relative\_to } S) = \text{openin } (\text{subtopology } X \ S)$
by (*force simp: relative_to_def openin_subtopology*)

lemma *topspace_subtopology [simp]*: $\text{topspace } (\text{subtopology } U \ V) = \text{topspace } U \cap V$
by (*auto simp: topspace_def openin_subtopology*)

lemma *topspace_subtopology_subset*:
 $S \subseteq \text{topspace } X \implies \text{topspace } (\text{subtopology } X \ S) = S$
by (*simp add: inf.absorb_iff2*)

lemma *closedin_subtopology*: $\text{closedin } (\text{subtopology } U \ V) \ S \longleftrightarrow (\exists T. \text{closedin } U \ T \wedge S = T \cap V)$
unfolding *closedin_def topspace_subtopology*
by (*auto simp: openin_subtopology*)

lemma *openin_subtopology_refl*: $\text{openin } (\text{subtopology } U \ V) \ V \longleftrightarrow V \subseteq \text{topspace } U$
unfolding *openin_subtopology*
by *auto (metis IntD1 in_mono openin_subset)*

lemma *subtopology_subtopology*:
 $\text{subtopology } (\text{subtopology } X \ S) \ T = \text{subtopology } X \ (S \cap T)$
proof –
have *eq*: $\bigwedge T'. (\exists S'. T' = S' \cap T \wedge (\exists T. \text{openin } X \ T \wedge S' = T \cap S)) = (\exists Sa. T' = Sa \cap (S \cap T) \wedge \text{openin } X \ Sa)$
by (*metis inf_assoc*)
have $\text{subtopology } (\text{subtopology } X \ S) \ T = \text{topology } (\lambda Ta. \exists Sa. Ta = Sa \cap T \wedge \text{openin } (\text{subtopology } X \ S) \ Sa)$
by (*simp add: subtopology_def*)
also have $\dots = \text{subtopology } X \ (S \cap T)$
by (*simp add: openin_subtopology_eq (simp add: subtopology_def)*)
finally show *?thesis* .
qed

```

lemma openin_subtopology_alt:
  openin (subtopology X U) S  $\longleftrightarrow$  S  $\in$  ( $\lambda T. U \cap T$ ) ‘ Collect (openin X)
by (simp add: image_iff inf_commute openin_subtopology)

lemma closedin_subtopology_alt:
  closedin (subtopology X U) S  $\longleftrightarrow$  S  $\in$  ( $\lambda T. U \cap T$ ) ‘ Collect (closedin X)
by (simp add: image_iff inf_commute closedin_subtopology)

lemma subtopology_superset:
  assumes UV: topspace U  $\subseteq$  V
  shows subtopology U V = U
proof –
  {
    fix S
    {
      fix T
      assume T: openin U T S = T  $\cap$  V
      from T openin_subset[OF T(1)] UV have eq: S = T
      by blast
      have openin U S
      unfolding eq using T by blast
    }
    moreover
    {
      assume S: openin U S
      then have  $\exists T. openin U T \wedge S = T \cap V$ 
      using openin_subset[OF S] UV by auto
    }
    ultimately have ( $\exists T. openin U T \wedge S = T \cap V$ )  $\longleftrightarrow$  openin U S
    by blast
  }
  then show ?thesis
  unfolding topology_eq openin_subtopology by blast
qed

lemma subtopology_topspace[simp]: subtopology U (topspace U) = U
by (simp add: subtopology_superset)

lemma subtopology_UNIV[simp]: subtopology U UNIV = U
by (simp add: subtopology_superset)

lemma subtopology_restrict:
  subtopology X (topspace X  $\cap$  S) = subtopology X S
by (metis subtopology_subtopology subtopology_topspace)

lemma openin_subtopology_empty:
  openin (subtopology U {}) S  $\longleftrightarrow$  S = {}
by (metis Int_empty_right openin_empty openin_subtopology)

```


lemma *closedin_subtopology_empty*:

$\text{closedin } (\text{subtopology } U \ \{\}) \ S \longleftrightarrow S = \{\}$

by (*metis* *Int_empty_right* *closedin_empty* *closedin_subtopology*)

lemma *closedin_subtopology_refl* [*simp*]:

$\text{closedin } (\text{subtopology } U \ X) \ X \longleftrightarrow X \subseteq \text{topspace } U$

by (*metis* *closedin_def* *closedin_topspace* *inf.absorb_iff2* *le_inf_iff* *topspace_subtopology*)

lemma *closedin_topspace_empty*: $\text{topspace } T = \{\} \implies (\text{closedin } T \ S \longleftrightarrow S = \{\})$

by (*simp* *add*: *closedin_def*)

lemma *open_in_topspace_empty*:

$\text{topspace } X = \{\} \implies \text{openin } X \ S \longleftrightarrow S = \{\}$

by (*simp* *add*: *openin_closedin_eq*)

lemma *openin_imp_subset*:

$\text{openin } (\text{subtopology } U \ S) \ T \implies T \subseteq S$

by (*metis* *Int_iff* *openin_subtopology_subsetI*)

lemma *closedin_imp_subset*:

$\text{closedin } (\text{subtopology } U \ S) \ T \implies T \subseteq S$

by (*simp* *add*: *closedin_def*)

lemma *openin_open_subtopology*:

$\text{openin } X \ S \implies \text{openin } (\text{subtopology } X \ S) \ T \longleftrightarrow \text{openin } X \ T \wedge T \subseteq S$

by (*metis* *inf.orderE* *openin_Int* *openin_imp_subset* *openin_subtopology*)

lemma *closedin_closed_subtopology*:

$\text{closedin } X \ S \implies (\text{closedin } (\text{subtopology } X \ S) \ T \longleftrightarrow \text{closedin } X \ T \wedge T \subseteq S)$

by (*metis* *closedin_Int* *closedin_imp_subset* *closedin_subtopology* *inf.orderE*)

lemma *openin_subtopology_Un*:

$\llbracket \text{openin } (\text{subtopology } X \ T) \ S; \text{openin } (\text{subtopology } X \ U) \ S \rrbracket$

$\implies \text{openin } (\text{subtopology } X \ (T \cup U)) \ S$

by (*simp* *add*: *openin_subtopology*) *blast*

lemma *closedin_subtopology_Un*:

$\llbracket \text{closedin } (\text{subtopology } X \ T) \ S; \text{closedin } (\text{subtopology } X \ U) \ S \rrbracket$

$\implies \text{closedin } (\text{subtopology } X \ (T \cup U)) \ S$

by (*simp* *add*: *closedin_subtopology*) *blast*

lemma *openin_trans_full*:

$\llbracket \text{openin } (\text{subtopology } X \ U) \ S; \text{openin } X \ U \rrbracket \implies \text{openin } X \ S$

by (*simp* *add*: *openin_open_subtopology*)

2.2.4 The canonical topology from the underlying type class

abbreviation *euclidean* :: 'a::topological_space topology
 where *euclidean* $\equiv$ *topology open*

abbreviation *top_of_set* :: 'a::topological_space set $\Rightarrow$ 'a topology
 where *top_of_set* $\equiv$ *subtopology (topology open)*

lemma *open_openin*: *open S* $\longleftrightarrow$ *openin euclidean S*
 by (*simp add: istopology_open topology_inverse*)

declare *open_openin* [*symmetric, simp*]

lemma *topspace_euclidean* [*simp*]: *topspace euclidean* = *UNIV*
 by (*force simp: topspace_def*)

lemma *topspace_euclidean_subtopology*[*simp*]: *topspace (top_of_set S)* = *S*
 by (*simp*)

lemma *closed_closedin*: *closed S* $\longleftrightarrow$ *closedin euclidean S*
 by (*simp add: closed_def closedin_def Compl_eq_Diff_UNIV*)

declare *closed_closedin* [*symmetric, simp*]

lemma *openin_subtopology_self* [*simp*]: *openin (top_of_set S) S*
 by (*metis openin_topspace topspace_euclidean_subtopology*)

The most basic facts about the usual topology and metric on R

abbreviation *euclideanreal* :: *real topology*
 where *euclideanreal* $\equiv$ *topology open*

2.2.5 Basic "localization" results are handy for connectedness.

lemma *openin_open*: *openin (top_of_set U) S* $\longleftrightarrow$ ($\exists T. \text{open } T \wedge (S = U \cap T)$)
 by (*auto simp: openin_subtopology*)

lemma *openin_Int_open*:
 $\llbracket \text{openin } (top\_of\_set\ U)\ S; \text{open } T \rrbracket$
 $\implies \text{openin } (top\_of\_set\ U)\ (S \cap T)$
 by (*metis open_Int Int_assoc openin_open*)

lemma *openin_open_Int[intro]*: *open S* $\implies \text{openin } (top\_of\_set\ U)\ (U \cap S)$
 by (*auto simp: openin_open*)

lemma *open_openin_trans[trans]*:
 $\text{open } S \implies \text{open } T \implies T \subseteq S \implies \text{openin } (top\_of\_set\ S)\ T$
 by (*metis Int_absorb1 openin_open_Int*)

```

lemma open_subset:  $S \subseteq T \implies \text{open } S \implies \text{openin } (\text{top\_of\_set } T) S$ 
  by (auto simp: openin_open)

lemma closedin_closed:  $\text{closedin } (\text{top\_of\_set } U) S \longleftrightarrow (\exists T. \text{closed } T \wedge S = U \cap T)$ 
  by (simp add: closedin_subtopology Int_ac)

lemma closedin_closed_Int:  $\text{closed } S \implies \text{closedin } (\text{top\_of\_set } U) (U \cap S)$ 
  by (metis closedin_closed)

lemma closed_subset:  $S \subseteq T \implies \text{closed } S \implies \text{closedin } (\text{top\_of\_set } T) S$ 
  by (auto simp: closedin_closed)

lemma closedin_closed_subset:
   $\llbracket \text{closedin } (\text{top\_of\_set } U) V; T \subseteq U; S = V \cap T \rrbracket$ 
   $\implies \text{closedin } (\text{top\_of\_set } T) S$ 
  by (metis (no_types, lifting) Int_assoc Int_commute closedin_closed inf.orderE)

lemma finite_imp_closedin:
  fixes  $S :: 'a::t1\_space \text{ set}$ 
  shows  $\llbracket \text{finite } S; S \subseteq T \rrbracket \implies \text{closedin } (\text{top\_of\_set } T) S$ 
  by (simp add: finite_imp_closed closed_subset)

lemma closedin_singleton [simp]:
  fixes  $a :: 'a::t1\_space$ 
  shows  $\text{closedin } (\text{top\_of\_set } U) \{a\} \longleftrightarrow a \in U$ 
using closedin_subset by (force intro: closed_subset)

lemma openin_euclidean_subtopology_iff:
  fixes  $S U :: 'a::metric\_space \text{ set}$ 
  shows  $\text{openin } (\text{top\_of\_set } U) S \longleftrightarrow$ 
 $S \subseteq U \wedge (\forall x \in S. \exists e > 0. \forall x' \in U. \text{dist } x' x < e \longrightarrow x' \in S)$ 
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  assume ?lhs
  then show ?rhs
    unfolding openin_open open_dist by blast
next
  define  $T$  where  $T = \{x. \exists a \in S. \exists d > 0. (\forall y \in U. \text{dist } y a < d \longrightarrow y \in S) \wedge \text{dist } x a < d\}$ 
  have 1:  $\forall x \in T. \exists e > 0. \forall y. \text{dist } y x < e \longrightarrow y \in T$ 
    unfolding T_def
    apply clarsimp
    apply (rule_tac  $x=d - \text{dist } x a$  in exI)
    by (metis add_0_left dist_commute dist_triangle_lt less_diff_eq)
  assume ?rhs then have 2:  $S = U \cap T$ 
    unfolding T_def
    by auto (metis dist_self)

```

```

from 1 2 show ?lhs
  unfolding openin_open open_dist by fast
qed

```

```

lemma connected_openin:
  connected S  $\longleftrightarrow$ 
   $\neg(\exists E1 E2. \text{openin } (\text{top\_of\_set } S) E1 \wedge$ 
     $\text{openin } (\text{top\_of\_set } S) E2 \wedge$ 
     $S \subseteq E1 \cup E2 \wedge E1 \cap E2 = \{\} \wedge E1 \neq \{\} \wedge E2 \neq \{\})$ 
  unfolding connected_def openin_open disjoint_iff_not_equal by blast

```

```

lemma connected_openin_eq:
  connected S  $\longleftrightarrow$ 
   $\neg(\exists E1 E2. \text{openin } (\text{top\_of\_set } S) E1 \wedge$ 
     $\text{openin } (\text{top\_of\_set } S) E2 \wedge$ 
     $E1 \cup E2 = S \wedge E1 \cap E2 = \{\} \wedge$ 
     $E1 \neq \{\} \wedge E2 \neq \{\})$ 
  unfolding connected_openin
  by (metis (no_types, lifting) Un_subset_iff openin_imp_subset subset_antisym)

```

```

lemma connected_closedin:
  connected S  $\longleftrightarrow$ 
   $(\nexists E1 E2.$ 
     $\text{closedin } (\text{top\_of\_set } S) E1 \wedge$ 
     $\text{closedin } (\text{top\_of\_set } S) E2 \wedge$ 
     $S \subseteq E1 \cup E2 \wedge E1 \cap E2 = \{\} \wedge E1 \neq \{\} \wedge E2 \neq \{\})$ 
  (is ?lhs = ?rhs)

```

```

proof
  assume ?lhs
  then show ?rhs
    by (auto simp add: connected_closed closedin_closed)
  next
    assume R: ?rhs
    then show ?lhs
      proof (clarsimp simp add: connected_closed closedin_closed)
        fix A B
        assume s_sub:  $S \subseteq A \cup B \wedge B \cap S \neq \{\}$ 
        and disj:  $A \cap B \cap S = \{\}$ 
        and cl: closed A closed B
        have  $S \cap (A \cup B) = S$ 
        using s_sub(1) by auto
        have  $S - A = B \cap S$ 
        using Diff_subset_conv Un_Diff_Int disj s_sub(1) by auto
        then have  $S \cap A = \{\}$ 
        by (metis Diff_Diff_Int Diff_disjoint Un_Diff_Int R cl closedin_closed_Int
          inf_commute order_refl s_sub(2))
        then show  $A \cap S = \{\}$ 
        by blast
      qed

```

qed

lemma *connected_closedin_eq*:

$$\begin{aligned} & \text{connected } S \longleftrightarrow \\ & \neg(\exists E1\ E2. \\ & \quad \text{closedin } (\text{top\_of\_set } S)\ E1 \wedge \\ & \quad \text{closedin } (\text{top\_of\_set } S)\ E2 \wedge \\ & \quad E1 \cup E2 = S \wedge E1 \cap E2 = \{\} \wedge \\ & \quad E1 \neq \{\} \wedge E2 \neq \{\}) \end{aligned}$$

unfolding *connected_closedin*

by (metis *Un_subset_iff closedin_imp_subset subset_antisym*)

These "transitivity" results are handy too

lemma *openin_trans[trans]*:

$$\begin{aligned} & \text{openin } (\text{top\_of\_set } T)\ S \implies \text{openin } (\text{top\_of\_set } U)\ T \implies \\ & \text{openin } (\text{top\_of\_set } U)\ S \end{aligned}$$

by (metis *openin_Int_open openin_open*)

lemma *openin_open_trans*: $\text{openin } (\text{top\_of\_set } T)\ S \implies \text{open } T \implies \text{open } S$

by (auto simp: *openin_open intro: openin_trans*)

lemma *closedin_trans[trans]*:

$$\begin{aligned} & \text{closedin } (\text{top\_of\_set } T)\ S \implies \text{closedin } (\text{top\_of\_set } U)\ T \implies \\ & \text{closedin } (\text{top\_of\_set } U)\ S \end{aligned}$$

by (auto simp: *closedin_closed closed_Inter Int_assoc*)

lemma *closedin_closed_trans*: $\text{closedin } (\text{top\_of\_set } T)\ S \implies \text{closed } T \implies \text{closed } S$

by (auto simp: *closedin_closed intro: closedin_trans*)

lemma *openin_subtopology_Int_subset*:

$$\llbracket \text{openin } (\text{top\_of\_set } u)\ (u \cap S); v \subseteq u \rrbracket \implies \text{openin } (\text{top\_of\_set } v)\ (v \cap S)$$

by (auto simp: *openin_subtopology*)

lemma *openin_open_eq*: $\text{open } s \implies (\text{openin } (\text{top\_of\_set } s)\ t \longleftrightarrow \text{open } t \wedge t \subseteq s)$

using *open_subset openin_open_trans openin_subset* **by** *fastforce*

2.2.6 Derived set (set of limit points)

definition *derived_set_of* :: $'a\ \text{topology} \Rightarrow 'a\ \text{set} \Rightarrow 'a\ \text{set}$ (**infixl** *derived'_set'_of* 80)

where $X\ \text{derived\_set\_of } S \equiv$

$$\begin{aligned} & \{x \in \text{topspace } X. \\ & \quad (\forall T. x \in T \wedge \text{openin } X\ T \longrightarrow (\exists y \neq x. y \in S \wedge y \in T))\} \end{aligned}$$

lemma *derived_set_of_restrict [simp]*:

$$X\ \text{derived\_set\_of } (\text{topspace } X \cap S) = X\ \text{derived\_set\_of } S$$

by (simp add: *derived_set_of_def*) (metis *openin_subset subset_iff*)

lemma *in_derived_set_of*:

$x \in X \text{ derived\_set\_of } S \iff x \in \text{topspace } X \wedge (\forall T. x \in T \wedge \text{openin } X \ T \longrightarrow (\exists y \neq x. y \in S \wedge y \in T))$
by (*simp add: derived_set_of_def*)

lemma *derived_set_of_subset_topspace*:

$X \text{ derived\_set\_of } S \subseteq \text{topspace } X$
by (*auto simp add: derived_set_of_def*)

lemma *derived_set_of_subtopology*:

$(\text{subtopology } X \ U) \text{ derived\_set\_of } S = U \cap (X \text{ derived\_set\_of } (U \cap S))$
by (*simp add: derived_set_of_def openin_subtopology blast*)

lemma *derived_set_of_subset_subtopology*:

$(\text{subtopology } X \ S) \text{ derived\_set\_of } T \subseteq S$
by (*simp add: derived_set_of_subtopology*)

lemma *derived_set_of_empty* [*simp*]: $X \text{ derived\_set\_of } \{\} = \{\}$

by (*auto simp: derived_set_of_def*)

lemma *derived_set_of_mono*:

$S \subseteq T \implies X \text{ derived\_set\_of } S \subseteq X \text{ derived\_set\_of } T$
unfolding *derived_set_of_def* **by** *blast*

lemma *derived_set_of_Un*:

$X \text{ derived\_set\_of } (S \cup T) = X \text{ derived\_set\_of } S \cup X \text{ derived\_set\_of } T$ (**is** *?lhs = ?rhs*)

proof

show *?lhs* $\subseteq$ *?rhs*

by (*clarsimp simp: in_derived_set_of*) (*metis IntE IntI openin_Int*)

show *?rhs* $\subseteq$ *?lhs*

by (*simp add: derived_set_of_mono*)

qed

lemma *derived_set_of_Union*:

$\text{finite } \mathcal{F} \implies X \text{ derived\_set\_of } (\bigcup \mathcal{F}) = (\bigcup S \in \mathcal{F}. X \text{ derived\_set\_of } S)$

proof (*induction* $\mathcal{F}$ *rule: finite_induct*)

case (*insert* $S \ \mathcal{F}$)

then show *?case*

by (*simp add: derived_set_of_Un*)

qed *auto*

lemma *derived_set_of_topspace*:

$X \text{ derived\_set\_of } (\text{topspace } X) = \{x \in \text{topspace } X. \neg \text{openin } X \ \{x\}\}$ (**is** *?lhs = ?rhs*)

proof

show *?lhs* $\subseteq$ *?rhs*

by (*auto simp: in_derived_set_of*)

```

show ?rhs  $\subseteq$  ?lhs
by (clarsimp simp: in_derived_set_of) (metis openin_closedin_eq openin_subopen
singletonD subset_eq)
qed

```

```

lemma discrete_topology_unique_derived_set:
  discrete_topology U = X  $\longleftrightarrow$  topspace X = U  $\wedge$  X derived_set_of U = {}
by (auto simp: discrete_topology_unique_derived_set_of_topspace)

```

```

lemma subtopology_eq_discrete_topology_eq:
  subtopology X U = discrete_topology U  $\longleftrightarrow$  U  $\subseteq$  topspace X  $\wedge$  U  $\cap$  X de-
rived_set_of U = {}
using discrete_topology_unique_derived_set [of U subtopology X U]
by (auto simp: eq_commute derived_set_of_subtopology)

```

```

lemma subtopology_eq_discrete_topology:
  S  $\subseteq$  topspace X  $\wedge$  S  $\cap$  X derived_set_of S = {}
 $\implies$  subtopology X S = discrete_topology S
by (simp add: subtopology_eq_discrete_topology_eq)

```

```

lemma subtopology_eq_discrete_topology_gen:
  S  $\cap$  X derived_set_of S = {}  $\implies$  subtopology X S = discrete_topology (topspace
X  $\cap$  S)
by (metis Int_lower1 derived_set_of_restrict inf_assoc inf_bot_right subtopol-
ogy_eq_discrete_topology_eq subtopology_subtopology subtopology_topspace)

```

```

lemma subtopology_discrete_topology [simp]:
  subtopology (discrete_topology U) S = discrete_topology (U  $\cap$  S)
proof -
have ( $\lambda T. \exists Sa. T = Sa \cap S \wedge Sa \subseteq U$ ) = ( $\lambda Sa. Sa \subseteq U \wedge Sa \subseteq S$ )
by force
then show ?thesis
by (simp add: subtopology_def) (simp add: discrete_topology_def)
qed

```

```

lemma openin_Int_derived_set_of_subset:
  openin X S  $\implies$  S  $\cap$  X derived_set_of T  $\subseteq$  X derived_set_of (S  $\cap$  T)
by (auto simp: derived_set_of_def)

```

```

lemma openin_Int_derived_set_of_eq:
assumes openin X S
shows S  $\cap$  X derived_set_of T = S  $\cap$  X derived_set_of (S  $\cap$  T) (is ?lhs =
?rhs)
proof
show ?lhs  $\subseteq$  ?rhs
by (simp add: asms openin_Int_derived_set_of_subset)
show ?rhs  $\subseteq$  ?lhs
by (metis derived_set_of_mono inf_commute inf_le1 inf_mono order_refl)
qed

```

2.2.7 Closure with respect to a topological space

definition *closure_of* :: 'a topology $\Rightarrow$ 'a set $\Rightarrow$ 'a set (**infixr** *closure'_of* 80)
where $X \text{ closure\_of } S \equiv \{x \in \text{topspace } X. \forall T. x \in T \wedge \text{openin } X T \longrightarrow (\exists y \in S. y \in T)\}$

lemma *closure_of_restrict*: $X \text{ closure\_of } S = X \text{ closure\_of } (\text{topspace } X \cap S)$
unfolding *closure_of_def*
using *openin_subset* **by** *blast*

lemma *in_closure_of*:
 $x \in X \text{ closure\_of } S \longleftrightarrow$
 $x \in \text{topspace } X \wedge (\forall T. x \in T \wedge \text{openin } X T \longrightarrow (\exists y. y \in S \wedge y \in T))$
by (*auto simp: closure_of_def*)

lemma *closure_of*: $X \text{ closure\_of } S = \text{topspace } X \cap (S \cup X \text{ derived\_set\_of } S)$
by (*fastforce simp: in_closure_of in_derived_set_of*)

lemma *closure_of_alt*: $X \text{ closure\_of } S = \text{topspace } X \cap S \cup X \text{ derived\_set\_of } S$
using *derived_set_of_subset_topspace* [*of X S*]
unfolding *closure_of_def in_derived_set_of*
by *safe (auto simp: in_derived_set_of)*

lemma *derived_set_of_subset_closure_of*:
 $X \text{ derived\_set\_of } S \subseteq X \text{ closure\_of } S$
by (*fastforce simp: closure_of_def in_derived_set_of*)

lemma *closure_of_subtopology*:
 $(\text{subtopology } X U) \text{ closure\_of } S = U \cap (X \text{ closure\_of } (U \cap S))$
unfolding *closure_of_def topspace_subtopology openin_subtopology*
by *safe (metis (full_types) IntI Int_iff inf.commute)+*

lemma *closure_of_empty* [*simp*]: $X \text{ closure\_of } \{\} = \{\}$
by (*simp add: closure_of_alt*)

lemma *closure_of_topspace* [*simp*]: $X \text{ closure\_of } \text{topspace } X = \text{topspace } X$
by (*simp add: closure_of*)

lemma *closure_of_UNIV* [*simp*]: $X \text{ closure\_of } \text{UNIV} = \text{topspace } X$
by (*simp add: closure_of*)

lemma *closure_of_subset_topspace*: $X \text{ closure\_of } S \subseteq \text{topspace } X$
by (*simp add: closure_of*)

lemma *closure_of_subset_subtopology*: $(\text{subtopology } X S) \text{ closure\_of } T \subseteq S$
by (*simp add: closure_of_subtopology*)

lemma *closure_of_mono*: $S \subseteq T \implies X \text{ closure\_of } S \subseteq X \text{ closure\_of } T$
by (*fastforce simp add: closure_of_def*)


```

lemma closure_of_subtopology_subset:
  (subtopology X U) closure_of S  $\subseteq$  (X closure_of S)
unfolding closure_of_subtopology
by clarsimp (meson closure_of_mono contra_subsetD inf.cobounded2)

lemma closure_of_subtopology_mono:
  T  $\subseteq$  U  $\implies$  (subtopology X T) closure_of S  $\subseteq$  (subtopology X U) closure_of S
unfolding closure_of_subtopology
by auto (meson closure_of_mono inf_mono subset_iff)

lemma closure_of_Un [simp]: X closure_of (S  $\cup$  T) = X closure_of S  $\cup$  X
closure_of T
by (simp add: Un_assoc Un_left_commute closure_of_alt derived_set_of_Un
inf_sup_distrib1)

lemma closure_of_Union:
  finite F  $\implies$  X closure_of ( $\bigcup$  F) = ( $\bigcup$  S  $\in$  F. X closure_of S)
by (induction F rule: finite_induct) auto

lemma closure_of_subset: S  $\subseteq$  topspace X  $\implies$  S  $\subseteq$  X closure_of S
by (auto simp: closure_of_def)

lemma closure_of_subset_Int: topspace X  $\cap$  S  $\subseteq$  X closure_of S
by (auto simp: closure_of_def)

lemma closure_of_subset_eq: S  $\subseteq$  topspace X  $\wedge$  X closure_of S  $\subseteq$  S  $\longleftrightarrow$  closedin
X S
proof -
  have openin X (topspace X - S)
  if  $\bigwedge x. \llbracket x \in \text{topspace } X; \forall T. x \in T \wedge \text{openin } X T \longrightarrow S \cap T \neq \{\} \rrbracket \implies x \in S$ 
  apply (subst openin_subopen)
  by (metis Diff_iff Diff_mono Diff_triv inf commute openin_subset order_refl
that)
  then show ?thesis
  by (auto simp: closedin_def closure_of_def disjoint_iff_not_equal)
qed

lemma closure_of_eq: X closure_of S = S  $\longleftrightarrow$  closedin X S
proof (cases S  $\subseteq$  topspace X)
  case True
  then show ?thesis
  by (metis closure_of_subset closure_of_subset_eq set_eq_subset)
next
  case False
  then show ?thesis
  using closure_of_closure_of_subset_eq by fastforce
qed

lemma closedin_contains_derived_set:

```

$closedin\ X\ S \longleftrightarrow X\ derived\_set\_of\ S \subseteq S \wedge S \subseteq topspace\ X$
proof (intro iffI conjI)
 show $closedin\ X\ S \implies X\ derived\_set\_of\ S \subseteq S$
 using closure_of_eq derived_set_of_subset_closure_of **by** fastforce
 show $closedin\ X\ S \implies S \subseteq topspace\ X$
 using closedin_subset **by** blast
 show $X\ derived\_set\_of\ S \subseteq S \wedge S \subseteq topspace\ X \implies closedin\ X\ S$
 by (metis closure_of_closure_of_eq inf.absorb_iff2 sup.orderE)
qed

lemma derived_set_subset_gen:

$X\ derived\_set\_of\ S \subseteq S \longleftrightarrow closedin\ X\ (topspace\ X \cap S)$

by (simp add: closedin_contains_derived_set derived_set_of_restrict derived_set_of_subset_topspace)

lemma derived_set_subset: $S \subseteq topspace\ X \implies (X\ derived\_set\_of\ S \subseteq S \longleftrightarrow closedin\ X\ S)$

by (simp add: closedin_contains_derived_set)

lemma closedin_derived_set:

$closedin\ (subtopology\ X\ T)\ S \longleftrightarrow$

$S \subseteq topspace\ X \wedge S \subseteq T \wedge (\forall x. x \in X\ derived\_set\_of\ S \wedge x \in T \longrightarrow x \in$

$S)$

by (auto simp: closedin_contains_derived_set derived_set_of_subtopology Int_absorb1)

lemma closedin_Int_closure_of:

$closedin\ (subtopology\ X\ S)\ T \longleftrightarrow S \cap X\ closure\_of\ T = T$

by (metis Int_left_absorb closure_of_eq closure_of_subtopology)

lemma closure_of_closedin: $closedin\ X\ S \implies X\ closure\_of\ S = S$

by (simp add: closure_of_eq)

lemma closure_of_eq_diff: $X\ closure\_of\ S = topspace\ X - \bigcup \{T. openin\ X\ T \wedge disjnt\ S\ T\}$

by (auto simp: closure_of_def disjnt_iff)

lemma closedin_closure_of [simp]: $closedin\ X\ (X\ closure\_of\ S)$

unfolding closure_of_eq_diff **by** blast

lemma closure_of_closure_of [simp]: $X\ closure\_of\ (X\ closure\_of\ S) = X\ closure\_of\ S$

by (simp add: closure_of_eq)

lemma closure_of_hull:

assumes $S \subseteq topspace\ X$ **shows** $X\ closure\_of\ S = (closedin\ X)\ hull\ S$

proof (rule hull_unique [THEN sym])

show $S \subseteq X\ closure\_of\ S$

by (simp add: closure_of_subset assms)

next

show $closedin\ X\ (X\ closure\_of\ S)$

```

  by simp
  show  $\bigwedge T. \llbracket S \subseteq T; \text{closedin } X \ T \rrbracket \implies X \text{ closure\_of } S \subseteq T$ 
    by (metis closure_of_eq closure_of_mono)
qed

```

```

lemma closure_of_minimal:
   $\llbracket S \subseteq T; \text{closedin } X \ T \rrbracket \implies (X \text{ closure\_of } S) \subseteq T$ 
  by (metis closure_of_eq closure_of_mono)

```

```

lemma closure_of_minimal_eq:
   $\llbracket S \subseteq \text{topspace } X; \text{closedin } X \ T \rrbracket \implies (X \text{ closure\_of } S) \subseteq T \longleftrightarrow S \subseteq T$ 
  by (meson closure_of_minimal closure_of_subset subset_trans)

```

```

lemma closure_of_unique:
   $\llbracket S \subseteq T; \text{closedin } X \ T; \bigwedge T'. \llbracket S \subseteq T'; \text{closedin } X \ T' \rrbracket \implies T \subseteq T' \rrbracket$ 
   $\implies X \text{ closure\_of } S = T$ 
  by (meson closedin_closure_of closedin_subset closure_of_minimal closure_of_subset
    eq_iff order.trans)

```

```

lemma closure_of_eq_empty_gen:  $X \text{ closure\_of } S = \{\} \longleftrightarrow \text{disjnt } (\text{topspace } X) S$ 
  unfolding disjnt_def closure_of_restrict [where S=S]
  using closure_of by fastforce

```

```

lemma closure_of_eq_empty:  $S \subseteq \text{topspace } X \implies X \text{ closure\_of } S = \{\} \longleftrightarrow S = \{\}$ 
  using closure_of_subset by fastforce

```

```

lemma openin_Int_closure_of_subset:
  assumes openin X S
  shows  $S \cap X \text{ closure\_of } T \subseteq X \text{ closure\_of } (S \cap T)$ 
proof -
  have  $S \cap X \text{ derived\_set\_of } T = S \cap X \text{ derived\_set\_of } (S \cap T)$ 
    by (meson assms openin_Int_derived_set_of_eq)
  moreover have  $S \cap (S \cap T) = S \cap T$ 
    by fastforce
  ultimately show ?thesis
    by (metis closure_of_alt inf.cobounded2 inf_left_commute inf_sup_distrib1)
qed

```

```

lemma closure_of_openin_Int_closure_of:
  assumes openin X S
  shows  $X \text{ closure\_of } (S \cap X \text{ closure\_of } T) = X \text{ closure\_of } (S \cap T)$ 
proof
  show  $X \text{ closure\_of } (S \cap X \text{ closure\_of } T) \subseteq X \text{ closure\_of } (S \cap T)$ 
    by (simp add: assms closure_of_minimal openin_Int_closure_of_subset)
next
  show  $X \text{ closure\_of } (S \cap T) \subseteq X \text{ closure\_of } (S \cap X \text{ closure\_of } T)$ 

```

by (metis Int_lower1 Int_subset_iff assms closedin_closure_of closure_of_minimal_eq
closure_of_mono inf_le2 le_infI1 openin_subset)
qed

lemma openin_Int_closure_of_eq:
assumes openin X S shows $S \cap X \text{ closure\_of } T = S \cap X \text{ closure\_of } (S \cap T)$
(is ?lhs = ?rhs)
proof
show ?lhs $\subseteq$?rhs
by (simp add: assms openin_Int_closure_of_subset)
show ?rhs $\subseteq$?lhs
by (metis closure_of_mono inf_commute inf_le1 inf_mono order_refl)
qed

lemma openin_Int_closure_of_eq_empty:
assumes openin X S shows $S \cap X \text{ closure\_of } T = \{\} \longleftrightarrow S \cap T = \{\}$ (is
?lhs = ?rhs)
proof
show ?lhs $\implies$?rhs
unfolding disjoint_iff
by (meson assms in_closure_of_in_mono openin_subset)
show ?rhs $\implies$?lhs
by (simp add: assms openin_Int_closure_of_eq)
qed

lemma closure_of_openin_Int_superset:
openin X S $\wedge S \subseteq X \text{ closure\_of } T$
 $\implies X \text{ closure\_of } (S \cap T) = X \text{ closure\_of } S$
by (metis closure_of_openin_Int_closure_of_inf.orderE)

lemma closure_of_openin_subtopology_Int_closure_of:
assumes S: openin (subtopology X U) S and $T \subseteq U$
shows $X \text{ closure\_of } (S \cap X \text{ closure\_of } T) = X \text{ closure\_of } (S \cap T)$ (is ?lhs =
?rhs)
proof
obtain S0 where S0: openin X S0 $S = S0 \cap U$
using assms by (auto simp: openin_subtopology)
show ?lhs $\subseteq$?rhs
proof –
have $S0 \cap X \text{ closure\_of } T = S0 \cap X \text{ closure\_of } (S0 \cap T)$
by (meson S0(1) openin_Int_closure_of_eq)
moreover have $S0 \cap T = S0 \cap U \cap T$
using $\langle T \subseteq U \rangle$ by fastforce
ultimately have $S \cap X \text{ closure\_of } T \subseteq X \text{ closure\_of } (S \cap T)$
using S0(2) by auto
then show ?thesis
by (meson closedin_closure_of closure_of_minimal)
qed
next

```

show ?rhs  $\subseteq$  ?lhs
proof -
  have  $T \cap S \subseteq T \cup X \text{ derived\_set\_of } T$ 
  by force
  then show ?thesis
    by (metis Int_subset_iff S closure_of closure_of_mono inf.cobounded2
inf.coboundedI2 inf_commute openin_closedin_eq topspace_subtopology)
  qed
qed

```

```

lemma closure_of_subtopology_open:
   $\text{openin } X \ U \vee S \subseteq U \implies (\text{subtopology } X \ U) \text{ closure\_of } S = U \cap X \text{ closure\_of } S$ 
by (metis closure_of_subtopology inf_absorb2 openin_Int_closure_of_eq)

```

```

lemma discrete_topology_closure_of:
   $(\text{discrete\_topology } U) \text{ closure\_of } S = U \cap S$ 
by (metis closedin_discrete_topology closure_of_restrict closure_of_unique discrete_topology_unique inf_sup_ord(1) order_refl)

```

Interior with respect to a topological space.

```

definition interior_of :: 'a topology  $\Rightarrow$  'a set  $\Rightarrow$  'a set (infixr interior'_of 80)
  where  $X \text{ interior\_of } S \equiv \{x. \exists T. \text{openin } X \ T \wedge x \in T \wedge T \subseteq S\}$ 

```

```

lemma interior_of_restrict:
   $X \text{ interior\_of } S = X \text{ interior\_of } (\text{topspace } X \cap S)$ 
using openin_subset by (auto simp: interior_of_def)

```

```

lemma interior_of_eq:  $(X \text{ interior\_of } S = S) \longleftrightarrow \text{openin } X \ S$ 
unfolding interior_of_def using openin_subopen by blast

```

```

lemma interior_of_openin:  $\text{openin } X \ S \implies X \text{ interior\_of } S = S$ 
by (simp add: interior_of_eq)

```

```

lemma interior_of_empty [simp]:  $X \text{ interior\_of } \{\} = \{\}$ 
by (simp add: interior_of_eq)

```

```

lemma interior_of_topspace [simp]:  $X \text{ interior\_of } (\text{topspace } X) = \text{topspace } X$ 
by (simp add: interior_of_eq)

```

```

lemma openin_interior_of [simp]:  $\text{openin } X \ (X \text{ interior\_of } S)$ 
unfolding interior_of_def
using openin_subopen by fastforce

```

```

lemma interior_of_interior_of [simp]:
   $X \text{ interior\_of } X \text{ interior\_of } S = X \text{ interior\_of } S$ 
by (simp add: interior_of_eq)

```

```

lemma interior_of_subset:  $X \text{ interior\_of } S \subseteq S$ 

```

by (*auto simp: interior_of_def*)

lemma *interior_of_subset_closure_of*: $X \text{ interior\_of } S \subseteq X \text{ closure\_of } S$
by (*metis closure_of_subset_Int dual_order.trans interior_of_restrict interior_of_subset*)

lemma *subset_interior_of_eq*: $S \subseteq X \text{ interior\_of } S \longleftrightarrow \text{openin } X S$
by (*metis interior_of_eq interior_of_subset subset_antisym*)

lemma *interior_of_mono*: $S \subseteq T \implies X \text{ interior\_of } S \subseteq X \text{ interior\_of } T$
by (*auto simp: interior_of_def*)

lemma *interior_of_maximal*: $\llbracket T \subseteq S; \text{openin } X T \rrbracket \implies T \subseteq X \text{ interior\_of } S$
by (*auto simp: interior_of_def*)

lemma *interior_of_maximal_eq*: $\text{openin } X T \implies T \subseteq X \text{ interior\_of } S \longleftrightarrow T \subseteq S$
by (*meson interior_of_maximal interior_of_subset order_trans*)

lemma *interior_of_unique*:
 $\llbracket T \subseteq S; \text{openin } X T; \bigwedge T'. \llbracket T' \subseteq S; \text{openin } X T' \rrbracket \implies T' \subseteq T \rrbracket \implies X \text{ interior\_of } S = T$
by (*simp add: interior_of_maximal_eq interior_of_subset subset_antisym*)

lemma *interior_of_subset_topspace*: $X \text{ interior\_of } S \subseteq \text{topspace } X$
by (*simp add: openin_subset*)

lemma *interior_of_subset_subtopology*: $(\text{subtopology } X S) \text{ interior\_of } T \subseteq S$
by (*meson openin_imp_subset openin_interior_of*)

lemma *interior_of_Int*: $X \text{ interior\_of } (S \cap T) = X \text{ interior\_of } S \cap X \text{ interior\_of } T$ (*is ?lhs = ?rhs*)

proof

show $?lhs \subseteq ?rhs$

by (*simp add: interior_of_mono*)

show $?rhs \subseteq ?lhs$

by (*meson inf_mono interior_of_maximal interior_of_subset openin_Int openin_interior_of*)

qed

lemma *interior_of_Inter_subset*: $X \text{ interior\_of } (\bigcap \mathcal{F}) \subseteq (\bigcap S \in \mathcal{F}. X \text{ interior\_of } S)$
by (*simp add: INT_greatest Inf_lower interior_of_mono*)

lemma *union_interior_of_subset*:
 $X \text{ interior\_of } S \cup X \text{ interior\_of } T \subseteq X \text{ interior\_of } (S \cup T)$
by (*simp add: interior_of_mono*)

lemma *interior_of_eq_empty*:
 $X \text{ interior\_of } S = \{\} \longleftrightarrow (\forall T. \text{openin } X T \wedge T \subseteq S \longrightarrow T = \{\})$

by (*metis* *bot.extremum_uniqueI interior_of_maximal interior_of_subset openin_interior_of*)

lemma *interior_of_eq_empty_alt*:

$X \text{ interior\_of } S = \{\} \iff (\forall T. \text{openin } X \ T \wedge T \neq \{\} \longrightarrow T - S \neq \{\})$

by (*auto simp: interior_of_eq_empty*)

lemma *interior_of_Union_openin_subsets*:

$\bigcup \{T. \text{openin } X \ T \wedge T \subseteq S\} = X \text{ interior\_of } S$

by (*rule interior_of_unique [symmetric]*) *auto*

lemma *interior_of_complement*:

$X \text{ interior\_of } (\text{topspace } X - S) = \text{topspace } X - X \text{ closure\_of } S$

by (*auto simp: interior_of_def closure_of_def*)

lemma *interior_of_closure_of*:

$X \text{ interior\_of } S = \text{topspace } X - X \text{ closure\_of } (\text{topspace } X - S)$

unfolding *interior_of_complement [symmetric]*

by (*metis Diff_Diff_Int interior_of_restrict*)

lemma *closure_of_interior_of*:

$X \text{ closure\_of } S = \text{topspace } X - X \text{ interior\_of } (\text{topspace } X - S)$

by (*simp add: interior_of_complement Diff_Diff_Int closure_of*)

lemma *closure_of_complement*: $X \text{ closure\_of } (\text{topspace } X - S) = \text{topspace } X - X \text{ interior\_of } S$

unfolding *interior_of_def closure_of_def*

by (*blast dest: openin_subset*)

lemma *interior_of_eq_empty_complement*:

$X \text{ interior\_of } S = \{\} \iff X \text{ closure\_of } (\text{topspace } X - S) = \text{topspace } X$

using *interior_of_subset_topspace [of X S] closure_of_complement* **by** *fastforce*

lemma *closure_of_eq_topspace*:

$X \text{ closure\_of } S = \text{topspace } X \iff X \text{ interior\_of } (\text{topspace } X - S) = \{\}$

using *closure_of_subset_topspace [of X S] interior_of_complement* **by** *fastforce*

lemma *interior_of_subtopology_subset*:

$U \cap X \text{ interior\_of } S \subseteq (\text{subtopology } X \ U) \text{ interior\_of } S$

by (*auto simp: interior_of_def openin_subtopology*)

lemma *interior_of_subtopology_subsets*:

$T \subseteq U \implies T \cap (\text{subtopology } X \ U) \text{ interior\_of } S \subseteq (\text{subtopology } X \ T) \text{ interior\_of } S$

by (*metis inf.absorb_iff2 interior_of_subtopology_subset subtopology_subtopology*)

lemma *interior_of_subtopology_mono*:

$\llbracket S \subseteq T; T \subseteq U \rrbracket \implies (\text{subtopology } X \ U) \text{ interior\_of } S \subseteq (\text{subtopology } X \ T) \text{ interior\_of } S$

by (*metis dual_order.trans inf.orderE inf_commute interior_of_subset inte-*

rior_of_subtopology_subsets)

lemma *interior_of_subtopology_open*:

assumes *openin* $X\ U$

shows $(\text{subtopology } X\ U)\ \text{interior\_of } S = U \cap X\ \text{interior\_of } S$

proof –

have $\forall A. U \cap X\ \text{closure\_of } (U \cap A) = U \cap X\ \text{closure\_of } A$

using *assms openin_Int_closure_of_eq* **by** *blast*

then have $\text{topspace } X \cap U - U \cap X\ \text{closure\_of } (\text{topspace } X \cap U - S) = U \cap$
 $(\text{topspace } X - X\ \text{closure\_of } (\text{topspace } X - S))$

by (*metis* (*no_types*) *Diff_Int_distrib* *Int_Diff_inf_commute*)

then show *?thesis*

unfolding *interior_of_closure_of_closure_of_subtopology_open* *topspace_subtopology*

using *openin_Int_closure_of_eq* [*OF assms*]

by (*metis assms closure_of_subtopology_open*)

qed

lemma *dense_intersects_open*:

$X\ \text{closure\_of } S = \text{topspace } X \longleftrightarrow (\forall T. \text{openin } X\ T \wedge T \neq \{\} \longrightarrow S \cap T \neq \{\})$

proof –

have $X\ \text{closure\_of } S = \text{topspace } X \longleftrightarrow (\text{topspace } X - X\ \text{interior\_of } (\text{topspace } X - S) = \text{topspace } X)$

by (*simp add: closure_of_interior_of*)

also have $\dots \longleftrightarrow X\ \text{interior\_of } (\text{topspace } X - S) = \{\}$

by (*simp add: closure_of_complement interior_of_eq_empty_complement*)

also have $\dots \longleftrightarrow (\forall T. \text{openin } X\ T \wedge T \neq \{\} \longrightarrow S \cap T \neq \{\})$

unfolding *interior_of_eq_empty_alt*

using *openin_subset* **by** *fastforce*

finally show *?thesis* .

qed

lemma *interior_of_closedin_union_empty_interior_of*:

assumes *closedin* $X\ S$ **and** *disj*: $X\ \text{interior\_of } T = \{\}$

shows $X\ \text{interior\_of } (S \cup T) = X\ \text{interior\_of } S$

proof –

have $X\ \text{closure\_of } (\text{topspace } X - T) = \text{topspace } X$

by (*metis* *Diff_Diff_Int disj closure_of_eq_topspace closure_of_restrict interior_of_closure_of*)

then show *?thesis*

unfolding *interior_of_closure_of*

by (*metis* *Diff_Un Diff_subset assms(1) closedin_def closure_of_openin_Int_superset*)

qed

lemma *interior_of_union_eq_empty*:

closedin $X\ S$

$\implies (X\ \text{interior\_of } (S \cup T) = \{\} \longleftrightarrow$

$X\ \text{interior\_of } S = \{\} \wedge X\ \text{interior\_of } T = \{\})$

by (*metis interior_of_closedin_union_empty_interior_of le_sup_iff subset_empty*)

union_interior_of_subset)

lemma *discrete_topology_interior_of* [simp]:
 (*discrete_topology* *U*) *interior_of* *S* = *U* $\cap$ *S*
 by (*simp* add: *interior_of_restrict* [*of* _ *S*] *interior_of_eq*)

2.2.8 Frontier with respect to topological space

definition *frontier_of* :: 'a topology $\Rightarrow$ 'a set $\Rightarrow$ 'a set (**infixr** *frontier'* of 80)
 where *X frontier_of* *S* $\equiv$ *X closure_of* *S* - *X interior_of* *S*

lemma *frontier_of_closures*:
 $X \text{ frontier\_of } S = X \text{ closure\_of } S \cap X \text{ closure\_of } (\text{topspace } X - S)$
 by (*metis* *Diff_Diff_Int* *closure_of_complement* *closure_of_subset_topspace* *double_diff* *frontier_of_def* *interior_of_subset_closure_of*)

lemma *interior_of_union_frontier_of* [simp]:
 $X \text{ interior\_of } S \cup X \text{ frontier\_of } S = X \text{ closure\_of } S$
 by (*simp* add: *frontier_of_def* *interior_of_subset_closure_of* *subset_antisym*)

lemma *frontier_of_restrict*: $X \text{ frontier\_of } S = X \text{ frontier\_of } (\text{topspace } X \cap S)$
 by (*metis* *closure_of_restrict* *frontier_of_def* *interior_of_restrict*)

lemma *closedin_frontier_of*: *closedin* *X* (*X frontier_of* *S*)
 by (*simp* add: *closedin_Int* *frontier_of_closures*)

lemma *frontier_of_subset_topspace*: $X \text{ frontier\_of } S \subseteq \text{topspace } X$
 by (*simp* add: *closedin_frontier_of* *closedin_subset*)

lemma *frontier_of_subset_subtopology*: (*subtopology* *X* *S*) *frontier_of* *T* $\subseteq$ *S*
 by (*metis* (*no_types*) *closedin_derived_set* *closedin_frontier_of*)

lemma *frontier_of_subtopology_subset*:
 $U \cap (\text{subtopology } X \text{ } U) \text{ frontier\_of } S \subseteq (X \text{ frontier\_of } S)$
proof –
 have $U \cap X \text{ interior\_of } S - \text{subtopology } X \text{ } U \text{ interior\_of } S = \{\}$
 by (*simp* add: *interior_of_subtopology_subset*)
 moreover have $X \text{ closure\_of } S \cap \text{subtopology } X \text{ } U \text{ closure\_of } S = \text{subtopology } X \text{ } U \text{ closure\_of } S$
 by (*meson* *closure_of_subtopology_subset* *inf.absorb_iff2*)
 ultimately show ?thesis
 unfolding *frontier_of_def*
 by blast
qed

lemma *frontier_of_subtopology_mono*:
 $\llbracket S \subseteq T; T \subseteq U \rrbracket \Longrightarrow (\text{subtopology } X \text{ } T) \text{ frontier\_of } S \subseteq (\text{subtopology } X \text{ } U) \text{ frontier\_of } S$

by (*simp* add: *frontier_of_def Diff_mono closure_of_subtopology_mono interior_of_subtopology_mono*)

lemma *clopenin_eq_frontier_of*:

$\text{closedin } X \ S \wedge \text{openin } X \ S \longleftrightarrow S \subseteq \text{topspace } X \wedge X \text{frontier\_of } S = \{\}$

proof (*cases* $S \subseteq \text{topspace } X$)

case *True*

then show *?thesis*

by (*metis* *Diff_eq_empty_iff closure_of_eq closure_of_subset_eq frontier_of_def interior_of_eq interior_of_subset interior_of_union frontier_of_sup_bot_right*)

next

case *False*

then show *?thesis*

by (*simp* add: *frontier_of_closures openin_closedin_eq*)

qed

lemma *frontier_of_eq_empty*:

$S \subseteq \text{topspace } X \implies (X \text{frontier\_of } S = \{\} \longleftrightarrow \text{closedin } X \ S \wedge \text{openin } X \ S)$

by (*simp* add: *clopenin_eq_frontier_of*)

lemma *frontier_of_openin*:

$\text{openin } X \ S \implies X \text{frontier\_of } S = X \text{closure\_of } S - S$

by (*metis* (*no_types*) *frontier_of_def interior_of_eq*)

lemma *frontier_of_openin_straddle_Int*:

assumes $\text{openin } X \ U \ U \cap X \text{frontier\_of } S \neq \{\}$

shows $U \cap S \neq \{\} \ U - S \neq \{\}$

proof –

have $U \cap (X \text{closure\_of } S \cap X \text{closure\_of } (\text{topspace } X - S)) \neq \{\}$

using *assms* **by** (*simp* add: *frontier_of_closures*)

then show $U \cap S \neq \{\}$

using *assms* *openin_Int_closure_of_eq_empty* **by** *fastforce*

show $U - S \neq \{\}$

proof –

have $\exists A. X \text{closure\_of } (A - S) \cap U \neq \{\}$

using $\langle U \cap (X \text{closure\_of } S \cap X \text{closure\_of } (\text{topspace } X - S)) \neq \{\} \rangle$ **by**

blast

then have $\neg U \subseteq S$

by (*metis* *Diff_disjoint Diff_eq_empty_iff Int_Diff assms(1) inf_commute openin_Int_closure_of_eq_empty*)

then show *?thesis*

by *blast*

qed

qed

lemma *frontier_of_subset_closedin*: $\text{closedin } X \ S \implies (X \text{frontier\_of } S) \subseteq S$

using *closure_of_eq frontier_of_def* **by** *fastforce*

lemma *frontier_of_empty* [*simp*]: $X \text{frontier\_of } \{\} = \{\}$

```

    by (simp add: frontier_of_def)

lemma frontier_of_topspace [simp]:  $X \text{ frontier\_of } \text{topspace } X = \{\}$ 
  by (simp add: frontier_of_def)

lemma frontier_of_subset_eq:
  assumes  $S \subseteq \text{topspace } X$ 
  shows  $(X \text{ frontier\_of } S) \subseteq S \longleftrightarrow \text{closedin } X S$ 
proof
  show  $X \text{ frontier\_of } S \subseteq S \implies \text{closedin } X S$ 
  by (metis assms closure_of_subset_eq interior_of_subset interior_of_union_frontier_of
le_sup_iff)
  show  $\text{closedin } X S \implies X \text{ frontier\_of } S \subseteq S$ 
  by (simp add: frontier_of_subset_closedin)
qed

lemma frontier_of_complement:  $X \text{ frontier\_of } (\text{topspace } X - S) = X \text{ frontier\_of } S$ 
  by (metis Diff_Diff_Int closure_of_restrict frontier_of_closures inf_commute)

lemma frontier_of_disjoint_eq:
  assumes  $S \subseteq \text{topspace } X$ 
  shows  $((X \text{ frontier\_of } S) \cap S = \{\} \longleftrightarrow \text{openin } X S)$ 
proof
  assume  $X \text{ frontier\_of } S \cap S = \{\}$ 
  then have  $\text{closedin } X (\text{topspace } X - S)$ 
  using assms closure_of_subset frontier_of_def interior_of_eq interior_of_subset
  by fastforce
  then show  $\text{openin } X S$ 
  using assms by (simp add: openin_closedin)
next
  show  $\text{openin } X S \implies X \text{ frontier\_of } S \cap S = \{\}$ 
  by (simp add: Diff_Diff_Int closedin_def frontier_of_openin inf_absorb_iff2
inf_commute)
qed

lemma frontier_of_disjoint_eq_alt:
   $S \subseteq (\text{topspace } X - X \text{ frontier\_of } S) \longleftrightarrow \text{openin } X S$ 
proof (cases  $S \subseteq \text{topspace } X$ )
  case True
  show ?thesis
  using True frontier_of_disjoint_eq by auto
next
  case False
  then show ?thesis
  by (meson Diff_subset openin_subset subset_trans)
qed

lemma frontier_of_Int:

```

```

      X frontier_of (S ∩ T) =
      X closure_of (S ∩ T) ∩ (X frontier_of S ∪ X frontier_of T)
proof -
  have *: U ⊆ S ∧ U ⊆ T ⇒ U ∩ (S ∩ A ∪ T ∩ B) = U ∩ (A ∪ B) for U S
  T A B :: 'a set
  by blast
  show ?thesis
  by (simp add: frontier_of_closures closure_of_mono Diff_Int * flip: clo-
sure_of_Un)
qed

lemma frontier_of_Int_subset: X frontier_of (S ∩ T) ⊆ X frontier_of S ∪ X
frontier_of T
by (simp add: frontier_of_Int)

lemma frontier_of_Int_closedin:
  assumes closedin X S closedin X T
  shows X frontier_of (S ∩ T) = X frontier_of S ∩ T ∪ S ∩ X frontier_of T (is
?lhs = ?rhs)
proof
  show ?lhs ⊆ ?rhs
  using assms by (force simp add: frontier_of_Int closedin_Int closure_of_closedin)
  show ?rhs ⊆ ?lhs
  using assms frontier_of_subset_closedin
  by (auto simp add: frontier_of_Int closedin_Int closure_of_closedin)
qed

lemma frontier_of_Un_subset: X frontier_of (S ∪ T) ⊆ X frontier_of S ∪ X
frontier_of T
by (metis Diff_Un frontier_of_Int_subset frontier_of_complement)

lemma frontier_of_Union_subset:
  finite F ⇒ X frontier_of (⋃ F) ⊆ (⋃ T ∈ F. X frontier_of T)
proof (induction F rule: finite_induct)
  case (insert A F)
  then show ?case
  using frontier_of_Un_subset by fastforce
qed simp

lemma frontier_of_frontier_of_subset:
  X frontier_of (X frontier_of S) ⊆ X frontier_of S
by (simp add: closedin_frontier_of frontier_of_subset_closedin)

lemma frontier_of_subtopology_open:
  openin X U ⇒ (subtopology X U) frontier_of S = U ∩ X frontier_of S
by (simp add: Diff_Int_distrib closure_of_subtopology_open frontier_of_def
interior_of_subtopology_open)

lemma discrete_topology_frontier_of [simp]:

```

(discrete_topology U) frontier_of S = {}
 by (simp add: Diff_eq discrete_topology_closure_of_frontier_of_closures)

2.2.9 Locally finite collections

definition locally_finite_in

where

locally_finite_in X $\mathcal{A} \longleftrightarrow$
 $(\bigcup \mathcal{A} \subseteq \text{topspace } X) \wedge$
 $(\forall x \in \text{topspace } X. \exists V. \text{openin } X V \wedge x \in V \wedge \text{finite } \{U \in \mathcal{A}. U \cap V \neq \{\}\})$

lemma finite_imp_locally_finite_in:

$\llbracket \text{finite } \mathcal{A}; \bigcup \mathcal{A} \subseteq \text{topspace } X \rrbracket \implies \text{locally\_finite\_in } X \mathcal{A}$

by (auto simp: locally_finite_in_def)

lemma locally_finite_in_subset:

assumes locally_finite_in X $\mathcal{A} \mathcal{B} \subseteq \mathcal{A}$

shows locally_finite_in X $\mathcal{B}$

proof –

have finite ($\mathcal{A} \cap \{U. U \cap V \neq \{\}\}$) $\implies$ finite ($\mathcal{B} \cap \{U. U \cap V \neq \{\}\}$) **for** V

by (meson $\langle \mathcal{B} \subseteq \mathcal{A} \rangle$ finite_subset inf_le1 inf_le2 le_inf_iff subset_trans)

then show ?thesis

using assms **unfolding** locally_finite_in_def Int_def **by** fastforce

qed

lemma locally_finite_in_refinement:

assumes $\mathcal{A}$: locally_finite_in X $\mathcal{A}$ **and** f: $\bigwedge S. S \in \mathcal{A} \implies f S \subseteq S$

shows locally_finite_in X ($f \text{ ` } \mathcal{A}$)

proof –

show ?thesis

unfolding locally_finite_in_def

proof safe

fix x

assume $x \in \text{topspace } X$

then obtain V **where** openin X V $x \in V$ finite { $U \in \mathcal{A}. U \cap V \neq \{\}$ }

using $\mathcal{A}$ **unfolding** locally_finite_in_def **by** blast

moreover have { $U \in \mathcal{A}. f U \cap V \neq \{\}$ } $\subseteq$ { $U \in \mathcal{A}. U \cap V \neq \{\}$ } **for** V

using f **by** blast

ultimately have finite { $U \in \mathcal{A}. f U \cap V \neq \{\}$ }

using finite_subset **by** blast

moreover have $f \text{ ` } \{U \in \mathcal{A}. f U \cap V \neq \{\}\} = \{U \in f \text{ ` } \mathcal{A}. U \cap V \neq \{\}\}$

by blast

ultimately have finite { $U \in f \text{ ` } \mathcal{A}. U \cap V \neq \{\}$ }

by (metis (no_types, lifting) finite_imageI)

then show $\exists V. \text{openin } X V \wedge x \in V \wedge \text{finite } \{U \in f \text{ ` } \mathcal{A}. U \cap V \neq \{\}\}$

using $\langle \text{openin } X V \rangle \langle x \in V \rangle$ **by** blast

next

show $\bigwedge x xa. \llbracket xa \in \mathcal{A}; x \in f xa \rrbracket \implies x \in \text{topspace } X$

```

    by (meson Sup_upper  $\mathcal{A}$  f locally_finite_in_def subset_iff)
  qed
qed

lemma locally_finite_in_subtopology:
  assumes  $\mathcal{A}$ : locally_finite_in  $X$   $\mathcal{A} \cup \mathcal{A} \subseteq S$ 
  shows locally_finite_in (subtopology  $X$   $S$ )  $\mathcal{A}$ 
  unfolding locally_finite_in_def
proof safe
  fix x
  assume x:  $x \in \text{topspace (subtopology } X \text{ } S)$ 
  then obtain  $V$  where openin  $X$   $V$   $x \in V$  and fin: finite  $\{U \in \mathcal{A}. U \cap V \neq \{\}\}$ 
  {}
  using  $\mathcal{A}$  unfolding locally_finite_in_def topspace_subtopology by blast
  show  $\exists V. \text{openin (subtopology } X \text{ } S) V \wedge x \in V \wedge \text{finite } \{U \in \mathcal{A}. U \cap V \neq \{\}\}$ 
  proof (intro exI conjI)
    show openin (subtopology  $X$   $S$ ) ( $S \cap V$ )
    by (simp add:  $\langle \text{openin } X \text{ } V \rangle$  openin_subtopology_Int2)
    have  $\{U \in \mathcal{A}. U \cap (S \cap V) \neq \{\}\} \subseteq \{U \in \mathcal{A}. U \cap V \neq \{\}\}$ 
    by auto
    with fin show finite  $\{U \in \mathcal{A}. U \cap (S \cap V) \neq \{\}\}$ 
    using finite_subset by auto
    show  $x \in S \cap V$ 
    using x  $\langle x \in V \rangle$  by (simp)
  qed
next
  show  $\bigwedge x A. \llbracket x \in A; A \in \mathcal{A} \rrbracket \implies x \in \text{topspace (subtopology } X \text{ } S)$ 
  using assms unfolding locally_finite_in_def topspace_subtopology by blast
qed

lemma closedin_locally_finite_Union:
  assumes clo:  $\bigwedge S. S \in \mathcal{A} \implies \text{closedin } X \text{ } S$  and  $\mathcal{A}$ : locally_finite_in  $X$   $\mathcal{A}$ 
  shows closedin  $X$  ( $\bigcup \mathcal{A}$ )
  using  $\mathcal{A}$  unfolding locally_finite_in_def closedin_def
proof clarify
  show openin  $X$  ( $\text{topspace } X - \bigcup \mathcal{A}$ )
  proof (subst openin_subopen, clarify)
    fix x
    assume x  $\in \text{topspace } X$  and  $x \notin \bigcup \mathcal{A}$ 
    then obtain  $V$  where openin  $X$   $V$   $x \in V$  and fin: finite  $\{U \in \mathcal{A}. U \cap V \neq \{\}\}$ 
    {}
    using  $\mathcal{A}$  unfolding locally_finite_in_def by blast
    let  $?T = V - \bigcup \{S \in \mathcal{A}. S \cap V \neq \{\}\}$ 
    show  $\exists T. \text{openin } X \text{ } T \wedge x \in T \wedge T \subseteq \text{topspace } X - \bigcup \mathcal{A}$ 
    proof (intro exI conjI)
      show openin  $X$   $?T$ 
      by (metis (no_types, lifting) fin  $\langle \text{openin } X \text{ } V \rangle$  clo closedin_Union mem_Collect_eq openin_diff)
    qed
  qed

```

```

    show  $x \in ?T$ 
      using  $\langle x \notin \bigcup \mathcal{A} \rangle \langle x \in V \rangle$  by auto
    show  $?T \subseteq \text{topspace } X - \bigcup \mathcal{A}$ 
      using  $\langle \text{openin } X \ V \rangle \text{openin\_subset}$  by auto
  qed
qed
qed

lemma locally_finite_in_closure:
  assumes  $\mathcal{A}$ : locally_finite_in  $X \ \mathcal{A}$ 
  shows locally_finite_in  $X \ ((\lambda S. X \text{ closure\_of } S) \ ' \mathcal{A})$ 
  using  $\mathcal{A}$  unfolding locally_finite_in_def
proof (intro conjI; clarsimp)
  fix  $x \ A$ 
  assume  $x \in X \text{ closure\_of } A$ 
  then show  $x \in \text{topspace } X$ 
    by (meson in_closure_of)
next
  fix  $x$ 
  assume  $x \in \text{topspace } X$  and  $\bigcup \mathcal{A} \subseteq \text{topspace } X$ 
  then obtain  $V$  where  $V$ : openin  $X \ V \ x \in V$  and fin: finite  $\{U \in \mathcal{A}. U \cap V \neq \{\}\}$ 
  using  $\mathcal{A}$  unfolding locally_finite_in_def by blast
  have eq:  $\{y \in f \ ' \mathcal{A}. Q \ y\} = f \ ' \{x. x \in \mathcal{A} \wedge Q(f \ x)\}$  for  $f \ Q$ 
    by blast
  have eq2:  $\{A \in \mathcal{A}. X \text{ closure\_of } A \cap V \neq \{\}\} = \{A \in \mathcal{A}. A \cap V \neq \{\}\}$ 
    using openin_Int_closure_of_eq_empty  $V$  by blast
  have finite  $\{U \in (\text{closure\_of}) \ X \ ' \mathcal{A}. U \cap V \neq \{\}\}$ 
    by (simp add: eq eq2 fin)
  with  $V$  show  $\exists V. \text{openin } X \ V \wedge x \in V \wedge \text{finite } \{U \in (\text{closure\_of}) \ X \ ' \mathcal{A}. U \cap V \neq \{\}\}$ 
    by blast
qed

```

```

lemma closedin_Union_locally_finite_closure:
  locally_finite_in  $X \ \mathcal{A} \implies \text{closedin } X \ (\bigcup ((\lambda S. X \text{ closure\_of } S) \ ' \mathcal{A}))$ 
  by (metis (mono_tags) closedin_closure_of closedin_locally_finite_Union imageE locally_finite_in_closure)

```

```

lemma closure_of_Union_subset:  $\bigcup ((\lambda S. X \text{ closure\_of } S) \ ' \mathcal{A}) \subseteq X \text{ closure\_of } (\bigcup \mathcal{A})$ 
  by clarify (meson Union_upper closure_of_mono subsetD)

```

```

lemma closure_of_locally_finite_Union:
  assumes locally_finite_in  $X \ \mathcal{A}$ 
  shows  $X \text{ closure\_of } (\bigcup \mathcal{A}) = \bigcup ((\lambda S. X \text{ closure\_of } S) \ ' \mathcal{A})$ 
proof (rule closure_of_unique)
  show  $\bigcup \mathcal{A} \subseteq \bigcup ((\text{closure\_of}) \ X \ ' \mathcal{A})$ 
    using assms by (simp add: SUP_upper2 Sup_le_iff closure_of_subset lo-

```

```

cally_finite_in_def)
show closedin X (⋃ ((closure_of) X 'A))
  using assms by (simp add: closedin_Union_locally_finite_closure)
show  $\bigwedge T'. \llbracket \bigcup \mathcal{A} \subseteq T'; \text{closedin } X T' \rrbracket \implies \bigcup ((\text{closure\_of}) X ' \mathcal{A}) \subseteq T'$ 
  by (simp add: Sup_le_iff closure_of_minimal)
qed

```

2.2.10 Continuous maps

We will need to deal with continuous maps in terms of topologies and not in terms of type classes, as defined below.

definition *continuous_map* **where**

```

continuous_map X Y f  $\equiv$ 
  ( $\forall x \in \text{topspace } X. f\ x \in \text{topspace } Y$ )  $\wedge$ 
  ( $\forall U. \text{openin } Y\ U \longrightarrow \text{openin } X\ \{x \in \text{topspace } X. f\ x \in U\}$ )

```

lemma *continuous_map*:

```

continuous_map X Y f  $\longleftrightarrow$ 
   $f ' (\text{topspace } X) \subseteq \text{topspace } Y \wedge (\forall U. \text{openin } Y\ U \longrightarrow \text{openin } X\ \{x \in \text{topspace } X. f\ x \in U\})$ 
  by (auto simp: continuous_map_def)

```

lemma *continuous_map_image_subset_topspace*:

```

continuous_map X Y f  $\implies f ' (\text{topspace } X) \subseteq \text{topspace } Y$ 
  by (auto simp: continuous_map_def)

```

lemma *continuous_map_on_empty*: $\text{topspace } X = \{\}$ $\implies \text{continuous\_map } X\ Y\ f$

```

  by (auto simp: continuous_map_def)

```

lemma *continuous_map_closedin*:

```

continuous_map X Y f  $\longleftrightarrow$ 
  ( $\forall x \in \text{topspace } X. f\ x \in \text{topspace } Y$ )  $\wedge$ 
  ( $\forall C. \text{closedin } Y\ C \longrightarrow \text{closedin } X\ \{x \in \text{topspace } X. f\ x \in C\}$ )

```

proof –

```

  have ( $\forall U. \text{openin } Y\ U \longrightarrow \text{openin } X\ \{x \in \text{topspace } X. f\ x \in U\}$ ) =
    ( $\forall C. \text{closedin } Y\ C \longrightarrow \text{closedin } X\ \{x \in \text{topspace } X. f\ x \in C\}$ )

```

```

  if  $\bigwedge x. x \in \text{topspace } X \implies f\ x \in \text{topspace } Y$ 

```

proof –

```

  have eq:  $\{x \in \text{topspace } X. f\ x \in \text{topspace } Y \wedge f\ x \notin C\} = (\text{topspace } X - \{x \in \text{topspace } X. f\ x \in C\})$  for  $C$ 

```

```

  using that by blast

```

```

  show ?thesis

```

```

  proof (intro iffI allI impI)

```

```

    fix  $C$ 

```

```

    assume  $\forall U. \text{openin } Y\ U \longrightarrow \text{openin } X\ \{x \in \text{topspace } X. f\ x \in U\}$  and
     $\text{closedin } Y\ C$ 

```

```

    then have  $\text{openin } X\ \{x \in \text{topspace } X. f\ x \in \text{topspace } Y - C\}$  by blast

```

```

    then show  $\text{closedin } X\ \{x \in \text{topspace } X. f\ x \in C\}$ 

```



```

      by (auto simp add: closedin_def eq)
    next
      fix U
      assume  $\forall C. \text{closedin } Y \ C \longrightarrow \text{closedin } X \ \{x \in \text{topspace } X. f \ x \in C\}$  and
    openin Y U
      then have  $\text{closedin } X \ \{x \in \text{topspace } X. f \ x \in \text{topspace } Y - U\}$  by blast
      then show  $\text{openin } X \ \{x \in \text{topspace } X. f \ x \in U\}$ 
        by (auto simp add: openin_closedin_eq eq)
      qed
    qed
  then show ?thesis
    by (auto simp: continuous_map_def)
  qed

```

```

lemma openin_continuous_map_preimage:
   $\llbracket \text{continuous\_map } X \ Y \ f; \text{openin } Y \ U \rrbracket \Longrightarrow \text{openin } X \ \{x \in \text{topspace } X. f \ x \in U\}$ 
  by (simp add: continuous_map_def)

```

```

lemma closedin_continuous_map_preimage:
   $\llbracket \text{continuous\_map } X \ Y \ f; \text{closedin } Y \ C \rrbracket \Longrightarrow \text{closedin } X \ \{x \in \text{topspace } X. f \ x \in C\}$ 
  by (simp add: continuous_map_def)

```

```

lemma openin_continuous_map_preimage_gen:
  assumes  $\text{continuous\_map } X \ Y \ f \text{ openin } X \ U \text{ openin } Y \ V$ 
  shows  $\text{openin } X \ \{x \in U. f \ x \in V\}$ 
proof -
  have eq:  $\{x \in U. f \ x \in V\} = U \cap \{x \in \text{topspace } X. f \ x \in V\}$ 
    using assms(2) openin_closedin_eq by fastforce
  show ?thesis
    unfolding eq
    using assms openin_continuous_map_preimage by fastforce
qed

```

```

lemma closedin_continuous_map_preimage_gen:
  assumes  $\text{continuous\_map } X \ Y \ f \text{ closedin } X \ U \text{ closedin } Y \ V$ 
  shows  $\text{closedin } X \ \{x \in U. f \ x \in V\}$ 
proof -
  have eq:  $\{x \in U. f \ x \in V\} = U \cap \{x \in \text{topspace } X. f \ x \in V\}$ 
    using assms(2) closedin_def by fastforce
  show ?thesis
    unfolding eq
    using assms closedin_continuous_map_preimage by fastforce
qed

```

```

lemma continuous_map_image_closure_subset:
  assumes  $\text{continuous\_map } X \ Y \ f$ 
  shows  $f \ ' (X \ \text{closure\_of } S) \subseteq Y \ \text{closure\_of } f \ ' S$ 
proof -

```

```

have *: f ' (topspace X)  $\subseteq$  topspace Y
  by (meson assms continuous_map)
have X_closure_of T  $\subseteq$  {x  $\in$  X_closure_of T. f x  $\in$  Y_closure_of (f ' T)} if T
 $\subseteq$  topspace X for T
proof (rule closure_of_minimal)
  show T  $\subseteq$  {x  $\in$  X_closure_of T. f x  $\in$  Y_closure_of f ' T}
    using closure_of_subset * that by (fastforce simp: in_closure_of)
next
  show closedin X {x  $\in$  X_closure_of T. f x  $\in$  Y_closure_of f ' T}
    using assms closedin_continuous_map_preimage_gen by fastforce
qed
then have f ' (X_closure_of (topspace X  $\cap$  S))  $\subseteq$  Y_closure_of (f ' (topspace X
 $\cap$  S))
  by blast
also have ...  $\subseteq$  Y_closure_of (topspace Y  $\cap$  f ' S)
  using * by (blast intro!: closure_of_mono)
finally have f ' (X_closure_of (topspace X  $\cap$  S))  $\subseteq$  Y_closure_of (topspace Y
 $\cap$  f ' S) .
  then show ?thesis
    by (metis closure_of_restrict)
qed

```

```

lemma continuous_map_subset_aux1: continuous_map X Y f  $\implies$ 
  ( $\forall$  S. f ' (X_closure_of S)  $\subseteq$  Y_closure_of f ' S)
  using continuous_map_image_closure_subset by blast

```

```

lemma continuous_map_subset_aux2:
  assumes  $\forall$  S. S  $\subseteq$  topspace X  $\implies$  f ' (X_closure_of S)  $\subseteq$  Y_closure_of f ' S
  shows continuous_map X Y f
  unfolding continuous_map_closedin
proof (intro conjI ballI allI impI)
  fix x
  assume x  $\in$  topspace X
  then show f x  $\in$  topspace Y
    using assms closure_of_subset_topspace by fastforce
next
  fix C
  assume closedin Y C
  then show closedin X {x  $\in$  topspace X. f x  $\in$  C}
  proof (clarsimp simp flip: closure_of_subset_eq, intro conjI)
    fix x
    assume x: x  $\in$  X_closure_of {x  $\in$  topspace X. f x  $\in$  C}
      and C  $\subseteq$  topspace Y and Y_closure_of C  $\subseteq$  C
    show x  $\in$  topspace X
      by (meson x in_closure_of)
    have {a  $\in$  topspace X. f a  $\in$  C}  $\subseteq$  topspace X
      by simp
    moreover have Y_closure_of f ' {a  $\in$  topspace X. f a  $\in$  C}  $\subseteq$  C
      by (simp add:  $\langle$ closedin Y C $\rangle$  closure_of_minimal_image_subset_iff)

```

```

ultimately have  $f' (X \text{ closure\_of } \{a \in \text{topspace } X. f a \in C\}) \subseteq C$ 
  using assms by blast
then show  $f x \in C$ 
  using x by auto
qed
qed

lemma continuous_map_eq_image_closure_subset:
  continuous_map  $X Y f \longleftrightarrow (\forall S. f' (X \text{ closure\_of } S) \subseteq Y \text{ closure\_of } f' S)$ 
  using continuous_map_subset_aux1 continuous_map_subset_aux2 by metis

lemma continuous_map_eq_image_closure_subset_alt:
  continuous_map  $X Y f \longleftrightarrow (\forall S. S \subseteq \text{topspace } X \longrightarrow f' (X \text{ closure\_of } S) \subseteq Y \text{ closure\_of } f' S)$ 
  using continuous_map_subset_aux1 continuous_map_subset_aux2 by metis

lemma continuous_map_eq_image_closure_subset_gen:
  continuous_map  $X Y f \longleftrightarrow$ 
     $f' (\text{topspace } X) \subseteq \text{topspace } Y \wedge$ 
     $(\forall S. f' (X \text{ closure\_of } S) \subseteq Y \text{ closure\_of } f' S)$ 
  using continuous_map_subset_aux1 continuous_map_subset_aux2 continuous_map_image_subset_topspace
  by metis

lemma continuous_map_closure_preimage_subset:
  continuous_map  $X Y f$ 
     $\implies X \text{ closure\_of } \{x \in \text{topspace } X. f x \in T\}$ 
     $\subseteq \{x \in \text{topspace } X. f x \in Y \text{ closure\_of } T\}$ 
  unfolding continuous_map_closedin
  by (rule closure_of_minimal) (use in_closure_of in ‹fastforce+›)

lemma continuous_map_frontier_frontier_preimage_subset:
  assumes continuous_map  $X Y f$ 
  shows  $X \text{ frontier\_of } \{x \in \text{topspace } X. f x \in T\} \subseteq \{x \in \text{topspace } X. f x \in Y \text{ frontier\_of } T\}$ 
  proof -
    have eq:  $\text{topspace } X - \{x \in \text{topspace } X. f x \in T\} = \{x \in \text{topspace } X. f x \in \text{topspace } Y - T\}$ 
    using assms unfolding continuous_map_def by blast
    have  $X \text{ closure\_of } \{x \in \text{topspace } X. f x \in T\} \subseteq \{x \in \text{topspace } X. f x \in Y \text{ closure\_of } T\}$ 
    by (simp add: assms continuous_map_closure_preimage_subset)
    moreover
    have  $X \text{ closure\_of } (\text{topspace } X - \{x \in \text{topspace } X. f x \in T\}) \subseteq \{x \in \text{topspace } X. f x \in Y \text{ closure\_of } (\text{topspace } Y - T)\}$ 
    using continuous_map_closure_preimage_subset [OF assms] eq by presburger
    ultimately show ?thesis
    by (auto simp: frontier_of_closures)
  qed

```

```

lemma topology_finer_continuous_id:
  assumes topspace X = topspace Y
  shows  $(\forall S. \text{openin } X \ S \longrightarrow \text{openin } Y \ S) \longleftrightarrow \text{continuous\_map } Y \ X \ \text{id}$  (is ?lhs
= ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
    unfolding continuous_map_def
    using assms openin_subopen openin_subset by fastforce
  show ?rhs  $\implies$  ?lhs
    unfolding continuous_map_def
    using assms openin_subopen topspace_def by fastforce
qed

```

```

lemma continuous_map_const [simp]:
  continuous_map X Y  $(\lambda x. C) \longleftrightarrow \text{topspace } X = \{\}$   $\vee C \in \text{topspace } Y$ 
proof (cases topspace X = \{\})
  case False
    show ?thesis
    proof (cases C \in topspace Y)
      case True
        with openin_subopen show ?thesis
        by (auto simp: continuous_map_def)
      next
        case False
        then show ?thesis
        unfolding continuous_map_def by fastforce
    qed
  qed (auto simp: continuous_map_on_empty)

```

```

declare continuous_map_const [THEN iffD2, continuous_intros]

```

```

lemma continuous_map_compose [continuous_intros]:
  assumes f: continuous_map X X' f and g: continuous_map X' X'' g
  shows continuous_map X X'' (g \circ f)
  unfolding continuous_map_def
proof (intro conjI ballI allI impI)
  fix x
  assume x \in topspace X
  then show  $(g \circ f) \ x \in \text{topspace } X''$ 
    using assms unfolding continuous_map_def by force
next
  fix U
  assume openin X'' U
  have eq: \{x \in topspace X. (g \circ f) \ x \in U\} = \{x \in topspace X. f \ x \in \{y. y \in
topspace X' \wedge g \ y \in U\}\}
    by auto (meson f continuous_map_def)
  show openin X \{x \in topspace X. (g \circ f) \ x \in U\}
    unfolding eq

```

```

    using assms unfolding continuous_map_def
    using ‹openin X'' U› by blast
qed

```

```

lemma continuous_map_eq:
  assumes continuous_map X X' f and  $\bigwedge x. x \in \text{topspace } X \implies f x = g x$  shows
    continuous_map X X' g
proof -
  have eq:  $\{x \in \text{topspace } X. f x \in U\} = \{x \in \text{topspace } X. g x \in U\}$  for U
    using assms by auto
  show ?thesis
    using assms by (simp add: continuous_map_def eq)
qed

```

```

lemma restrict_continuous_map [simp]:
   $\text{topspace } X \subseteq S \implies \text{continuous\_map } X X' (\text{restrict } f S) \longleftrightarrow \text{continuous\_map } X X' f$ 
  by (auto simp: elim!: continuous_map_eq)

```

```

lemma continuous_map_in_subtopology:
  continuous_map X (subtopology X' S) f  $\longleftrightarrow$  continuous_map X X' f  $\wedge$  f '
  ( $\text{topspace } X \subseteq S$ )
  (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof -
    have  $\bigwedge A. f ' (X \text{ closure\_of } A) \subseteq \text{subtopology } X' S \text{ closure\_of } f ' A$ 
      by (meson L continuous_map_image_closure_subset)
    then show ?thesis
      by (metis (no_types) closure_of_subset_subtopology closure_of_subtopology_subset
        closure_of_topospace continuous_map_eq image_closure_subset dual_order.trans)
  qed
next
  assume R: ?rhs
  then have eq:  $\{x \in \text{topspace } X. f x \in U\} = \{x \in \text{topspace } X. f x \in U \wedge f x \in S\}$  for U
    by auto
  show ?lhs
    using R
    unfolding continuous_map
    by (auto simp: openin_subtopology eq)
qed

```

```

lemma continuous_map_from_subtopology:
  continuous_map X X' f  $\implies$  continuous_map (subtopology X S) X' f
  by (auto simp: continuous_map openin_subtopology)

```

lemma *continuous_map_into_fulltopology*:

$\text{continuous\_map } X \text{ (subtopology } X' T) f \implies \text{continuous\_map } X X' f$
by (*auto simp: continuous_map_in_subtopology*)

lemma *continuous_map_into_subtopology*:

$\llbracket \text{continuous\_map } X X' f; f ' \text{topspace } X \subseteq T \rrbracket \implies \text{continuous\_map } X \text{ (subtopology } X' T) f$
by (*auto simp: continuous_map_in_subtopology*)

lemma *continuous_map_from_subtopology_mono*:

$\llbracket \text{continuous\_map (subtopology } X T) X' f; S \subseteq T \rrbracket$
 $\implies \text{continuous\_map (subtopology } X S) X' f$

by (*metis inf.absorb_iff2 continuous_map_from_subtopology subtopology_subtopology*)

lemma *continuous_map_from_discrete_topology [simp]*:

$\text{continuous\_map (discrete\_topology } U) X f \longleftrightarrow f ' U \subseteq \text{topspace } X$
by (*auto simp: continuous_map_def*)

lemma *continuous_map_iff_continuous [simp]*: $\text{continuous\_map (top\_of\_set } S) \text{ euclidean } g = \text{continuous\_on } S g$

by (*fastforce simp add: continuous_map_openin_subtopology continuous_on_open_invariant*)

lemma *continuous_map_iff_continuous2 [simp]*: $\text{continuous\_map euclidean euclidean } g = \text{continuous\_on UNIV } g$

by (*metis continuous_map_iff_continuous subtopology_UNIV*)

lemma *continuous_map_openin_preimage_eq*:

$\text{continuous\_map } X Y f \longleftrightarrow$
 $f ' (\text{topspace } X) \subseteq \text{topspace } Y \wedge (\forall U. \text{openin } Y U \longrightarrow \text{openin } X (\text{topspace } X \cap f - ' U))$
by (*auto simp: continuous_map_def vimage_def Int_def*)

lemma *continuous_map_closedin_preimage_eq*:

$\text{continuous\_map } X Y f \longleftrightarrow$
 $f ' (\text{topspace } X) \subseteq \text{topspace } Y \wedge (\forall U. \text{closedin } Y U \longrightarrow \text{closedin } X (\text{topspace } X \cap f - ' U))$
by (*auto simp: continuous_map_closedin vimage_def Int_def*)

lemma *continuous_map_square_root*: $\text{continuous\_map euclideanreal euclideanreal } \text{sqrt}$

by (*simp add: continuous_at_imp_continuous_on isCont_real_sqrt*)

lemma *continuous_map_sqrt [continuous_intros]*:

$\text{continuous\_map } X \text{ euclideanreal } f \implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. \text{sqrt}(f x))$

by (*meson continuous_map_compose continuous_map_eq continuous_map_square_root o_apply*)

lemma *continuous_map_id [simp, continuous_intros]*: $\text{continuous\_map } X X \text{ id}$

unfolding *continuous_map_def* **using** *openin_subopen topspace_def* **by** *fastforce*

declare *continuous_map_id* [*unfolded id_def, simp, continuous_intros*]

lemma *continuous_map_id_subt* [*simp*]: *continuous_map* (*subtopology X S*) *X id*
by (*simp add: continuous_map_from_subtopology*)

declare *continuous_map_id_subt* [*unfolded id_def, simp*]

lemma *continuous_map_alt*:

continuous_map T1 T2 f
 $= ((\forall U. \text{openin } T2 \ U \longrightarrow \text{openin } T1 \ (f - ' U \cap \text{topspace } T1)) \wedge f - ' \text{topspace } T1 \subseteq \text{topspace } T2)$
by (*auto simp: continuous_map_def vimage_def image_def Collect_conj_eq inf_commute*)

lemma *continuous_map_open* [*intro*]:

continuous_map T1 T2 f $\Longrightarrow$ *openin T2 U* $\Longrightarrow$ *openin T1 (f - ' U \cap \text{topspace}(T1))*
unfolding *continuous_map_alt* **by** *auto*

lemma *continuous_map_preimage_topspace* [*intro*]:

assumes *continuous_map T1 T2 f*
shows $f - ' (\text{topspace } T2) \cap \text{topspace } T1 = \text{topspace } T1$
using *assms* **unfolding** *continuous_map_def* **by** *auto*

2.2.11 Open and closed maps (not a priori assumed continuous)

definition *open_map* :: '*a topology* $\Rightarrow$ '*b topology* $\Rightarrow$ ('*a* $\Rightarrow$ '*b*) $\Rightarrow$ bool

where *open_map X1 X2 f* $\equiv \forall U. \text{openin } X1 \ U \longrightarrow \text{openin } X2 \ (f - ' U)$

definition *closed_map* :: '*a topology* $\Rightarrow$ '*b topology* $\Rightarrow$ ('*a* $\Rightarrow$ '*b*) $\Rightarrow$ bool

where *closed_map X1 X2 f* $\equiv \forall U. \text{closedin } X1 \ U \longrightarrow \text{closedin } X2 \ (f - ' U)$

lemma *open_map_imp_subset_topspace*:

open_map X1 X2 f $\Longrightarrow f - ' (\text{topspace } X1) \subseteq \text{topspace } X2$
unfolding *open_map_def* **by** (*simp add: openin_subset*)

lemma *open_map_on_empty*:

topspace X = {} $\Longrightarrow$ *open_map X Y f*
by (*metis empty_iff imageE in_mono open_map_def openin_subopen openin_subset*)

lemma *closed_map_on_empty*:

topspace X = {} $\Longrightarrow$ *closed_map X Y f*
by (*simp add: closed_map_def closedin_topspace_empty*)

lemma *closed_map_const*:

```

      closed_map X Y (λx. c) <=> topspace X = {} ∨ closedin Y {c}
proof (cases topspace X = {})
  case True
    then show ?thesis
      by (simp add: closed_map_on_empty)
  next
    case False
    then show ?thesis
      by (auto simp: closed_map_def image_constant_conv)
qed

```

```

lemma open_map_imp_subset:
   $\llbracket \text{open\_map } X1\ X2\ f; S \subseteq \text{topspace } X1 \rrbracket \implies f' S \subseteq \text{topspace } X2$ 
  by (meson order_trans open_map_imp_subset_topspace_subset_image_iff)

```

```

lemma topology_finer_open_id:
   $(\forall S. \text{openin } X\ S \longrightarrow \text{openin } X'\ S) \longleftrightarrow \text{open\_map } X\ X'\ \text{id}$ 
  unfolding open_map_def by auto

```

```

lemma open_map_id: open_map X X id
  unfolding open_map_def by auto

```

```

lemma open_map_eq:
   $\llbracket \text{open\_map } X\ X'\ f; \bigwedge x. x \in \text{topspace } X \implies f\ x = g\ x \rrbracket \implies \text{open\_map } X\ X'\ g$ 
  unfolding open_map_def
  by (metis image_cong openin_subset_subset_iff)

```

```

lemma open_map_inclusion_eq:
   $\text{open\_map } (\text{subtopology } X\ S)\ X\ \text{id} \longleftrightarrow \text{openin } X\ (\text{topspace } X \cap S)$ 
proof -
  have *:  $\text{openin } X\ (T \cap S)$  if  $\text{openin } X\ (S \cap \text{topspace } X)$  openin X T for T
  proof -
    have  $T \subseteq \text{topspace } X$ 
    using that by (simp add: openin_subset)
    with that show  $\text{openin } X\ (T \cap S)$ 
    by (metis inf.absorb1 inf.left_commute inf_commute openin_Int)
  qed
  show ?thesis
    by (fastforce simp add: open_map_def Int_commute openin_subtopology_alt
intro: *)
qed

```

```

lemma open_map_inclusion:
   $\text{openin } X\ S \implies \text{open\_map } (\text{subtopology } X\ S)\ X\ \text{id}$ 
  by (simp add: open_map_inclusion_eq openin_Int)

```

```

lemma open_map_compose:
   $\llbracket \text{open\_map } X\ X'\ f; \text{open\_map } X'\ X''\ g \rrbracket \implies \text{open\_map } X\ X''\ (g \circ f)$ 
  by (metis (no_types, lifting) image_comp open_map_def)

```



```

lemma closed_map_imp_subset_topspace:
   $\text{closed\_map } X1\ X2\ f \implies f' (topspace\ X1) \subseteq topspace\ X2$ 
by (simp add: closed_map_def closedin_subset)

lemma closed_map_imp_subset:
   $\llbracket \text{closed\_map } X1\ X2\ f; S \subseteq topspace\ X1 \rrbracket \implies f' S \subseteq topspace\ X2$ 
using closed_map_imp_subset_topspace by blast

lemma topology_finer_closed_id:
   $(\forall S. \text{closedin } X\ S \longrightarrow \text{closedin } X'\ S) \longleftrightarrow \text{closed\_map } X\ X'\ id$ 
by (simp add: closed_map_def)

lemma closed_map_id:  $\text{closed\_map } X\ X\ id$ 
by (simp add: closed_map_def)

lemma closed_map_eq:
   $\llbracket \text{closed\_map } X\ X'\ f; \bigwedge x. x \in topspace\ X \implies f\ x = g\ x \rrbracket \implies \text{closed\_map } X\ X'\ g$ 
unfolding closed_map_def
by (metis image_cong closedin_subset subset_iff)

lemma closed_map_compose:
   $\llbracket \text{closed\_map } X\ X'\ f; \text{closed\_map } X'\ X''\ g \rrbracket \implies \text{closed\_map } X\ X''\ (g \circ f)$ 
by (metis (no_types, lifting) closed_map_def image_comp)

lemma closed_map_inclusion_eq:
   $\text{closed\_map } (\text{subtopology } X\ S)\ X\ id \longleftrightarrow \text{closedin } X\ (topspace\ X \cap S)$ 
proof -
  have *:  $\text{closedin } X\ (T \cap S)$  if  $\text{closedin } X\ (S \cap topspace\ X)$  closedin } X\ T for  $T$ 
proof -
  have  $T \subseteq topspace\ X$ 
  using that by (simp add: closedin_subset)
  with that show  $\text{closedin } X\ (T \cap S)$ 
  by (metis inf.absorb1 inf.left_commute inf_commute closedin_Int)
qed
show ?thesis
by (fastforce simp add: closed_map_def Int_commute closedin_subtopology_alt intro: *)
qed

lemma closed_map_inclusion:  $\text{closedin } X\ S \implies \text{closed\_map } (\text{subtopology } X\ S)\ X\ id$ 
by (simp add: closed_map_inclusion_eq closedin_Int)

lemma open_map_into_subtopology:
   $\llbracket \text{open\_map } X\ X'\ f; f' (topspace\ X) \subseteq S \rrbracket \implies \text{open\_map } X\ (\text{subtopology } X'\ S)\ f$ 
unfolding open_map_def openin_subtopology

```

using *openin_subset* **by** *fastforce*

lemma *closed_map_into_subtopology*:

$\llbracket \text{closed\_map } X \ X' \ f; f' \text{ ' } \text{topspace } X \subseteq S \rrbracket \implies \text{closed\_map } X \ (\text{subtopology } X' \ S) \ f$

unfolding *closed_map_def* *closedin_subtopology*

using *closedin_subset* **by** *fastforce*

lemma *open_map_into_discrete_topology*:

$\text{open\_map } X \ (\text{discrete\_topology } U) \ f \longleftrightarrow f' \text{ ' } (\text{topspace } X) \subseteq U$

unfolding *open_map_def* *openin_discrete_topology* **using** *openin_subset* **by** *blast*

lemma *closed_map_into_discrete_topology*:

$\text{closed\_map } X \ (\text{discrete\_topology } U) \ f \longleftrightarrow f' \text{ ' } (\text{topspace } X) \subseteq U$

unfolding *closed_map_def* *closedin_discrete_topology* **using** *closedin_subset* **by** *blast*

lemma *bijective_open_imp_closed_map*:

$\llbracket \text{open\_map } X \ X' \ f; f' \text{ ' } (\text{topspace } X) = \text{topspace } X'; \text{inj\_on } f \ (\text{topspace } X) \rrbracket \implies \text{closed\_map } X \ X' \ f$

unfolding *open_map_def* *closed_map_def* *closedin_def*

by *auto* (*metis* *Diff_subset inj_on_image_set_diff*)

lemma *bijective_closed_imp_open_map*:

$\llbracket \text{closed\_map } X \ X' \ f; f' \text{ ' } (\text{topspace } X) = \text{topspace } X'; \text{inj\_on } f \ (\text{topspace } X) \rrbracket \implies \text{open\_map } X \ X' \ f$

unfolding *closed_map_def* *open_map_def* *openin_closedin_eq*

by *auto* (*metis* *Diff_subset inj_on_image_set_diff*)

lemma *open_map_from_subtopology*:

$\llbracket \text{open\_map } X \ X' \ f; \text{openin } X \ U \rrbracket \implies \text{open\_map } (\text{subtopology } X \ U) \ X' \ f$

unfolding *open_map_def* *openin_subtopology_alt* **by** *blast*

lemma *closed_map_from_subtopology*:

$\llbracket \text{closed\_map } X \ X' \ f; \text{closedin } X \ U \rrbracket \implies \text{closed\_map } (\text{subtopology } X \ U) \ X' \ f$

unfolding *closed_map_def* *closedin_subtopology_alt* **by** *blast*

lemma *open_map_restriction*:

assumes *f*: $\text{open\_map } X \ X' \ f$ **and** *U*: $\{x \in \text{topspace } X. f \ x \in V\} = U$

shows $\text{open\_map } (\text{subtopology } X \ U) \ (\text{subtopology } X' \ V) \ f$

unfolding *open_map_def*

proof *clarsimp*

fix *W*

assume $\text{openin } (\text{subtopology } X \ U) \ W$

then obtain *T* **where** $\text{openin } X \ T \ W = T \cap U$

by (*meson* *openin_subtopology*)

with *f* *U* **have** $f' \text{ ' } W = (f' \text{ ' } T) \cap V$

unfolding *open_map_def* *openin_closedin_eq* **by** *auto*

```

    then show openin (subtopology X' V) (f ' W)
      by (metis ‹openin X T› f open_map_def openin_subtopology_Int)
qed

```

```

lemma closed_map_restriction:
  assumes f: closed_map X X' f and U: {x ∈ topspace X. f x ∈ V} = U
  shows closed_map (subtopology X U) (subtopology X' V) f
  unfolding closed_map_def
proof clarsimp
  fix W
  assume closedin (subtopology X U) W
  then obtain T where closedin X T W = T ∩ U
    by (meson closedin_subtopology)
  with f U have f ' W = (f ' T) ∩ V
    unfolding closed_map_def closedin_def by auto
  then show closedin (subtopology X' V) (f ' W)
    by (metis ‹closedin X T› closed_map_def closedin_subtopology f)
qed

```

2.2.12 Quotient maps

```

definition quotient_map where
  quotient_map X X' f ‹↔›
    f ' (topspace X) = topspace X' ∧
    (∀ U. U ⊆ topspace X' ‹→› (openin X {x. x ∈ topspace X ∧ f x ∈ U} ‹↔›
openin X' U))

```

```

lemma quotient_map_eq:
  assumes quotient_map X X' f ∧ x. x ∈ topspace X ‹⇒› f x = g x
  shows quotient_map X X' g
proof -
  have eq: {x ∈ topspace X. f x ∈ U} = {x ∈ topspace X. g x ∈ U} for U
    using assms by auto
  show ?thesis
    using assms
    unfolding quotient_map_def
    by (metis (mono_tags, lifting) eq image_cong)
qed

```

```

lemma quotient_map_compose:
  assumes f: quotient_map X X' f and g: quotient_map X' X'' g
  shows quotient_map X X'' (g ∘ f)
  unfolding quotient_map_def
proof (intro conjI allI impI)
  show (g ∘ f) ' topspace X = topspace X''
    using assms by (simp only: image_comp [symmetric]) (simp add: quotient_map_def)
next
  fix U''
  assume U'' ⊆ topspace X''

```

```

define  $U'$  where  $U' \equiv \{y \in \text{topspace } X'. g \ y \in U''\}$ 
have  $U' \subseteq \text{topspace } X'$ 
  by (auto simp add:  $U'_\text{def}$ )
then have  $U': \text{openin } X \ \{x \in \text{topspace } X. f \ x \in U'\} = \text{openin } X' \ U'$ 
  using assms unfolding quotient_map_def by simp
have eq:  $\{x \in \text{topspace } X. f \ x \in \text{topspace } X' \wedge g \ (f \ x) \in U''\} = \{x \in \text{topspace } X. (g \circ f) \ x \in U''\}$ 
  using f quotient_map_def by fastforce
have  $\text{openin } X \ \{x \in \text{topspace } X. (g \circ f) \ x \in U''\} = \text{openin } X \ \{x \in \text{topspace } X. f \ x \in U'\}$ 
  using assms by (simp add: quotient_map_def  $U'_\text{def}$  eq)
also have  $\dots = \text{openin } X'' \ U''$ 
  using  $U'_\text{def}$   $\langle U'' \subseteq \text{topspace } X'' \rangle$   $U' \ g$  quotient_map_def by fastforce
finally show  $\text{openin } X \ \{x \in \text{topspace } X. (g \circ f) \ x \in U''\} = \text{openin } X'' \ U''$ .
qed

```

```

lemma quotient_map_from_composition:
  assumes  $f: \text{continuous\_map } X \ X' \ f$  and  $g: \text{continuous\_map } X' \ X'' \ g$  and  $gf: \text{quotient\_map } X \ X'' \ (g \circ f)$ 
  shows  $\text{quotient\_map } X' \ X'' \ g$ 
  unfolding quotient_map_def
proof (intro conjI allI impI)
  show  $g \text{ 'topspace } X' = \text{topspace } X''$ 
    using assms unfolding continuous_map_def quotient_map_def by fastforce
next
  fix  $U'' :: 'c \text{ set}$ 
  assume  $U'': U'' \subseteq \text{topspace } X''$ 
  have eq:  $\{x \in \text{topspace } X. g \ (f \ x) \in U''\} = \{x \in \text{topspace } X. f \ x \in \{y. y \in \text{topspace } X' \wedge g \ y \in U''\}\}$ 
    using continuous_map_def f by fastforce
  show  $\text{openin } X' \ \{x \in \text{topspace } X'. g \ x \in U''\} = \text{openin } X'' \ U''$ 
    using assms unfolding continuous_map_def quotient_map_def
    by (metis (mono_tags, lifting) Collect_cong  $U'' \text{ comp\_apply } eq$ )
qed

```

```

lemma quotient_imp_continuous_map:
   $\text{quotient\_map } X \ X' \ f \implies \text{continuous\_map } X \ X' \ f$ 
  by (simp add: continuous_map openin_subset quotient_map_def)

```

```

lemma quotient_imp_surjective_map:
   $\text{quotient\_map } X \ X' \ f \implies f \text{ 'topspace } X = \text{topspace } X'$ 
  by (simp add: quotient_map_def)

```

```

lemma quotient_map_closedin:
   $\text{quotient\_map } X \ X' \ f \longleftrightarrow$ 
   $f \text{ 'topspace } X = \text{topspace } X' \wedge$ 
   $(\forall U. U \subseteq \text{topspace } X' \longrightarrow (\text{closedin } X \ \{x. x \in \text{topspace } X \wedge f \ x \in U\} \longleftrightarrow \text{closedin } X' \ U))$ 
proof –

```

```

have eq: (topspace X - {x ∈ topspace X. f x ∈ U}) = {x ∈ topspace X. f x ∈
topspace X' ∧ f x ∉ U}
  if f ' topspace X = topspace X' U' ⊆ topspace X' for U'
    using that by auto
have (∀ U ⊆ topspace X'. openin X {x ∈ topspace X. f x ∈ U} = openin X' U)
=
  (∀ U ⊆ topspace X'. closedin X {x ∈ topspace X. f x ∈ U} = closedin X'
U)
  if f ' topspace X = topspace X'
proof (rule iffI; intro allI impI subsetI)
  fix U'
    assume *[rule_format]: ∀ U ⊆ topspace X'. openin X {x ∈ topspace X. f x ∈
U} = openin X' U
    and U': U' ⊆ topspace X'
    show closedin X {x ∈ topspace X. f x ∈ U'} = closedin X' U'
      using U' by (auto simp add: closedin_def simp flip: * [of topspace X' - U']
eq [OF that])
    next
      fix U' :: 'b set
      assume *[rule_format]: ∀ U ⊆ topspace X'. closedin X {x ∈ topspace X. f x ∈
U} = closedin X' U
      and U': U' ⊆ topspace X'
      show openin X {x ∈ topspace X. f x ∈ U'} = openin X' U'
        using U' by (auto simp add: openin_closedin_eq simp flip: * [of topspace
X' - U'] eq [OF that])
      qed
    then show ?thesis
      unfolding quotient_map_def by force
  qed

lemma continuous_open_imp_quotient_map:
  assumes continuous_map X X' f and om: open_map X X' f and feq: f '
(topspace X) = topspace X'
  shows quotient_map X X' f
proof -
  { fix U
    assume U: U ⊆ topspace X' and openin X {x ∈ topspace X. f x ∈ U}
    then have ope: openin X' (f ' {x ∈ topspace X. f x ∈ U})
      using om unfolding open_map_def by blast
    then have openin X' U
      using U feq by (subst openin_subopen) force
    }
  moreover have openin X {x ∈ topspace X. f x ∈ U} if U ⊆ topspace X' and
openin X' U for U
    using that assms unfolding continuous_map_def by blast
  ultimately show ?thesis
    unfolding quotient_map_def using assms by blast
  qed

```

lemma *continuous_closed_imp_quotient_map*:
assumes *continuous_map* $X\ X'\ f$ **and** *om*: *closed_map* $X\ X'\ f$ **and** *feq*: $f\ '\ (topspace\ X) = topspace\ X'$
shows *quotient_map* $X\ X'\ f$
proof –
have $f\ '\ \{x \in topspace\ X. f\ x \in U\} = U$ **if** $U \subseteq topspace\ X'$ **for** U
using *that feq* **by** *auto*
with *assms* **show** *?thesis*
unfolding *quotient_map_closedin* *closed_map_def* *continuous_map_closedin*
by *auto*
qed

lemma *continuous_open_quotient_map*:
 $\llbracket continuous\_map\ X\ X'\ f; open\_map\ X\ X'\ f \rrbracket \implies quotient\_map\ X\ X'\ f \longleftrightarrow f\ '\ (topspace\ X) = topspace\ X'$
by (*meson continuous_open_imp_quotient_map quotient_map_def*)

lemma *continuous_closed_quotient_map*:
 $\llbracket continuous\_map\ X\ X'\ f; closed\_map\ X\ X'\ f \rrbracket \implies quotient\_map\ X\ X'\ f \longleftrightarrow f\ '\ (topspace\ X) = topspace\ X'$
by (*meson continuous_closed_imp_quotient_map quotient_map_def*)

lemma *injective_quotient_map*:
assumes *inj_on* $f\ (topspace\ X)$
shows *quotient_map* $X\ X'\ f \longleftrightarrow$
 $continuous\_map\ X\ X'\ f \wedge open\_map\ X\ X'\ f \wedge closed\_map\ X\ X'\ f \wedge f\ '\ (topspace\ X) = topspace\ X'$
(is ?lhs = ?rhs)

proof
assume $L: ?lhs$
have *open_map* $X\ X'\ f$
proof (*clarsimp simp add: open_map_def*)
fix U
assume *openin* $X\ U$
then have $U \subseteq topspace\ X$
by (*simp add: openin_subset*)
moreover have $\{x \in topspace\ X. f\ x \in f\ '\ U\} = U$
using $\langle U \subseteq topspace\ X \rangle$ *assms inj_onD* **by** *fastforce*
ultimately show *openin* $X'\ (f\ '\ U)$
using L **unfolding** *quotient_map_def*
by (*metis (no_types, lifting) Collect_cong openin_X_U image_mono*)
qed
moreover have *closed_map* $X\ X'\ f$
proof (*clarsimp simp add: closed_map_def*)
fix U
assume *closedin* $X\ U$
then have $U \subseteq topspace\ X$
by (*simp add: closedin_subset*)
moreover have $\{x \in topspace\ X. f\ x \in f\ '\ U\} = U$

```

    using  $\langle U \subseteq \text{topspace } X \rangle$  assms inj_onD by fastforce
  ultimately show closedin  $X'$  ( $f \restriction U$ )
    using L unfolding quotient_map_closedin
    by (metis (no_types, lifting) Collect_cong  $\langle \text{closedin } X \ U \rangle$  image_mono)
qed
ultimately show ?rhs
  using L by (simp add: quotient_imp_continuous_map quotient_imp_surjective_map)
next
  assume ?rhs
  then show ?lhs
    by (simp add: continuous_closed_imp_quotient_map)
qed

```

```

lemma continuous_compose_quotient_map:
  assumes  $f$ : quotient_map  $X \ X'$   $f$  and  $g$ : continuous_map  $X \ X''$  ( $g \circ f$ )
  shows continuous_map  $X' \ X''$   $g$ 
  unfolding quotient_map_def continuous_map_def
proof (intro conjI ballI allI impI)
  show  $\bigwedge x'. x' \in \text{topspace } X' \implies g \ x' \in \text{topspace } X''$ 
    using assms unfolding quotient_map_def
    by (metis (no_types, opaque_lifting) continuous_map_image_subset_topspace
  image_comp image_subset_iff)
next
  fix  $U'' :: 'c \text{ set}$ 
  assume  $U''$ : openin  $X''$   $U''$ 
  have  $f \restriction \text{topspace } X = \text{topspace } X'$ 
    by (simp add: f quotient_imp_surjective_map)
  then have eq:  $\{x \in \text{topspace } X. f \ x \in \text{topspace } X' \wedge g \ (f \ x) \in U\} = \{x \in \text{topspace } X. g \ (f \ x) \in U\}$  for  $U$ 
    by auto
  have openin  $X \ \{x \in \text{topspace } X. f \ x \in \text{topspace } X' \wedge g \ (f \ x) \in U''\}$ 
    unfolding eq using  $U''$  g openin_continuous_map_preimage by fastforce
  then have *: openin  $X \ \{x \in \text{topspace } X. f \ x \in \{x \in \text{topspace } X'. g \ x \in U''\}\}$ 
    by auto
  show openin  $X' \ \{x \in \text{topspace } X'. g \ x \in U''\}$ 
    using f unfolding quotient_map_def
    by (metis (no_types) Collect_subset *)
qed

```

```

lemma continuous_compose_quotient_map_eq:
  quotient_map  $X \ X'$   $f \implies$  continuous_map  $X \ X''$  ( $g \circ f$ )  $\longleftrightarrow$  continuous_map
 $X' \ X''$   $g$ 
  using continuous_compose_quotient_map continuous_map_compose quotient_imp_continuous_map
  by blast

```

```

lemma quotient_map_compose_eq:
  quotient_map  $X \ X'$   $f \implies$  quotient_map  $X \ X''$  ( $g \circ f$ )  $\longleftrightarrow$  quotient_map  $X'$ 
 $X''$   $g$ 
  by (meson continuous_compose_quotient_map_eq quotient_imp_continuous_map

```

quotient_map_compose quotient_map_from_composition)

lemma *quotient_map_restriction*:

assumes *quo*: *quotient_map* $X\ Y\ f$ **and** $U: \{x \in \text{topspace } X. f\ x \in V\} = U$ **and**
disj: *openin* $Y\ V \vee \text{closedin } Y\ V$

shows *quotient_map* (*subtopology* $X\ U$) (*subtopology* $Y\ V$) f

using *disj*

proof

assume $V: \text{openin } Y\ V$

with U **have** $sub: U \subseteq \text{topspace } X\ V \subseteq \text{topspace } Y$

by (*auto simp: openin_subset*)

have $fim: f \text{ 'topspace } X = \text{topspace } Y$

and $Y: \bigwedge U. U \subseteq \text{topspace } Y \implies \text{openin } X\ \{x \in \text{topspace } X. f\ x \in U\} = \text{openin } Y\ U$

using *quo* **unfolding** *quotient_map_def* **by** *auto*

have *openin* $X\ U$

using $U\ V\ Y\ sub(2)$ **by** *blast*

show *?thesis*

unfolding *quotient_map_def*

proof (*intro conjI allI impI*)

show $f \text{ 'topspace } (\text{subtopology } X\ U) = \text{topspace } (\text{subtopology } Y\ V)$

using $sub\ U\ fim$ **by** (*auto*)

next

fix $Y' :: 'b\ set$

assume $Y' \subseteq \text{topspace } (\text{subtopology } Y\ V)$

then have $Y' \subseteq \text{topspace } Y\ Y' \subseteq V$

by (*simp_all*)

then have $eq: \{x \in \text{topspace } X. x \in U \wedge f\ x \in Y'\} = \{x \in \text{topspace } X. f\ x \in Y'\}$

using U **by** *blast*

then show *openin* (*subtopology* $X\ U$) $\{x \in \text{topspace } (\text{subtopology } X\ U). f\ x \in Y'\} = \text{openin } (\text{subtopology } Y\ V)\ Y'$

using $U\ V\ Y\ \langle \text{openin } X\ U \rangle\ \langle Y' \subseteq \text{topspace } Y \rangle\ \langle Y' \subseteq V \rangle$

by (*simp add: openin_open_subtopology eq*) (*auto simp: openin_closedin_eq*)

qed

next

assume $V: \text{closedin } Y\ V$

with U **have** $sub: U \subseteq \text{topspace } X\ V \subseteq \text{topspace } Y$

by (*auto simp: closedin_subset*)

have $fim: f \text{ 'topspace } X = \text{topspace } Y$

and $Y: \bigwedge U. U \subseteq \text{topspace } Y \implies \text{closedin } X\ \{x \in \text{topspace } X. f\ x \in U\} = \text{closedin } Y\ U$

using *quo* **unfolding** *quotient_map_closedin* **by** *auto*

have *closedin* $X\ U$

using $U\ V\ Y\ sub(2)$ **by** *blast*

show *?thesis*

unfolding *quotient_map_closedin*

proof (*intro conjI allI impI*)

show $f \text{ 'topspace } (\text{subtopology } X\ U) = \text{topspace } (\text{subtopology } Y\ V)$


```

    using sub U fim by (auto)
  next
    fix Y' :: 'b set
    assume Y' ⊆ topspace (subtopology Y V)
    then have Y' ⊆ topspace Y Y' ⊆ V
      by (simp_all)
    then have eq: {x ∈ topspace X. x ∈ U ∧ f x ∈ Y'} = {x ∈ topspace X. f x ∈
Y'}
      using U by blast
    then show closedin (subtopology X U) {x ∈ topspace (subtopology X U). f x ∈
Y'} = closedin (subtopology Y V) Y'
      using U V Y ⟨closedin X U⟩ ⟨Y' ⊆ topspace Y⟩ ⟨Y' ⊆ V⟩
      by (simp add: closedin_closed_subtopology eq) (auto simp: closedin_def)
    qed
  qed

lemma quotient_map_saturated_open:
  quotient_map X Y f ⟷
    continuous_map X Y f ∧ f' (topspace X) = topspace Y ∧
    (∀ U. openin X U ∧ {x ∈ topspace X. f x ∈ f' U} ⊆ U ⟶ openin Y (f'
U))
  (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  then have fim: f' topspace X = topspace Y
    and Y: ⋀ U. U ⊆ topspace Y ⟹ openin Y U = openin X {x ∈ topspace X. f
x ∈ U}
  unfolding quotient_map_def by auto
  show ?rhs
  proof (intro conjI allI impI)
    show continuous_map X Y f
      by (simp add: L quotient_imp_continuous_map)
    show f' topspace X = topspace Y
      by (simp add: fim)
  next
    fix U :: 'a set
    assume U: openin X U ∧ {x ∈ topspace X. f x ∈ f' U} ⊆ U
    then have sub: f' U ⊆ topspace Y and eq: {x ∈ topspace X. f x ∈ f' U} =
U
      using fim openin_subset by fastforce+
    show openin Y (f' U)
      by (simp add: sub Y eq U)
    qed
  next
    assume ?rhs
    then have YX: ⋀ U. openin Y U ⟹ openin X {x ∈ topspace X. f x ∈ U}
      and fim: f' topspace X = topspace Y
      and XY: ⋀ U. ⌊openin X U; {x ∈ topspace X. f x ∈ f' U} ⊆ U⌋ ⟹ openin
Y (f' U)

```

```

    by (auto simp: quotient_map_def continuous_map_def)
  show ?lhs
  proof (simp add: quotient_map_def fim, intro allI impI iffI)
    fix U :: 'b set
    assume U ⊆ topspace Y and X: openin X {x ∈ topspace X. f x ∈ U}
    have feq: f ' {x ∈ topspace X. f x ∈ U} = U
      using ⟨U ⊆ topspace Y⟩ fim by auto
    show openin Y U
      using XY [OF X] by (simp add: feq)
  next
    fix U :: 'b set
    assume U ⊆ topspace Y and Y: openin Y U
    show openin X {x ∈ topspace X. f x ∈ U}
      by (metis YX [OF Y])
  qed
qed

```

2.2.13 Separated Sets

definition *separatedin* :: 'a topology ⇒ 'a set ⇒ 'a set ⇒ bool

where *separatedin* X S T ≡
 $S \subseteq \text{topspace } X \wedge T \subseteq \text{topspace } X \wedge$
 $S \cap X \text{ closure\_of } T = \{\} \wedge T \cap X \text{ closure\_of } S = \{\}$

lemma *separatedin_empty* [simp]:
 $\text{separatedin } X \ S \ \{\} \longleftrightarrow S \subseteq \text{topspace } X$
 $\text{separatedin } X \ \{\} \ S \longleftrightarrow S \subseteq \text{topspace } X$
by (simp_all add: separatedin_def)

lemma *separatedin_refl* [simp]:
 $\text{separatedin } X \ S \ S \longleftrightarrow S = \{\}$
proof –
 have $\bigwedge x. [\![\text{separatedin } X \ S \ S; x \in S]\!] \implies \text{False}$
 by (metis all_not_in_conv closure_of_subset inf.orderE separatedin_def)
 then show ?thesis
 by auto
qed

lemma *separatedin_sym*:
 $\text{separatedin } X \ S \ T \longleftrightarrow \text{separatedin } X \ T \ S$
by (auto simp: separatedin_def)

lemma *separatedin_imp_disjoint*:
 $\text{separatedin } X \ S \ T \implies \text{disjnt } S \ T$
by (meson closure_of_subset disjnt_def disjnt_subset2 separatedin_def)

lemma *separatedin_mono*:
 $[\![\text{separatedin } X \ S \ T; S' \subseteq S; T' \subseteq T]\!] \implies \text{separatedin } X \ S' \ T'$
unfolding separatedin_def

using closure_of_mono by blast

lemma separatedin_open_sets:

$\llbracket \text{openin } X \ S; \text{openin } X \ T \rrbracket \implies \text{separatedin } X \ S \ T \longleftrightarrow \text{disjnt } S \ T$

unfolding disjnt_def separatedin_def

by (auto simp: openin_Int_closure_of_eq_empty openin_subset)

lemma separatedin_closed_sets:

$\llbracket \text{closedin } X \ S; \text{closedin } X \ T \rrbracket \implies \text{separatedin } X \ S \ T \longleftrightarrow \text{disjnt } S \ T$

unfolding closure_of_eq disjnt_def separatedin_def

by (metis closedin_def closure_of_eq inf_commute)

lemma separatedin_subtopology:

$\text{separatedin } (\text{subtopology } X \ U) \ S \ T \longleftrightarrow S \subseteq U \wedge T \subseteq U \wedge \text{separatedin } X \ S \ T$ (is ?lhs = ?rhs)

by (auto simp: separatedin_def closure_of_subtopology Int_ac disjoint_iff elim!: inf.orderE)

lemma separatedin_discrete_topology:

$\text{separatedin } (\text{discrete\_topology } U) \ S \ T \longleftrightarrow S \subseteq U \wedge T \subseteq U \wedge \text{disjnt } S \ T$

by (metis openin_discrete_topology separatedin_def separatedin_open_sets topspace_discrete_topology)

lemma separated_eq_distinguishable:

$\text{separatedin } X \ \{x\} \ \{y\} \longleftrightarrow$

$x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge$

$(\exists U. \text{openin } X \ U \wedge x \in U \wedge (y \notin U)) \wedge$

$(\exists v. \text{openin } X \ v \wedge y \in v \wedge (x \notin v))$

by (force simp: separatedin_def closure_of_def)

lemma separatedin_Un [simp]:

$\text{separatedin } X \ S \ (T \cup U) \longleftrightarrow \text{separatedin } X \ S \ T \wedge \text{separatedin } X \ S \ U$

$\text{separatedin } X \ (S \cup T) \ U \longleftrightarrow \text{separatedin } X \ S \ U \wedge \text{separatedin } X \ T \ U$

by (auto simp: separatedin_def)

lemma separatedin_Union:

$\text{finite } \mathcal{F} \implies \text{separatedin } X \ S \ (\bigcup \mathcal{F}) \longleftrightarrow S \subseteq \text{topspace } X \wedge (\forall T \in \mathcal{F}. \text{separatedin } X \ S \ T)$

$\text{finite } \mathcal{F} \implies \text{separatedin } X \ (\bigcup \mathcal{F}) \ S \longleftrightarrow (\forall T \in \mathcal{F}. \text{separatedin } X \ S \ T) \wedge S \subseteq \text{topspace } X$

by (auto simp: separatedin_def closure_of_Union)

lemma separatedin_openin_diff:

$\llbracket \text{openin } X \ S; \text{openin } X \ T \rrbracket \implies \text{separatedin } X \ (S - T) \ (T - S)$

unfolding separatedin_def

by (metis Diff_Int_distrib2 Diff_disjoint Diff_empty Diff_mono empty_Diff empty_subsetI openin_Int_closure_of_eq_empty openin_subset)

lemma separatedin_closedin_diff:

assumes closedin $X \ S$ closedin $X \ T$

```

shows separatedin X (S - T) (T - S)
proof -
  have S - T ⊆ topspace X T - S ⊆ topspace X
    using assms closedin_subset by auto
  with assms show ?thesis
    by (simp add: separatedin_def Diff_Int_distrib2 closure_of_minimal inf_absorb2)
qed

```

```

lemma separation_closedin_Un_gen:
  separatedin X S T ⟷
    S ⊆ topspace X ∧ T ⊆ topspace X ∧ disjnt S T ∧
    closedin (subtopology X (S ∪ T)) S ∧
    closedin (subtopology X (S ∪ T)) T
  by (auto simp add: separatedin_def closedin_Int_closure_of disjnt_iff dest: clo-
    sure_of_subset)

```

```

lemma separation_openin_Un_gen:
  separatedin X S T ⟷
    S ⊆ topspace X ∧ T ⊆ topspace X ∧ disjnt S T ∧
    openin (subtopology X (S ∪ T)) S ∧
    openin (subtopology X (S ∪ T)) T
  unfolding openin_closedin_eq topspace_subtopology separation_closedin_Un_gen
  disjnt_def
  by (auto simp: Diff_triv Int_commute Un_Diff inf_absorb1 topspace_def)

```

2.2.14 Homeomorphisms

(1-way and 2-way versions may be useful in places)

```

definition homeomorphic_map :: 'a topology ⇒ 'b topology ⇒ ('a ⇒ 'b) ⇒ bool
where
  homeomorphic_map X Y f ≡ quotient_map X Y f ∧ inj_on f (topspace X)

```

```

definition homeomorphic_maps :: 'a topology ⇒ 'b topology ⇒ ('a ⇒ 'b) ⇒ ('b
⇒ 'a) ⇒ bool
where
  homeomorphic_maps X Y f g ≡
    continuous_map X Y f ∧ continuous_map Y X g ∧
    (∀ x ∈ topspace X. g(f x) = x) ∧ (∀ y ∈ topspace Y. f(g y) = y)

```

```

lemma homeomorphic_map_eq:
  ⟦homeomorphic_map X Y f; ∧ x. x ∈ topspace X ⟹ f x = g x⟧ ⟹ homeo-
  morphic_map X Y g
  by (meson homeomorphic_map_def inj_on_cong quotient_map_eq)

```

```

lemma homeomorphic_maps_eq:
  ⟦homeomorphic_maps X Y f g;
  ∧ x. x ∈ topspace X ⟹ f x = f' x; ∧ y. y ∈ topspace Y ⟹ g y = g' y⟧
  ⟹ homeomorphic_maps X Y f' g'

```

```

unfolding homeomorphic_maps_def
by (metis continuous_map_eq continuous_map_eq_image_closure_subset_gen
image_subset_iff)

```

```

lemma homeomorphic_maps_sym:
  homeomorphic_maps X Y f g  $\longleftrightarrow$  homeomorphic_maps Y X g f
by (auto simp: homeomorphic_maps_def)

```

```

lemma homeomorphic_maps_id:
  homeomorphic_maps X Y id id  $\longleftrightarrow$  Y = X (is ?lhs = ?rhs)

```

```

proof
  assume L: ?lhs
  then have topspace X = topspace Y
    by (auto simp: homeomorphic_maps_def continuous_map_def)
  with L show ?rhs
    unfolding homeomorphic_maps_def
    by (metis topology_finer_continuous_id topology_eq)
next
  assume ?rhs
  then show ?lhs
    unfolding homeomorphic_maps_def by auto
qed

```

```

lemma homeomorphic_map_id [simp]: homeomorphic_map X Y id  $\longleftrightarrow$  Y = X
  (is ?lhs = ?rhs)

```

```

proof
  assume L: ?lhs
  then have eq: topspace X = topspace Y
    by (auto simp: homeomorphic_map_def continuous_map_def quotient_map_def)
  then have  $\bigwedge S. \text{openin } X \ S \longrightarrow \text{openin } Y \ S$ 
    by (meson L homeomorphic_map_def injective_quotient_map topology_finer_open_id)
  then show ?rhs
    using L unfolding homeomorphic_map_def
    by (metis eq_quotient_imp_continuous_map topology_eq topology_finer_continuous_id)
next
  assume ?rhs
  then show ?lhs
    unfolding homeomorphic_map_def
    by (simp add: closed_map_id continuous_closed_imp_quotient_map)
qed

```

```

lemma homeomorphic_map_compose:
  assumes homeomorphic_map X Y f homeomorphic_map Y X'' g
  shows homeomorphic_map X X'' (g  $\circ$  f)
proof –
  have inj_on g (f ` topspace X)
    by (metis (no_types) assms homeomorphic_map_def quotient_imp_surjective_map)
  then show ?thesis
    using assms by (meson comp_inj_on homeomorphic_map_def quotient_map_compose_eq)

```

qed

lemma *homeomorphic_maps_compose*:

$\text{homeomorphic\_maps } X \ Y \ f \ h \wedge$
 $\text{homeomorphic\_maps } Y \ X'' \ g \ k$
 $\implies \text{homeomorphic\_maps } X \ X'' \ (g \circ f) \ (h \circ k)$

unfolding *homeomorphic_maps_def*

by (*auto simp: continuous_map_compose; simp add: continuous_map_def*)

lemma *homeomorphic_eq_everything_map*:

$\text{homeomorphic\_map } X \ Y \ f \longleftrightarrow$
 $\text{continuous\_map } X \ Y \ f \wedge \text{open\_map } X \ Y \ f \wedge \text{closed\_map } X \ Y \ f \wedge$
 $f \text{ ' } (\text{topspace } X) = \text{topspace } Y \wedge \text{inj\_on } f \ (\text{topspace } X)$

unfolding *homeomorphic_map_def*

by (*force simp: injective_quotient_map intro: injective_quotient_map*)

lemma *homeomorphic_imp_continuous_map*:

$\text{homeomorphic\_map } X \ Y \ f \implies \text{continuous\_map } X \ Y \ f$

by (*simp add: homeomorphic_eq_everything_map*)

lemma *homeomorphic_imp_open_map*:

$\text{homeomorphic\_map } X \ Y \ f \implies \text{open\_map } X \ Y \ f$

by (*simp add: homeomorphic_eq_everything_map*)

lemma *homeomorphic_imp_closed_map*:

$\text{homeomorphic\_map } X \ Y \ f \implies \text{closed\_map } X \ Y \ f$

by (*simp add: homeomorphic_eq_everything_map*)

lemma *homeomorphic_imp_surjective_map*:

$\text{homeomorphic\_map } X \ Y \ f \implies f \text{ ' } (\text{topspace } X) = \text{topspace } Y$

by (*simp add: homeomorphic_eq_everything_map*)

lemma *homeomorphic_imp_injective_map*:

$\text{homeomorphic\_map } X \ Y \ f \implies \text{inj\_on } f \ (\text{topspace } X)$

by (*simp add: homeomorphic_eq_everything_map*)

lemma *bijective_open_imp_homeomorphic_map*:

$\llbracket \text{continuous\_map } X \ Y \ f; \text{open\_map } X \ Y \ f; f \text{ ' } (\text{topspace } X) = \text{topspace } Y; \text{inj\_on } f \ (\text{topspace } X) \rrbracket$

$\implies \text{homeomorphic\_map } X \ Y \ f$

by (*simp add: homeomorphic_map_def continuous_open_imp_quotient_map*)

lemma *bijective_closed_imp_homeomorphic_map*:

$\llbracket \text{continuous\_map } X \ Y \ f; \text{closed\_map } X \ Y \ f; f \text{ ' } (\text{topspace } X) = \text{topspace } Y; \text{inj\_on } f \ (\text{topspace } X) \rrbracket$

$\implies \text{homeomorphic\_map } X \ Y \ f$

by (*simp add: continuous_closed_quotient_map homeomorphic_map_def*)

lemma *open_eq_continuous_inverse_map*:

```

    assumes X:  $\bigwedge x. x \in \text{topspace } X \implies f x \in \text{topspace } Y \wedge g(f x) = x$ 
    and Y:  $\bigwedge y. y \in \text{topspace } Y \implies g y \in \text{topspace } X \wedge f(g y) = y$ 
    shows  $\text{open\_map } X Y f \longleftrightarrow \text{continuous\_map } Y X g$ 
  proof -
    have eq:  $\{x \in \text{topspace } Y. g x \in U\} = f^{-1} U$  if  $\text{openin } X U$  for  $U$ 
    using  $\text{openin\_subset}$  [OF that] by (force simp:  $X Y \text{ image\_iff}$ )
    show ?thesis
    by (auto simp:  $Y \text{ open\_map\_def continuous\_map\_def}$  eq)
  qed

lemma closed_eq_continuous_inverse_map:
  assumes X:  $\bigwedge x. x \in \text{topspace } X \implies f x \in \text{topspace } Y \wedge g(f x) = x$ 
  and Y:  $\bigwedge y. y \in \text{topspace } Y \implies g y \in \text{topspace } X \wedge f(g y) = y$ 
  shows  $\text{closed\_map } X Y f \longleftrightarrow \text{continuous\_map } Y X g$ 
  proof -
    have eq:  $\{x \in \text{topspace } Y. g x \in U\} = f^{-1} U$  if  $\text{closedin } X U$  for  $U$ 
    using  $\text{closedin\_subset}$  [OF that] by (force simp:  $X Y \text{ image\_iff}$ )
    show ?thesis
    by (auto simp:  $Y \text{ closed\_map\_def continuous\_map\_closedin}$  eq)
  qed

lemma homeomorphic_maps_map:
  homeomorphic_maps  $X Y f g \longleftrightarrow$ 
    homeomorphic_map  $X Y f \wedge \text{homeomorphic\_map } Y X g \wedge$ 
     $(\forall x \in \text{topspace } X. g(f x) = x) \wedge (\forall y \in \text{topspace } Y. f(g y) = y)$ 
  (is ?lhs = ?rhs)
  proof
    assume ?lhs
    then have L:  $\text{continuous\_map } X Y f \text{ continuous\_map } Y X g \forall x \in \text{topspace } X. g$ 
     $(f x) = x \forall x' \in \text{topspace } Y. f(g x') = x'$ 
    by (auto simp: homeomorphic_maps_def)
    show ?rhs
    proof (intro conjI bijective_open_imp_homeomorphic_map L)
      show  $\text{open\_map } X Y f$ 
      using L using  $\text{open\_eq\_continuous\_inverse\_map}$  [of concl:  $X Y f g$ ] by
      (simp add: continuous_map_def)
      show  $\text{open\_map } Y X g$ 
      using L using  $\text{open\_eq\_continuous\_inverse\_map}$  [of concl:  $Y X g f$ ] by
      (simp add: continuous_map_def)
      show  $f^{-1} \text{topspace } X = \text{topspace } Y g^{-1} \text{topspace } Y = \text{topspace } X$ 
      using L by (force simp: continuous_map_closedin)+
      show  $\text{inj\_on } f (\text{topspace } X) \text{ inj\_on } g (\text{topspace } Y)$ 
      using L unfolding inj_on_def by metis+
    qed
  next
    assume ?rhs
    then show ?lhs
    by (auto simp: homeomorphic_maps_def homeomorphic_imp_continuous_map)
  qed

```

```

lemma homeomorphic_maps_imp_map:
  homeomorphic_maps X Y f g  $\implies$  homeomorphic_map X Y f
  using homeomorphic_maps_map by blast

lemma homeomorphic_map_maps:
  homeomorphic_map X Y f  $\longleftrightarrow$  ( $\exists g$ . homeomorphic_maps X Y f g)
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then have L: continuous_map X Y f open_map X Y f closed_map X Y f
    f ' (topspace X) = topspace Y inj_on f (topspace X)
    by (auto simp: homeomorphic_eq_everything_map)
  have X:  $\bigwedge x. x \in \text{topspace } X \implies f\ x \in \text{topspace } Y \wedge \text{inv\_into } (\text{topspace } X) f (f\ x) = x$ 
    using L by auto
  have Y:  $\bigwedge y. y \in \text{topspace } Y \implies \text{inv\_into } (\text{topspace } X) f\ y \in \text{topspace } X \wedge f\ (\text{inv\_into } (\text{topspace } X) f\ y) = y$ 
    by (simp add: L f_inv_into_f inv_into_inv)
  have homeomorphic_maps X Y f (inv_into (topspace X) f)
    unfolding homeomorphic_maps_def
  proof (intro conjI L)
    show continuous_map Y X (inv_into (topspace X) f)
      by (simp add: L X Y flip: open_eq_continuous_inverse_map [where f=f])
  next
    show  $\forall x \in \text{topspace } X. \text{inv\_into } (\text{topspace } X) f (f\ x) = x$ 
       $\forall y \in \text{topspace } Y. f\ (\text{inv\_into } (\text{topspace } X) f\ y) = y$ 
      using X Y by auto
  qed
  then show ?rhs
    by metis
next
  assume ?rhs
  then show ?lhs
    using homeomorphic_maps_map by blast
qed

```

```

lemma homeomorphic_maps_involution:
   $\llbracket \text{continuous\_map } X\ X\ f; \bigwedge x. x \in \text{topspace } X \implies f(f\ x) = x \rrbracket \implies \text{homeomorphic\_maps } X\ X\ f\ f$ 
  by (auto simp: homeomorphic_maps_def)

```

```

lemma homeomorphic_map_involution:
   $\llbracket \text{continuous\_map } X\ X\ f; \bigwedge x. x \in \text{topspace } X \implies f(f\ x) = x \rrbracket \implies \text{homeomorphic\_map } X\ X\ f$ 
  using homeomorphic_maps_involution homeomorphic_maps_map by blast

```

```

lemma homeomorphic_map_openness:
  assumes hom: homeomorphic_map X Y f and U:  $U \subseteq \text{topspace } X$ 

```



```

shows openin Y (f ' U)  $\longleftrightarrow$  openin X U
proof -
  obtain g where homeomorphic_maps X Y f g
    using assms by (auto simp: homeomorphic_map_maps)
  then have g: homeomorphic_map Y X g and gf:  $\bigwedge x. x \in \text{topspace } X \implies g(f$ 
 $x) = x$ 
    by (auto simp: homeomorphic_maps_map)
  then have openin X U  $\implies$  openin Y (f ' U)
    using hom homeomorphic_imp_open_map open_map_def by blast
  show openin Y (f ' U) = openin X U
proof
  assume L: openin Y (f ' U)
  have U = g ' (f ' U)
    using U gf by force
  then show openin X U
    by (metis L homeomorphic_imp_open_map open_map_def g)
next
  assume openin X U
  then show openin Y (f ' U)
    using hom homeomorphic_imp_open_map open_map_def by blast
qed
qed

```

```

lemma homeomorphic_map_closedness:
  assumes hom: homeomorphic_map X Y f and U:  $U \subseteq \text{topspace } X$ 
  shows closedin Y (f ' U)  $\longleftrightarrow$  closedin X U
proof -
  obtain g where homeomorphic_maps X Y f g
    using assms by (auto simp: homeomorphic_map_maps)
  then have g: homeomorphic_map Y X g and gf:  $\bigwedge x. x \in \text{topspace } X \implies g(f$ 
 $x) = x$ 
    by (auto simp: homeomorphic_maps_map)
  then have closedin X U  $\implies$  closedin Y (f ' U)
    using hom homeomorphic_imp_closed_map closed_map_def by blast
  show closedin Y (f ' U) = closedin X U
proof
  assume L: closedin Y (f ' U)
  have U = g ' (f ' U)
    using U gf by force
  then show closedin X U
    by (metis L homeomorphic_imp_closed_map closed_map_def g)
next
  assume closedin X U
  then show closedin Y (f ' U)
    using hom homeomorphic_imp_closed_map closed_map_def by blast
qed
qed

```

lemma *homeomorphic_map_openness_eq*:
 $\text{homeomorphic\_map } X \ Y \ f \implies \text{openin } X \ U \longleftrightarrow U \subseteq \text{topspace } X \wedge \text{openin } Y \ (f \, ' \, U)$
by (meson *homeomorphic_map_openness openin_closedin_eq*)

lemma *homeomorphic_map_closedness_eq*:
 $\text{homeomorphic\_map } X \ Y \ f \implies \text{closedin } X \ U \longleftrightarrow U \subseteq \text{topspace } X \wedge \text{closedin } Y \ (f \, ' \, U)$
by (meson *closedin_subset homeomorphic_map_closedness*)

lemma *all_openin_homeomorphic_image*:
assumes *homeomorphic_map* $X \ Y \ f$
shows $(\forall V. \text{openin } Y \ V \longrightarrow P \ V) \longleftrightarrow (\forall U. \text{openin } X \ U \longrightarrow P(f \, ' \, U))$ (**is** *?lhs = ?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (meson *assms homeomorphic_map_openness_eq*)
next
assume *?rhs*
then show *?lhs*
by (metis (*no_types, lifting*) *assms homeomorphic_imp_surjective_map homeomorphic_map_openness openin_subset subset_image_iff*)
qed

lemma *all_closedin_homeomorphic_image*:
assumes *homeomorphic_map* $X \ Y \ f$
shows $(\forall V. \text{closedin } Y \ V \longrightarrow P \ V) \longleftrightarrow (\forall U. \text{closedin } X \ U \longrightarrow P(f \, ' \, U))$ (**is** *?lhs = ?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (meson *assms homeomorphic_map_closedness_eq*)
next
assume *?rhs*
then show *?lhs*
by (metis (*no_types, lifting*) *assms homeomorphic_imp_surjective_map homeomorphic_map_closedness closedin_subset subset_image_iff*)
qed

lemma *homeomorphic_map_derived_set_of*:
assumes *hom*: *homeomorphic_map* $X \ Y \ f$ **and** *S*: $S \subseteq \text{topspace } X$
shows $Y \text{ derived\_set\_of } (f \, ' \, S) = f \, ' \, (X \text{ derived\_set\_of } S)$
proof –
have *fim*: $f \, ' \, (\text{topspace } X) = \text{topspace } Y$ **and** *inj*: *inj_on* $f \ (\text{topspace } X)$
using *hom* **by** (auto *simp: homeomorphic_eq_everything_map*)
have *iff*: $(\forall T. x \in T \wedge \text{openin } X \ T \longrightarrow (\exists y. y \neq x \wedge y \in S \wedge y \in T)) =$
 $(\forall T. T \subseteq \text{topspace } Y \longrightarrow f \, x \in T \longrightarrow \text{openin } Y \ T \longrightarrow (\exists y. y \neq f \, x \wedge$

```

y ∈ f ' S ∧ y ∈ T))
  if x ∈ topspace X for x
proof -
  have §: (x ∈ T ∧ openin X T) = (T ⊆ topspace X ∧ f x ∈ f ' T ∧ openin Y
(f ' T)) for T
  by (meson hom homeomorphic_map_openness_eq inj inj_on_image_mem_iff
that)
  moreover have (∃ y. y ≠ x ∧ y ∈ S ∧ y ∈ T) = (∃ y. y ≠ f x ∧ y ∈ f ' S ∧
y ∈ f ' T) (is ?lhs = ?rhs)
  if T ⊆ topspace X ∧ f x ∈ f ' T ∧ openin Y (f ' T) for T
proof
  show ?lhs ⇒ ?rhs
  by (meson § imageI inj inj_on_eq_iff inj_on_subset that)
  show ?rhs ⇒ ?lhs
  using S inj inj_onD that by fastforce
qed
ultimately show ?thesis
by (auto simp flip: fim simp: all_subset_image)
qed
have *: [T = f ' S; ∧x. x ∈ S ⇒ P x ⇔ Q(f x)] ⇒ {y. y ∈ T ∧ Q y} = f
' {x ∈ S. P x} for T S P Q
by auto
show ?thesis
unfolding derived_set_of_def
by (rule *) (use fim iff openin_subset in force)+
qed

```

```

lemma homeomorphic_map_closure_of:
  assumes hom: homeomorphic_map X Y f and S: S ⊆ topspace X
  shows Y closure_of (f ' S) = f ' (X closure_of S)
  unfolding closure_of
  using homeomorphic_imp_surjective_map [OF hom] S
  by (auto simp: in_derived_set_of homeomorphic_map_derived_set_of [OF assms])

```

```

lemma homeomorphic_map_interior_of:
  assumes hom: homeomorphic_map X Y f and S: S ⊆ topspace X
  shows Y interior_of (f ' S) = f ' (X interior_of S)
proof -
  { fix y
    assume y ∈ topspace Y and y ∉ Y closure_of (topspace Y - f ' S)
    then have y ∈ f ' (topspace X - X closure_of (topspace X - S))
      using homeomorphic_eq_everything_map [THEN iffD1, OF hom] homeo-
morphic_map_closure_of [OF hom]
    by (metis DiffI Diff_subset S closure_of_subset_topspace inj_on_image_set_diff)
  }
  moreover
  { fix x
    assume x ∈ topspace X

```

```

    then have  $f x \in \text{topspace } Y$ 
      using  $\text{hom homeomorphic\_imp\_surjective\_map}$  by blast }
  moreover
  { fix  $x$ 
    assume  $x \in \text{topspace } X$  and  $x \notin X \text{ closure\_of } (\text{topspace } X - S)$  and  $f x \in Y$ 
      closure\_of  $(\text{topspace } Y - f^{-1} S)$ 
    then have False
      using  $\text{homeomorphic\_map\_closure\_of}$  [OF  $\text{hom}$ ]  $\text{hom}$ 
      unfolding  $\text{homeomorphic\_eq\_everything\_map}$ 
      by ( $\text{metis Diff\_subset } S \text{ closure\_of\_subset\_topspace inj\_on\_image\_mem\_iff}$ 
         $\text{inj\_on\_image\_set\_diff}$ )
    }
  ultimately show ?thesis
    by ( $\text{auto simp: interior\_of\_closure\_of}$ )
qed

```

```

lemma  $\text{homeomorphic\_map\_frontier\_of}$ :
  assumes  $\text{hom: homeomorphic\_map } X Y f$  and  $S: S \subseteq \text{topspace } X$ 
  shows  $Y \text{ frontier\_of } (f^{-1} S) = f^{-1} (X \text{ frontier\_of } S)$ 
  unfolding  $\text{frontier\_of\_def}$ 
proof ( $\text{intro equalityI subsetI DiffI}$ )
  fix  $y$ 
  assume  $y \in Y \text{ closure\_of } f^{-1} S - Y \text{ interior\_of } f^{-1} S$ 
  then show  $y \in f^{-1} (X \text{ closure\_of } S - X \text{ interior\_of } S)$ 
    using  $S \text{ hom homeomorphic\_map\_closure\_of homeomorphic\_map\_interior\_of}$ 
  by fastforce
next
  fix  $y$ 
  assume  $y \in f^{-1} (X \text{ closure\_of } S - X \text{ interior\_of } S)$ 
  then show  $y \in Y \text{ closure\_of } f^{-1} S$ 
    using  $S \text{ hom homeomorphic\_map\_closure\_of}$  by fastforce
next
  fix  $x$ 
  assume  $x \in f^{-1} (X \text{ closure\_of } S - X \text{ interior\_of } S)$ 
  then obtain  $y$  where  $x = f y$   $y \in X \text{ closure\_of } S$   $y \notin X \text{ interior\_of } S$ 
    by blast
  then have  $y \in \text{topspace } X$ 
    by ( $\text{simp add: in\_closure\_of}$ )
  then have  $f y \notin f^{-1} (X \text{ interior\_of } S)$ 
    by ( $\text{meson hom homeomorphic\_map\_def inj\_on\_image\_mem\_iff interior\_of\_subset\_topspace}$ 
       $y(3)$ )
  then show  $x \notin Y \text{ interior\_of } f^{-1} S$ 
    using  $S \text{ hom homeomorphic\_map\_interior\_of } y(1)$  by blast
qed

```

```

lemma  $\text{homeomorphic\_maps\_subtopologies}$ :
   $\llbracket \text{homeomorphic\_maps } X Y f g; f^{-1} (\text{topspace } X \cap S) = \text{topspace } Y \cap T \rrbracket$ 
   $\implies \text{homeomorphic\_maps } (\text{subtopology } X S) (\text{subtopology } Y T) f g$ 
  unfolding  $\text{homeomorphic\_maps\_def}$ 

```

by (force simp: continuous_map_from_subtopology continuous_map_in_subtopology)

lemma *homeomorphic_maps_subtopologies_alt*:

$\llbracket \text{homeomorphic\_maps } X \ Y \ f \ g; f' (topspace \ X \cap S) \subseteq T; g' (topspace \ Y \cap T) \subseteq S \rrbracket$

$\implies \text{homeomorphic\_maps } (\text{subtopology } X \ S) \ (\text{subtopology } Y \ T) \ f \ g$

unfolding *homeomorphic_maps_def*

by (force simp: continuous_map_from_subtopology continuous_map_in_subtopology)

lemma *homeomorphic_map_subtopologies*:

$\llbracket \text{homeomorphic\_map } X \ Y \ f; f' (topspace \ X \cap S) = topspace \ Y \cap T \rrbracket$

$\implies \text{homeomorphic\_map } (\text{subtopology } X \ S) \ (\text{subtopology } Y \ T) \ f$

by (meson *homeomorphic_map_maps homeomorphic_maps_subtopologies*)

lemma *homeomorphic_map_subtopologies_alt*:

assumes *hom*: *homeomorphic_map* *X* *Y* *f*

and *S*: $\bigwedge x. \llbracket x \in topspace \ X; f \ x \in topspace \ Y \rrbracket \implies f \ x \in T \longleftrightarrow x \in S$

shows *homeomorphic_map* (*subtopology* *X* *S*) (*subtopology* *Y* *T*) *f*

proof –

have *homeomorphic_maps* (*subtopology* *X* *S*) (*subtopology* *Y* *T*) *f* *g*

if *homeomorphic_maps* *X* *Y* *f* *g* **for** *g*

proof (rule *homeomorphic_maps_subtopologies* [*OF that*])

show $f' (topspace \ X \cap S) = topspace \ Y \cap T$

using *that S*

apply (*auto simp*: *homeomorphic_maps_def continuous_map_def*)

by (*metis IntI image_iff*)

qed

then show *?thesis*

using *hom* **by** (*meson homeomorphic_map_maps*)

qed

2.2.15 Relation of homeomorphism between topological spaces

definition *homeomorphic_space* (*infixr* *homeomorphic'_space* 50)

where *X* *homeomorphic_space* *Y* $\equiv \exists f \ g. \text{homeomorphic\_maps } X \ Y \ f \ g$

lemma *homeomorphic_space_refl*: *X* *homeomorphic_space* *X*

by (*meson homeomorphic_maps_id homeomorphic_space_def*)

lemma *homeomorphic_space_sym*:

X *homeomorphic_space* *Y* $\longleftrightarrow$ *Y* *homeomorphic_space* *X*

unfolding *homeomorphic_space_def* **by** (*metis homeomorphic_maps_sym*)

lemma *homeomorphic_space_trans* [*trans*]:

$\llbracket X1 \text{ homeomorphic\_space } X2; X2 \text{ homeomorphic\_space } X3 \rrbracket \implies X1 \text{ homeomorphic\_space } X3$

unfolding *homeomorphic_space_def* **by** (*metis homeomorphic_maps_compose*)

lemma *homeomorphic_space*:

$X \text{ homeomorphic\_space } Y \longleftrightarrow (\exists f. \text{homeomorphic\_map } X Y f)$
by (simp add: homeomorphic_map_maps homeomorphic_space_def)

lemma homeomorphic_maps_imp_homeomorphic_space:
 $\text{homeomorphic\_maps } X Y f g \implies X \text{ homeomorphic\_space } Y$
unfolding homeomorphic_space_def **by** metis

lemma homeomorphic_map_imp_homeomorphic_space:
 $\text{homeomorphic\_map } X Y f \implies X \text{ homeomorphic\_space } Y$
unfolding homeomorphic_map_maps
using homeomorphic_space_def **by** blast

lemma homeomorphic_empty_space:
 $X \text{ homeomorphic\_space } Y \implies \text{topspace } X = \{\} \longleftrightarrow \text{topspace } Y = \{\}$
by (metis homeomorphic_imp_surjective_map homeomorphic_space_image_is_empty)

lemma homeomorphic_empty_space_eq:
assumes $\text{topspace } X = \{\}$
shows $X \text{ homeomorphic\_space } Y \longleftrightarrow \text{topspace } Y = \{\}$
proof –
have $\forall f t. \text{continuous\_map } X (t::'b \text{ topology}) f$
using assms continuous_map_on_empty **by** blast
then show ?thesis
by (metis (no_types) assms continuous_map_on_empty empty_iff homeomorphic_empty_space homeomorphic_maps_def homeomorphic_space_def)
qed

2.2.16 Connected topological spaces

definition connected_space :: 'a topology $\Rightarrow$ bool **where**
 $\text{connected\_space } X \equiv$
 $\neg(\exists E1 E2. \text{openin } X E1 \wedge \text{openin } X E2 \wedge$
 $\text{topspace } X \subseteq E1 \cup E2 \wedge E1 \cap E2 = \{\} \wedge E1 \neq \{\} \wedge E2 \neq \{\})$

definition connectedin :: 'a topology $\Rightarrow$ 'a set $\Rightarrow$ bool **where**
 $\text{connectedin } X S \equiv S \subseteq \text{topspace } X \wedge \text{connected\_space } (\text{subtopology } X S)$

lemma connected_spaceD:
 $\llbracket \text{connected\_space } X;$
 $\text{openin } X U; \text{openin } X V; \text{topspace } X \subseteq U \cup V; U \cap V = \{\}; U \neq \{\}; V \neq$
 $\{\} \rrbracket \implies \text{False}$
by (auto simp: connected_space_def)

lemma connectedin_subset_topospace: $\text{connectedin } X S \implies S \subseteq \text{topspace } X$
by (simp add: connectedin_def)

lemma connectedin_topospace:
 $\text{connectedin } X (\text{topspace } X) \longleftrightarrow \text{connected\_space } X$
by (simp add: connectedin_def)

lemma *connected_space_subtopology*:

$\text{connectedin } X \ S \implies \text{connected\_space } (\text{subtopology } X \ S)$

by (*simp add: connectedin_def*)

lemma *connectedin_subtopology*:

$\text{connectedin } (\text{subtopology } X \ S) \ T \longleftrightarrow \text{connectedin } X \ T \wedge T \subseteq S$

by (*force simp: connectedin_def subtopology_subtopology inf_absorb2*)

lemma *connected_space_eq*:

$\text{connected\_space } X \longleftrightarrow$

$(\nexists E1 \ E2. \text{openin } X \ E1 \wedge \text{openin } X \ E2 \wedge E1 \cup E2 = \text{topspace } X \wedge E1 \cap E2$

$= \{\} \wedge E1 \neq \{\} \wedge E2 \neq \{\})$

unfolding *connected_space_def*

by (*metis openin_Un openin_subset subset_antisym*)

lemma *connected_space_closedin*:

$\text{connected\_space } X \longleftrightarrow$

$(\nexists E1 \ E2. \text{closedin } X \ E1 \wedge \text{closedin } X \ E2 \wedge \text{topspace } X \subseteq E1 \cup E2 \wedge$
 $E1 \cap E2 = \{\} \wedge E1 \neq \{\} \wedge E2 \neq \{\})$ (**is** *?lhs = ?rhs*)

proof

assume *?lhs*

then have *L*: $\bigwedge E1 \ E2. \llbracket \text{openin } X \ E1; E1 \cap E2 = \{\}; \text{topspace } X \subseteq E1 \cup E2; \text{openin } X \ E2 \rrbracket \implies E1 = \{\} \vee E2 = \{\}$

by (*simp add: connected_space_def*)

show *?rhs*

unfolding *connected_space_def*

proof *clarify*

fix *E1 E2*

assume *closedin X E1 and closedin X E2 and topspace X ⊆ E1 ∪ E2 and E1 ∩ E2 = {}*

and *E1 ≠ {} and E2 ≠ {}*

have *E1 ∪ E2 = topspace X*

by (*meson Un_subset_iff <closedin X E1> <closedin X E2> <topspace X ⊆ E1 ∪ E2> closedin_def subset_antisym*)

then have *topspace X − E2 = E1*

using *<E1 ∩ E2 = {}>* **by** *fastforce*

then have *topspace X = E1*

using *<E1 ≠ {}>* *L* *<closedin X E1> <closedin X E2>* **by** *blast*

then show *False*

using *<E1 ∩ E2 = {}> <E1 ∪ E2 = topspace X> <E2 ≠ {}>* **by** *blast*

qed

next

assume *R: ?rhs*

show *?lhs*

unfolding *connected_space_def*

proof *clarify*

fix *E1 E2*

assume *openin X E1 and openin X E2 and topspace X ⊆ E1 ∪ E2 and E1*

```

 $\cap E2 = \{\}$ 
  and  $E1 \neq \{\}$  and  $E2 \neq \{\}$ 
  have  $E1 \cup E2 = \text{topspace } X$ 
    by (meson Un_subset_iff  $\langle \text{openin } X E1 \rangle \langle \text{openin } X E2 \rangle \langle \text{topspace } X \subseteq E1 \cup E2 \rangle$ 
    openin_closedin_eq subset_antisym)
  then have  $\text{topspace } X - E2 = E1$ 
    using  $\langle E1 \cap E2 = \{\} \rangle$  by fastforce
  then have  $\text{topspace } X = E1$ 
    using  $\langle E1 \neq \{\} \rangle$  R  $\langle \text{openin } X E1 \rangle \langle \text{openin } X E2 \rangle$  by blast
  then show False
    using  $\langle E1 \cap E2 = \{\} \rangle \langle E1 \cup E2 = \text{topspace } X \rangle \langle E2 \neq \{\} \rangle$  by blast
qed
qed

```

lemma *connected_space_closedin_eq*:

```

connected_space  $X \longleftrightarrow$ 
  ( $\nexists E1 E2. \text{closedin } X E1 \wedge \text{closedin } X E2 \wedge$ 
     $E1 \cup E2 = \text{topspace } X \wedge E1 \cap E2 = \{\} \wedge E1 \neq \{\} \wedge E2 \neq \{\}$ )
  by (metis closedin_Un closedin_def connected_space_closedin subset_antisym)

```

lemma *connected_space_clopen_in*:

```

connected_space  $X \longleftrightarrow$ 
  ( $\forall T. \text{openin } X T \wedge \text{closedin } X T \longrightarrow T = \{\} \vee T = \text{topspace } X$ )
proof -
  have eq:  $\text{openin } X E1 \wedge \text{openin } X E2 \wedge E1 \cup E2 = \text{topspace } X \wedge E1 \cap E2 = \{\} \wedge P$ 
     $\longleftrightarrow E2 = \text{topspace } X - E1 \wedge \text{openin } X E1 \wedge \text{openin } X E2 \wedge P$  for  $E1 E2$ 
  P
  using openin_subset by blast
  show ?thesis
    unfolding connected_space_eq eq closedin_def
    by (auto simp: openin_closedin_eq)
qed

```

lemma *connectedin*:

```

connectedin  $X S \longleftrightarrow$ 
   $S \subseteq \text{topspace } X \wedge$ 
  ( $\nexists E1 E2. \text{openin } X E1 \wedge \text{openin } X E2 \wedge$ 
     $S \subseteq E1 \cup E2 \wedge E1 \cap E2 \cap S = \{\} \wedge E1 \cap S \neq \{\} \wedge E2 \cap S \neq \{\}$ )
  (is ?lhs = ?rhs)
proof -
  have *: ( $\exists E1:: 'a \text{ set}. \exists E2:: 'a \text{ set}. (\exists T1:: 'a \text{ set}. P1 T1 \wedge E1 = f1 T1) \wedge$ 
    ( $\exists T2:: 'a \text{ set}. P2 T2 \wedge E2 = f2 T2) \wedge$ 
     $R E1 E2) \longleftrightarrow (\exists T1 T2. P1 T1 \wedge P2 T2 \wedge R(f1 T1) (f2 T2))$  for  $P1$ 
     $f1 P2 f2 R$ 
  by auto
  show ?thesis
    unfolding connectedin_def connected_space_def openin_subtopology topspace_subtopology

```



```

*
  by (intro conj_cong arg_cong [where f=Not] ex_cong1; blast dest!: openin_subset)
qed

lemma connectedin_iff_connected [simp]: connectedin euclidean S  $\longleftrightarrow$  connected
S
  by (simp add: connected_def connectedin)

lemma connectedin_closedin:
  connectedin X S  $\longleftrightarrow$ 
    S  $\subseteq$  topspace X  $\wedge$ 
     $\neg(\exists E1 E2. \text{closedin } X E1 \wedge \text{closedin } X E2 \wedge$ 
      S  $\subseteq (E1 \cup E2) \wedge$ 
       $(E1 \cap E2 \cap S = \{\}) \wedge$ 
       $\neg(E1 \cap S = \{\}) \wedge \neg(E2 \cap S = \{\}))$ 
proof -
  have *:  $(\exists E1:: 'a \text{ set}. \exists E2:: 'a \text{ set}. (\exists T1:: 'a \text{ set}. P1 T1 \wedge E1 = f1 T1) \wedge$ 
 $(\exists T2:: 'a \text{ set}. P2 T2 \wedge E2 = f2 T2) \wedge$ 
 $R E1 E2) \longleftrightarrow (\exists T1 T2. P1 T1 \wedge P2 T2 \wedge R(f1 T1) (f2 T2))$  for P1
  f1 P2 f2 R
  by auto
  show ?thesis
  unfolding connectedin_def connected_space_closedin closedin_subtopology topspace_subtopology
  *
  by (intro conj_cong arg_cong [where f=Not] ex_cong1; blast dest!: openin_subset)
qed

lemma connectedin_empty [simp]: connectedin X  $\{\}$ 
  by (simp add: connectedin)

lemma connected_space_topspace_empty:
  topspace X =  $\{\}$   $\implies$  connected_space X
  using connectedin_topspace by fastforce

lemma connectedin_sing [simp]: connectedin X  $\{a\} \longleftrightarrow a \in \text{topspace } X$ 
  by (simp add: connectedin)

lemma connectedin_absolute [simp]:
  connectedin (subtopology X S) S  $\longleftrightarrow$  connectedin X S
  by (simp add: connectedin_subtopology)

lemma connectedin_Union:
  assumes  $\mathcal{U}: \bigwedge S. S \in \mathcal{U} \implies \text{connectedin } X S$  and ne:  $\bigcap \mathcal{U} \neq \{\}$ 
  shows connectedin X  $(\bigcup \mathcal{U})$ 
proof -
  have  $\bigcup \mathcal{U} \subseteq \text{topspace } X$ 
  using  $\mathcal{U}$  by (simp add: Union_least connectedin_def)
  moreover have False
  if openin X E1 openin X E2 and cover:  $\bigcup \mathcal{U} \subseteq E1 \cup E2$  and disj:  $E1 \cap E2$ 

```

```

 $\cap \bigcup \mathcal{U} = \{\}$ 
  and overlap1:  $E1 \cap \bigcup \mathcal{U} \neq \{\}$  and overlap2:  $E2 \cap \bigcup \mathcal{U} \neq \{\}$ 
  for  $E1\ E2$ 
  proof -
    have disjS:  $E1 \cap E2 \cap S = \{\}$  if  $S \in \mathcal{U}$  for  $S$ 
      using Diff_triv that disj by auto
    have coverS:  $S \subseteq E1 \cup E2$  if  $S \in \mathcal{U}$  for  $S$ 
      using that cover by blast
    have  $\mathcal{U} \neq \{\}$ 
      using overlap1 by blast
    obtain  $a$  where  $a: \bigwedge U. U \in \mathcal{U} \implies a \in U$ 
      using ne by force
    with  $\langle \mathcal{U} \neq \{\} \rangle$  have  $a \in \bigcup \mathcal{U}$ 
      by blast
    then consider  $a \in E1 \mid a \in E2$ 
      using  $\langle \bigcup \mathcal{U} \subseteq E1 \cup E2 \rangle$  by auto
    then show False
  proof cases
    case 1
      then obtain  $b\ S$  where  $b \in E2\ b \in S\ S \in \mathcal{U}$ 
        using overlap2 by blast
      then show ?thesis
        using 1  $\langle \text{openin } X\ E1 \rangle \langle \text{openin } X\ E2 \rangle$  disjS coverS  $a$   $[OF\ \langle S \in \mathcal{U} \rangle]\ \mathcal{U}[OF\ \langle S \in \mathcal{U} \rangle]$ 
        unfolding connectedin
        by (meson disjoint_iff_not_equal)
    next
      case 2
        then obtain  $b\ S$  where  $b \in E1\ b \in S\ S \in \mathcal{U}$ 
          using overlap1 by blast
        then show ?thesis
          using 2  $\langle \text{openin } X\ E1 \rangle \langle \text{openin } X\ E2 \rangle$  disjS coverS  $a$   $[OF\ \langle S \in \mathcal{U} \rangle]\ \mathcal{U}[OF\ \langle S \in \mathcal{U} \rangle]$ 
          unfolding connectedin
          by (meson disjoint_iff_not_equal)
  qed
qed
ultimately show ?thesis
  unfolding connectedin by blast
qed

```

lemma *connectedin_Un*:

```

 $\llbracket \text{connectedin } X\ S; \text{connectedin } X\ T; S \cap T \neq \{\} \rrbracket \implies \text{connectedin } X\ (S \cup T)$ 
  using connectedin_Union [of  $\{S, T\}$ ] by auto

```

lemma *connected_space_subconnected*:

```

 $\text{connected\_space } X \longleftrightarrow (\forall x \in \text{topspace } X. \forall y \in \text{topspace } X. \exists S. \text{connectedin } X\ S \wedge x \in S \wedge y \in S)$  (is ?lhs = ?rhs)

```

proof

```

    assume ?lhs
    then show ?rhs
      using connectedin_topspace by blast
next
  assume R [rule_format]: ?rhs
  have False if openin X U openin X V and disj:  $U \cap V = \{\}$  and cover: topspace
 $X \subseteq U \cup V$ 
    and  $U \neq \{\}$   $V \neq \{\}$  for U V
  proof -
    obtain u v where  $u \in U$   $v \in V$ 
    using  $\langle U \neq \{\} \rangle \langle V \neq \{\} \rangle$  by auto
    then obtain T where  $u \in T$   $v \in T$  and T: connectedin X T
    using R [of u v] that
    by (meson  $\langle \text{openin } X \ U \rangle \langle \text{openin } X \ V \rangle \text{subsetD openin\_subset}$ )
    then show False
      using that unfolding connectedin
      by (metis IntI  $\langle u \in U \rangle \langle v \in V \rangle \text{empty\_iff inf\_bot\_left subset\_trans}$ )
    qed
  then show ?lhs
    by (auto simp: connected_space_def)
qed

```

```

lemma connectedin_intermediate_closure_of:
  assumes connectedin X S  $S \subseteq T$   $T \subseteq X \text{closure\_of } S$ 
  shows connectedin X T
proof -
  have S:  $S \subseteq \text{topspace } X$  and T:  $T \subseteq \text{topspace } X$ 
  using assms by (meson closure_of_subset_topspace dual_order.trans) +
  have §:  $\bigwedge E1 \ E2. [\text{openin } X \ E1; \text{openin } X \ E2; E1 \cap S = \{\} \vee E2 \cap S = \{\}]$ 
 $\implies E1 \cap T = \{\} \vee E2 \cap T = \{\}$ 
  using assms unfolding disjoint_iff by (meson in_closure_of subsetD)
  then show ?thesis
    using assms
    unfolding connectedin_closure_of_subset_topspace S T
    by (metis Int_empty_right T dual_order.trans inf.orderE inf_left_commute)
qed

```

```

lemma connectedin_closure_of:
  connectedin X S  $\implies \text{connectedin } X (X \text{closure\_of } S)$ 
  by (meson closure_of_subset connectedin_def connectedin_intermediate_closure_of subset_refl)

```

```

lemma connectedin_separation:
  connectedin X S  $\longleftrightarrow$ 
     $S \subseteq \text{topspace } X \wedge$ 
     $(\nexists C1 \ C2. C1 \cup C2 = S \wedge C1 \neq \{\} \wedge C2 \neq \{\} \wedge C1 \cap X \text{closure\_of } C2$ 
 $= \{\} \wedge C2 \cap X \text{closure\_of } C1 = \{\})$  (is ?lhs = ?rhs)
  unfolding connectedin_def connected_space_closedin_eq closedin_Int_closure_of
topspace_subtopology

```

```

apply (intro conj_cong refl arg_cong [where f=Not])
apply (intro ex_cong1 iffI, blast)
using closure_of_subset_Int by force

```

```

lemma connectedin_eq_not_separated:
  connectedin X S  $\longleftrightarrow$ 
    S  $\subseteq$  topspace X  $\wedge$ 
    ( $\nexists$  C1 C2. C1  $\cup$  C2 = S  $\wedge$  C1  $\neq$  {}  $\wedge$  C2  $\neq$  {}  $\wedge$  separatedin X C1 C2)
unfolding separatedin_def by (metis connectedin_separation sup.boundedE)

```

```

lemma connectedin_eq_not_separated_subset:
  connectedin X S  $\longleftrightarrow$ 
    S  $\subseteq$  topspace X  $\wedge$  ( $\nexists$  C1 C2. S  $\subseteq$  C1  $\cup$  C2  $\wedge$  S  $\cap$  C1  $\neq$  {}  $\wedge$  S  $\cap$  C2  $\neq$  {}
 $\wedge$  separatedin X C1 C2)
proof -
  have  $\forall$  C1 C2. S  $\subseteq$  C1  $\cup$  C2  $\longrightarrow$  S  $\cap$  C1 = {}  $\vee$  S  $\cap$  C2 = {}  $\vee$   $\neg$  separatedin
  X C1 C2
  if  $\bigwedge$  C1 C2. C1  $\cup$  C2 = S  $\longrightarrow$  C1 = {}  $\vee$  C2 = {}  $\vee$   $\neg$  separatedin X C1 C2
  proof (intro allI)
    fix C1 C2
    show S  $\subseteq$  C1  $\cup$  C2  $\longrightarrow$  S  $\cap$  C1 = {}  $\vee$  S  $\cap$  C2 = {}  $\vee$   $\neg$  separatedin X C1
    C2
    using that [of S  $\cap$  C1 S  $\cap$  C2]
    by (auto simp: separatedin_mono)
  qed
  then show ?thesis
    by (metis Un_Int_eq(1) Un_Int_eq(2) connectedin_eq_not_separated or-
    der_refl)
qed

```

```

lemma connected_space_eq_not_separated:
  connected_space X  $\longleftrightarrow$ 
    ( $\nexists$  C1 C2. C1  $\cup$  C2 = topspace X  $\wedge$  C1  $\neq$  {}  $\wedge$  C2  $\neq$  {}  $\wedge$  separatedin X C1
    C2)
by (simp add: connectedin_eq_not_separated flip: connectedin_topspace)

```

```

lemma connected_space_eq_not_separated_subset:
  connected_space X  $\longleftrightarrow$ 
    ( $\nexists$  C1 C2. topspace X  $\subseteq$  C1  $\cup$  C2  $\wedge$  C1  $\neq$  {}  $\wedge$  C2  $\neq$  {}  $\wedge$  separatedin X C1
    C2)
by (metis connected_space_eq_not_separated le_sup_iff separatedin_def sub-
    set_antisym)

```

```

lemma connectedin_subset_separated_union:
   $\llbracket$  connectedin X C; separatedin X S T; C  $\subseteq$  S  $\cup$  T  $\rrbracket \implies$  C  $\subseteq$  S  $\vee$  C  $\subseteq$  T
unfolding connectedin_eq_not_separated_subset by blast

```

```

lemma connectedin_nonseparated_union:
  assumes connectedin X S connectedin X T  $\neg$  separatedin X S T

```

```

shows connectedin  $X$   $(S \cup T)$ 
proof -
  have  $\bigwedge C1\ C2. \llbracket T \subseteq C1 \cup C2; S \subseteq C1 \cup C2 \rrbracket \implies$ 
     $S \cap C1 = \{\} \wedge T \cap C1 = \{\} \vee S \cap C2 = \{\} \wedge T \cap C2 = \{\} \vee \neg$ 
separatedin  $X\ C1\ C2$ 
  using assms
  unfolding connectedin_eq_not_separated_subset
  by (metis (no_types, lifting) assms connectedin_subset_separated_union inf.orderE
separatedin_empty(1) separatedin_mono separatedin_sym)
  then show ?thesis
  unfolding connectedin_eq_not_separated_subset
  by (simp add: assms(1) assms(2) connectedin_subset_topspace Int_Un_distrib2)
qed

```

lemma *connected_space_closures*:

```

connected_space  $X \longleftrightarrow$ 
   $(\nexists e1\ e2. e1 \cup e2 = \text{topspace } X \wedge X \text{ closure\_of } e1 \cap X \text{ closure\_of } e2 = \{\})$ 
 $\wedge e1 \neq \{\} \wedge e2 \neq \{\}$ 
  (is ?lhs = ?rhs)

```

```

proof
  assume ?lhs
  then show ?rhs
    unfolding connected_space_closedin_eq
    by (metis Un_upper1 Un_upper2 closedin_closure_of_closure_of_Un closure_of_eq_empty closure_of_topspace)
  next
    assume ?rhs
    then show ?lhs
      unfolding connected_space_closedin_eq
      by (metis closure_of_eq)
qed

```

lemma *connectedin_inter_frontier_of*:

```

assumes connectedin  $X\ S\ S \cap T \neq \{\} \ S - T \neq \{\}$ 
shows  $S \cap X \text{ frontier\_of } T \neq \{\}$ 
proof -
  have  $S \subseteq \text{topspace } X$  and *:
     $\bigwedge E1\ E2. \text{openin } X\ E1 \longrightarrow \text{openin } X\ E2 \longrightarrow E1 \cap E2 \cap S = \{\} \longrightarrow S \subseteq E1$ 
 $\cup E2 \longrightarrow E1 \cap S = \{\} \vee E2 \cap S = \{\}$ 
  using  $\langle \text{connectedin } X\ S \rangle$  by (auto simp: connectedin)
  moreover
    have  $S - (\text{topspace } X \cap T) \neq \{\}$ 
    using assms(3) by blast
  moreover
    have  $S \cap \text{topspace } X \cap T \neq \{\}$ 
    using assms(1) assms(2) connectedin by fastforce
  moreover
    have False if  $S \cap T \neq \{\} \ S - T \neq \{\} \ T \subseteq \text{topspace } X\ S \cap X \text{ frontier\_of } T = \{\}$ 
for  $T$ 

```

```

proof –
  have null:  $S \cap (X \text{ closure\_of } T - X \text{ interior\_of } T) = \{\}$ 
    using that unfolding frontier_of_def by blast
  have  $X \text{ interior\_of } T \cap (\text{topspace } X - X \text{ closure\_of } T) \cap S = \{\}$ 
    by (metis Diff_disjoint inf_bot_left interior_of_Int interior_of_complement
interior_of_empty)
  moreover have  $S \subseteq X \text{ interior\_of } T \cup (\text{topspace } X - X \text{ closure\_of } T)$ 
    using that  $\langle S \subseteq \text{topspace } X \rangle$  null by auto
  moreover have  $S \cap X \text{ interior\_of } T \neq \{\}$ 
    using closure_of_subset that(1) that(3) null by fastforce
  ultimately have  $S \cap X \text{ interior\_of } (\text{topspace } X - T) = \{\}$ 
    by (metis * inf_commute interior_of_complement openin_interior_of)
  then have  $\text{topspace } (\text{subtopology } X S) \cap X \text{ interior\_of } T = S$ 
    using  $\langle S \subseteq \text{topspace } X \rangle$  interior_of_complement null by fastforce
  then show ?thesis
    using that by (metis Diff_eq_empty_iff inf_le2 interior_of_subset subset_trans)
qed
ultimately show ?thesis
  by (metis Int_lower1 frontier_of_restrict inf_assoc)
qed

lemma connectedin_continuous_map_image:
  assumes  $f$ : continuous_map  $X$   $Y$   $f$  and connectedin  $X$   $S$ 
  shows connectedin  $Y$  ( $f \text{ ` } S$ )
proof –
  have  $S \subseteq \text{topspace } X$  and *:
     $\bigwedge E1 E2. \text{openin } X E1 \longrightarrow \text{openin } X E2 \longrightarrow E1 \cap E2 \cap S = \{\} \longrightarrow S \subseteq E1$ 
 $\cup E2 \longrightarrow E1 \cap S = \{\} \vee E2 \cap S = \{\}$ 
    using  $\langle \text{connectedin } X S \rangle$  by (auto simp: connectedin)
  show ?thesis
    unfolding connectedin_connected_space_def
  proof (intro conjI notI; clarify)
    show  $f x \in \text{topspace } Y$  if  $x \in S$  for  $x$ 
      using  $\langle S \subseteq \text{topspace } X \rangle$  continuous_map_image_subset_topspace  $f$  that by
blast
    next
      fix  $U V$ 
      let ? $U = \{x \in \text{topspace } X. f x \in U\}$ 
      let ? $V = \{x \in \text{topspace } X. f x \in V\}$ 
      assume  $UV$ :  $\text{openin } Y U \text{ openin } Y V \text{ } f \text{ ` } S \subseteq U \cup V \text{ } U \cap V \cap f \text{ ` } S = \{\} \text{ } U$ 
 $\cap f \text{ ` } S \neq \{\} \text{ } V \cap f \text{ ` } S \neq \{\}$ 
      then have 1:  $?U \cap ?V \cap S = \{\}$ 
        by auto
      have 2:  $\text{openin } X ?U \text{ openin } X ?V$ 
        using  $\langle \text{openin } Y U \rangle \langle \text{openin } Y V \rangle$  continuous_map  $f$  by fastforce+
      show False
        using * [of ? $U$  ? $V$ ]  $UV$   $\langle S \subseteq \text{topspace } X \rangle$ 
by (auto simp: 1 2)

```

qed
qed

lemma *homeomorphic_connected_space:*

$X \text{ homeomorphic\_space } Y \implies \text{connected\_space } X \longleftrightarrow \text{connected\_space } Y$

unfolding *homeomorphic_space_def homeomorphic_maps_def*

by (*metis connected_space_subconnected connectedin_continuous_map_image connectedin_topspace continuous_map_image_subset_topspace image_eqI image_subset_iff*)

lemma *homeomorphic_map_connectedness:*

assumes $f: \text{homeomorphic\_map } X \ Y \ f$ **and** $U: U \subseteq \text{topspace } X$

shows $\text{connectedin } Y \ (f \, ' \, U) \longleftrightarrow \text{connectedin } X \ U$

proof –

have $1: f \, ' \, U \subseteq \text{topspace } Y \longleftrightarrow U \subseteq \text{topspace } X$

using $U \ f \ \text{homeomorphic\_imp\_surjective\_map}$ **by** *blast*

moreover have $\text{connected\_space } (\text{subtopology } Y \ (f \, ' \, U)) \longleftrightarrow \text{connected\_space } (\text{subtopology } X \ U)$

proof (*rule homeomorphic_connected_space*)

have $f \, ' \, U \subseteq \text{topspace } Y$

by (*simp add: U 1*)

then have $\text{topspace } Y \cap f \, ' \, U = f \, ' \, U$

by (*simp add: subset_antisym*)

then show $\text{subtopology } Y \ (f \, ' \, U) \text{ homeomorphic\_space } \text{subtopology } X \ U$

by (*metis (no_types) Int_subset_iff U f homeomorphic_map_imp_homeomorphic_space homeomorphic_map_subtopologies homeomorphic_space_sym subset_antisym subset_refl*)

qed

ultimately show *?thesis*

by (*auto simp: connectedin_def*)

qed

lemma *homeomorphic_map_connectedness_eq:*

$\text{homeomorphic\_map } X \ Y \ f$

$\implies \text{connectedin } X \ U \longleftrightarrow$

$U \subseteq \text{topspace } X \wedge \text{connectedin } Y \ (f \, ' \, U)$

using *homeomorphic_map_connectedness connectedin_subset_topspace* **by** *metis*

lemma *connectedin_discrete_topology:*

$\text{connectedin } (\text{discrete\_topology } U) \ S \longleftrightarrow S \subseteq U \wedge (\exists a. S \subseteq \{a\})$

proof (*cases* $S \subseteq U$)

case *True*

show *?thesis*

proof (*cases* $S = \{ \}$)

case *False*

moreover have $\text{connectedin } (\text{discrete\_topology } U) \ S \longleftrightarrow (\exists a. S = \{a\})$

proof

show $\text{connectedin } (\text{discrete\_topology } U) \ S \implies \exists a. S = \{a\}$

using *False connectedin_inter_frontier_of_insert_Diff* **by** *fastforce*

qed (*use True in auto*)

```

      ultimately show ?thesis
      by auto
    qed simp
  next
    case False
    then show ?thesis
      by (simp add: connectedin_def)
  qed

```

```

lemma connected_space_discrete_topology:
  connected_space (discrete_topology U)  $\longleftrightarrow$  ( $\exists a. U \subseteq \{a\}$ )
  by (metis connectedin_discrete_topology connectedin_topspace order_refl topspace_discrete_topology)

```

2.2.17 Compact sets

```

definition compactin where
  compactin X S  $\longleftrightarrow$ 
    S  $\subseteq$  topspace X  $\wedge$ 
    ( $\forall \mathcal{U}. (\forall U \in \mathcal{U}. \text{openin } X U) \wedge S \subseteq \bigcup \mathcal{U}$ 
       $\longrightarrow (\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge S \subseteq \bigcup \mathcal{F})$ )

```

```

definition compact_space where
  compact_space X  $\equiv$  compactin X (topspace X)

```

```

lemma compact_space_alt:
  compact_space X  $\longleftrightarrow$ 
    ( $\forall \mathcal{U}. (\forall U \in \mathcal{U}. \text{openin } X U) \wedge \text{topspace } X \subseteq \bigcup \mathcal{U}$ 
       $\longrightarrow (\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge \text{topspace } X \subseteq \bigcup \mathcal{F})$ )
  by (simp add: compact_space_def compactin_def)

```

```

lemma compact_space:
  compact_space X  $\longleftrightarrow$ 
    ( $\forall \mathcal{U}. (\forall U \in \mathcal{U}. \text{openin } X U) \wedge \bigcup \mathcal{U} = \text{topspace } X$ 
       $\longrightarrow (\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge \bigcup \mathcal{F} = \text{topspace } X)$ )
  unfolding compact_space_alt
  using openin_subset by fastforce

```

```

lemma compactinD:
   $\llbracket \text{compactin } X S; \bigwedge U. U \in \mathcal{U} \implies \text{openin } X U; S \subseteq \bigcup \mathcal{U} \rrbracket \implies \exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge S \subseteq \bigcup \mathcal{F}$ 
  by (auto simp: compactin_def)

```

```

lemma compactin_euclidean_iff [simp]: compactin euclidean S  $\longleftrightarrow$  compact S
  by (simp add: compact_eq_Heine_Borel compactin_def) meson

```

```

lemma compactin_absolute [simp]:
  compactin (subtopology X S) S  $\longleftrightarrow$  compactin X S

```

```

proof -
  have eq: ( $\forall U \in \mathcal{U}. \exists Y. \text{openin } X Y \wedge U = Y \cap S$ )  $\longleftrightarrow \mathcal{U} \subseteq (\lambda Y. Y \cap S)$ 

```



```

{y. openin X y} for U
  by auto
  show ?thesis
  by (auto simp: compactin_def openin_subtopology eq_imp_conjL all_subset_image
ex_finite_subset_image)
qed

```

```

lemma compactin_subspace: compactin X S  $\longleftrightarrow$  S  $\subseteq$  topspace X  $\wedge$  compact_space
(subtopology X S)
  unfolding compact_space_def topspace_subtopology
  by (metis compactin_absolute compactin_def inf.absorb2)

```

```

lemma compact_space_subtopology: compactin X S  $\implies$  compact_space (subtopology
X S)
  by (simp add: compactin_subspace)

```

```

lemma compactin_subtopology: compactin (subtopology X S) T  $\longleftrightarrow$  compactin X
T  $\wedge$  T  $\subseteq$  S
  by (metis compactin_subspace inf.absorb_iff2 le_inf_iff subtopology_subtopology
topspace_subtopology)

```

```

lemma compactin_subset_topspace: compactin X S  $\implies$  S  $\subseteq$  topspace X
  by (simp add: compactin_subspace)

```

```

lemma compactin_contractive:
   $\llbracket$ compactin X' S; topspace X' = topspace X;
 $\bigwedge$  U. openin X U  $\implies$  openin X' U $\rrbracket \implies$  compactin X S
  by (simp add: compactin_def)

```

```

lemma finite_imp_compactin:
   $\llbracket$ S  $\subseteq$  topspace X; finite S $\rrbracket \implies$  compactin X S
  by (metis compactin_subspace compact_space_finite_UnionD inf.absorb_iff2 or-
der_refl topspace_subtopology)

```

```

lemma compactin_empty [iff]: compactin X {}
  by (simp add: finite_imp_compactin)

```

```

lemma compact_space_topspace_empty:
  topspace X = {}  $\implies$  compact_space X
  by (simp add: compact_space_def)

```

```

lemma finite_imp_compactin_eq:
  finite S  $\implies$  (compactin X S  $\longleftrightarrow$  S  $\subseteq$  topspace X)
  using compactin_subset_topspace finite_imp_compactin by blast

```

```

lemma compactin_sing [simp]: compactin X {a}  $\longleftrightarrow$  a  $\in$  topspace X
  by (simp add: finite_imp_compactin_eq)

```

```

lemma closed_compactin:

```

```

    assumes  $XK$ : compactin  $X$   $K$  and  $C \subseteq K$  and  $XC$ : closedin  $X$   $C$ 
    shows compactin  $X$   $C$ 
    unfolding compactin_def
  proof (intro conjI allI impI)
    show  $C \subseteq \text{topspace } X$ 
      by (simp add:  $XC$  closedin_subset)
  next
    fix  $\mathcal{U} :: 'a \text{ set set}$ 
    assume  $\mathcal{U}$ :  $\text{Ball } \mathcal{U} (\text{openin } X) \wedge C \subseteq \bigcup \mathcal{U}$ 
    have  $(\forall U \in \text{insert } (\text{topspace } X - C) \mathcal{U}. \text{openin } X U)$ 
      using  $XC$   $\mathcal{U}$  by blast
    moreover have  $K \subseteq \bigcup (\text{insert } (\text{topspace } X - C) \mathcal{U})$ 
      using  $\mathcal{U}$   $XK$  compactin_subset_topspace by fastforce
    ultimately obtain  $\mathcal{F}$  where finite  $\mathcal{F}$   $\mathcal{F} \subseteq \text{insert } (\text{topspace } X - C) \mathcal{U}$   $K \subseteq \bigcup \mathcal{F}$ 
      using assms unfolding compactin_def by metis
    moreover have openin  $X$   $(\text{topspace } X - C)$ 
      using  $XC$  by auto
    ultimately show  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge C \subseteq \bigcup \mathcal{F}$ 
      using  $\langle C \subseteq K \rangle$ 
      by (rule_tac  $x = \mathcal{F} - \{\text{topspace } X - C\}$  in exI) auto
  qed

```

```

lemma closedin_compact_space:
   $\llbracket \text{compact\_space } X; \text{closedin } X S \rrbracket \implies \text{compactin } X S$ 
  by (simp add: closed_compactin closedin_subset compact_space_def)

```

```

lemma compact_Int_closedin:
  assumes compactin  $X$   $S$  closedin  $X$   $T$  shows compactin  $X$   $(S \cap T)$ 
  proof -
    have compactin (subtopology  $X$   $S$ )  $(S \cap T)$ 
      by (metis assms closedin_compact_space closedin_subtopology compactin_subspace
        inf_commute)
    then show ?thesis
      by (simp add: compactin_subtopology)
  qed

```

```

lemma closed_Int_compactin:  $\llbracket \text{closedin } X S; \text{compactin } X T \rrbracket \implies \text{compactin } X (S \cap T)$ 
  by (metis compact_Int_closedin inf_commute)

```

```

lemma compactin_Un:
  assumes  $S$ : compactin  $X$   $S$  and  $T$ : compactin  $X$   $T$  shows compactin  $X$   $(S \cup T)$ 
  unfolding compactin_def
  proof (intro conjI allI impI)
    show  $S \cup T \subseteq \text{topspace } X$ 
      using assms by (auto simp: compactin_def)
  next
    fix  $\mathcal{U} :: 'a \text{ set set}$ 

```

```

assume  $\mathcal{U}$ : Ball  $\mathcal{U}$  (openin  $X$ )  $\wedge S \cup T \subseteq \bigcup \mathcal{U}$ 
with  $S$  obtain  $\mathcal{F}$  where  $\mathcal{V}$ : finite  $\mathcal{F}$   $\mathcal{F} \subseteq \mathcal{U}$   $S \subseteq \bigcup \mathcal{F}$ 
  unfolding compactin_def by (meson sup.bounded_iff)
obtain  $\mathcal{W}$  where finite  $\mathcal{W}$   $\mathcal{W} \subseteq \mathcal{U}$   $T \subseteq \bigcup \mathcal{W}$ 
  using  $\mathcal{U}$   $T$ 
  unfolding compactin_def by (meson sup.bounded_iff)
with  $\mathcal{V}$  show  $\exists \mathcal{V}. \text{finite } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge S \cup T \subseteq \bigcup \mathcal{V}$ 
  by (rule_tac  $x = \mathcal{F} \cup \mathcal{W}$  in exI) auto
qed

```

lemma compactin_Union:

```

 $\llbracket \text{finite } \mathcal{F}; \bigwedge S. S \in \mathcal{F} \implies \text{compactin } X S \rrbracket \implies \text{compactin } X (\bigcup \mathcal{F})$ 
by (induction rule: finite_induct) (simp_all add: compactin_Un)

```

lemma compactin_subtopology_imp_compact:

```

assumes compactin (subtopology  $X$   $S$ )  $K$  shows compactin  $X$   $K$ 
using assms
proof (clarsimp simp add: compactin_def)
  fix  $\mathcal{U}$ 
  define  $\mathcal{V}$  where  $\mathcal{V} \equiv (\lambda U. U \cap S) \text{ ` } \mathcal{U}$ 
  assume  $K \subseteq \text{topspace } X$  and  $K \subseteq S$  and  $\forall x \in \mathcal{U}. \text{openin } X x$  and  $K \subseteq \bigcup \mathcal{U}$ 
  then have  $\forall V \in \mathcal{V}. \text{openin } (\text{subtopology } X S) V$   $K \subseteq \bigcup \mathcal{V}$ 
    unfolding  $\mathcal{V}$ _def by (auto simp: openin_subtopology)
  moreover
  assume  $\forall \mathcal{U}. (\forall x \in \mathcal{U}. \text{openin } (\text{subtopology } X S) x) \wedge K \subseteq \bigcup \mathcal{U} \longrightarrow (\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge K \subseteq \bigcup \mathcal{F})$ 
  ultimately obtain  $\mathcal{F}$  where finite  $\mathcal{F}$   $\mathcal{F} \subseteq \mathcal{V}$   $K \subseteq \bigcup \mathcal{F}$ 
    by meson
  then have  $\mathcal{F}$ :  $\exists U. U \in \mathcal{U} \wedge V = U \cap S$  if  $V \in \mathcal{F}$  for  $V$ 
    unfolding  $\mathcal{V}$ _def using that by blast
  let  $?F = (\lambda F. @U. U \in \mathcal{U} \wedge F = U \cap S) \text{ ` } \mathcal{F}$ 
  show  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge K \subseteq \bigcup \mathcal{F}$ 
  proof (intro exI conjI)
    show finite  $?F$ 
      using  $\langle \text{finite } \mathcal{F} \rangle$  by blast
    show  $?F \subseteq \mathcal{U}$ 
      using someI_ex [OF  $\mathcal{F}$ ] by blast
    show  $K \subseteq \bigcup ?F$ 
      using  $K \subseteq \bigcup \mathcal{F}$  by blast
  proof clarsimp
    fix  $x$ 
    assume  $x \in K$ 
    then show  $\exists V \in \mathcal{F}. x \in (\text{SOME } U. U \in \mathcal{U} \wedge V = U \cap S)$ 
      using  $\langle K \subseteq \bigcup \mathcal{F} \rangle$  someI_ex [OF  $\mathcal{F}$ ]
      by (metis (no_types, lifting) IntD1 Union_iff subsetCE)
  qed
qed
qed
qed

```

lemma compact_imp_compactin_subtopology:

```

    assumes compactin X K  $K \subseteq S$  shows compactin (subtopology X S) K
    using assms
  proof (clarsimp simp add: compactin_def)
    fix  $\mathcal{U} :: 'a \text{ set set}$ 
    define  $\mathcal{V}$  where  $\mathcal{V} \equiv \{V. \text{openin } X \ V \wedge (\exists U \in \mathcal{U}. U = V \cap S)\}$ 
    assume  $K \subseteq S$  and  $K \subseteq \text{topspace } X$  and  $\forall U \in \mathcal{U}. \text{openin } (\text{subtopology } X \ S) \ U$ 
    and  $K \subseteq \bigcup \mathcal{U}$ 
    then have  $\forall V \in \mathcal{V}. \text{openin } X \ V \ K \subseteq \bigcup \mathcal{V}$ 
      unfolding  $\mathcal{V\_def}$  by (fastforce simp: subset_eq openin_subtopology)+
    moreover
    assume  $\forall \mathcal{U}. (\forall U \in \mathcal{U}. \text{openin } X \ U) \wedge K \subseteq \bigcup \mathcal{U} \longrightarrow (\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge$ 
 $K \subseteq \bigcup \mathcal{F})$ 
    ultimately obtain  $\mathcal{F}$  where  $\text{finite } \mathcal{F} \ \mathcal{F} \subseteq \mathcal{V} \ K \subseteq \bigcup \mathcal{F}$ 
    by meson
    let  $?F = (\lambda F. F \cap S) \ ` \ \mathcal{F}$ 
    show  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge K \subseteq \bigcup \mathcal{F}$ 
    proof (intro exI conjI)
      show finite  $?F$ 
      using  $\langle \text{finite } \mathcal{F} \rangle$  by blast
      show  $?F \subseteq \mathcal{U}$ 
      using  $\mathcal{V\_def} \ \langle \mathcal{F} \subseteq \mathcal{V} \rangle$  by blast
      show  $K \subseteq \bigcup ?F$ 
      using  $\langle K \subseteq \bigcup \mathcal{F} \rangle$  assms(2) by auto
    qed
  qed

```

```

proposition compact_space_fip:
  compact_space X  $\longleftrightarrow$ 
    ( $\forall \mathcal{U}. (\forall C \in \mathcal{U}. \text{closedin } X \ C) \wedge (\forall \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \longrightarrow \bigcap \mathcal{F} \neq \{\}) \longrightarrow$ 
 $\bigcap \mathcal{U} \neq \{\})$ 
    (is _ = ?rhs)
  proof (cases topspace X =  $\{\}$ )
    case True
    then show ?thesis
    unfolding compact_space_def
    by (metis Sup_bot_conv(1) closedin_topspace_empty compactin_empty finite.emptyI
    finite_UnionD order_refl)
  next
    case False
    show ?thesis
    proof safe
      fix  $\mathcal{U} :: 'a \text{ set set}$ 
      assume * [rule_format]:  $\forall \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \longrightarrow \bigcap \mathcal{F} \neq \{\}$ 
      define  $\mathcal{V}$  where  $\mathcal{V} \equiv (\lambda S. \text{topspace } X - S) \ ` \ \mathcal{U}$ 
      assume clo:  $\forall C \in \mathcal{U}. \text{closedin } X \ C$  and [simp]:  $\bigcap \mathcal{U} = \{\}$ 
      then have  $\forall V \in \mathcal{V}. \text{openin } X \ V \ \text{topspace } X \subseteq \bigcup \mathcal{V}$ 
        by (auto simp:  $\mathcal{V\_def}$ )
      moreover assume [unfolded compact_space_alt, rule_format, of  $\mathcal{V}$ ]: com-

```

```

pact_space X
ultimately obtain  $\mathcal{F}$  where  $\mathcal{F}$ : finite  $\mathcal{F}$   $\mathcal{F} \subseteq \mathcal{U}$   $\text{topspace } X \subseteq \text{topspace } X - \bigcap \mathcal{F}$ 
  by (auto simp: ex_finite_subset_image  $\mathcal{V\_def}$ )
moreover have  $\mathcal{F} \neq \{\}$ 
  using  $\mathcal{F} \langle \text{topspace } X \neq \{\} \rangle$  by blast
ultimately show False
  using * [of  $\mathcal{F}$ ]
  by auto (metis Diff_iff Inter_iff clo closedin_def subsetD)
next
assume  $R$  [rule_format]: ?rhs
show compact_space X
  unfolding compact_space_alt
proof clarify
  fix  $\mathcal{U} :: 'a \text{ set set}$ 
  define  $\mathcal{V}$  where  $\mathcal{V} \equiv (\lambda S. \text{topspace } X - S) \text{ ` } \mathcal{U}$ 
  assume  $\forall C \in \mathcal{U}. \text{openin } X \ C$  and  $\text{topspace } X \subseteq \bigcup \mathcal{U}$ 
  with  $\langle \text{topspace } X \neq \{\} \rangle$  have *:  $\forall V \in \mathcal{V}. \text{closedin } X \ V$   $\mathcal{U} \neq \{\}$ 
    by (auto simp:  $\mathcal{V\_def}$ )
  show  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge \text{topspace } X \subseteq \bigcup \mathcal{F}$ 
  proof (rule ccontr; simp)
    assume  $\forall \mathcal{F} \subseteq \mathcal{U}. \text{finite } \mathcal{F} \longrightarrow \neg \text{topspace } X \subseteq \bigcup \mathcal{F}$ 
    then have  $\forall \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{V} \longrightarrow \bigcap \mathcal{F} \neq \{\}$ 
      by (simp add:  $\mathcal{V\_def}$  all_finite_subset_image)
    with  $\langle \text{topspace } X \subseteq \bigcup \mathcal{U} \rangle$  show False
      using  $R$  [of  $\mathcal{V}$ ] * by (simp add:  $\mathcal{V\_def}$ )
  qed
qed
qed
qed
qed

corollary compactin_fip:
  compactin  $X \ S \longleftrightarrow$ 
     $S \subseteq \text{topspace } X \wedge$ 
     $(\forall \mathcal{U}. (\forall C \in \mathcal{U}. \text{closedin } X \ C) \wedge (\forall \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \longrightarrow S \cap \bigcap \mathcal{F} \neq \{\}) \longrightarrow$ 
 $S \cap \bigcap \mathcal{U} \neq \{\})$ 
proof (cases  $S = \{\}$ )
  case False
  show ?thesis
proof (cases  $S \subseteq \text{topspace } X$ )
  case True
  then have compactin  $X \ S \longleftrightarrow$ 
     $(\forall \mathcal{U}. \mathcal{U} \subseteq (\lambda T. S \cap T) \text{ ` } \{T. \text{closedin } X \ T\} \longrightarrow$ 
     $(\forall \mathcal{F}. \text{finite } \mathcal{F} \longrightarrow \mathcal{F} \subseteq \mathcal{U} \longrightarrow \bigcap \mathcal{F} \neq \{\}) \longrightarrow \bigcap \mathcal{U} \neq \{\})$ 
    by (simp add: compact_space_fip compactin_subspace closedin_subtopology
    image_def subset_eq Int_commute imp_conjL)
  also have ... =  $(\forall \mathcal{U} \subseteq \text{Collect } (\text{closedin } X). (\forall \mathcal{F}. \text{finite } \mathcal{F} \longrightarrow \mathcal{F} \subseteq (\bigcap) S \text{ ` } \mathcal{U} \longrightarrow \bigcap \mathcal{F} \neq \{\}) \longrightarrow \bigcap ((\bigcap) S \text{ ` } \mathcal{U}) \neq \{\})$ 
    by (simp add: all_subset_image)

```

also have $\dots = (\forall \mathcal{U}. (\forall C \in \mathcal{U}. \text{closedin } X \ C) \wedge (\forall \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \longrightarrow S \cap \bigcap \mathcal{F} \neq \{\}) \longrightarrow S \cap \bigcap \mathcal{U} \neq \{\})$
proof –
have $\text{eq}: ((\forall \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \longrightarrow \bigcap ((\bigcap) S \text{ ‘ } \mathcal{F}) \neq \{\}) \longrightarrow \bigcap ((\bigcap) S \text{ ‘ } \mathcal{U}) \neq \{\}) \longleftrightarrow ((\forall \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \longrightarrow S \cap \bigcap \mathcal{F} \neq \{\}) \longrightarrow S \cap \bigcap \mathcal{U} \neq \{\})$ **for** $\mathcal{U}$
by *simp* (use $\langle S \neq \{\} \rangle$ **in** *blast*)
show *?thesis*
unfolding *imp_conjL* [*symmetric*] *all_finite_subset_image* *eq* **by** *blast*
qed
finally show *?thesis*
using *True* **by** *simp*
qed (*simp* *add: compactin_subspace*)
qed *force*

corollary *compact_space_imp_nest*:
fixes $C :: \text{nat} \Rightarrow 'a \text{ set}$
assumes *compact_space* X **and** *clo*: $\bigwedge n. \text{closedin } X \ (C \ n)$
and *ne*: $\bigwedge n. C \ n \neq \{\}$ **and** *inc*: $\bigwedge^m n. m \leq n \implies C \ n \subseteq C \ m$
shows $(\bigcap n. C \ n) \neq \{\}$
proof –
let $\mathcal{U} = \text{range } (\lambda n. \bigcap m \leq n. C \ m)$
have *closedin* $X \ A$ **if** $A \in \mathcal{U}$ **for** A
using *that clo* **by** *auto*
moreover have $(\bigcap n \in K. \bigcap m \leq n. C \ m) \neq \{\}$ **if** *finite* K **for** K
proof –
obtain n **where** $\bigwedge k. k \in K \implies k \leq n$
using *Max.coboundedI* $\langle \text{finite } K \rangle$ **by** *blast*
with *inc* **have** $C \ n \subseteq (\bigcap n \in K. \bigcap m \leq n. C \ m)$
by *blast*
with *ne* [*of n*] **show** *?thesis*
by *blast*
qed
ultimately show *?thesis*
using $\langle \text{compact\_space } X \rangle$ [*unfolded compact_space_fip*, *rule_format*, *of* $\mathcal{U}$]
by (*simp* *add: all_finite_subset_image INT_extend_simps UN_atMost_UNIV del: INT_simps*)
qed

lemma *compactin_discrete_topology*:
 $\text{compactin } (\text{discrete\_topology } X) \ S \longleftrightarrow S \subseteq X \wedge \text{finite } S$ (**is** *?lhs* = *?rhs*)
proof (*intro iffI conjI*)
assume $L: ?lhs$
then show $S \subseteq X$
by (*auto simp: compactin_def*)
have $*$: $\bigwedge \mathcal{U}. \text{Ball } \mathcal{U} \ (\text{openin } (\text{discrete\_topology } X)) \wedge S \subseteq \bigcup \mathcal{U} \implies (\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge S \subseteq \bigcup \mathcal{F})$
using L **by** (*auto simp: compactin_def*)

```

show finite S
  using * [of ( $\lambda x. \{x\}$ ) 'X']  $\langle S \subseteq X \rangle$ 
  by clarsimp (metis UN_singleton finite_subset_image infinite_super)
next
  assume ?rhs
  then show ?lhs
    by (simp add: finite_imp_compactin)
qed

```

```

lemma compact_space_discrete_topology: compact_space(discrete_topology X)  $\longleftrightarrow$ 
finite X
  by (simp add: compactin_discrete_topology compact_space_def)

```

```

lemma compact_space_imp_Bolzano_Weierstrass:
  assumes compact_space X infinite S  $S \subseteq \text{topspace } X$ 
  shows X derived_set_of S  $\neq \{\}$ 
proof
  assume X: X derived_set_of S =  $\{\}$ 
  then have closedin X S
    by (simp add: closedin_contains_derived_set assms)
  then have compactin X S
    by (rule closedin_compact_space [OF  $\langle \text{compact\_space } X \rangle$ ])
  with X show False
    by (metis  $\langle \text{infinite } S \rangle$  compactin_subspace compact_space_discrete_topology
inf_bot_right subtopology_eq_discrete_topology_eq)
qed

```

```

lemma compactin_imp_Bolzano_Weierstrass:
   $\llbracket \text{compactin } X S; \text{infinite } T \wedge T \subseteq S \rrbracket \implies S \cap X \text{ derived\_set\_of } T \neq \{\}$ 
  using compact_space_imp_Bolzano_Weierstrass [of subtopology X S]
  by (simp add: compactin_subspace derived_set_of_subtopology inf_absorb2)

```

```

lemma compact_closure_of_imp_Bolzano_Weierstrass:
   $\llbracket \text{compactin } X (X \text{ closure\_of } S); \text{infinite } T; T \subseteq S; T \subseteq \text{topspace } X \rrbracket \implies X$ 
   $\text{derived\_set\_of } T \neq \{\}$ 
  using closure_of_mono closure_of_subset compactin_imp_Bolzano_Weierstrass
  by fastforce

```

```

lemma discrete_compactin_eq_finite:
   $S \cap X \text{ derived\_set\_of } S = \{\} \implies \text{compactin } X S \longleftrightarrow S \subseteq \text{topspace } X \wedge \text{finite } S$ 
  by (meson compactin_imp_Bolzano_Weierstrass finite_imp_compactin_eq order_refl)

```

```

lemma discrete_compact_space_eq_finite:
   $X \text{ derived\_set\_of } (\text{topspace } X) = \{\} \implies (\text{compact\_space } X \longleftrightarrow \text{finite}(\text{topspace } X))$ 
  by (metis compact_space_discrete_topology discrete_topology_unique_derived_set)

```

```

lemma image_compactin:
  assumes cpt: compactin X S and cont: continuous_map X Y f
  shows compactin Y (f ' S)
  unfolding compactin_def
proof (intro conjI allI impI)
  show f ' S  $\subseteq$  topspace Y
  using compactin_subset_topspace cont continuous_map_image_subset_topspace
cpt by blast
next
  fix  $\mathcal{U} :: 'b \text{ set set}$ 
  assume  $\mathcal{U}$ : Ball  $\mathcal{U}$  (openin Y)  $\wedge f ' S \subseteq \bigcup \mathcal{U}$ 
  define  $\mathcal{V}$  where  $\mathcal{V} \equiv (\lambda U. \{x \in \text{topspace } X. f x \in U\}) ' \mathcal{U}$ 
  have  $S \subseteq \text{topspace } X$ 
  and *:  $\bigwedge \mathcal{U}. [\forall U \in \mathcal{U}. \text{openin } X U; S \subseteq \bigcup \mathcal{U}] \implies \exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge S$ 
 $\subseteq \bigcup \mathcal{F}$ 
  using cpt by (auto simp: compactin_def)
  obtain  $\mathcal{F}$  where  $\mathcal{F}$ : finite  $\mathcal{F}$   $\mathcal{F} \subseteq \mathcal{V}$   $S \subseteq \bigcup \mathcal{F}$ 
proof -
  have 1:  $\forall U \in \mathcal{V}. \text{openin } X U$ 
  unfolding  $\mathcal{V}$ _def using  $\mathcal{U}$  cont[unfolded continuous_map] by blast
  have 2:  $S \subseteq \bigcup \mathcal{V}$ 
  unfolding  $\mathcal{V}$ _def using compactin_subset_topspace cpt  $\mathcal{U}$  by fastforce
  show thesis
  using * [OF 1 2] that by metis
qed
have  $\forall v \in \mathcal{V}. \exists U. U \in \mathcal{U} \wedge v = \{x \in \text{topspace } X. f x \in U\}$ 
  using  $\mathcal{V}$ _def by blast
then obtain U where U:  $\forall v \in \mathcal{V}. U v \in \mathcal{U} \wedge v = \{x \in \text{topspace } X. f x \in U v\}$ 
  by metis
show  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge f ' S \subseteq \bigcup \mathcal{F}$ 
proof (intro conjI exI)
  show finite (U '  $\mathcal{F}$ )
  by (simp add:  $\langle \text{finite } \mathcal{F} \rangle$ )
next
  show U '  $\mathcal{F} \subseteq \mathcal{U}$ 
  using  $\langle \mathcal{F} \subseteq \mathcal{V} \rangle$  U by auto
next
  show  $f ' S \subseteq \bigcup (U ' \mathcal{F})$ 
  using  $\mathcal{F}(2-3)$  U UnionE subset_eq U by fastforce
qed
qed

```

```

lemma homeomorphic_compact_space:
  assumes X homeomorphic_space Y
  shows compact_space X  $\longleftrightarrow$  compact_space Y
  using homeomorphic_space_sym
  by (metis assms compact_space_def homeomorphic_eq_everything_map home-
omorphi-compactin)

```



```

lemma homeomorphic_map_compactness:
  assumes hom: homeomorphic_map X Y f and U:  $U \subseteq \text{topspace } X$ 
  shows  $\text{compactin } Y (f \text{ ` } U) \longleftrightarrow \text{compactin } X U$ 
proof –
  have  $f \text{ ` } U \subseteq \text{topspace } Y$ 
  using hom U homeomorphic_imp_surjective_map by blast
  moreover have homeomorphic_map (subtopology X U) (subtopology Y ( $f \text{ ` } U$ ))
  f
  using U hom homeomorphic_imp_surjective_map by (blast intro: homeomor-
phic_map_subtopologies)
  then have  $\text{compact\_space (subtopology } Y (f \text{ ` } U)) = \text{compact\_space (subtopology } X U)$ 
  using homeomorphic_compact_space homeomorphic_map_imp_homeomorphic_space
by blast
  ultimately show ?thesis
  by (simp add: compactin_subspace U)
qed

```

```

lemma homeomorphic_map_compactness_eq:
  homeomorphic_map X Y f
   $\implies \text{compactin } X U \longleftrightarrow U \subseteq \text{topspace } X \wedge \text{compactin } Y (f \text{ ` } U)$ 
  by (meson compactin_subset_topspace homeomorphic_map_compactness)

```

2.2.18 Embedding maps

```

definition embedding_map
  where embedding_map X Y f  $\equiv \text{homeomorphic\_map } X (\text{subtopology } Y (f \text{ ` } (\text{topspace } X))) f$ 

```

```

lemma embedding_map_eq:
   $\llbracket \text{embedding\_map } X Y f; \bigwedge x. x \in \text{topspace } X \implies f x = g x \rrbracket \implies \text{embedding\_map } X Y g$ 
  unfolding embedding_map_def
  by (metis homeomorphic_map_eq image_cong)

```

```

lemma embedding_map_compose:
  assumes embedding_map X X' f embedding_map X' X'' g
  shows embedding_map X X'' ( $g \circ f$ )
proof –
  have hm: homeomorphic_map X (subtopology X' ( $f \text{ ` } \text{topspace } X$ )) f homeomor-
phic_map X' (subtopology X'' ( $g \text{ ` } \text{topspace } X'$ )) g
  using assms by (auto simp: embedding_map_def)
  then obtain C where  $g \text{ ` } \text{topspace } X' \cap C = (g \circ f) \text{ ` } \text{topspace } X$ 
  by (metis (no_types) Int_absorb1 continuous_map_image_subset_topspace
continuous_map_in_subtopology homeomorphic_eq_everything_map image_comp
image_mono)
  then have homeomorphic_map (subtopology X' ( $f \text{ ` } \text{topspace } X$ )) (subtopology
X'' ( $(g \circ f) \text{ ` } \text{topspace } X$ )) g

```

```

    by (metis hm homeomorphic_imp_surjective_map homeomorphic_map_subtopologies
image_comp subtopology_subtopology topspace_subtopology)
    then show ?thesis
    unfolding embedding_map_def
    using hm(1) homeomorphic_map_compose by blast
qed

```

```

lemma surjective_embedding_map:
  embedding_map X Y f  $\wedge$  f ' (topspace X) = topspace Y  $\longleftrightarrow$  homeomorphic_map
X Y f
  by (force simp: embedding_map_def homeomorphic_eq_everything_map)

```

```

lemma embedding_map_in_subtopology:
  embedding_map X (subtopology Y S) f  $\longleftrightarrow$  embedding_map X Y f  $\wedge$  f ' (topspace
X)  $\subseteq$  S (is ?lhs = ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
    unfolding embedding_map_def
    by (metis continuous_map_in_subtopology homeomorphic_imp_continuous_map
inf_absorb2 subtopology_subtopology)
qed (simp add: embedding_map_def inf_absorb_iff2 subtopology_subtopology)

```

```

lemma injective_open_imp_embedding_map:
   $\llbracket$ continuous_map X Y f; open_map X Y f; inj_on f (topspace X) $\rrbracket \implies$  embed-
ding_map X Y f
  unfolding embedding_map_def
  by (simp add: continuous_map_in_subtopology continuous_open_quotient_map
eq_iff homeomorphic_map_def open_map_imp_subset open_map_into_subtopology)

```

```

lemma injective_closed_imp_embedding_map:
   $\llbracket$ continuous_map X Y f; closed_map X Y f; inj_on f (topspace X) $\rrbracket \implies$  embed-
ding_map X Y f
  unfolding embedding_map_def
  by (simp add: closed_map_imp_subset closed_map_into_subtopology continu-
ous_closed_quotient_map
continuous_map_in_subtopology dual_order.eq_iff homeomor-
phic_map_def)

```

```

lemma embedding_map_imp_homeomorphic_space:
  embedding_map X Y f  $\implies$  X homeomorphic_space (subtopology Y (f ' (topspace
X)))
  unfolding embedding_map_def
  using homeomorphic_space by blast

```

```

lemma embedding_imp_closed_map:
   $\llbracket$ embedding_map X Y f; closedin Y (f ' topspace X) $\rrbracket \implies$  closed_map X Y f
  unfolding closed_map_def
  by (auto simp: closedin_closed_subtopology embedding_map_def homeomorphic_map_closedness_eq)

```

2.2.19 Retraction and section maps

definition *retraction_maps* :: $'a \text{ topology} \Rightarrow 'b \text{ topology} \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow \text{bool}$

where *retraction_maps* $X Y f g \equiv$
 $\text{continuous\_map } X Y f \wedge \text{continuous\_map } Y X g \wedge (\forall x \in \text{topspace } Y. f(g x) = x)$

definition *section_map* :: $'a \text{ topology} \Rightarrow 'b \text{ topology} \Rightarrow ('a \Rightarrow 'b) \Rightarrow \text{bool}$

where *section_map* $X Y f \equiv \exists g. \text{retraction\_maps } Y X g f$

definition *retraction_map* :: $'a \text{ topology} \Rightarrow 'b \text{ topology} \Rightarrow ('a \Rightarrow 'b) \Rightarrow \text{bool}$

where *retraction_map* $X Y f \equiv \exists g. \text{retraction\_maps } X Y f g$

lemma *retraction_maps_eq*:

$\llbracket \text{retraction\_maps } X Y f g; \bigwedge x. x \in \text{topspace } X \Longrightarrow f x = f' x; \bigwedge x. x \in \text{topspace } Y \Longrightarrow g x = g' x \rrbracket$

$\Longrightarrow \text{retraction\_maps } X Y f' g'$

unfolding *retraction_maps_def* **by** (metis (no_types, lifting) *continuous_map_def continuous_map_eq*)

lemma *section_map_eq*:

$\llbracket \text{section\_map } X Y f; \bigwedge x. x \in \text{topspace } X \Longrightarrow f x = g x \rrbracket \Longrightarrow \text{section\_map } X Y g$

unfolding *section_map_def* **using** *retraction_maps_eq* **by** blast

lemma *retraction_map_eq*:

$\llbracket \text{retraction\_map } X Y f; \bigwedge x. x \in \text{topspace } X \Longrightarrow f x = g x \rrbracket \Longrightarrow \text{retraction\_map } X Y g$

unfolding *retraction_map_def* **using** *retraction_maps_eq* **by** blast

lemma *homeomorphic_imp_retraction_maps*:

$\text{homeomorphic\_maps } X Y f g \Longrightarrow \text{retraction\_maps } X Y f g$

by (simp add: *homeomorphic_maps_def* *retraction_maps_def*)

lemma *section_and_retraction_eq_homeomorphic_map*:

$\text{section\_map } X Y f \wedge \text{retraction\_map } X Y f \longleftrightarrow \text{homeomorphic\_map } X Y f$
(is ?lhs = ?rhs)

proof

assume ?lhs

then obtain $g g'$ **where** $f: \text{continuous\_map } X Y f$

and $g: \text{continuous\_map } Y X g \forall x \in \text{topspace } X. g(f x) = x$

and $g': \text{continuous\_map } Y X g' \forall x \in \text{topspace } Y. f(g' x) = x$

by (auto simp: *retraction_map_def* *retraction_maps_def* *section_map_def*)

then have *homeomorphic_maps* $X Y f g$

by (force simp add: *homeomorphic_maps_def* *continuous_map_def*)

then show ?rhs

using *homeomorphic_map_maps* **by** blast

next

assume ?rhs

```

then show ?lhs
  unfolding retraction_map_def section_map_def
  by (meson homeomorphic_imp_retraction_maps homeomorphic_map_maps
homeomorphic_maps_sym)
qed

```

```

lemma section_imp_embedding_map:
  section_map X Y f  $\implies$  embedding_map X Y f
  unfolding section_map_def embedding_map_def homeomorphic_map_maps re-
traction_maps_def homeomorphic_maps_def
  by (force simp: continuous_map_in_subtopology continuous_map_from_subtopology)

```

```

lemma retraction_imp_quotient_map:
  assumes retraction_map X Y f
  shows quotient_map X Y f
  unfolding quotient_map_def
proof (intro conjI subsetI allI impI)
  show f 'topspace X = topspace Y
    using assms by (force simp: retraction_map_def retraction_maps_def contin-
uous_map_def)
next
  fix U
  assume U:  $U \subseteq \text{topspace } Y$ 
  have openin Y U
    if  $\forall x \in \text{topspace } Y. g\ x \in \text{topspace } X \ \forall x \in \text{topspace } Y. f\ (g\ x) = x$ 
    openin Y  $\{x \in \text{topspace } Y. g\ x \in \{x \in \text{topspace } X. f\ x \in U\}\}$  for g
  using openin_subopen U that by fastforce
  then show openin X  $\{x \in \text{topspace } X. f\ x \in U\} = \text{openin } Y\ U$ 
    using assms by (auto simp: retraction_map_def retraction_maps_def contin-
uous_map_def)
qed

```

```

lemma retraction_maps_compose:
   $\llbracket \text{retraction\_maps } X\ Y\ f\ f'; \text{retraction\_maps } Y\ Z\ g\ g' \rrbracket \implies \text{retraction\_maps } X\ Z\ (g \circ f)\ (f' \circ g')$ 
  by (clarsimp simp: retraction_maps_def continuous_map_compose) (simp add: continuous_map_def)

```

```

lemma retraction_map_compose:
   $\llbracket \text{retraction\_map } X\ Y\ f; \text{retraction\_map } Y\ Z\ g \rrbracket \implies \text{retraction\_map } X\ Z\ (g \circ f)$ 
  by (meson retraction_map_def retraction_maps_compose)

```

```

lemma section_map_compose:
   $\llbracket \text{section\_map } X\ Y\ f; \text{section\_map } Y\ Z\ g \rrbracket \implies \text{section\_map } X\ Z\ (g \circ f)$ 
  by (meson retraction_maps_compose section_map_def)

```

```

lemma surjective_section_eq_homeomorphic_map:
  section_map X Y f  $\wedge$  f 'topspace X = topspace Y  $\longleftrightarrow$  homeomorphic_map

```

$X \ Y \ f$

by (*meson section_and_retraction_eq_homeomorphic_map section_imp_embedding_map surjective_embedding_map*)

lemma *surjective_retraction_or_section_map*:

$f \text{ ' } (topspace \ X) = topspace \ Y \implies retraction\_map \ X \ Y \ f \vee section\_map \ X \ Y$
 $f \longleftrightarrow retraction\_map \ X \ Y \ f$

using *section_and_retraction_eq_homeomorphic_map surjective_section_eq_homeomorphic_map*
by *fastforce*

lemma *retraction_imp_surjective_map*:

$retraction\_map \ X \ Y \ f \implies f \text{ ' } (topspace \ X) = topspace \ Y$

by (*simp add: retraction_imp_quotient_map quotient_imp_surjective_map*)

lemma *section_imp_injective_map*:

$\llbracket section\_map \ X \ Y \ f; x \in topspace \ X; y \in topspace \ X \rrbracket \implies f \ x = f \ y \longleftrightarrow x = y$

by (*metis (mono_tags, opaque_lifting) retraction_maps_def section_map_def*)

lemma *retraction_maps_to_retract_maps*:

$retraction\_maps \ X \ Y \ r \ s$

$\implies retraction\_maps \ X \ (subtopology \ X \ (s \text{ ' } (topspace \ Y))) \ (s \circ r) \ id$

unfolding *retraction_maps_def*

by (*auto simp: continuous_map_compose continuous_map_into_subtopology continuous_map_from_subtopology*)

2.2.20 Continuity

lemma *continuous_on_open*:

$continuous\_on \ S \ f \longleftrightarrow$

$(\forall T. \ openin \ (top\_of\_set \ (f \text{ ' } S)) \ T \longrightarrow$
 $\quad \quad \quad \openin \ (top\_of\_set \ S) \ (S \cap f \text{ ' } T))$

unfolding *continuous_on_open_invariant openin_open Int_def vimage_def Int_commute*

by (*simp add: imp_ex imageI conj_commute eq_commute cong: conj_cong*)

lemma *continuous_on_closed*:

$continuous\_on \ S \ f \longleftrightarrow$

$(\forall T. \ closedin \ (top\_of\_set \ (f \text{ ' } S)) \ T \longrightarrow$
 $\quad \quad \quad \closedin \ (top\_of\_set \ S) \ (S \cap f \text{ ' } T))$

unfolding *continuous_on_closed_invariant closedin_closed Int_def vimage_def Int_commute*

by (*simp add: imp_ex imageI conj_commute eq_commute cong: conj_cong*)

lemma *continuous_on_imp_closedin*:

assumes $continuous\_on \ S \ f \ closedin \ (top\_of\_set \ (f \text{ ' } S)) \ T$

shows $\closedin \ (top\_of\_set \ S) \ (S \cap f \text{ ' } T)$

using *assms continuous_on_closed* **by** *blast*

lemma *continuous_map_subtopology_eu* [*simp*]:

$continuous\_map \ (top\_of\_set \ S) \ (subtopology \ euclidean \ T) \ h \longleftrightarrow continuous\_on$

$S \cap h \wedge h^{-1} S \subseteq T$
by (*simp add: continuous_map_in_subtopology*)

lemma *continuous_map_euclidean_top_of_set*:
assumes *eq*: $f^{-1} S = \text{UNIV}$ **and** *cont*: *continuous_on UNIV f*
shows *continuous_map euclidean (top_of_set S) f*
by (*simp add: cont continuous_map_into_subtopology eq image_subset_iff_subset_vimage*)

2.2.21 Half-global and completely global cases

lemma *continuous_openin_preimage_gen*:
assumes *continuous_on S f open T*
shows *openin (top_of_set S) (S ∩ f⁻¹ T)*
proof –
have *: $(S \cap f^{-1} T) = (S \cap f^{-1} (T \cap f^{-1} S))$
by *auto*
have *openin (top_of_set (f⁻¹ S)) (T ∩ f⁻¹ S)*
using *openin_open_Int[of T f⁻¹ S, OF assms(2)]* **unfolding** *openin_open* **by**
auto
then show *?thesis*
using *assms(1)[unfolded continuous_on_open, THEN spec[where x=T ∩ f⁻¹ S]]*
using * **by** *auto*
qed

lemma *continuous_closedin_preimage*:
assumes *continuous_on S f and closed T*
shows *closedin (top_of_set S) (S ∩ f⁻¹ T)*
proof –
have *: $(S \cap f^{-1} T) = (S \cap f^{-1} (T \cap f^{-1} S))$
by *auto*
have *closedin (top_of_set (f⁻¹ S)) (T ∩ f⁻¹ S)*
using *closedin_closed_Int[of T f⁻¹ S, OF assms(2)]*
by (*simp add: Int_commute*)
then show *?thesis*
using *assms(1)[unfolded continuous_on_closed, THEN spec[where x=T ∩ f⁻¹ S]]*
using * **by** *auto*
qed

lemma *continuous_openin_preimage_eq*:
 $\text{continuous\_on } S \ f \longleftrightarrow (\forall T. \text{open } T \longrightarrow \text{openin } (\text{top\_of\_set } S) (S \cap f^{-1} T))$
by (*metis Int_commute continuous_on_open_invariant open_openin openin_subtopology*)

lemma *continuous_closedin_preimage_eq*:
 $\text{continuous\_on } S \ f \longleftrightarrow$
 $(\forall T. \text{closed } T \longrightarrow \text{closedin } (\text{top\_of\_set } S) (S \cap f^{-1} T))$
by (*metis Int_commute closedin_closed continuous_on_closed_invariant*)

```

lemma continuous_open_preimage:
  assumes contf: continuous_on S f and open S open T
  shows open (S ∩ f - ' T)
proof-
  obtain U where open U (S ∩ f - ' T) = S ∩ U
  using continuous_openin_preimage_gen[OF contf ⟨open T⟩]
  unfolding openin_open by auto
  then show ?thesis
  using open_Int[of S U, OF ⟨open S⟩] by auto
qed

lemma continuous_closed_preimage:
  assumes contf: continuous_on S f and closed S closed T
  shows closed (S ∩ f - ' T)
proof-
  obtain U where closed U (S ∩ f - ' T) = S ∩ U
  using continuous_closedin_preimage[OF contf ⟨closed T⟩]
  unfolding closedin_closed by auto
  then show ?thesis using closed_Int[of S U, OF ⟨closed S⟩] by auto
qed

lemma continuous_open_vimage: open S ⟹ (⋀x. continuous (at x) f) ⟹ open
(f - ' S)
  by (metis continuous_on_eq_continuous_within open_vimage)

lemma continuous_closed_vimage: closed S ⟹ (⋀x. continuous (at x) f) ⟹
closed (f - ' S)
  by (simp add: closed_vimage continuous_on_eq_continuous_within)

lemma Times_in_interior_subtopology:
  assumes (x, y) ∈ U openin (top_of_set (S × T)) U
  obtains V W where openin (top_of_set S) V x ∈ V
    openin (top_of_set T) W y ∈ W (V × W) ⊆ U
proof -
  from assms obtain E where open E U = S × T ∩ E (x, y) ∈ E x ∈ S y ∈ T
  by (auto simp: openin_open)
  from open_prod_elim[OF ⟨open E⟩ ⟨(x, y) ∈ E⟩]
  obtain E1 E2 where open E1 open E2 (x, y) ∈ E1 × E2 E1 × E2 ⊆ E
  by blast
  show ?thesis
proof
  show openin (top_of_set S) (E1 ∩ S)
    openin (top_of_set T) (E2 ∩ T)
  using ⟨open E1⟩ ⟨open E2⟩
  by (auto simp: openin_open)
  show x ∈ E1 ∩ S y ∈ E2 ∩ T
  using ⟨(x, y) ∈ E1 × E2⟩ ⟨x ∈ S⟩ ⟨y ∈ T⟩ by auto
  show (E1 ∩ S) × (E2 ∩ T) ⊆ U
  using ⟨E1 × E2 ⊆ E⟩ ⟨U = _⟩

```

```

      by (auto simp: )
    qed
  qed

lemma closedin_Times:
  closedin (top_of_set S) S'  $\implies$  closedin (top_of_set T) T'  $\implies$ 
    closedin (top_of_set (S  $\times$  T)) (S'  $\times$  T')
  unfolding closedin_closed using closed_Times by blast

lemma openin_Times:
  openin (top_of_set S) S'  $\implies$  openin (top_of_set T) T'  $\implies$ 
    openin (top_of_set (S  $\times$  T)) (S'  $\times$  T')
  unfolding openin_open using open_Times by blast

lemma openin_Times_eq:
  fixes S :: 'a::topological_space set and T :: 'b::topological_space set
  shows
    openin (top_of_set (S  $\times$  T)) (S'  $\times$  T')  $\longleftrightarrow$ 
      S' = {}  $\vee$  T' = {}  $\vee$  openin (top_of_set S) S'  $\wedge$  openin (top_of_set T) T'
    (is ?lhs = ?rhs)
  proof (cases S' = {}  $\vee$  T' = {})
    case True
      then show ?thesis by auto
    next
      case False
        then obtain x y where x  $\in$  S' y  $\in$  T'
          by blast
        show ?thesis
          proof
            assume ?lhs
            have openin (top_of_set S) S'
              proof (subst openin_subopen, clarify)
                show  $\exists U. \text{openin } (top\_of\_set S) U \wedge x \in U \wedge U \subseteq S' \text{ if } x \in S' \text{ for } x$ 
                  using that  $\langle y \in T' \rangle \text{ Times\_in\_interior\_subtopology } [OF\_ \langle ?lhs \rangle, \text{ of } x y]$ 
                  by simp (metis mem_Sigma_iff subsetD subsetI)
              qed
            moreover have openin (top_of_set T) T'
              proof (subst openin_subopen, clarify)
                show  $\exists U. \text{openin } (top\_of\_set T) U \wedge y \in U \wedge U \subseteq T' \text{ if } y \in T' \text{ for } y$ 
                  using that  $\langle x \in S' \rangle \text{ Times\_in\_interior\_subtopology } [OF\_ \langle ?lhs \rangle, \text{ of } x y]$ 
                  by simp (metis mem_Sigma_iff subsetD subsetI)
              qed
            ultimately show ?rhs
              by simp
          next
            assume ?rhs
            with False show ?lhs
              by (simp add: openin_Times)
          qed
        qed
      qed
    qed
  qed

```


qed

lemma *Lim_transform_within_openin*:
assumes $f: (f \longrightarrow l) \text{ (at } a \text{ within } T)$
and $\text{openin } (\text{top\_of\_set } T) \ S \ a \in S$
and $\text{eq}: \bigwedge x. \llbracket x \in S; x \neq a \rrbracket \implies f \ x = g \ x$
shows $(g \longrightarrow l) \text{ (at } a \text{ within } T)$
proof –
have $\forall_F x \text{ in at } a \text{ within } T. x \in T \wedge x \neq a$
by (*simp add: eventually_at_filter*)
moreover
from $\langle \text{openin } \_ \_ \rangle$ **obtain** U **where** $\text{open } U \ S = T \cap U$
by (*auto simp: openin_open*)
then have $a \in U$ **using** $\langle a \in S \rangle$ **by** *auto*
from *topological_tendstoD[OF tendsto_ident_at $\langle \text{open } U \rangle \langle a \in U \rangle$]*
have $\forall_F x \text{ in at } a \text{ within } T. x \in U$ **by** *auto*
ultimately
have $\forall_F x \text{ in at } a \text{ within } T. f \ x = g \ x$
by *eventually_elim (auto simp: $\langle S = \_ \rangle$ eq)*
with f **show** *?thesis*
by (*rule Lim_transform_eventually*)
qed

lemma *continuous_on_open_gen*:
assumes $f: S \subseteq T$
shows $\text{continuous\_on } S \ f \longleftrightarrow$
 $(\forall U. \text{openin } (\text{top\_of\_set } T) \ U$
 $\longrightarrow \text{openin } (\text{top\_of\_set } S) \ (S \cap f - ' U))$
 $(\text{is } ?lhs = ?rhs)$
proof
assume *?lhs*
then show *?rhs*
by (*clarsimp simp add: continuous_openin_preimage_eq openin_open*)
 $(\text{metis Int\_assoc assms image_subset_iff_subset_vimage inf.absorb_iff1})$
next
assume $R \text{ [rule\_format]: } ?rhs$
show *?lhs*
proof (*clarsimp simp add: continuous_openin_preimage_eq*)
fix $U::'a \text{ set}$
assume $\text{open } U$
then have $\text{openin } (\text{top\_of\_set } S) \ (S \cap f - ' (U \cap T))$
by (*metis R inf_commute openin_open*)
then show $\text{openin } (\text{top\_of\_set } S) \ (S \cap f - ' U)$
by (*metis Int_assoc Int_commute assms image_subset_iff_subset_vimage*
 $\text{inf.absorb_iff2 vimage\_Int}$)
qed
qed

lemma *continuous_openin_preimage*:

$\llbracket \text{continuous\_on } S f; f ' S \subseteq T; \text{openin } (\text{top\_of\_set } T) U \rrbracket$
 $\implies \text{openin } (\text{top\_of\_set } S) (S \cap f - ' U)$
by (simp add: continuous_on_open_gen)

lemma continuous_on_closed_gen:
assumes $f ' S \subseteq T$
shows continuous_on $S f \longleftrightarrow$
 $(\forall U. \text{closedin } (\text{top\_of\_set } T) U$
 $\longrightarrow \text{closedin } (\text{top\_of\_set } S) (S \cap f - ' U))$
(is ?lhs = ?rhs)
proof -
have *: $U \subseteq T \implies S \cap f - ' (T - U) = S - (S \cap f - ' U)$ **for** U
using assms **by** blast
show ?thesis
proof
assume $L: ?lhs$
show ?rhs
proof clarify
fix U
assume closedin (top_of_set T) U
then show closedin (top_of_set S) $(S \cap f - ' U)$
using L **unfolding** continuous_on_open_gen [OF assms]
by (metis * closedin_def inf_le1 topspace_euclidean_subtopology)
qed
next
assume R [rule_format]: ?rhs
show ?lhs
unfolding continuous_on_open_gen [OF assms]
by (metis * R inf_le1 openin_closedin_eq topspace_euclidean_subtopology)
qed
qed

lemma continuous_closedin_preimage_gen:
assumes continuous_on $S f$ $f ' S \subseteq T$ closedin (top_of_set T) U
shows closedin (top_of_set S) $(S \cap f - ' U)$
using assms continuous_on_closed_gen **by** blast

lemma continuous_transform_within_openin:
assumes continuous (at a within T) f
and openin (top_of_set T) S $a \in S$
and eq: $\bigwedge x. x \in S \implies f x = g x$
shows continuous (at a within T) g
using assms **by** (simp add: Lim_transform_within_openin continuous_within)

2.2.22 The topology generated by some (open) subsets

In the definition below of a generated topology, the *Empty* case is not necessary, as it follows from *UN* taking for K the empty set. However, it is convenient to have, and is never a problem in proofs, so I prefer to write it

down explicitly.

We do not require $UNIV$ to be an open set, as this will not be the case in applications. (We are thinking of a topology on a subset of $UNIV$, the remaining part of $UNIV$ being irrelevant.)

inductive *generate_topology_on* **for** S **where**

Empty: *generate_topology_on* $S \ \{\}$
 $|$ *Int:* *generate_topology_on* $S \ a \implies \text{generate\_topology\_on } S \ b \implies \text{generate\_topology\_on } S \ (a \cap b)$
 $|$ *UN:* $(\bigwedge k. k \in K \implies \text{generate\_topology\_on } S \ k) \implies \text{generate\_topology\_on } S \ (\bigcup K)$
 $|$ *Basis:* $s \in S \implies \text{generate\_topology\_on } S \ s$

lemma *istopology_generate_topology_on:*

istopology (*generate_topology_on* S)

unfolding *istopology_def* **by** (*auto intro: generate_topology_on.intros*)

The basic property of the topology generated by a set S is that it is the smallest topology containing all the elements of S :

lemma *generate_topology_on_coarsest:*

assumes $T: \text{istopology } T \ \bigwedge s. s \in S \implies T \ s$

and *gen:* *generate_topology_on* $S \ s0$

shows $T \ s0$

using *gen*

by (*induct rule: generate_topology_on.induct*) (*use T in <auto simp: istopology_def>*)

abbreviation *topology_generated_by::* ($'a \text{ set set}$) $\Rightarrow$ ($'a \text{ topology}$)

where *topology_generated_by* $S \equiv \text{topology } (\text{generate\_topology\_on } S)$

lemma *openin_topology_generated_by_iff:*

openin (*topology_generated_by* S) $s \longleftrightarrow \text{generate\_topology\_on } S \ s$

using *topology_inverse* [*OF istopology_generate_topology_on[of S]*] **by** *simp*

lemma *openin_topology_generated_by:*

openin (*topology_generated_by* S) $s \implies \text{generate\_topology\_on } S \ s$

using *openin_topology_generated_by_iff* **by** *auto*

lemma *topology_generated_by_topspace [simp]:*

topspace (*topology_generated_by* S) = $(\bigcup S)$

proof

{
 $\text{fix } s \text{ assume } \text{openin } (\text{topology\_generated\_by } S) \ s$
 $\text{then have } \text{generate\_topology\_on } S \ s \text{ by (rule openin\_topology\_generated\_by)}$
 $\text{then have } s \subseteq (\bigcup S) \text{ by (induct, auto)}$
 }
 $\text{then show } \text{topspace } (\text{topology\_generated\_by } S) \subseteq (\bigcup S)$
 $\text{unfolding topspace\_def by auto}$

next

$\text{have } \text{generate\_topology\_on } S \ (\bigcup S)$

```

    using generate_topology_on.UN[OF generate_topology_on.Basis, of S S] by
simp
    then show  $(\bigcup S) \subseteq \text{topspace } (\text{topology\_generated\_by } S)$ 
    unfolding topspace_def using openin_topology_generated_by_iff by auto
qed

```

```

lemma topology_generated_by_Basis:
   $s \in S \implies \text{openin } (\text{topology\_generated\_by } S) s$ 
  by (simp only: openin_topology_generated_by_iff, auto simp: generate_topology_on.Basis)

```

```

lemma generate_topology_on_Inter:
   $\llbracket \text{finite } \mathcal{F}; \bigwedge K. K \in \mathcal{F} \implies \text{generate\_topology\_on } S K; \mathcal{F} \neq \{\} \rrbracket \implies \text{generate\_topology\_on } S (\bigcap \mathcal{F})$ 
  by (induction  $\mathcal{F}$  rule: finite_induct; force intro: generate_topology_on.intros)

```

2.2.23 Topology bases and sub-bases

```

lemma istopology_base_alt:
   $\text{istopology } (\text{arbitrary\_union\_of } P) \longleftrightarrow$ 
   $(\forall S T. (\text{arbitrary\_union\_of } P) S \wedge (\text{arbitrary\_union\_of } P) T$ 
     $\longrightarrow (\text{arbitrary\_union\_of } P) (S \cap T))$ 
  by (simp add: istopology_def) (blast intro: arbitrary_union_of_Union)

```

```

lemma istopology_base_eq:
   $\text{istopology } (\text{arbitrary\_union\_of } P) \longleftrightarrow$ 
   $(\forall S T. P S \wedge P T \longrightarrow (\text{arbitrary\_union\_of } P) (S \cap T))$ 
  by (simp add: istopology_base_alt arbitrary_union_of_Int_eq)

```

```

lemma istopology_base:
   $(\bigwedge S T. \llbracket P S; P T \rrbracket \implies P(S \cap T)) \implies \text{istopology } (\text{arbitrary\_union\_of } P)$ 
  by (simp add: arbitrary_def istopology_base_eq union_of_inc)

```

```

lemma openin_topology_base_unique:
   $\text{openin } X = \text{arbitrary\_union\_of } P \longleftrightarrow$ 
   $(\forall V. P V \longrightarrow \text{openin } X V) \wedge (\forall U x. \text{openin } X U \wedge x \in U \longrightarrow (\exists V. P V$ 
 $\wedge x \in V \wedge V \subseteq U))$ 
  (is ?lhs = ?rhs)

```

```

proof
  assume ?lhs
  then show ?rhs
    by (auto simp: union_of_def arbitrary_def)
next
  assume R: ?rhs
  then have *:  $\exists U \subseteq \text{Collect } P. \bigcup U = S$  if  $\text{openin } X S$  for  $S$ 
    using that by (rule_tac  $x = \{V. P V \wedge V \subseteq S\}$  in exI) fastforce
  from R show ?lhs
    by (fastforce simp add: union_of_def arbitrary_def intro: *)
qed

```

```

lemma topology_base_unique:
  assumes  $\bigwedge S. P\ S \implies \text{openin } X\ S$ 
     $\bigwedge U\ x. \llbracket \text{openin } X\ U; x \in U \rrbracket \implies \exists B. P\ B \wedge x \in B \wedge B \subseteq U$ 
  shows  $\text{topology } (\text{arbitrary\_union\_of } P) = X$ 
proof -
  have  $X = \text{topology } (\text{openin } X)$ 
    by (simp add: openin_inverse)
  also from assms have  $\text{openin } X = \text{arbitrary\_union\_of } P$ 
    by (subst openin_topology_base_unique) auto
  finally show ?thesis ..
qed

lemma topology_bases_eq_aux:
   $\llbracket (\text{arbitrary\_union\_of } P)\ S; \bigwedge U\ x. \llbracket P\ U; x \in U \rrbracket \implies \exists V. Q\ V \wedge x \in V \wedge V \subseteq U \rrbracket$ 
   $\implies (\text{arbitrary\_union\_of } Q)\ S$ 
by (metis arbitrary_union_of_alt arbitrary_union_of_idempot)

lemma topology_bases_eq:
   $\llbracket \bigwedge U\ x. \llbracket P\ U; x \in U \rrbracket \implies \exists V. Q\ V \wedge x \in V \wedge V \subseteq U; \bigwedge V\ x. \llbracket Q\ V; x \in V \rrbracket \implies \exists U. P\ U \wedge x \in U \wedge U \subseteq V \rrbracket$ 
   $\implies \text{topology } (\text{arbitrary\_union\_of } P) = \text{topology } (\text{arbitrary\_union\_of } Q)$ 
by (fastforce intro: arg_cong [where f=topology] elim: topology_bases_eq_aux)

lemma istopology_subbase:
   $\text{istopology } (\text{arbitrary\_union\_of } (\text{finite\_intersection\_of } P\ \text{relative\_to } S))$ 
by (simp add: finite_intersection_of_Int istopology_base_relative_to_Int)

lemma openin_subbase:
   $\text{openin } (\text{topology } (\text{arbitrary\_union\_of } (\text{finite\_intersection\_of } B\ \text{relative\_to } U)))$ 
 $S$ 
 $\longleftrightarrow (\text{arbitrary\_union\_of } (\text{finite\_intersection\_of } B\ \text{relative\_to } U))\ S$ 
by (simp add: istopology_subbase topology_inverse')

lemma topspace_subbase [simp]:
   $\text{topspace}(\text{topology } (\text{arbitrary\_union\_of } (\text{finite\_intersection\_of } B\ \text{relative\_to } U)))$ 
 $= U$  (is ?lhs = _)
proof
  show  $?lhs \subseteq U$ 
    by (metis arbitrary_union_of_relative_to openin_subbase openin_topspace
      relative_to_imp_subset)
  show  $U \subseteq ?lhs$ 
    by (metis arbitrary_union_of_inc finite_intersection_of_empty inf.orderE
      istopology_subbase openin_subset_relative_to_inc subset_UNIV topology_inverse')
qed

lemma minimal_topology_subbase:

```

```

    assumes  $X: \bigwedge S. P\ S \implies \text{openin } X\ S$  and  $\text{openin } X\ U$ 
    and  $S: \text{openin}(\text{topology}(\text{arbitrary\_union\_of } (\text{finite\_intersection\_of } P \text{ relative\_to } U)))\ S$ 
  shows  $\text{openin } X\ S$ 
  proof -
    have  $(\text{arbitrary\_union\_of } (\text{finite\_intersection\_of } P \text{ relative\_to } U))\ S$ 
    using  $S$   $\text{openin\_subbase}$  by blast
    with  $X \langle \text{openin } X\ U \rangle$  show ?thesis
    by (force simp add:  $\text{union\_of\_def}$   $\text{intersection\_of\_def}$   $\text{relative\_to\_def}$  intro:  $\text{openin\_Int\_Inter}$ )
  qed

```

```

lemma istopology_subbase_UNIV:
   $\text{istopology } (\text{arbitrary\_union\_of } (\text{finite\_intersection\_of } P))$ 
  by (simp add:  $\text{istopology\_base\_finite\_intersection\_of\_Int}$ )

```

```

lemma generate_topology_on_eq:
   $\text{generate\_topology\_on } S = \text{arbitrary\_union\_of } \text{finite}' \text{ intersection\_of } (\lambda x. x \in S)$  (is ?lhs = ?rhs)
  proof (intro ext iffI)
    fix A
    assume ?lhs A
    then show ?rhs A
    proof induction
      case (Int a b)
      then show ?case
      by (metis (mono_tags, lifting)  $\text{istopology\_base\_alt\_finite}' \text{ intersection\_of\_Int}$   $\text{istopology\_base}$ )
    next
      case (UN K)
      then show ?case
      by (simp add:  $\text{arbitrary\_union\_of\_Union}$ )
    next
      case (Basis s)
      then show ?case
      by (simp add:  $\text{Sup\_upper}$   $\text{arbitrary\_union\_of\_inc}$   $\text{finite}' \text{ intersection\_of\_inc}$   $\text{relative\_to\_subset}$ )
    qed auto
  next
    fix A
    assume ?rhs A
    then obtain  $\mathcal{U}$  where  $\mathcal{U}: \bigwedge T. T \in \mathcal{U} \implies \exists \mathcal{F}. \text{finite}' \mathcal{F} \wedge \mathcal{F} \subseteq S \wedge \bigcap \mathcal{F} = T$ 
    and eq:  $A = \bigcup \mathcal{U}$ 
    unfolding  $\text{union\_of\_def}$   $\text{intersection\_of\_def}$  by auto
    show ?lhs A
    unfolding eq
    proof (rule  $\text{generate\_topology\_on.UN}$ )
      fix T

```

```

    assume  $T \in \mathcal{U}$ 
    with  $\mathcal{U}$  obtain  $\mathcal{F}$  where  $\text{finite}' \mathcal{F} \mathcal{F} \subseteq S \cap \mathcal{F} = T$ 
    by blast
    have  $\text{generate\_topology\_on } S (\cap \mathcal{F})$ 
    proof (rule  $\text{generate\_topology\_on\_Inter}$ )
      show  $\text{finite } \mathcal{F} \mathcal{F} \neq \{\}$ 
      by (auto simp:  $\langle \text{finite}' \mathcal{F} \rangle$ )
      show  $\bigwedge K. K \in \mathcal{F} \implies \text{generate\_topology\_on } S K$ 
      by (metis  $\langle \mathcal{F} \subseteq S \rangle \text{generate\_topology\_on.simps subset\_iff}$ )
    qed
    then show  $\text{generate\_topology\_on } S T$ 
    using  $\langle \cap \mathcal{F} = T \rangle$  by blast
  qed
qed

lemma continuous_on_generated_topo_iff:
  continuous_map T1 (topology_generated_by S)  $f \longleftrightarrow$ 
    (( $\forall U. U \in S \longrightarrow \text{openin } T1 (f - 'U \cap \text{topspace}(T1))$ )  $\wedge$  ( $f(\text{topspace } T1) \subseteq$ 
    ( $\bigcup S$ )))
  unfolding continuous_map_alt topology_generated_by_topspace
  proof (auto simp add: topology_generated_by_Basis)
    assume  $H: \forall U. U \in S \longrightarrow \text{openin } T1 (f - 'U \cap \text{topspace } T1)$ 
    fix  $U$  assume  $\text{openin } (\text{topology\_generated\_by } S) U$ 
    then have  $\text{generate\_topology\_on } S U$  by (rule  $\text{openin\_topology\_generated\_by}$ )
    then show  $\text{openin } T1 (f - 'U \cap \text{topspace } T1)$ 
    proof (induct)
      fix  $a b$ 
      assume  $H: \text{openin } T1 (f - 'a \cap \text{topspace } T1) \text{ openin } T1 (f - 'b \cap \text{topspace } T1)$ 
      have  $f - '(a \cap b) \cap \text{topspace } T1 = (f - 'a \cap \text{topspace } T1) \cap (f - 'b \cap \text{topspace } T1)$ 
      by auto
      then show  $\text{openin } T1 (f - '(a \cap b) \cap \text{topspace } T1)$  using  $H$  by auto
    next
      fix  $K$ 
      assume  $H: \text{openin } T1 (f - 'k \cap \text{topspace } T1)$  if  $k \in K$  for  $k$ 
      define  $L$  where  $L = \{f - 'k \cap \text{topspace } T1 \mid k. k \in K\}$ 
      have *:  $\text{openin } T1 l$  if  $l \in L$  for  $l$  using that  $H$  unfolding  $L\_def$  by auto
      have  $\text{openin } T1 (\bigcup L)$  using  $\text{openin\_Union}[OF *]$  by simp
      moreover have  $(\bigcup L) = (f - ' \bigcup K \cap \text{topspace } T1)$  unfolding  $L\_def$  by auto
      ultimately show  $\text{openin } T1 (f - ' \bigcup K \cap \text{topspace } T1)$  by simp
    qed (auto simp add:  $H$ )
  qed

lemma continuous_on_generated_topo:
  assumes  $\bigwedge U. U \in S \implies \text{openin } T1 (f - 'U \cap \text{topspace}(T1))$ 
     $f(\text{topspace } T1) \subseteq (\bigcup S)$ 
  shows  $\text{continuous\_map } T1 (\text{topology\_generated\_by } S) f$ 
  using assms continuous_on_generated_topo_iff by blast

```

2.2.24 Pullback topology

Pulling back a topology by map gives again a topology. *subtopology* is a special case of this notion, pulling back by the identity. We introduce the general notion as we will need it to define the strong operator topology on the space of continuous linear operators, by pulling back the product topology on the space of all functions.

pullback_topology $A f T$ is the pullback of the topology T by the map f on the set A .

definition *pullback_topology*::('a set) $\Rightarrow$ ('a $\Rightarrow$ 'b) $\Rightarrow$ ('b topology) $\Rightarrow$ ('a topology)
where *pullback_topology* $A f T = \text{topology } (\lambda S. \exists U. \text{openin } T U \wedge S = f - 'U \cap A)$

lemma *istopology_pullback_topology*:
istopology $(\lambda S. \exists U. \text{openin } T U \wedge S = f - 'U \cap A)$
unfolding *istopology_def* **proof** (auto)
fix K **assume** $\forall S \in K. \exists U. \text{openin } T U \wedge S = f - 'U \cap A$
then have $\exists U. \forall S \in K. \text{openin } T (U \cap S) \wedge S = f - '(U \cap S) \cap A$
by (rule bchoice)
then obtain U **where** $U: \forall S \in K. \text{openin } T (U \cap S) \wedge S = f - '(U \cap S) \cap A$
by blast
define V **where** $V = (\bigcup S \in K. U \cap S)$
have $\text{openin } T V \bigcup K = f - 'V \cap A$ **unfolding** *V_def* **using** U **by** auto
then show $\exists V. \text{openin } T V \wedge \bigcup K = f - 'V \cap A$ **by** auto
qed

lemma *openin_pullback_topology*:
 $\text{openin } (\text{pullback\_topology } A f T) S \longleftrightarrow (\exists U. \text{openin } T U \wedge S = f - 'U \cap A)$
unfolding *pullback_topology_def* *topology_inverse* [*OF istopology_pullback_topology*]
by auto

lemma *topspace_pullback_topology*:
 $\text{topspace } (\text{pullback\_topology } A f T) = f - '(\text{topspace } T) \cap A$
by (auto simp add: *topspace_def* *openin_pullback_topology*)

proposition *continuous_map_pullback* [intro]:
assumes *continuous_map* $T1 T2 g$
shows *continuous_map* $(\text{pullback\_topology } A f T1) T2 (g \circ f)$
unfolding *continuous_map_alt*
proof (auto)
fix $U::'b \text{ set}$ **assume** $\text{openin } T2 U$
then have $\text{openin } T1 (g - 'U \cap \text{topspace } T1)$
using *assms* **unfolding** *continuous_map_alt* **by** auto
have $(g \circ f) - 'U \cap \text{topspace } (\text{pullback\_topology } A f T1) = (g \circ f) - 'U \cap A \cap f - '(\text{topspace } T1)$
unfolding *topspace_pullback_topology* **by** auto
also have $\dots = f - '(g - 'U \cap \text{topspace } T1) \cap A$
by auto


```

    also have openin (pullback_topology A f T1) (...)
      unfolding openin_pullback_topology using ‹openin T1 (g-‘U ∩ topspace T1)›
    by auto
    finally show openin (pullback_topology A f T1) ((g ∘ f) -‘ U ∩ topspace
      (pullback_topology A f T1))
      by auto
  next
    fix x assume x ∈ topspace (pullback_topology A f T1)
    then have f x ∈ topspace T1
      unfolding topspace_pullback_topology by auto
    then show g (f x) ∈ topspace T2
      using assms unfolding continuous_map_def by auto
  qed

```

```

proposition continuous_map_pullback' [intro]:
  assumes continuous_map T1 T2 (f o g) topspace T1 ⊆ g-‘A
  shows continuous_map T1 (pullback_topology A f T2) g
unfolding continuous_map_alt
proof (auto)
  fix U assume openin (pullback_topology A f T2) U
  then have ∃ V. openin T2 V ∧ U = f-‘V ∩ A
    unfolding openin_pullback_topology by auto
  then obtain V where openin T2 V U = f-‘V ∩ A
    by blast
  then have g -‘ U ∩ topspace T1 = g-‘(f-‘V ∩ A) ∩ topspace T1
    by blast
  also have ... = (f o g)-‘V ∩ (g-‘A ∩ topspace T1)
    by auto
  also have ... = (f o g)-‘V ∩ topspace T1
    using assms(2) by auto
  also have openin T1 (...)
    using assms(1) ‹openin T2 V› by auto
  finally show openin T1 (g -‘ U ∩ topspace T1) by simp
next
  fix x assume x ∈ topspace T1
  have (f o g) x ∈ topspace T2
    using assms(1) ‹x ∈ topspace T1› unfolding continuous_map_def by auto
  then have g x ∈ f-‘(topspace T2)
    unfolding comp_def by blast
  moreover have g x ∈ A using assms(2) ‹x ∈ topspace T1› by blast
  ultimately show g x ∈ topspace (pullback_topology A f T2)
    unfolding topspace_pullback_topology by blast
qed

```

2.2.25 Proper maps (not a priori assumed continuous)

```

definition proper_map
  where
    proper_map X Y f ≡

```

$\text{closed\_map } X \ Y \ f \wedge (\forall y \in \text{topspace } Y. \text{compactin } X \ \{x \in \text{topspace } X. f \ x = y\})$

lemma *proper_imp_closed_map*:
 $\text{proper\_map } X \ Y \ f \implies \text{closed\_map } X \ Y \ f$
by (*simp add: proper_map_def*)

lemma *proper_map_imp_subset_topspace*:
 $\text{proper\_map } X \ Y \ f \implies f' (\text{topspace } X) \subseteq \text{topspace } Y$
by (*simp add: closed_map_imp_subset_topspace proper_map_def*)

lemma *closed_injective_imp_proper_map*:
assumes $f: \text{closed\_map } X \ Y \ f$ **and** $\text{inj: inj\_on } f \ (\text{topspace } X)$
shows $\text{proper\_map } X \ Y \ f$
unfolding *proper_map_def*
proof (*clarsimp simp: f*)
show $\text{compactin } X \ \{x \in \text{topspace } X. f \ x = y\}$
if $y \in \text{topspace } Y$ **for** y
proof –
have $\{x \in \text{topspace } X. f \ x = y\} = \{\} \vee (\exists a \in \text{topspace } X. \{x \in \text{topspace } X. f \ x = y\} = \{a\})$
using *inj_on_eq_iff* [*OF inj*] **by** *auto*
then show *?thesis*
using *that* **by** (*metis (no_types, lifting) compactin_empty compactin_singleton*)
qed
qed

lemma *injective_imp_proper_eq_closed_map*:
 $\text{inj\_on } f \ (\text{topspace } X) \implies (\text{proper\_map } X \ Y \ f \longleftrightarrow \text{closed\_map } X \ Y \ f)$
using *closed_injective_imp_proper_map proper_imp_closed_map* **by** *blast*

lemma *homeomorphic_imp_proper_map*:
 $\text{homeomorphic\_map } X \ Y \ f \implies \text{proper\_map } X \ Y \ f$
by (*simp add: closed_injective_imp_proper_map homeomorphic_eq_everything_map*)

lemma *compactin_proper_map_preimage*:
assumes $f: \text{proper\_map } X \ Y \ f$ **and** $\text{compactin } Y \ K$
shows $\text{compactin } X \ \{x. x \in \text{topspace } X \wedge f \ x \in K\}$
proof –
have $f' (\text{topspace } X) \subseteq \text{topspace } Y$
by (*simp add: f proper_map_imp_subset_topspace*)
have $*$: $\bigwedge y. y \in \text{topspace } Y \implies \text{compactin } X \ \{x \in \text{topspace } X. f \ x = y\}$
using f **by** (*auto simp: proper_map_def*)
show *?thesis*
unfolding *compactin_def*
proof *clarsimp*
show $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge \{x \in \text{topspace } X. f \ x \in K\} \subseteq \bigcup \mathcal{F}$
if $\mathcal{U}: \forall U \in \mathcal{U}. \text{openin } X \ U$ **and** $\text{sub: } \{x \in \text{topspace } X. f \ x \in K\} \subseteq \bigcup \mathcal{U}$
for $\mathcal{U}$

```

proof –
  have  $\forall y \in K. \exists \mathcal{V}. \text{finite } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \{x \in \text{topspace } X. f\ x = y\} \subseteq \bigcup \mathcal{V}$ 
proof
  fix  $y$ 
  assume  $y \in K$ 
  then have  $\text{compactin } X\ \{x \in \text{topspace } X. f\ x = y\}$ 
    by (metis *  $\langle \text{compactin } Y\ K \rangle$  compactin_subspace subsetD)
  with  $\langle y \in K \rangle$  show  $\exists \mathcal{V}. \text{finite } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \{x \in \text{topspace } X. f\ x = y\} \subseteq$ 
 $\bigcup \mathcal{V}$ 
    unfolding compactin_def using  $\mathcal{U}$  sub by fastforce
qed
  then obtain  $\mathcal{V}$  where  $\mathcal{V}: \bigwedge y. y \in K \implies \text{finite } (\mathcal{V}\ y) \wedge \mathcal{V}\ y \subseteq \mathcal{U} \wedge \{x \in$ 
 $\text{topspace } X. f\ x = y\} \subseteq \bigcup (\mathcal{V}\ y)$ 
    by (metis (full_types))
  define  $F$  where  $F \equiv \lambda y. \text{topspace } Y - f\ '(\text{topspace } X - \bigcup (\mathcal{V}\ y))$ 
  have  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq F\ 'K \wedge K \subseteq \bigcup \mathcal{F}$ 
proof (rule compactinD [OF  $\langle \text{compactin } Y\ K \rangle$ ])
  have  $\bigwedge x. x \in K \implies \text{closedin } Y\ (f\ '(\text{topspace } X - \bigcup (\mathcal{V}\ x)))$ 
    using  $f$  unfolding proper_map_def closed_map_def
    by (meson  $\mathcal{U}\ \mathcal{V}$  openin_Union openin_closedin_eq subsetD)
  then show  $\text{openin } Y\ U$  if  $U \in F\ 'K$  for  $U$ 
    using that by (auto simp: F_def)
  show  $K \subseteq \bigcup (F\ 'K)$ 
    using  $\mathcal{V}\ \langle \text{compactin } Y\ K \rangle$  unfolding  $F\_def$  compactin_def by fastforce
qed
  then obtain  $J$  where  $\text{finite } J\ J \subseteq K$  and  $J: K \subseteq \bigcup (F\ 'J)$ 
    by (auto simp: ex_finite_subset_image)
  show ?thesis
    unfolding  $F\_def$ 
proof (intro exI conjI)
  show  $\text{finite } (\bigcup (\mathcal{V}\ 'J))$ 
    using  $\mathcal{V}\ \langle J \subseteq K \rangle\ \langle \text{finite } J \rangle$  by blast
  show  $\bigcup (\mathcal{V}\ 'J) \subseteq \mathcal{U}$ 
    using  $\mathcal{V}\ \langle J \subseteq K \rangle$  by blast
  show  $\{x \in \text{topspace } X. f\ x \in K\} \subseteq \bigcup (\bigcup (\mathcal{V}\ 'J))$ 
    using  $J\ \langle J \subseteq K \rangle$  unfolding  $F\_def$  by auto
qed
qed
qed
qed

```

lemma *compact_space_proper_map_preimage:*

assumes $f: \text{proper\_map } X\ Y\ f$ **and** $\text{fim}: f\ '(\text{topspace } X) = \text{topspace } Y$ **and** *compact_space Y*

shows *compact_space X*

proof –

have $\text{eq}: \text{topspace } X = \{x \in \text{topspace } X. f\ x \in \text{topspace } Y\}$
using *fim* **by** *blast*

```

moreover have compactin Y (topspace Y)
  using <compact_space Y> compact_space_def by auto
ultimately show ?thesis
  unfolding compact_space_def
  using eq f compactin_proper_map_preimage by fastforce
qed

```

```

lemma proper_map_alt:
  proper_map X Y f  $\longleftrightarrow$ 
    closed_map X Y f  $\wedge$  ( $\forall K$ . compactin Y K  $\longrightarrow$  compactin X  $\{x. x \in \text{topspace } X \wedge f x \in K\}$ )
proof (intro iffI conjI allI impI)
  show compactin X  $\{x \in \text{topspace } X. f x \in K\}$ 
    if proper_map X Y f and compactin Y K for K
    using that by (simp add: compactin_proper_map_preimage)
  show proper_map X Y f
    if f: closed_map X Y f  $\wedge$  ( $\forall K$ . compactin Y K  $\longrightarrow$  compactin X  $\{x \in \text{topspace } X. f x \in K\}$ )
proof -
  have compactin X  $\{x \in \text{topspace } X. f x = y\}$  if  $y \in \text{topspace } Y$  for y
  proof -
    have compactin X  $\{x \in \text{topspace } X. f x \in \{y\}\}$ 
      using f compactin_sing that by fastforce
    then show ?thesis
      by auto
  qed
  with f show ?thesis
    by (auto simp: proper_map_def)
qed
qed (simp add: proper_imp_closed_map)

```

```

lemma proper_map_on_empty:
  topspace X =  $\{\}$   $\implies$  proper_map X Y f
by (auto simp: proper_map_def closed_map_on_empty)

```

```

lemma proper_map_id [simp]:
  proper_map X X id
proof (clarsimp simp: proper_map_alt closed_map_id)
  fix K
  assume K: compactin X K
  then have  $\{a \in \text{topspace } X. a \in K\} = K$ 
    by (simp add: compactin_subspace subset_antisym subset_iff)
  then show compactin X  $\{a \in \text{topspace } X. a \in K\}$ 
    using K by auto
qed

```

```

lemma proper_map_compose:
  assumes proper_map X Y f proper_map Y Z g
  shows proper_map X Z (g  $\circ$  f)

```

```

proof –
  have closed_map X Y f and f:  $\bigwedge K. \text{compactin } Y \ K \implies \text{compactin } X \ \{x \in \text{topspace } X. f \ x \in K\}$ 
  and closed_map Y Z g and g:  $\bigwedge K. \text{compactin } Z \ K \implies \text{compactin } Y \ \{x \in \text{topspace } Y. g \ x \in K\}$ 
  using assms by (auto simp: proper_map_alt)
  show ?thesis
  unfolding proper_map_alt
  proof (intro conjI allI impI)
    show closed_map X Z (g  $\circ$  f)
    using  $\langle \text{closed\_map } X \ Y \ f \rangle \langle \text{closed\_map } Y \ Z \ g \rangle \text{closed\_map\_compose}$  by
    blast
    have  $\{x \in \text{topspace } X. g \ (f \ x) \in K\} = \{x \in \text{topspace } X. f \ x \in \{b \in \text{topspace } Y. g \ b \in K\}\}$  for K
    using  $\langle \text{closed\_map } X \ Y \ f \rangle \text{closed\_map\_imp\_subset\_topspace}$  by blast
    then show  $\text{compactin } X \ \{x \in \text{topspace } X. (g \circ f) \ x \in K\}$ 
    if  $\text{compactin } Z \ K$  for K
    using f [OF g [OF that]] by auto
  qed
qed

```

```

lemma proper_map_const:
  proper_map X Y ( $\lambda x. c$ )  $\longleftrightarrow \text{compact\_space } X \wedge (\text{topspace } X = \{\}) \vee \text{closedin } Y \ \{c\}$ 
proof (cases  $\text{topspace } X = \{\}$ )
  case True
    then show ?thesis
    by (simp add: compact_space_topspace_empty proper_map_on_empty)
  next
  case False
    have *:  $\text{compactin } X \ \{x \in \text{topspace } X. c = y\}$  if  $\text{compact\_space } X$  for y
    proof (cases  $c = y$ )
    case True
      then show ?thesis
      using compact_space_def  $\langle \text{compact\_space } X \rangle$  by auto
    qed auto
    then show ?thesis
    using closed_compactin_closedin_subset
    by (force simp: False proper_map_def closed_map_const compact_space_def)
  qed

```

```

lemma proper_map_inclusion:
   $s \subseteq \text{topspace } X$ 
   $\implies \text{proper\_map } (\text{subtopology } X \ s) \ X \ \text{id} \longleftrightarrow \text{closedin } X \ s \wedge (\forall k. \text{compactin } X \ k \longrightarrow \text{compactin } X \ (s \cap k))$ 
  by (auto simp: proper_map_alt closed_map_inclusion_eq inf.absorb_iff2 Collect_conj_eq compactin_subtopology intro: closed_Int_compactin)

```

2.2.26 Perfect maps (proper, continuous and surjective)

definition *perfect_map*

where $\text{perfect\_map } X \ Y \ f \equiv \text{continuous\_map } X \ Y \ f \wedge \text{proper\_map } X \ Y \ f \wedge f$
 $\text{' (topspace } X) = \text{topspace } Y$

lemma *homeomorphic_imp_perfect_map*:

$\text{homeomorphic\_map } X \ Y \ f \implies \text{perfect\_map } X \ Y \ f$

by (*simp add: homeomorphic_eq_everything_map homeomorphic_imp_proper_map perfect_map_def*)

lemma *perfect_imp_quotient_map*:

$\text{perfect\_map } X \ Y \ f \implies \text{quotient\_map } X \ Y \ f$

by (*simp add: continuous_closed_imp_quotient_map perfect_map_def proper_map_def*)

lemma *homeomorphic_eq_injective_perfect_map*:

$\text{homeomorphic\_map } X \ Y \ f \longleftrightarrow \text{perfect\_map } X \ Y \ f \wedge \text{inj\_on } f \ (\text{topspace } X)$

using *homeomorphic_imp_perfect_map homeomorphic_map_def perfect_imp_quotient_map*
by *blast*

lemma *perfect_injective_eq_homeomorphic_map*:

$\text{perfect\_map } X \ Y \ f \wedge \text{inj\_on } f \ (\text{topspace } X) \longleftrightarrow \text{homeomorphic\_map } X \ Y \ f$

by (*simp add: homeomorphic_eq_injective_perfect_map*)

lemma *perfect_map_id* [*simp*]: $\text{perfect\_map } X \ X \ \text{id}$

by (*simp add: homeomorphic_imp_perfect_map*)

lemma *perfect_map_compose*:

$\llbracket \text{perfect\_map } X \ Y \ f; \text{perfect\_map } Y \ Z \ g \rrbracket \implies \text{perfect\_map } X \ Z \ (g \circ f)$

by (*meson continuous_map_compose perfect_imp_quotient_map perfect_map_def proper_map_compose quotient_map_compose_eq quotient_map_def*)

lemma *perfect_imp_continuous_map*:

$\text{perfect\_map } X \ Y \ f \implies \text{continuous\_map } X \ Y \ f$

using *perfect_map_def* **by** *blast*

lemma *perfect_imp_closed_map*:

$\text{perfect\_map } X \ Y \ f \implies \text{closed\_map } X \ Y \ f$

by (*simp add: perfect_map_def proper_map_def*)

lemma *perfect_imp_proper_map*:

$\text{perfect\_map } X \ Y \ f \implies \text{proper\_map } X \ Y \ f$

by (*simp add: perfect_map_def*)

lemma *perfect_imp_surjective_map*:

$\text{perfect\_map } X \ Y \ f \implies f \text{' (topspace } X) = \text{topspace } Y$

by (*simp add: perfect_map_def*)

end

2.3 Abstract Topology 2

theory *Abstract_Topology_2*

imports

Elementary_Topology

Abstract_Topology

HOL-Library.Indicator_Function

begin

Combination of Elementary and Abstract Topology

lemma *approachable_lt_le2*:

$(\exists (d::\text{real}) > 0. \forall x. Q\ x \longrightarrow f\ x < d \longrightarrow P\ x) \longleftrightarrow (\exists d > 0. \forall x. f\ x \leq d \longrightarrow Q\ x \longrightarrow P\ x)$

apply *auto*

apply (*rule_tac* $x=d/2$ **in** *exI*, *auto*)

done

lemma *triangle_lemma*:

fixes $x\ y\ z :: \text{real}$

assumes $x: 0 \leq x$

and $y: 0 \leq y$

and $z: 0 \leq z$

and $xy: x^2 \leq y^2 + z^2$

shows $x \leq y + z$

proof –

have $y^2 + z^2 \leq y^2 + 2 * y * z + z^2$

using $z\ y$ **by** *simp*

with xy **have** $th: x^2 \leq (y + z)^2$

by (*simp add: power2_eq_square field_simps*)

from $y\ z$ **have** $yz: y + z \geq 0$

by *arith*

from *power2_le_imp_le[OF th yz]* **show** *?thesis* .

qed

lemma *isCont_indicator*:

fixes $x :: 'a::t2\_space$

shows *isCont* (*indicator* $A :: 'a \Rightarrow \text{real}$) $x = (x \notin \text{frontier } A)$

proof *auto*

fix x

assume *cts_at*: *isCont* (*indicator* $A :: 'a \Rightarrow \text{real}$) x **and** *fr*: $x \in \text{frontier } A$

with *continuous_at_open* **have** $1: \forall V::\text{real set. open } V \wedge \text{indicator } A\ x \in V$

$\longrightarrow$

$(\exists U::'a\ \text{set. open } U \wedge x \in U \wedge (\forall y \in U. \text{indicator } A\ y \in V))$ **by** *auto*

show *False*

proof (*cases* $x \in A$)

assume $x: x \in A$

hence *indicator* $A\ x \in (\{0 < .. < 2\} :: \text{real set})$ **by** *simp*

with 1 **obtain** U **where** $U: \text{open } U\ x \in U\ \forall y \in U. \text{indicator } A\ y \in (\{0 < .. < 2\}$

$:: \text{real set})$

```

    using open_greaterThanLessThan by metis
    hence  $\forall y \in U. \text{indicator } A \ y > (0::\text{real})$ 
    unfolding greaterThanLessThan_def by auto
    hence  $U \subseteq A$  using indicator_eq_0_iff by force
    hence  $x \in \text{interior } A$  using U interiorI by auto
    thus ?thesis using fr unfolding frontier_def by simp
next
  assume  $x: x \notin A$ 
  hence  $\text{indicator } A \ x \in (\{-1 < .. < 1\} :: \text{real set})$  by simp
  with 1 obtain U where  $U: \text{open } U \ x \in U \ \forall y \in U. \text{indicator } A \ y \in (\{-1 < .. < 1\} :: \text{real set})$ 
  using 1 open_greaterThanLessThan by metis
  hence  $\forall y \in U. \text{indicator } A \ y < (1::\text{real})$ 
  unfolding greaterThanLessThan_def by auto
  hence  $U \subseteq -A$  by auto
  hence  $x \in \text{interior } (-A)$  using U interiorI by auto
  thus ?thesis using fr interior_complement unfolding frontier_def by auto
qed
next
  assume nfr:  $x \notin \text{frontier } A$ 
  hence  $x \in \text{interior } A \vee x \in \text{interior } (-A)$ 
  by (auto simp: frontier_def closure_interior)
  thus isCont ((indicator A)::'a  $\Rightarrow$  real) x
  proof
    assume int:  $x \in \text{interior } A$ 
    then obtain U where  $U: \text{open } U \ x \in U \ U \subseteq A$  unfolding interior_def by
  auto
    hence  $\forall y \in U. \text{indicator } A \ y = (1::\text{real})$  unfolding indicator_def by auto
    hence continuous_on U (indicator A) by (simp add: indicator_eq_1_iff)
    thus ?thesis using U continuous_on_eq_continuous_at by auto
  next
    assume ext:  $x \in \text{interior } (-A)$ 
    then obtain U where  $U: \text{open } U \ x \in U \ U \subseteq -A$  unfolding interior_def by
  auto
    then have continuous_on U (indicator A)
    using continuous_on_topological by (auto simp: subset_iff)
    thus ?thesis using U continuous_on_eq_continuous_at by auto
  qed
qed

```

lemma *closedin_limpt*:

```

  closedin (top_of_set T) S  $\longleftrightarrow$   $S \subseteq T \wedge (\forall x. x \text{ islimpt } S \wedge x \in T \longrightarrow x \in S)$ 
  apply (simp add: closedin_closed, safe)
  apply (simp add: closed_limpt islimpt_subset)
  apply (rule_tac x=closure S in exI, simp)
  apply (force simp: closure_def)
  done

```

lemma *closedin_closed_eq*: $\text{closed } S \implies \text{closedin } (\text{top\_of\_set } S) \ T \longleftrightarrow \text{closed}$

$T \wedge T \subseteq S$
by (meson closedin_limpt closed_subset closedin_closed_trans)

lemma *connected_closed_set*:

closed S
 $\implies \text{connected } S \longleftrightarrow (\nexists A B. \text{closed } A \wedge \text{closed } B \wedge A \neq \{\} \wedge B \neq \{\} \wedge A \cup B = S \wedge A \cap B = \{\})$
unfolding *connected_closedin_eq closedin_closed_eq connected_closedin_eq* **by** *blast*

If a connected set is written as the union of two nonempty closed sets, then these sets have to intersect.

lemma *connected_as_closed_union*:

assumes *connected C C = A $\cup$ B closed A closed B A $\neq$ {} B $\neq$ {}*
shows *A $\cap$ B $\neq$ {}*
by (metis *assms closed_Un connected_closed_set*)

lemma *closedin_subset_trans*:

closedin (top_of_set U) S $\implies$ S $\subseteq$ T $\implies$ T $\subseteq$ U $\implies$
closedin (top_of_set T) S
by (meson *closedin_limpt subset_iff*)

lemma *openin_subset_trans*:

openin (top_of_set U) S $\implies$ S $\subseteq$ T $\implies$ T $\subseteq$ U $\implies$
openin (top_of_set T) S
by (auto *simp: openin_open*)

lemma *closedin_compact*:

$\llbracket \text{compact } S; \text{closedin (top_of_set S) } T \rrbracket \implies \text{compact } T$
by (metis *closedin_closed compact_Int_closed*)

lemma *closedin_compact_eq*:

fixes *S :: 'a::t2_space set*
shows
compact S
 $\implies (\text{closedin (top_of_set S) } T \longleftrightarrow \text{compact } T \wedge T \subseteq S)$

by (metis *closedin_imp_subset closedin_compact closed_subset compact_imp_closed*)

2.3.1 Closure

lemma *eucclidean_closure_of [simp]*: *eucclidean_closure_of S = closure S*

by (auto *simp: closure_of_def closure_def islimpt_def*)

lemma *closure_openin_Int_closure*:

assumes *ope: openin (top_of_set U) S and T $\subseteq$ U*
shows *closure(S $\cap$ closure T) = closure(S $\cap$ T)*

proof

obtain *V* **where** *open V and S: S = U $\cap$ V*

```

    using ope using openin_open by metis
  show closure (S ∩ closure T) ⊆ closure (S ∩ T)
  proof (clarsimp simp: S)
    fix x
    assume x ∈ closure (U ∩ V ∩ closure T)
    then have V ∩ closure T ⊆ A ⟹ x ∈ closure A for A
      by (metis closure_mono subsetD inf.coboundedI2 inf_assoc)
    then have x ∈ closure (T ∩ V)
      by (metis ⟨open V⟩ closure_closure inf_commute open_Int_closure_subset)
    then show x ∈ closure (U ∩ V ∩ T)
      by (metis ⟨T ⊆ U⟩ inf.absorb_iff2 inf_assoc inf_commute)
  qed
next
  show closure (S ∩ T) ⊆ closure (S ∩ closure T)
  by (meson Int_mono closure_mono closure_subset order_refl)
qed

```

corollary *infinite_openin*:

```

  fixes S :: 'a :: t1_space set
  shows ⟦openin (top_of_set U) S; x ∈ S; x islimpt U⟧ ⟹ infinite S
  by (clarsimp simp add: openin_open islimpt_eq_acc_point inf_commute)

```

lemma *closure_Int_ballI*:

```

  assumes ∧U. ⟦openin (top_of_set S) U; U ≠ {}⟧ ⟹ T ∩ U ≠ {}
  shows S ⊆ closure T

```

proof (clarsimp simp: closure_iff_nhds_not_empty)

fix x and A and V

assume x ∈ S V ⊆ A open V x ∈ V T ∩ A = {}

then have openin (top_of_set S) (A ∩ V ∩ S)

by (auto simp: openin_open intro!: exI[where x=V])

moreover have A ∩ V ∩ S ≠ {} using ⟨x ∈ V⟩ ⟨V ⊆ A⟩ ⟨x ∈ S⟩

by auto

ultimately have T ∩ (A ∩ V ∩ S) ≠ {}

by (rule assms)

with ⟨T ∩ A = {}⟩ show False by auto

qed

2.3.2 Frontier

lemma *euclidean_interior_of* [simp]: euclidean interior_of S = interior S

by (auto simp: interior_of_def interior_def)

lemma *euclidean_frontier_of* [simp]: euclidean frontier_of S = frontier S

by (auto simp: frontier_of_def frontier_def)

lemma *connected_Int_frontier*:

⟦connected s; s ∩ t ≠ {}; s - t ≠ {}⟧ ⟹ (s ∩ frontier t ≠ {})

apply (simp add: frontier_interiors connected_openin, safe)

apply (drule_tac x=s ∩ interior t in spec, safe)

```

  apply (drule_tac [2] x=s  $\cap$  interior  $(-t)$  in spec)
  apply (auto simp: disjoint_eq_subset_Compl dest: interior_subset [THEN subsetD])
done

```

2.3.3 Compactness

lemma *openin_delete*:

```

  fixes a :: 'a :: t1_space
  shows openin (top_of_set u) s
         $\implies$  openin (top_of_set u) (s - {a})
by (metis Int_Diff open_delete openin_open)

```

lemma *compact_eq_openin_cover*:

```

compact S  $\longleftrightarrow$ 
  ( $\forall C. (\forall c \in C. \text{openin } (\text{top\_of\_set } S) c) \wedge S \subseteq \bigcup C \longrightarrow$ 
   ( $\exists D \subseteq C. \text{finite } D \wedge S \subseteq \bigcup D$ ))
proof safe
  fix C
  assume compact S and  $\forall c \in C. \text{openin } (\text{top\_of\_set } S) c$  and  $S \subseteq \bigcup C$ 
  then have  $\forall c \in \{T. \text{open } T \wedge S \cap T \in C\}. \text{open } c$  and  $S \subseteq \bigcup \{T. \text{open } T \wedge S$ 
 $\cap T \in C\}$ 
    unfolding openin_open by force+
  with  $\langle \text{compact } S \rangle$  obtain D where  $D \subseteq \{T. \text{open } T \wedge S \cap T \in C\}$  and finite
  D and  $S \subseteq \bigcup D$ 
    by (meson compactE)
  then have  $\text{image } (\lambda T. S \cap T) D \subseteq C \wedge \text{finite } (\text{image } (\lambda T. S \cap T) D) \wedge S \subseteq$ 
 $\bigcup (\text{image } (\lambda T. S \cap T) D)$ 
    by auto
  then show  $\exists D \subseteq C. \text{finite } D \wedge S \subseteq \bigcup D ..$ 

```

next

```

assume 1:  $\forall C. (\forall c \in C. \text{openin } (\text{top\_of\_set } S) c) \wedge S \subseteq \bigcup C \longrightarrow$ 
  ( $\exists D \subseteq C. \text{finite } D \wedge S \subseteq \bigcup D$ )
show compact S
proof (rule compactI)
  fix C
  let ?C =  $\text{image } (\lambda T. S \cap T) C$ 
  assume  $\forall t \in C. \text{open } t$  and  $S \subseteq \bigcup C$ 
  then have  $(\forall c \in ?C. \text{openin } (\text{top\_of\_set } S) c) \wedge S \subseteq \bigcup ?C$ 
    unfolding openin_open by auto
  with 1 obtain D where  $D \subseteq ?C$  and finite D and  $S \subseteq \bigcup D$ 
    by metis
  let ?D =  $\text{inv\_into } C (\lambda T. S \cap T) ` D$ 
  have  $?D \subseteq C \wedge \text{finite } ?D \wedge S \subseteq \bigcup ?D$ 
proof (intro conjI)
  from  $\langle D \subseteq ?C \rangle$  show  $?D \subseteq C$ 
    by (fast intro: inv_into_into)
  from  $\langle \text{finite } D \rangle$  show finite ?D
    by (rule finite_imageI)

```

```

    from  $\langle S \subseteq \bigcup D \rangle$  show  $S \subseteq \bigcup ?D$ 
    apply (rule subset_trans)
    by (metis Int_Union Int_lower2  $\langle D \subseteq (\cap) S \text{ ' } C \rangle$  image_inv_into_cancel)
  qed
  then show  $\exists D \subseteq C. \text{finite } D \wedge S \subseteq \bigcup D ..$ 
  qed
qed

```

2.3.4 Continuity

```

lemma interior_image_subset:
  assumes  $\text{inj } f \wedge x. \text{continuous } (\text{at } x) f$ 
  shows  $\text{interior } (f \text{ ' } S) \subseteq f \text{ ' } (\text{interior } S)$ 
proof
  fix x assume  $x \in \text{interior } (f \text{ ' } S)$ 
  then obtain T where as:  $\text{open } T \ x \in T \ T \subseteq f \text{ ' } S ..$ 
  then have  $x \in f \text{ ' } S$  by auto
  then obtain y where  $y \in S \ x = f \ y$  by auto
  have  $\text{open } (f \text{ - ' } T)$ 
    using assms  $\langle \text{open } T \rangle$  by (simp add: continuous_at_imp_continuous_on
open_vimage)
  moreover have  $y \in \text{vimage } f \ T$ 
    using  $\langle x = f \ y \rangle \langle x \in T \rangle$  by simp
  moreover have  $\text{vimage } f \ T \subseteq S$ 
    using  $\langle T \subseteq \text{image } f \ S \rangle \langle \text{inj } f \rangle$  unfolding inj_on_def subset_eq by auto
  ultimately have  $y \in \text{interior } S ..$ 
  with  $\langle x = f \ y \rangle$  show  $x \in f \text{ ' } \text{interior } S ..$ 
qed

```

2.3.5 Equality of continuous functions on closure and related results

```

lemma continuous_closedin_preimage_constant:
  fixes  $f :: \_ \Rightarrow 'b::t1\_space$ 
  shows  $\text{continuous\_on } S \ f \Longrightarrow \text{closedin } (\text{top\_of\_set } S) \ \{x \in S. f \ x = a\}$ 
  using continuous_closedin_preimage[of  $S \ f \ \{a\}$ ] by (simp add: vimage_def Collect_conj_eq)

```

```

lemma continuous_closed_preimage_constant:
  fixes  $f :: \_ \Rightarrow 'b::t1\_space$ 
  shows  $\text{continuous\_on } S \ f \Longrightarrow \text{closed } S \Longrightarrow \text{closed } \{x \in S. f \ x = a\}$ 
  using continuous_closed_preimage[of  $S \ f \ \{a\}$ ] by (simp add: vimage_def Collect_conj_eq)

```

```

lemma continuous_constant_on_closure:
  fixes  $f :: \_ \Rightarrow 'b::t1\_space$ 
  assumes  $\text{continuous\_on } (\text{closure } S) \ f$ 
  and  $\bigwedge x. x \in S \Longrightarrow f \ x = a$ 
  and  $x \in \text{closure } S$ 

```

```

shows  $f x = a$ 
  using continuous_closed_preimage_constant[of closure S f a]
    assms closure_minimal[of S  $\{x \in \text{closure } S. f x = a\}$ ] closure_subset
  unfolding subset_eq
  by auto

```

```

lemma image_closure_subset:
  assumes conf: continuous_on (closure S) f
    and closed T
    and  $f^{-1}(S) \subseteq T$ 
  shows  $f^{-1}(\text{closure } S) \subseteq T$ 
proof -
  have  $S \subseteq \{x \in \text{closure } S. f x \in T\}$ 
    using assms(3) closure_subset by auto
  moreover have closed  $(\text{closure } S \cap f^{-1}(T))$ 
    using continuous_closed_preimage[OF conf]  $\langle \text{closed } T \rangle$  by auto
  ultimately have  $\text{closure } S = (\text{closure } S \cap f^{-1}(T))$ 
    using closure_minimal[of S  $(\text{closure } S \cap f^{-1}(T))$ ] by auto
  then show ?thesis by auto
qed

```

2.3.6 A function constant on a set

```

definition constant_on (infixl (constant'_on) 50)
  where  $f \text{ constant\_on } A \equiv \exists y. \forall x \in A. f x = y$ 

```

```

lemma constant_on_subset:  $\llbracket f \text{ constant\_on } A; B \subseteq A \rrbracket \implies f \text{ constant\_on } B$ 
  unfolding constant_on_def by blast

```

```

lemma injective_not_constant:
  fixes  $S :: 'a::\{\text{perfect\_space}\} \text{ set}$ 
  shows  $\llbracket \text{open } S; \text{inj\_on } f S; f \text{ constant\_on } S \rrbracket \implies S = \{\}$ 
unfolding constant_on_def
by (metis equals0I inj_on_contraD islimpt_UNIV islimpt_def)

```

```

lemma constant_on_closureI:
  fixes  $f :: \_ \Rightarrow 'b::t1\_space$ 
  assumes cof:  $f \text{ constant\_on } S$  and conf: continuous_on (closure S) f
  shows  $f \text{ constant\_on } (\text{closure } S)$ 
using continuous_constant_on_closure [OF conf] cof unfolding constant_on_def
by metis

```

2.3.7 Continuity relative to a union.

```

lemma continuous_on_Un_local:
   $\llbracket \text{closedin } (\text{top\_of\_set } (s \cup t)) \ s; \text{closedin } (\text{top\_of\_set } (s \cup t)) \ t;$ 
     $\text{continuous\_on } s \ f; \text{continuous\_on } t \ f \rrbracket$ 
 $\implies \text{continuous\_on } (s \cup t) \ f$ 
unfolding continuous_on_closedin_limpt
by (metis Lim_trivial_limit Lim_within_union Un_iff trivial_limit_within)

```

lemma *continuous_on_cases_local*:

$\llbracket \text{closedin } (\text{top\_of\_set } (s \cup t)) \ s; \text{closedin } (\text{top\_of\_set } (s \cup t)) \ t; \\ \text{continuous\_on } s \ f; \text{continuous\_on } t \ g; \\ \bigwedge x. \llbracket x \in s \wedge \neg P \ x \vee x \in t \wedge P \ x \rrbracket \implies f \ x = g \ x \rrbracket \\ \implies \text{continuous\_on } (s \cup t) \ (\lambda x. \text{if } P \ x \text{ then } f \ x \text{ else } g \ x) \\ \text{by (rule continuous\_on\_Un\_local) (auto intro: continuous\_on\_eq)}$

lemma *continuous_on_cases_le*:

fixes $h :: 'a :: \text{topological\_space} \Rightarrow \text{real}$
assumes $\text{continuous\_on } \{t \in s. h \ t \leq a\} \ f$
and $\text{continuous\_on } \{t \in s. a \leq h \ t\} \ g$
and $h: \text{continuous\_on } s \ h$
and $\bigwedge t. \llbracket t \in s; h \ t = a \rrbracket \implies f \ t = g \ t$
shows $\text{continuous\_on } s \ (\lambda t. \text{if } h \ t \leq a \text{ then } f(t) \text{ else } g(t))$
proof –
have $s = (s \cap h - 'atMost \ a) \cup (s \cap h - 'atLeast \ a)$
by *force*
have $1: \text{closedin } (\text{top\_of\_set } s) \ (s \cap h - 'atMost \ a)$
by (rule *continuous_closedin_preimage* [OF $h \ \text{closed\_atMost}$])
have $2: \text{closedin } (\text{top\_of\_set } s) \ (s \cap h - 'atLeast \ a)$
by (rule *continuous_closedin_preimage* [OF $h \ \text{closed\_atLeast}$])
have $\text{eq}: s \cap h - '\{..a\} = \{t \in s. h \ t \leq a\} \ s \cap h - '\{a..\} = \{t \in s. a \leq h \ t\}$
by *auto*
show *?thesis*
apply (rule *continuous_on_subset* [of $s, OF \ \_\text{order\_refl}$])
apply (*subst* s)
apply (rule *continuous_on_cases_local*)
using $1 \ 2 \ s \ \text{assms}$ **apply** (*auto simp: eq*)
done

qed

lemma *continuous_on_cases_1*:

fixes $s :: \text{real set}$
assumes $\text{continuous\_on } \{t \in s. t \leq a\} \ f$
and $\text{continuous\_on } \{t \in s. a \leq t\} \ g$
and $a \in s \implies f \ a = g \ a$
shows $\text{continuous\_on } s \ (\lambda t. \text{if } t \leq a \text{ then } f(t) \text{ else } g(t))$
using *assms*
by (*auto intro: continuous_on_cases_le* [where $h = \text{id}$, *simplified*])

2.3.8 Inverse function property for open/closed maps

lemma *continuous_on_inverse_open_map*:

assumes *contf*: $\text{continuous\_on } S \ f$
and *imf*: $f ' S = T$
and *injf*: $\bigwedge x. x \in S \implies g \ (f \ x) = x$
and *oo*: $\bigwedge U. \text{openin } (\text{top\_of\_set } S) \ U \implies \text{openin } (\text{top\_of\_set } T) \ (f ' U)$
shows $\text{continuous\_on } T \ g$

proof –

from *imf injf* **have** *gTS*: $g \circ T = S$

by *force*

from *imf injf* **have** *fU*: $U \subseteq S \implies (f \circ U) = T \cap g \circ U$ **for** *U*

by *force*

show *?thesis*

by (*simp add: continuous_on_open [of T g] gTS*) (*metis openin_imp_subset fU oo*)

qed

lemma *continuous_on_inverse_closed_map*:

assumes *contf*: *continuous_on S f*

and *imf*: $f \circ S = T$

and *injf*: $\bigwedge x. x \in S \implies g(f x) = x$

and *oo*: $\bigwedge U. \text{closedin } (\text{top\_of\_set } S) U \implies \text{closedin } (\text{top\_of\_set } T) (f \circ U)$

shows *continuous_on T g*

proof –

from *imf injf* **have** *gTS*: $g \circ T = S$

by *force*

from *imf injf* **have** *fU*: $U \subseteq S \implies (f \circ U) = T \cap g \circ U$ **for** *U*

by *force*

show *?thesis*

by (*simp add: continuous_on_closed [of T g] gTS*) (*metis closedin_imp_subset fU oo*)

qed

lemma *homeomorphism_injective_open_map*:

assumes *contf*: *continuous_on S f*

and *imf*: $f \circ S = T$

and *injf*: *inj_on f S*

and *oo*: $\bigwedge U. \text{openin } (\text{top\_of\_set } S) U \implies \text{openin } (\text{top\_of\_set } T) (f \circ U)$

obtains *g* **where** *homeomorphism S T f g*

proof

have *continuous_on T (inv_into S f)*

by (*metis contf continuous_on_inverse_open_map imf injf inv_into_f_f oo*)

with *imf injf contf* **show** *homeomorphism S T f (inv_into S f)*

by (*auto simp: homeomorphism_def*)

qed

lemma *homeomorphism_injective_closed_map*:

assumes *contf*: *continuous_on S f*

and *imf*: $f \circ S = T$

and *injf*: *inj_on f S*

and *oo*: $\bigwedge U. \text{closedin } (\text{top\_of\_set } S) U \implies \text{closedin } (\text{top\_of\_set } T) (f \circ U)$

obtains *g* **where** *homeomorphism S T f g*

proof

have *continuous_on T (inv_into S f)*

by (*metis contf continuous_on_inverse_closed_map imf injf inv_into_f_f oo*)

with *imf injf contf* **show** *homeomorphism S T f (inv_into S f)*

by (auto simp: homeomorphism_def)
qed

lemma *homeomorphism_imp_open_map*:
 assumes *hom*: *homeomorphism* *S* *T* *f* *g*
 and *oo*: *openin* (*top_of_set* *S*) *U*
 shows *openin* (*top_of_set* *T*) (*f* ' *U*)
proof –
 from *hom oo* have [*simp*]: $f \text{ ' } U = T \cap g \text{ - ' } U$
 using *openin_subset* **by** (fastforce simp: *homeomorphism_def* *rev_image_eqI*)
 from *hom* have *continuous_on* *T* *g*
 unfolding *homeomorphism_def* **by** blast
 moreover have $g \text{ ' } T = S$
 by (*metis hom homeomorphism_def*)
 ultimately show ?thesis
 by (*simp add: continuous_on_open oo*)
 qed

lemma *homeomorphism_imp_closed_map*:
 assumes *hom*: *homeomorphism* *S* *T* *f* *g*
 and *oo*: *closedin* (*top_of_set* *S*) *U*
 shows *closedin* (*top_of_set* *T*) (*f* ' *U*)
proof –
 from *hom oo* have [*simp*]: $f \text{ ' } U = T \cap g \text{ - ' } U$
 using *closedin_subset* **by** (fastforce simp: *homeomorphism_def* *rev_image_eqI*)
 from *hom* have *continuous_on* *T* *g*
 unfolding *homeomorphism_def* **by** blast
 moreover have $g \text{ ' } T = S$
 by (*metis hom homeomorphism_def*)
 ultimately show ?thesis
 by (*simp add: continuous_on_closed oo*)
 qed

2.3.9 Seperability

lemma *subset_second_countable*:
 obtains $\mathcal{B} :: 'a :: \text{second\_countable\_topology set set}$
 where *countable* $\mathcal{B}$
 $\{\} \notin \mathcal{B}$
 $\bigwedge C. C \in \mathcal{B} \implies \text{openin}(\text{top\_of\_set } S) C$
 $\bigwedge T. \text{openin}(\text{top\_of\_set } S) T \implies \exists \mathcal{U}. \mathcal{U} \subseteq \mathcal{B} \wedge T = \bigcup \mathcal{U}$
proof –
 obtain $\mathcal{B} :: 'a \text{ set set}$
 where *countable* $\mathcal{B}$
 and *opeB*: $\bigwedge C. C \in \mathcal{B} \implies \text{openin}(\text{top\_of\_set } S) C$
 and $\mathcal{B}$: $\bigwedge T. \text{openin}(\text{top\_of\_set } S) T \implies \exists \mathcal{U}. \mathcal{U} \subseteq \mathcal{B} \wedge T = \bigcup \mathcal{U}$
proof –
 obtain $\mathcal{C} :: 'a \text{ set set}$
 where *countable* $\mathcal{C}$ and *ope*: $\bigwedge C. C \in \mathcal{C} \implies \text{open } C$


```

    and  $\mathcal{C}$ :  $\bigwedge S. \text{open } S \implies \exists U. U \subseteq \mathcal{C} \wedge S = \bigcup U$ 
  by (metis univ_second_countable that)
show ?thesis
proof
  show countable (( $\lambda C. S \cap C$ ) '  $\mathcal{C}$ )
    by (simp add: 'countable  $\mathcal{C}$ ')
  show  $\bigwedge C. C \in (\cap) S \text{ ' } \mathcal{C} \implies \text{openin } (\text{top\_of\_set } S) C$ 
    using ope by auto
  show  $\bigwedge T. \text{openin } (\text{top\_of\_set } S) T \implies \exists \mathcal{U} \subseteq (\cap) S \text{ ' } \mathcal{C}. T = \bigcup \mathcal{U}$ 
    by (metis  $\mathcal{C}$  image_mono inf_Sup openin_open)
qed
qed
show ?thesis
proof
  show countable ( $\mathcal{B} - \{\{\}\}$ )
    using 'countable  $\mathcal{B}$ ' by blast
  show  $\bigwedge C. \llbracket C \in \mathcal{B} - \{\{\}\} \rrbracket \implies \text{openin } (\text{top\_of\_set } S) C$ 
    by (simp add: '  $\bigwedge C. C \in \mathcal{B} \implies \text{openin } (\text{top\_of\_set } S) C$  ')
  show  $\exists \mathcal{U} \subseteq \mathcal{B} - \{\{\}\}. T = \bigcup \mathcal{U}$  if  $\text{openin } (\text{top\_of\_set } S) T$  for  $T$ 
    using  $\mathcal{B}$  [OF that]
    apply clarify
    apply (rule_tac  $x=\mathcal{U} - \{\{\}\}$  in exI, auto)
    done
qed auto
qed

lemma Lindelof_openin:
  fixes  $\mathcal{F} :: 'a::\text{second\_countable\_topology}$  set set
  assumes  $\bigwedge S. S \in \mathcal{F} \implies \text{openin } (\text{top\_of\_set } U) S$ 
  obtains  $\mathcal{F}'$  where  $\mathcal{F}' \subseteq \mathcal{F}$  countable  $\mathcal{F}' \cup \mathcal{F}' = \bigcup \mathcal{F}$ 
proof -
  have  $\bigwedge S. S \in \mathcal{F} \implies \exists T. \text{open } T \wedge S = U \cap T$ 
    using assms by (simp add: openin_open)
  then obtain  $tf$  where  $tf: \bigwedge S. S \in \mathcal{F} \implies \text{open } (tf\ S) \wedge (S = U \cap tf\ S)$ 
    by metis
  have [simp]:  $\bigwedge \mathcal{F}'. \mathcal{F}' \subseteq \mathcal{F} \implies \bigcup \mathcal{F}' = U \cap \bigcup (tf \text{ ' } \mathcal{F}')$ 
    using  $tf$  by fastforce
  obtain  $\mathcal{G}$  where countable  $\mathcal{G} \wedge \mathcal{G} \subseteq tf \text{ ' } \mathcal{F} \cup \mathcal{G} = \bigcup (tf \text{ ' } \mathcal{F})$ 
    using  $tf$  by (force intro: Lindelof [of  $tf \text{ ' } \mathcal{F}$ ])
  then obtain  $\mathcal{F}'$  where  $\mathcal{F}': \mathcal{F}' \subseteq \mathcal{F}$  countable  $\mathcal{F}' \cup \mathcal{F}' = \bigcup \mathcal{F}$ 
    by (clarsimp simp add: countable_subset_image)
  then show ?thesis ..
qed

```

2.3.10 Closed Maps

```

lemma continuous_imp_closed_map:
  fixes  $f :: 'a::t2\_space \Rightarrow 'b::t2\_space$ 
  assumes closedin ( $\text{top\_of\_set } S$ )  $U$ 

```

$continuous\_on\ S\ f\ f^{-1} S = T\ compact\ S$
shows $closedin\ (top\_of\_set\ T)\ (f^{-1} U)$
by (*metis* *assms* $closedin\_compact\_eq\ compact\_continuous\_image\ continuous\_on\_subset\_subset\_image\_iff$)

lemma *closed_map_restrict*:
assumes $cloU$: $closedin\ (top\_of\_set\ (S \cap f^{-1} T'))\ U$
and cc : $\bigwedge U. closedin\ (top\_of\_set\ S)\ U \implies closedin\ (top\_of\_set\ T)\ (f^{-1} U)$
and $T' \subseteq T$
shows $closedin\ (top\_of\_set\ T')\ (f^{-1} U)$
proof –
obtain V **where** $closed\ V\ U = S \cap f^{-1} T' \cap V$
using $cloU$ **by** (*auto simp: closedin_closed*)
with $cc\ [of\ S \cap V]\ \langle T' \subseteq T \rangle$ **show** *?thesis*
by (*fastforce simp add: closedin_closed*)
qed

2.3.11 Open Maps

lemma *open_map_restrict*:
assumes $opeU$: $openin\ (top\_of\_set\ (S \cap f^{-1} T'))\ U$
and oo : $\bigwedge U. openin\ (top\_of\_set\ S)\ U \implies openin\ (top\_of\_set\ T)\ (f^{-1} U)$
and $T' \subseteq T$
shows $openin\ (top\_of\_set\ T')\ (f^{-1} U)$
proof –
obtain V **where** $open\ V\ U = S \cap f^{-1} T' \cap V$
using $opeU$ **by** (*auto simp: openin_open*)
with $oo\ [of\ S \cap V]\ \langle T' \subseteq T \rangle$ **show** *?thesis*
by (*fastforce simp add: openin_open*)
qed

2.3.12 Quotient maps

lemma *quotient_map_imp_continuous_open*:
assumes T : $f^{-1} S \subseteq T$
and ope : $\bigwedge U. U \subseteq T \implies (openin\ (top\_of\_set\ S)\ (S \cap f^{-1} U) \longleftrightarrow openin\ (top\_of\_set\ T)\ U)$
shows $continuous\_on\ S\ f$
proof –
have [*simp*]: $S \cap f^{-1} f^{-1} S = S$ **by** *auto*
show *?thesis*
by (*meson* $T\ continuous\_on\_open\_gen\ ope\ openin\_imp\_subset$)
qed

lemma *quotient_map_imp_continuous_closed*:
assumes T : $f^{-1} S \subseteq T$
and ope : $\bigwedge U. U \subseteq T \implies (closedin\ (top\_of\_set\ S)\ (S \cap f^{-1} U) \longleftrightarrow closedin\ (top\_of\_set\ T)\ U)$

```

    shows continuous_on S f
  proof -
    have [simp]:  $S \cap f^{-1} f^{-1} S = S$  by auto
    show ?thesis
      by (meson T closedin_imp_subset continuous_on_closed_gen ope)
  qed

```

```

lemma open_map_imp_quotient_map:
  assumes conf: continuous_on S f
    and T:  $T \subseteq f^{-1} S$ 
    and ope:  $\bigwedge T. \text{openin}(\text{top\_of\_set } S) T \implies \text{openin}(\text{top\_of\_set } (f^{-1} S)) (f^{-1} T)$ 
  shows openin (top_of_set S) ( $S \cap f^{-1} T$ ) =
    openin (top_of_set (f^{-1} S)) T
  proof -
    have  $T = f^{-1} (S \cap f^{-1} T)$ 
    using T by blast
    then show ?thesis
      using ope conf continuous_on_open by metis
  qed

```

```

lemma closed_map_imp_quotient_map:
  assumes conf: continuous_on S f
    and T:  $T \subseteq f^{-1} S$ 
    and ope:  $\bigwedge T. \text{closedin}(\text{top\_of\_set } S) T \implies \text{closedin}(\text{top\_of\_set } (f^{-1} S)) (f^{-1} T)$ 
  shows openin (top_of_set S) ( $S \cap f^{-1} T$ )  $\longleftrightarrow$ 
    openin (top_of_set (f^{-1} S)) T
    (is ?lhs = ?rhs)
  proof
    assume ?lhs
    then have *:  $\text{closedin}(\text{top\_of\_set } S) (S - (S \cap f^{-1} T))$ 
    using closedin_diff by fastforce
    have [simp]:  $(f^{-1} S - f^{-1} (S - (S \cap f^{-1} T))) = T$ 
    using T by blast
    show ?rhs
      using ope [OF *, unfolded closedin_def] by auto
  next
    assume ?rhs
    with conf show ?lhs
      by (auto simp: continuous_on_open)
  qed

```

```

lemma continuous_right_inverse_imp_quotient_map:
  assumes conf: continuous_on S f and imf:  $f^{-1} S \subseteq T$ 
    and contg: continuous_on T g and img:  $g^{-1} T \subseteq S$ 
    and fg [simp]:  $\bigwedge y. y \in T \implies f(g y) = y$ 
    and U:  $U \subseteq T$ 
  shows openin (top_of_set S) ( $S \cap f^{-1} U$ )  $\longleftrightarrow$ 

```

```

      openin (top_of_set T) U
    (is ?lhs = ?rhs)
  proof -
    have f:  $\bigwedge Z. \text{openin } (\text{top\_of\_set } (f \text{ ' } S)) Z \implies$ 
       $\text{openin } (\text{top\_of\_set } S) (S \cap f \text{ ' } Z)$ 
    and g:  $\bigwedge Z. \text{openin } (\text{top\_of\_set } (g \text{ ' } T)) Z \implies$ 
       $\text{openin } (\text{top\_of\_set } T) (T \cap g \text{ ' } Z)$ 
    using contf contg by (auto simp: continuous_on_open)
    show ?thesis
  proof
    have  $T \cap g \text{ ' } (g \text{ ' } T \cap (S \cap f \text{ ' } U)) = \{x \in T. f (g x) \in U\}$ 
    using imf img by blast
    also have ... = U
    using U by auto
    finally have eq:  $T \cap g \text{ ' } (g \text{ ' } T \cap (S \cap f \text{ ' } U)) = U$  .
    assume ?lhs
    then have *:  $\text{openin } (\text{top\_of\_set } (g \text{ ' } T)) (g \text{ ' } T \cap (S \cap f \text{ ' } U))$ 
    by (meson img openin_Int openin_subtopology_Int_subset openin_subtopology_self)
    show ?rhs
    using g [OF *] eq by auto
  next
    assume rhs: ?rhs
    show ?lhs
    by (metis f fg image_eqI image_subset_iff imf img openin_subopen openin_subtopology_self
      openin_trans rhs)
  qed
qed

lemma continuous_left_inverse_imp_quotient_map:
  assumes continuous_on S f
    and continuous_on (f ' S) g
    and  $\bigwedge x. x \in S \implies g(f x) = x$ 
    and  $U \subseteq f \text{ ' } S$ 
  shows  $\text{openin } (\text{top\_of\_set } S) (S \cap f \text{ ' } U) \longleftrightarrow$ 
     $\text{openin } (\text{top\_of\_set } (f \text{ ' } S)) U$ 
apply (rule continuous_right_inverse_imp_quotient_map)
using assms apply force+
done

lemma continuous_imp_quotient_map:
  fixes f :: 'a::t2_space  $\Rightarrow$  'b::t2_space
  assumes continuous_on S f f ' S = T compact S U  $\subseteq$  T
  shows  $\text{openin } (\text{top\_of\_set } S) (S \cap f \text{ ' } U) \longleftrightarrow$ 
     $\text{openin } (\text{top\_of\_set } T) U$ 
  by (metis (no_types, lifting) assms closed_map_imp_quotient_map continuous_imp_closed_map)

```

2.3.13 Pasting lemmas for functions, for of casewise definitions

on open sets

lemma *pasting_lemma*:

```

assumes ope:  $\bigwedge i. i \in I \implies \text{openin } X (T i)$ 
and cont:  $\bigwedge i. i \in I \implies \text{continuous\_map}(\text{subtopology } X (T i)) Y (f i)$ 
and f:  $\bigwedge i j x. \llbracket i \in I; j \in I; x \in \text{topspace } X \cap T i \cap T j \rrbracket \implies f i x = f j x$ 
and g:  $\bigwedge x. x \in \text{topspace } X \implies \exists j. j \in I \wedge x \in T j \wedge g x = f j x$ 
shows  $\text{continuous\_map } X Y g$ 
unfolding  $\text{continuous\_map\_openin\_preimage\_eq}$ 
proof (intro conjI allI impI)
  show  $g \text{ 'topspace } X \subseteq \text{topspace } Y$ 
  using g cont  $\text{continuous\_map\_image\_subset\_topspace}$  by fastforce
next
  fix U
  assume Y:  $\text{openin } Y U$ 
  have T:  $T i \subseteq \text{topspace } X$  if  $i \in I$  for i
  using ope by (simp add:  $\text{openin\_subset}$  that)
  have *:  $\text{topspace } X \cap g \text{ ' } U = (\bigcup i \in I. T i \cap f i \text{ ' } U)$ 
  using f g T by fastforce
  have  $\bigwedge i. i \in I \implies \text{openin } X (T i \cap f i \text{ ' } U)$ 
  using cont unfolding  $\text{continuous\_map\_openin\_preimage\_eq}$ 
  by (metis Y T inf.commute inf_absorb1 ope  $\text{topspace\_subtopology}$   $\text{openin\_trans\_full}$ )
  then show  $\text{openin } X (\text{topspace } X \cap g \text{ ' } U)$ 
  by (auto simp: *)
qed

```

lemma *pasting_lemma_exists*:

```

assumes X:  $\text{topspace } X \subseteq (\bigcup i \in I. T i)$ 
and ope:  $\bigwedge i. i \in I \implies \text{openin } X (T i)$ 
and cont:  $\bigwedge i. i \in I \implies \text{continuous\_map}(\text{subtopology } X (T i)) Y (f i)$ 
and f:  $\bigwedge i j x. \llbracket i \in I; j \in I; x \in \text{topspace } X \cap T i \cap T j \rrbracket \implies f i x = f j x$ 
obtains g where  $\text{continuous\_map } X Y g \wedge \bigwedge x i. \llbracket i \in I; x \in \text{topspace } X \cap T i \rrbracket \implies g x = f i x$ 
proof
  let ?h =  $\lambda x. f (\text{SOME } i. i \in I \wedge x \in T i) x$ 
  show  $\text{continuous\_map } X Y ?h$ 
  apply (rule pasting_lemma [OF ope cont])
  apply (blast intro: f)+
  by (metis (no_types, lifting) UN_E X subsetD someI_ex)
  show  $f (\text{SOME } i. i \in I \wedge x \in T i) x = f i x$  if  $i \in I$   $x \in \text{topspace } X \cap T i$  for
  i x
  by (metis (no_types, lifting) IntD2 IntI f someI_ex that)
qed

```

lemma *pasting_lemma_locally_finite*:

```

assumes fin:  $\bigwedge x. x \in \text{topspace } X \implies \exists V. \text{openin } X V \wedge x \in V \wedge \text{finite } \{i \in I. T i \cap V \neq \{\}\}$ 

```

```

    and clo:  $\bigwedge i. i \in I \implies \text{closedin } X (T i)$ 
    and cont:  $\bigwedge i. i \in I \implies \text{continuous\_map}(\text{subtopology } X (T i)) Y (f i)$ 
    and f:  $\bigwedge i j x. \llbracket i \in I; j \in I; x \in \text{topspace } X \cap T i \cap T j \rrbracket \implies f i x = f j x$ 
    and g:  $\bigwedge x. x \in \text{topspace } X \implies \exists j. j \in I \wedge x \in T j \wedge g x = f j x$ 
  shows continuous_map X Y g
  unfolding continuous_map_closedin_preimage_eq
proof (intro conjI allI impI)
  show g '  $\text{topspace } X \subseteq \text{topspace } Y$ 
    using g cont continuous_map_image_subset_topspace by fastforce
next
fix U
assume Y: closedin Y U
have T:  $T i \subseteq \text{topspace } X$  if  $i \in I$  for i
  using clo by (simp add: closedin_subset that)
have *:  $\text{topspace } X \cap g^{-1} U = (\bigcup i \in I. T i \cap f i^{-1} U)$ 
  using f g T by fastforce
have cTf:  $\bigwedge i. i \in I \implies \text{closedin } X (T i \cap f i^{-1} U)$ 
  using cont unfolding continuous_map_closedin_preimage_eq topspace_subtopology
  by (simp add: Int_absorb1 T Y clo closedin_closed_subtopology)
have sub:  $\{Z \in (\lambda i. T i \cap f i^{-1} U) ' I. Z \cap V \neq \{\}\}$ 
   $\subseteq (\lambda i. T i \cap f i^{-1} U) ' \{i \in I. T i \cap V \neq \{\}\}$  for V
  by auto
have 1:  $(\bigcup i \in I. T i \cap f i^{-1} U) \subseteq \text{topspace } X$ 
  using T by blast
then have lf: locally_finite_in X (( $\lambda i. T i \cap f i^{-1} U$ ) ' I)
  unfolding locally_finite_in_def
  using finite_subset [OF sub] fin by force
show closedin X ( $\text{topspace } X \cap g^{-1} U$ )
  apply (subst *)
  apply (rule closedin_locally_finite_Union)
  apply (auto intro: cTf lf)
done
qed

```

Likewise on closed sets, with a finiteness assumption

lemma pasting_lemma_closed:

```

  assumes fin: finite I
    and clo:  $\bigwedge i. i \in I \implies \text{closedin } X (T i)$ 
    and cont:  $\bigwedge i. i \in I \implies \text{continuous\_map}(\text{subtopology } X (T i)) Y (f i)$ 
    and f:  $\bigwedge i j x. \llbracket i \in I; j \in I; x \in \text{topspace } X \cap T i \cap T j \rrbracket \implies f i x = f j x$ 
    and g:  $\bigwedge x. x \in \text{topspace } X \implies \exists j. j \in I \wedge x \in T j \wedge g x = f j x$ 
  shows continuous_map X Y g
  using pasting_lemma_locally_finite [OF _ clo cont f g] fin by auto

```

lemma pasting_lemma_exists_locally_finite:

```

  assumes fin:  $\bigwedge x. x \in \text{topspace } X \implies \exists V. \text{openin } X V \wedge x \in V \wedge \text{finite } \{i \in$ 
 $I. T i \cap V \neq \{\}\}$ 
    and X:  $\text{topspace } X \subseteq \bigcup (T ' I)$ 

```

```

    and clo:  $\bigwedge i. i \in I \implies \text{closedin } X (T i)$ 
    and cont:  $\bigwedge i. i \in I \implies \text{continuous\_map}(\text{subtopology } X (T i)) Y (f i)$ 
    and f:  $\bigwedge i j x. \llbracket i \in I; j \in I; x \in \text{topspace } X \cap T i \cap T j \rrbracket \implies f i x = f j x$ 
    and g:  $\bigwedge x. x \in \text{topspace } X \implies \exists j. j \in I \wedge x \in T j \wedge g x = f j x$ 
    obtains g where continuous_map X Y g  $\bigwedge x i. \llbracket i \in I; x \in \text{topspace } X \cap T i \rrbracket$ 
 $\implies g x = f i x$ 

```

proof

```

    show continuous_map X Y ( $\lambda x. f(@i. i \in I \wedge x \in T i) x$ )
    apply (rule pasting_lemma_locally_finite [OF fin])
    apply (blast intro: assms)+
    by (metis (no_types, lifting) UN_E X set_rev_mp someI_ex)

```

next

```

    fix x i
    assume i  $\in I$  and  $x \in \text{topspace } X \cap T i$ 
    show f (SOME i.  $i \in I \wedge x \in T i$ ) x = f i x
    apply (rule someI2_ex)
    using  $\langle i \in I \rangle \langle x \in \text{topspace } X \cap T i \rangle$  apply blast
    by (meson Int_iff  $\langle i \in I \rangle \langle x \in \text{topspace } X \cap T i \rangle$  f)

```

qed

lemma pasting_lemma_exists_closed:

```

    assumes fin: finite I
    and X:  $\text{topspace } X \subseteq \bigcup (T ` I)$ 
    and clo:  $\bigwedge i. i \in I \implies \text{closedin } X (T i)$ 
    and cont:  $\bigwedge i. i \in I \implies \text{continuous\_map}(\text{subtopology } X (T i)) Y (f i)$ 
    and f:  $\bigwedge i j x. \llbracket i \in I; j \in I; x \in \text{topspace } X \cap T i \cap T j \rrbracket \implies f i x = f j x$ 
    obtains g where continuous_map X Y g  $\bigwedge x i. \llbracket i \in I; x \in \text{topspace } X \cap T i \rrbracket$ 
 $\implies g x = f i x$ 

```

proof

```

    show continuous_map X Y ( $\lambda x. f (SOME i. i \in I \wedge x \in T i) x$ )
    apply (rule pasting_lemma_closed [OF  $\langle \text{finite } I \rangle$  clo cont])
    apply (blast intro: f)+
    by (metis (mono_tags, lifting) UN_iff X someI_ex subset_iff)

```

next

```

    fix x i
    assume i  $\in I$   $x \in \text{topspace } X \cap T i$ 
    then show f (SOME i.  $i \in I \wedge x \in T i$ ) x = f i x
    by (metis (no_types, lifting) IntD2 IntI f someI_ex)

```

qed

lemma continuous_map_cases:

```

    assumes f: continuous_map (subtopology X (X closure_of {x. P x})) Y f
    and g: continuous_map (subtopology X (X closure_of {x.  $\neg P x$ })) Y g
    and fg:  $\bigwedge x. x \in X \text{ frontier\_of } \{x. P x\} \implies f x = g x$ 
    shows continuous_map X Y ( $\lambda x. \text{if } P x \text{ then } f x \text{ else } g x$ )

```

proof (rule pasting_lemma_closed)

```

    let ?f =  $\lambda b. \text{if } b \text{ then } f \text{ else } g$ 
    let ?g =  $\lambda x. \text{if } P x \text{ then } f x \text{ else } g x$ 
    let ?T =  $\lambda b. \text{if } b \text{ then } X \text{ closure\_of } \{x. P x\} \text{ else } X \text{ closure\_of } \{x. \sim P x\}$ 

```

```

show finite {True,False} by auto
have eq: topspace X - Collect P = topspace X  $\cap$  {x.  $\neg$  P x}
by blast
show ?f i x = ?f j x
if i  $\in$  {True,False} j  $\in$  {True,False} and x: x  $\in$  topspace X  $\cap$  ?T i  $\cap$  ?T j for
i j x
proof -
have f x = g x
if i  $\neg$  j
apply (rule fg)
unfolding frontier_of_closures eq
using x that closure_of_restrict by fastforce
moreover
have g x = f x
if x  $\in$  X closure_of {x.  $\neg$  P x} x  $\in$  X closure_of Collect P  $\neg$  i j for x
apply (rule fg [symmetric])
unfolding frontier_of_closures eq
using x that closure_of_restrict by fastforce
ultimately show ?thesis
using that by (auto simp flip: closure_of_restrict)
qed
show  $\exists j. j \in \{True, False\} \wedge x \in ?T j \wedge (if P x then f x else g x) = ?f j x$ 
if x  $\in$  topspace X for x
apply simp
apply safe
apply (metis Int_iff closure_of inf_sup_absorb mem_Collect_eq that)
by (metis DiffI eq closure_of_subset_Int contra_subsetD mem_Collect_eq
that)
qed (auto simp: f g)

```

lemma continuous_map_cases_alt:

```

assumes f: continuous_map (subtopology X (X closure_of {x  $\in$  topspace X. P
x})) Y f
and g: continuous_map (subtopology X (X closure_of {x  $\in$  topspace X.  $\sim$  P
x})) Y g
and fg:  $\bigwedge x. x \in X$  frontier_of {x  $\in$  topspace X. P x}  $\implies$  f x = g x
shows continuous_map X Y ( $\lambda x. if P x then f x else g x$ )
apply (rule continuous_map_cases)
using assms
apply (simp_all add: Collect_conj_eq closure_of_restrict [symmetric] fron-
tier_of_restrict [symmetric])
done

```

lemma continuous_map_cases_function:

```

assumes contp: continuous_map X Z p
and contf: continuous_map (subtopology X {x  $\in$  topspace X. p x  $\in$  Z closure_of
U}) Y f
and contg: continuous_map (subtopology X {x  $\in$  topspace X. p x  $\in$  Z closure_of
(topspace Z - U)}) Y g

```



```

    and fg:  $\bigwedge x. \llbracket x \in \text{topspace } X; p \ x \in Z \text{ frontier\_of } U \rrbracket \implies f \ x = g \ x$ 
  shows continuous_map X Y ( $\lambda x. \text{if } p \ x \in U \text{ then } f \ x \text{ else } g \ x$ )
proof (rule continuous_map_cases_alt)
  show continuous_map (subtopology X (X closure_of {x  $\in$  topspace X. p x  $\in$  U})) Y f
  proof (rule continuous_map_from_subtopology_mono)
    let ?T = {x  $\in$  topspace X. p x  $\in$  Z closure_of U}
    show continuous_map (subtopology X ?T) Y f
      by (simp add: contf)
    show X closure_of {x  $\in$  topspace X. p x  $\in$  U}  $\subseteq$  ?T
      by (rule continuous_map_closure_preimage_subset [OF contp])
  qed
  show continuous_map (subtopology X (X closure_of {x  $\in$  topspace X. p x  $\notin$  U})) Y g
  proof (rule continuous_map_from_subtopology_mono)
    let ?T = {x  $\in$  topspace X. p x  $\in$  Z closure_of (topspace Z - U)}
    show continuous_map (subtopology X ?T) Y g
      by (simp add: contg)
    have X closure_of {x  $\in$  topspace X. p x  $\notin$  U}  $\subseteq$  X closure_of {x  $\in$  topspace X. p x  $\in$  topspace Z - U}
      apply (rule closure_of_mono)
      using continuous_map_closedin contp by fastforce
    then show X closure_of {x  $\in$  topspace X. p x  $\notin$  U}  $\subseteq$  ?T
      by (rule order_trans [OF continuous_map_closure_preimage_subset [OF contp]])
  qed
next
  show f x = g x if x  $\in$  X frontier_of {x  $\in$  topspace X. p x  $\in$  U} for x
    using that continuous_map_frontier_frontier_preimage_subset [OF contp, of U] fg by blast
qed

```

2.3.14 Retractions

definition *retraction* :: ($'a::\text{topological\_space}$) set $\Rightarrow$ $'a$ set $\Rightarrow$ ($'a \Rightarrow 'a$) $\Rightarrow$ bool
where *retraction* S T r $\longleftrightarrow$
 $T \subseteq S \wedge \text{continuous\_on } S \ r \wedge r \restriction S \subseteq T \wedge (\forall x \in T. r \ x = x)$

definition *retract_of* (infixl retract'_of 50) **where**
 $T \text{ retract\_of } S \longleftrightarrow (\exists r. \text{retraction } S \ T \ r)$

lemma *retraction_idempotent*: *retraction* S T r $\implies x \in S \implies r \ (r \ x) = r \ x$
unfolding *retraction_def* **by** auto

Preservation of fixpoints under (more general notion of) retraction

lemma *invertible_fixpoint_property*:
fixes S :: $'a::\text{topological\_space}$ set
and T :: $'b::\text{topological\_space}$ set
assumes *contt*: *continuous_on* T i

```

    and  $i \text{ ' } T \subseteq S$ 
    and  $\text{contr: continuous\_on } S \text{ } r$ 
    and  $r \text{ ' } S \subseteq T$ 
    and  $\text{ri: } \bigwedge y. y \in T \implies r \text{ (} i \text{ } y) = y$ 
    and  $\text{FP: } \bigwedge f. \llbracket \text{continuous\_on } S \text{ } f; f \text{ ' } S \subseteq S \rrbracket \implies \exists x \in S. f \text{ } x = x$ 
    and  $\text{contg: continuous\_on } T \text{ } g$ 
    and  $g \text{ ' } T \subseteq T$ 
    obtains  $y$  where  $y \in T$  and  $g \text{ } y = y$ 
  proof -
    have  $\exists x \in S. (i \circ g \circ r) \text{ } x = x$ 
    proof (rule FP)
      show continuous_on  $S \text{ (} i \circ g \circ r)$ 
        by (meson contt contr assms(4) contg assms(8) continuous_on_compose
            continuous_on_subset)
      show  $(i \circ g \circ r) \text{ ' } S \subseteq S$ 
        using assms(2,4,8) by force
    qed
    then obtain  $x$  where  $x \in S \text{ (} i \circ g \circ r) \text{ } x = x \text{ ..}$ 
    then have *:  $g \text{ (} r \text{ } x) \in T$ 
      using assms(4,8) by auto
    have  $r \text{ ((} i \circ g \circ r) \text{ } x) = r \text{ } x$ 
      using  $x$  by auto
    then show ?thesis
      using * ri that by auto
  qed

lemma homeomorphic_fixpoint_property:
  fixes  $S :: \text{'a::topological\_space set}$ 
  and  $T :: \text{'b::topological\_space set}$ 
  assumes  $S \text{ homeomorphic } T$ 
  shows  $(\forall f. \text{continuous\_on } S \text{ } f \wedge f \text{ ' } S \subseteq S \longrightarrow (\exists x \in S. f \text{ } x = x)) \longleftrightarrow$ 
     $(\forall g. \text{continuous\_on } T \text{ } g \wedge g \text{ ' } T \subseteq T \longrightarrow (\exists y \in T. g \text{ } y = y))$ 
    (is ?lhs = ?rhs)
  proof -
    obtain  $r \text{ } i$  where  $r$ :
       $\forall x \in S. i \text{ (} r \text{ } x) = x \text{ } r \text{ ' } S = T \text{ continuous\_on } S \text{ } r$ 
       $\forall y \in T. r \text{ (} i \text{ } y) = y \text{ } i \text{ ' } T = S \text{ continuous\_on } T \text{ } i$ 
    using assms unfolding homeomorphic_def homeomorphism_def by blast
    show ?thesis
    proof
      assume ?lhs
      with  $r$  show ?rhs
        by (metis invertible_fixpoint_property[of  $T \text{ } i \text{ } S \text{ } r$ ] order_refl)
    next
      assume ?rhs
      with  $r$  show ?lhs
        by (metis invertible_fixpoint_property[of  $S \text{ } r \text{ } T \text{ } i$ ] order_refl)
    qed
  qed

```

```

lemma retract_fixpoint_property:
  fixes f :: 'a::topological_space  $\Rightarrow$  'b::topological_space
    and S :: 'a set
  assumes T retract_of S
    and FP:  $\bigwedge f. \llbracket \text{continuous\_on } S \text{ } f; f \text{ ' } S \subseteq S \rrbracket \Longrightarrow \exists x \in S. f \text{ } x = x$ 
    and contg: continuous_on T g
    and g ' T  $\subseteq$  T
  obtains y where y  $\in$  T and g y = y
proof -
  obtain h where retraction S T h
    using assms(1) unfolding retract_of_def ..
  then show ?thesis
    unfolding retraction_def
    using invertible_fixpoint_property[OF continuous_on_id _ _ _ FP]
    by (metis assms(4) contg image_ident that)
qed

```

```

lemma retraction:
  retraction S T r  $\longleftrightarrow$ 
    T  $\subseteq$  S  $\wedge$  continuous_on S r  $\wedge$  r ' S = T  $\wedge$  ( $\forall x \in T. r \text{ } x = x$ )
  by (force simp: retraction_def)

```

```

lemma retractionE: — yields properties normalized wrt. simp – less likely to loop
  assumes retraction S T r
  obtains T = r ' S r ' S  $\subseteq$  S continuous_on S r  $\wedge$  x. x  $\in$  S  $\Longrightarrow$  r (r x) = r x
proof (rule that)
  from retraction [of S T r] assms
  have T  $\subseteq$  S continuous_on S r r ' S = T and  $\forall x \in T. r \text{ } x = x$ 
    by simp_all
  then show T = r ' S r ' S  $\subseteq$  S continuous_on S r
    by simp_all
  from  $\langle \forall x \in T. r \text{ } x = x \rangle$  have r x = x if x  $\in$  T for x
    using that by simp
  with  $\langle r \text{ ' } S = T \rangle$  show r (r x) = r x if x  $\in$  S for x
    using that by auto
qed

```

```

lemma retract_ofE: — yields properties normalized wrt. simp – less likely to loop
  assumes T retract_of S
  obtains r where T = r ' S r ' S  $\subseteq$  S continuous_on S r  $\wedge$  x. x  $\in$  S  $\Longrightarrow$  r (r x) = r x
proof -
  from assms obtain r where retraction S T r
    by (auto simp add: retract_of_def)
  with that show thesis
    by (auto elim: retractionE)
qed

```

lemma *retract_of_imp_extensible*:
assumes S *retract_of* T **and** *continuous_on* S f **and** $f \text{ ' } S \subseteq U$
obtains g **where** *continuous_on* T g $g \text{ ' } T \subseteq U \wedge x. x \in S \implies g \ x = f \ x$
proof –
from $\langle S$ *retract_of* $T \rangle$ **obtain** r **where** *retraction* T S r
by (*auto simp add: retract_of_def*)
show *thesis*
by (*rule that* $[of \ f \circ \ r]$)
(use $\langle$ *continuous_on* S $f \rangle$ $\langle f \text{ ' } S \subseteq U \rangle$ $\langle$ *retraction* T S $r \rangle$ **in** $\langle$ *auto simp:*
continuous_on_compose2 *retraction* $\rangle$
qed

lemma *idempotent_imp_retraction*:
assumes *continuous_on* S f **and** $f \text{ ' } S \subseteq S$ **and** $\wedge x. x \in S \implies f(f \ x) = f \ x$
shows *retraction* S $(f \text{ ' } S)$ f
by (*simp add: assms retraction*)

lemma *retraction_subset*:
assumes *retraction* S T r **and** $T \subseteq s'$ **and** $s' \subseteq S$
shows *retraction* s' T r
unfolding *retraction_def*
by (*metis assms continuous_on_subset image_mono retraction*)

lemma *retract_of_subset*:
assumes T *retract_of* S **and** $T \subseteq s'$ **and** $s' \subseteq S$
shows T *retract_of* s'
by (*meson assms retract_of_def retraction_subset*)

lemma *retraction_refl* [*simp*]: *retraction* S S $(\lambda x. x)$
by (*simp add: retraction*)

lemma *retract_of_refl* [*iff*]: S *retract_of* S
unfolding *retract_of_def* *retraction_def*
using *continuous_on_id* **by** *blast*

lemma *retract_of_imp_subset*:
 S *retract_of* $T \implies S \subseteq T$
by (*simp add: retract_of_def retraction_def*)

lemma *retract_of_empty* [*simp*]:
 $(\{\}$ *retract_of* $S) \longleftrightarrow S = \{\}$ $(S$ *retract_of* $\{\}) \longleftrightarrow S = \{\}$
by (*auto simp: retract_of_def retraction_def*)

lemma *retract_of_singleton* [*iff*]: $(\{x\}$ *retract_of* $S) \longleftrightarrow x \in S$
unfolding *retract_of_def* *retraction_def* **by** *force*

lemma *retraction_comp*:
 $\llbracket$ *retraction* S T f ; *retraction* T U g $\rrbracket$
 $\implies$ *retraction* S U $(g \circ f)$

apply (auto simp: retraction_def intro: continuous_on_compose2)
by blast

lemma retract_of_trans [trans]:
 assumes S retract_of T and T retract_of U
 shows S retract_of U
using assms **by** (auto simp: retract_of_def intro: retraction_comp)

lemma closedin_retract:
 fixes $S :: 'a :: t2\_space$ set
 assumes S retract_of T
 shows closedin (top_of_set T) S
proof –
 obtain r **where** $r: S \subseteq T$ continuous_on T r $r^{-1} T \subseteq S \wedge x. x \in S \implies r\ x = x$
using assms **by** (auto simp: retract_of_def retraction_def)
 have $S = \{x \in T. x = r\ x\}$
using r **by** auto
 also have $\dots = T \cap ((\lambda x. (x, r\ x)) -' (\{y. \exists x. y = (x, x)\}))$
unfolding vimage_def mem_Times_iff fst_conv snd_conv
using r
by auto
 also have closedin (top_of_set T) $\dots$
by (rule continuous_closedin_preimage) (auto intro!: closed_diagonal continuous_on_Pair r)
finally show ?thesis .
qed

lemma closedin_self [simp]: closedin (top_of_set S) S
by simp

lemma retract_of_closed:
 fixes $S :: 'a :: t2\_space$ set
 shows $\llbracket \text{closed } T; S \text{ retract\_of } T \rrbracket \implies \text{closed } S$
by (metis closedin_retract closedin_closed_eq)

lemma retract_of_compact:
 $\llbracket \text{compact } T; S \text{ retract\_of } T \rrbracket \implies \text{compact } S$
by (metis compact_continuous_image retract_of_def retraction)

lemma retract_of_connected:
 $\llbracket \text{connected } T; S \text{ retract\_of } T \rrbracket \implies \text{connected } S$
by (metis Topological_Spaces.connected_continuous_image retract_of_def retraction)

lemma retraction_openin_vimage_iff:
 $\text{openin } (\text{top\_of\_set } S) (S \cap r^{-1} U) \longleftrightarrow \text{openin } (\text{top\_of\_set } T) U$
if retraction: retraction S T r **and** $U \subseteq T$
using retraction **apply** (rule retractionE)
apply (rule continuous_right_inverse_imp_quotient_map [where $g=r$])

```

using  $\langle U \subseteq T \rangle$  apply (auto elim: continuous_on_subset)
done

```

lemma *retract_of_Times*:

```

 $\llbracket S \text{ retract\_of } s'; T \text{ retract\_of } t' \rrbracket \implies (S \times T) \text{ retract\_of } (s' \times t')$ 
apply (simp add: retract_of_def retraction_def Sigma_mono, clarify)
apply (rename_tac f g)
apply (rule_tac x= $\lambda z. ((f \circ \text{fst}) z, (g \circ \text{snd}) z)$  in exI)
apply (rule conjI continuous_intros | erule continuous_on_subset | force)+
done

```

2.3.15 Retractions on a topological space

definition *retract_of_space* :: 'a set $\Rightarrow$ 'a topology $\Rightarrow$ bool (**infix** *retract'_of'_space* 50)

```

where S retract_of_space X
 $\equiv S \subseteq \text{topspace } X \wedge (\exists r. \text{continuous\_map } X (\text{subtopology } X S) r \wedge (\forall x \in S. r x = x))$ 

```

lemma *retract_of_space_retraction_maps*:

```

S retract_of_space X  $\longleftrightarrow S \subseteq \text{topspace } X \wedge (\exists r. \text{retraction\_maps } X (\text{subtopology } X S) r \text{ id})$ 

```

```

by (auto simp: retract_of_space_def retraction_maps_def)

```

lemma *retract_of_space_section_map*:

```

S retract_of_space X  $\longleftrightarrow S \subseteq \text{topspace } X \wedge \text{section\_map } (\text{subtopology } X S) X \text{ id}$ 

```

```

unfolding retract_of_space_def retraction_maps_def section_map_def
by (auto simp: continuous_map_from_subtopology)

```

lemma *retract_of_space_imp_subset*:

```

S retract_of_space X  $\implies S \subseteq \text{topspace } X$ 
by (simp add: retract_of_space_def)

```

lemma *retract_of_space_topspace*:

```

 $\text{topspace } X \text{ retract\_of\_space } X$ 
using retract_of_space_def by force

```

lemma *retract_of_space_empty* [simp]:

```

 $\{\}$  retract_of_space X  $\longleftrightarrow \text{topspace } X = \{\}$ 
by (auto simp: continuous_map_def retract_of_space_def)

```

lemma *retract_of_space_singleton* [simp]:

```

 $\{a\}$  retract_of_space X  $\longleftrightarrow a \in \text{topspace } X$ 

```

proof –

```

have continuous_map X (subtopology X {a}) ( $\lambda x. a$ )  $\wedge (\lambda x. a) a = a$  if  $a \in \text{topspace } X$ 

```

```

using that by simp

```

```

then show ?thesis

```

by (force simp: retract_of_space_def)
qed

lemma *retract_of_space_clopen*:
 assumes *openin* X S *closedin* X S $S = \{\}$ $\implies$ *topspace* $X = \{\}$
 shows S *retract_of_space* X
proof (cases $S = \{\}$)
 case *False*
 then obtain a where $a \in S$
 by *blast*
 show ?thesis
 unfolding *retract_of_space_def*
proof (intro *exI conjI*)
 show $S \subseteq \text{topspace } X$
 by (simp add: *assms closedin_subset*)
 have *continuous_map* X X ($\lambda x.$ if $x \in S$ then x else a)
proof (rule *continuous_map_cases*)
 show *continuous_map* (subtopology X (X *closure_of* $\{x. x \in S\}$)) X ($\lambda x.$ x)
 by (simp add: *continuous_map_from_subtopology*)
 show *continuous_map* (subtopology X (X *closure_of* $\{x. x \notin S\}$)) X ($\lambda x.$ a)
 using $\langle S \subseteq \text{topspace } X \rangle \langle a \in S \rangle$ by *force*
 show $x = a$ if $x \in X$ *frontier_of* $\{x. x \in S\}$ for x
 using *assms that clopenin_eq_frontier_of* by *fastforce*
 qed
 then show *continuous_map* X (subtopology X S) ($\lambda x.$ if $x \in S$ then x else a)
 using $\langle S \subseteq \text{topspace } X \rangle \langle a \in S \rangle$ by (auto simp: *continuous_map_in_subtopology*)
 qed *auto*
 qed (use *assms* in *auto*)

lemma *retract_of_space_disjoint_union*:
 assumes *openin* X S *openin* X T and ST : *disjnt* S T $S \cup T = \text{topspace } X$ and
 $S = \{\} \implies \text{topspace } X = \{\}$
 shows S *retract_of_space* X
proof (rule *retract_of_space_clopen*)
 have $S \cap T = \{\}$
 by (meson ST *disjnt_def*)
 then have $S = \text{topspace } X - T$
 using ST by *auto*
 then show *closedin* X S
 using $\langle \text{openin } X T \rangle$ by *blast*
 qed (auto simp: *assms*)

lemma *retraction_maps_section_image1*:
 assumes *retraction_maps* X Y r s
 shows s ' (*topspace* Y) *retract_of_space* X
 unfolding *retract_of_space_section_map*
proof
 show s ' *topspace* $Y \subseteq \text{topspace } X$
 using *assms continuous_map_image_subset_topspace retraction_maps_def*

by *blast*
 show *section_map* (*subtopology* *X* (*s* ‘ *topspace* *Y*)) *X id*
 unfolding *section_map_def*
 using *assms retraction_maps_to_retract_maps* by *blast*
 qed

lemma *retraction_maps_section_image2*:
 retraction_maps *X Y r s*
 $\implies$ *subtopology* *X* (*s* ‘ (*topspace* *Y*)) *homeomorphic_space Y*
 using *embedding_map_imp_homeomorphic_space homeomorphic_space_sym*
section_imp_embedding_map
 section_map_def by *blast*

2.3.16 Paths and path-connectedness

definition *pathin* :: 'a topology $\Rightarrow$ (real $\Rightarrow$ 'a) $\Rightarrow$ bool where
 pathin *X g* $\equiv$ *continuous_map* (*subtopology* *euclideanreal* {0..1}) *X g*

lemma *pathin_compose*:
 $\llbracket \text{pathin } X \text{ g; continuous\_map } X \text{ Y f} \rrbracket \implies \text{pathin } Y (f \circ g)$
 by (*simp add: continuous_map_compose pathin_def*)

lemma *pathin_subtopology*:
 pathin (*subtopology* *X S*) *g* $\longleftrightarrow$ *pathin* *X g* $\wedge$ ($\forall x \in \{0..1\}. g \ x \in S$)
 by (*auto simp: pathin_def continuous_map_in_subtopology*)

lemma *pathin_const*:
 pathin *X* ($\lambda x. a$) $\longleftrightarrow a \in \text{topspace } X$
 by (*simp add: pathin_def*)

lemma *path_start_in_topspace*: *pathin* *X g* $\implies g \ 0 \in \text{topspace } X$
 by (*force simp: pathin_def continuous_map*)

lemma *path_finish_in_topspace*: *pathin* *X g* $\implies g \ 1 \in \text{topspace } X$
 by (*force simp: pathin_def continuous_map*)

lemma *path_image_subset_topspace*: *pathin* *X g* $\implies g \text{ ‘ } (\{0..1\}) \subseteq \text{topspace } X$
 by (*force simp: pathin_def continuous_map*)

definition *path_connected_space* :: 'a topology $\Rightarrow$ bool
 where *path_connected_space* *X* $\equiv \forall x \in \text{topspace } X. \forall y \in \text{topspace } X. \exists g.$
 pathin *X g* $\wedge g \ 0 = x \wedge g \ 1 = y$

definition *path_connectedin* :: 'a topology $\Rightarrow$ 'a set $\Rightarrow$ bool
 where *path_connectedin* *X S* $\equiv S \subseteq \text{topspace } X \wedge \text{path\_connected\_space}(\text{subtopology } X \ S)$

lemma *path_connectedin_absolute* [*simp*]:
 path_connectedin (*subtopology* *X S*) *S* $\longleftrightarrow \text{path\_connectedin } X \ S$


```

by (simp add: path_connectedin_def subtopology_subtopology)

lemma path_connectedin_subset_topspace:
  path_connectedin X S  $\implies$  S  $\subseteq$  topspace X
by (simp add: path_connectedin_def)

lemma path_connectedin_subtopology:
  path_connectedin (subtopology X S) T  $\longleftrightarrow$  path_connectedin X T  $\wedge$  T  $\subseteq$  S
by (auto simp: path_connectedin_def subtopology_subtopology inf.absorb2)

lemma path_connectedin:
  path_connectedin X S  $\longleftrightarrow$ 
    S  $\subseteq$  topspace X  $\wedge$ 
    ( $\forall x \in S. \forall y \in S. \exists g. \text{pathin } X \ g \wedge g \text{ ` } \{0..1\} \subseteq S \wedge g \ 0 = x \wedge g \ 1 = y$ )
unfolding path_connectedin_def path_connected_space_def pathin_def continuous_map_in_subtopology
by (intro conj_cong refl ball_cong) (simp_all add: inf.absorb_iff2)

lemma path_connectedin_topspace:
  path_connectedin X (topspace X)  $\longleftrightarrow$  path_connected_space X
by (simp add: path_connectedin_def)

lemma path_connected_imp_connected_space:
  assumes path_connected_space X
  shows connected_space X
proof -
  have *:  $\exists S. \text{connectedin } X \ S \wedge g \ 0 \in S \wedge g \ 1 \in S$  if pathin X g for g
  proof (intro exI conjI)
    have continuous_map (subtopology euclideanreal {0..1}) X g
      using connectedin_absolute that by (simp add: pathin_def)
    then show connectedin X (g ` {0..1})
      by (rule connectedin_continuous_map_image) auto
  qed auto
  show ?thesis
    using assms
    by (auto intro: * simp add: path_connected_space_def connected_space_subconnected
      Ball_def)
qed

lemma path_connectedin_imp_connectedin:
  path_connectedin X S  $\implies$  connectedin X S
by (simp add: connectedin_def path_connected_imp_connected_space path_connectedin_def)

lemma path_connected_space_topspace_empty:
  topspace X = {}  $\implies$  path_connected_space X
by (simp add: path_connected_space_def)

lemma path_connectedin_empty [simp]: path_connectedin X {}
by (simp add: path_connectedin)

```

lemma *path_connectedin_singleton* [simp]: $\text{path\_connectedin } X \{a\} \longleftrightarrow a \in \text{topspace } X$

proof

show $\text{path\_connectedin } X \{a\} \implies a \in \text{topspace } X$

by (simp add: path_connectedin)

show $a \in \text{topspace } X \implies \text{path\_connectedin } X \{a\}$

unfolding path_connectedin

using pathin_const **by** fastforce

qed

lemma *path_connectedin_continuous_map_image*:

assumes $f: \text{continuous\_map } X \ Y$ **and** $S: \text{path\_connectedin } X \ S$

shows $\text{path\_connectedin } Y \ (f \ ' \ S)$

proof –

have $fX: f \ ' \ (\text{topspace } X) \subseteq \text{topspace } Y$

by (metis f continuous_map_image_subset_topospace)

show ?thesis

unfolding path_connectedin

proof (intro conjI ballI; clarify?)

fix x

assume $x \in S$

show $f \ x \in \text{topspace } Y$

by (meson S fX $\langle x \in S \rangle$ image_subset_iff path_connectedin_subset_topospace

set_mp)

next

fix $x \ y$

assume $x \in S$ **and** $y \in S$

then obtain g **where** $g: \text{pathin } X \ g \ ' \ \{0..1\} \subseteq S \ g \ 0 = x \ g \ 1 = y$

using S **by** (force simp: path_connectedin)

show $\exists g. \text{pathin } Y \ g \wedge g \ ' \ \{0..1\} \subseteq f \ ' \ S \wedge g \ 0 = f \ x \wedge g \ 1 = f \ y$

proof (intro exI conjI)

show $\text{pathin } Y \ (f \circ g)$

using $\langle \text{pathin } X \ g \rangle$ f pathin_compose **by** auto

qed (use g in auto)

qed

qed

lemma *path_connectedin_discrete_topology*:

$\text{path\_connectedin } (\text{discrete\_topology } U) \ S \longleftrightarrow S \subseteq U \wedge (\exists a. S \subseteq \{a\})$

apply safe

using path_connectedin_subset_topospace **apply** fastforce

apply (meson connectedin_discrete_topology path_connectedin_imp_connectedin)

using subset_singletonD **by** fastforce

lemma *path_connected_space_discrete_topology*:

$\text{path\_connected\_space } (\text{discrete\_topology } U) \longleftrightarrow (\exists a. U \subseteq \{a\})$

by (metis path_connectedin_discrete_topology path_connectedin_topospace path_connected_space_topospace subset_singletonD topspace_discrete_topology)

lemma *homeomorphic_path_connected_space_imp*:

$\llbracket \text{path\_connected\_space } X; X \text{ homeomorphic\_space } Y \rrbracket \implies \text{path\_connected\_space } Y$

unfolding *homeomorphic_space_def homeomorphic_maps_def*

by (*metis* (*no_types*, *opaque_lifting*) *continuous_map_closedin continuous_map_image_subset_topspace imageI order_class.order.antisym path_connectedin_continuous_map_image path_connectedin_topspace subsetI*)

lemma *homeomorphic_path_connected_space*:

$X \text{ homeomorphic\_space } Y \implies \text{path\_connected\_space } X \longleftrightarrow \text{path\_connected\_space } Y$

by (*meson* *homeomorphic_path_connected_space_imp homeomorphic_space_sym*)

lemma *homeomorphic_map_path_connectedness*:

assumes *homeomorphic_map* $X \ Y \ f \ U \subseteq \text{topspace } X$

shows $\text{path\_connectedin } Y \ (f \ ' \ U) \longleftrightarrow \text{path\_connectedin } X \ U$

unfolding *path_connectedin_def*

proof (*intro conj_cong homeomorphic_path_connected_space*)

show $(f \ ' \ U \subseteq \text{topspace } Y) = (U \subseteq \text{topspace } X)$

using *assms homeomorphic_imp_surjective_map* **by** *blast*

next

assume $U \subseteq \text{topspace } X$

show $\text{subtopology } Y \ (f \ ' \ U) \text{ homeomorphic\_space } \text{subtopology } X \ U$

using *assms unfolding homeomorphic_eq_everything_map*

by (*metis* (*no_types*, *opaque_lifting*) *assms homeomorphic_map_subtopologies homeomorphic_space homeomorphic_space_sym image_mono inf.absorb_iff2*)

qed

lemma *homeomorphic_map_path_connectedness_eq*:

$\text{homeomorphic\_map } X \ Y \ f \implies \text{path\_connectedin } X \ U \longleftrightarrow U \subseteq \text{topspace } X \wedge \text{path\_connectedin } Y \ (f \ ' \ U)$

by (*meson* *homeomorphic_map_path_connectedness path_connectedin_def*)

2.3.17 Connected components

definition *connected_component_of* :: $'a \text{ topology} \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$

where $\text{connected\_component\_of } X \ x \ y \equiv$

$\exists T. \text{connectedin } X \ T \wedge x \in T \wedge y \in T$

abbreviation *connected_component_of_set*

where $\text{connected\_component\_of\_set } X \ x \equiv \text{Collect } (\text{connected\_component\_of } X \ x)$

definition *connected_components_of* :: $'a \text{ topology} \Rightarrow ('a \text{ set}) \text{ set}$

where $\text{connected\_components\_of } X \equiv \text{connected\_component\_of\_set } X \ ' \ \text{topspace } X$

lemma *connected_component_in_topospace:*

connected_component_of X x y $\implies x \in \text{topspace } X \wedge y \in \text{topspace } X$

by (*meson connected_component_of_def connectedin_subset_topospace in_mono*)

lemma *connected_component_of_refl:*

connected_component_of X x x $\longleftrightarrow x \in \text{topspace } X$

by (*meson connected_component_in_topospace connected_component_of_def connectedinSing insertI1*)

lemma *connected_component_of_sym:*

connected_component_of X x y $\longleftrightarrow \text{connected\_component\_of } X y x$

by (*meson connected_component_of_def*)

lemma *connected_component_of_trans:*

[[connected_component_of X x y; connected_component_of X y z]]

$\implies \text{connected\_component\_of } X x z$

unfolding *connected_component_of_def*

using *connectedin_Un* **by** *blast*

lemma *connected_component_of_mono:*

[[connected_component_of (subtopology X S) x y; $S \subseteq T$]]

$\implies \text{connected\_component\_of } (\text{subtopology } X T) x y$

by (*metis connected_component_of_def connectedin_subtopology inf.absorb_iff2 subtopology_subtopology*)

lemma *connected_component_of_set:*

connected_component_of_set X x = {y. $\exists T. \text{connectedin } X T \wedge x \in T \wedge y \in T$ }

by (*meson connected_component_of_def*)

lemma *connected_component_of_subset_topospace:*

connected_component_of_set X x $\subseteq \text{topspace } X$

using *connected_component_in_topospace* **by** *force*

lemma *connected_component_of_eq_empty:*

connected_component_of_set X x = {} $\longleftrightarrow (x \notin \text{topspace } X)$

using *connected_component_in_topospace connected_component_of_refl* **by** *fastforce*

lemma *connected_space_iff_connected_component:*

connected_space X $\longleftrightarrow (\forall x \in \text{topspace } X. \forall y \in \text{topspace } X. \text{connected\_component\_of } X x y)$

by (*simp add: connected_component_of_def connected_space_subconnected*)

lemma *connected_space_imp_connected_component_of:*

[[connected_space X; a $\in \text{topspace } X$; b $\in \text{topspace } X$]]

$\implies \text{connected\_component\_of } X a b$

by (*simp add: connected_space_iff_connected_component*)

lemma *connected_space_connected_component_set*:
 $\text{connected\_space } X \longleftrightarrow (\forall x \in \text{topspace } X. \text{connected\_component\_of\_set } X \ x = \text{topspace } X)$
using *connected_component_of_subset_topspace connected_space_iff_connected_component*
by *fastforce*

lemma *connected_component_of_maximal*:
 $\llbracket \text{connectedin } X \ S; x \in S \rrbracket \implies S \subseteq \text{connected\_component\_of\_set } X \ x$
by (*meson Ball_Collect connected_component_of_def*)

lemma *connected_component_of_equiv*:
 $\text{connected\_component\_of } X \ x \ y \longleftrightarrow$
 $x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge \text{connected\_component\_of } X \ x = \text{connected\_component\_of } X \ y$
apply (*simp add: connected_component_in_topspace fun_eq_iff*)
by (*meson connected_component_of_refl connected_component_of_sym connected_component_of_trans*)

lemma *connected_component_of_disjoint*:
 $\text{disjnt } (\text{connected\_component\_of\_set } X \ x) (\text{connected\_component\_of\_set } X \ y)$
 $\longleftrightarrow \sim (\text{connected\_component\_of } X \ x \ y)$
using *connected_component_of_equiv unfolding disjnt_iff* **by** *force*

lemma *connected_component_of_eq*:
 $\text{connected\_component\_of } X \ x = \text{connected\_component\_of } X \ y \longleftrightarrow$
 $(x \notin \text{topspace } X) \wedge (y \notin \text{topspace } X) \vee$
 $x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge$
 $\text{connected\_component\_of } X \ x \ y$
by (*metis Collect_empty_eq_bot connected_component_of_eq_empty connected_component_of_equiv*)

lemma *connectedin_connected_component_of*:
 $\text{connectedin } X \ (\text{connected\_component\_of\_set } X \ x)$
proof –
have $\text{connected\_component\_of\_set } X \ x = \bigcup \{T. \text{connectedin } X \ T \wedge x \in T\}$
by (*auto simp: connected_component_of_def*)
then show *?thesis*
apply (*rule ssubst*)
by (*blast intro: connectedin_Union*)
qed

lemma *Union_connected_components_of*:
 $\bigcup (\text{connected\_components\_of } X) = \text{topspace } X$
unfolding *connected_components_of_def*
apply (*rule equalityI*)
apply (*simp add: SUP_least connected_component_of_subset_topspace*)
using *connected_component_of_refl* **by** *fastforce*

lemma *connected_components_of_maximal*:

$\llbracket C \in \text{connected\_components\_of } X; \text{connectedin } X \ S; \sim \text{disjnt } C \ S \rrbracket \implies S \subseteq C$
unfolding *connected_components_of_def disjnt_def*
apply *clarify*
by (*metis Int_emptyI connected_component_of_def connected_component_of_trans mem_Collect_eq*)

lemma *pairwise_disjoint_connected_components_of*:
 $\text{pairwise disjnt } (\text{connected\_components\_of } X)$
unfolding *connected_components_of_def pairwise_def*
apply *clarify*
by (*metis connected_component_of_disjoint connected_component_of_equiv*)

lemma *complement_connected_components_of_Union*:
 $C \in \text{connected\_components\_of } X$
 $\implies \text{topspace } X - C = \bigcup (\text{connected\_components\_of } X - \{C\})$
apply (*rule equalityI*)
using *Union_connected_components_of* **apply** *fastforce*
by (*metis Diff_cancel Diff_subset Union_connected_components_of cSup_singleton diff_Union_pairwise_disjoint equalityE insert_subsetI pairwise_disjoint_connected_components_of*)

lemma *nonempty_connected_components_of*:
 $C \in \text{connected\_components\_of } X \implies C \neq \{\}$
unfolding *connected_components_of_def*
by (*metis (no_types, lifting) connected_component_of_eq_empty imageE*)

lemma *connected_components_of_subset*:
 $C \in \text{connected\_components\_of } X \implies C \subseteq \text{topspace } X$
using *Union_connected_components_of* **by** *fastforce*

lemma *connectedin_connected_components_of*:
assumes $C \in \text{connected\_components\_of } X$
shows $\text{connectedin } X \ C$
proof –
have $C \in \text{connected\_component\_of\_set } X \text{ ‘ } \text{topspace } X$
using *assms connected_components_of_def* **by** *blast*
then show *?thesis*
using *connectedin_connected_component_of* **by** *fastforce*
qed

lemma *connected_component_in_connected_components_of*:
 $\text{connected\_component\_of\_set } X \ a \in \text{connected\_components\_of } X \longleftrightarrow a \in \text{topspace } X$
apply (*rule iffI*)
using *connected_component_of_eq_empty nonempty_connected_components_of*
apply *fastforce*
by (*simp add: connected_components_of_def*)

lemma *connected_space_iff_components_eq*:
 $\text{connected\_space } X \longleftrightarrow (\forall C \in \text{connected\_components\_of } X. \forall C' \in \text{con-}$

```

nected_components_of X. C = C')
  apply (rule iffI)
  apply (force simp: connected_components_of_def connected_space_connected_component_set
image_iff)
  by (metis connected_component_in_connected_components_of connected_component_of_refl
connected_space_iff_connected_component mem_Collect_eq)

```

```

lemma connected_components_of_eq_empty:
  connected_components_of X = {}  $\longleftrightarrow$  topspace X = {}
  by (simp add: connected_components_of_def)

```

```

lemma connected_components_of_empty_space:
  topspace X = {}  $\implies$  connected_components_of X = {}
  by (simp add: connected_components_of_eq_empty)

```

```

lemma connected_components_of_subset_singleton:
  connected_components_of X  $\subseteq$  {S}  $\longleftrightarrow$  connected_space X  $\wedge$  (topspace X =
{}  $\vee$  topspace X = S)
proof (cases topspace X = {})
  case True
  then show ?thesis
  by (simp add: connected_components_of_empty_space connected_space_topspace_empty)
next
  case False
  then show ?thesis
  by (metis (no_types, opaque_lifting) Union_connected_components_of ccpo_Sup_singleton
connected_components_of_eq_empty connected_space_iff_components_eq
insertI1 singletonD
subsetI subset_singleton_iff)
qed

```

```

lemma connected_space_iff_components_subset_singleton:
  connected_space X  $\longleftrightarrow$  ( $\exists a.$  connected_components_of X  $\subseteq$  {a})
  by (simp add: connected_components_of_subset_singleton)

```

```

lemma connected_components_of_eq_singleton:
  connected_components_of X = {S}
 $\longleftrightarrow$  connected_space X  $\wedge$  topspace X  $\neq$  {}  $\wedge$  S = topspace X
  by (metis ccpo_Sup_singleton connected_components_of_subset_singleton insert_not_empty
subset_singleton_iff)

```

```

lemma connected_components_of_connected_space:
  connected_space X  $\implies$  connected_components_of X = (if topspace X = {} then
{} else {topspace X})
  by (simp add: connected_components_of_eq_empty connected_components_of_eq_singleton)

```

```

lemma exists_connected_component_of_superset:
  assumes connectedin X S and ne: topspace X  $\neq$  {}
  shows  $\exists C. C \in$  connected_components_of X  $\wedge$  S  $\subseteq$  C

```

```

proof (cases S = {})
  case True
    then show ?thesis
      using ne_connected_components_of_def by blast
  next
    case False
    then show ?thesis
      by (meson all_not_in_conv assms(1) connected_component_in_connected_components_of
connected_component_of_maximal_connectedin_subset_topspace in_mono)
qed

```

```

lemma closedin_connected_components_of:
  assumes C ∈ connected_components_of X
  shows closedin X C
proof –
  obtain x where x ∈ topspace X and x: C = connected_component_of_set X x
    using assms by (auto simp: connected_components_of_def)
  have connected_component_of_set X x ⊆ topspace X
    by (simp add: connected_component_of_subset_topspace)
  moreover have X closure_of connected_component_of_set X x ⊆ connected_component_of_set
X x
  proof (rule connected_component_of_maximal)
    show connectedin X (X closure_of connected_component_of_set X x)
    by (simp add: connectedin_closure_of_connectedin_connected_component_of)
    show x ∈ X closure_of connected_component_of_set X x
    by (simp add: ⟨x ∈ topspace X⟩ closure_of connected_component_of_refl)
  qed
  ultimately
  show ?thesis
    using closure_of_subset_eq x by auto
qed

```

```

lemma closedin_connected_component_of:
  closedin X (connected_component_of_set X x)
  by (metis closedin_connected_components_of closedin_empty connected_component_in_connected_co
connected_component_of_eq_empty)

```

```

lemma connected_component_of_eq_overlap:
  connected_component_of_set X x = connected_component_of_set X y ⟷
    (x ∉ topspace X) ∧ (y ∉ topspace X) ∨
    ∼(connected_component_of_set X x ∩ connected_component_of_set X y =
{})
  using connected_component_of_equiv by fastforce

```

```

lemma connected_component_of_nonoverlap:
  connected_component_of_set X x ∩ connected_component_of_set X y = {}
  ⟷
    (x ∉ topspace X) ∨ (y ∉ topspace X) ∨
    ∼(connected_component_of_set X x = connected_component_of_set X y)

```


by (*metis* *connected_component_of_eq_empty* *connected_component_of_eq_overlap* *inf.idem*)

lemma *connected_component_of_overlap*:
 $\sim(\text{connected\_component\_of\_set } X \ x \cap \text{connected\_component\_of\_set } X \ y = \{\})$
 $\longleftrightarrow$
 $x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge$
 $\text{connected\_component\_of\_set } X \ x = \text{connected\_component\_of\_set } X \ y$
by (*meson* *connected_component_of_nonoverlap*)

end

2.4 Connected Components

theory *Connected*
imports
Abstract_Topology_2
begin

2.4.1 Connectedness

lemma *connected_local*:
 $\text{connected } S \longleftrightarrow$
 $\neg (\exists e1 \ e2.$
 $\text{openin } (\text{top\_of\_set } S) \ e1 \wedge$
 $\text{openin } (\text{top\_of\_set } S) \ e2 \wedge$
 $S \subseteq e1 \cup e2 \wedge$
 $e1 \cap e2 = \{\} \wedge$
 $e1 \neq \{\} \wedge$
 $e2 \neq \{\})$
unfolding *connected_def* *openin_open*
by *safe blast+*

lemma *exists_diff*:
fixes $P :: 'a \text{ set} \Rightarrow \text{bool}$
shows $(\exists S. P \ (- \ S)) \longleftrightarrow (\exists S. P \ S)$
(is $?lhs \longleftrightarrow ?rhs$ **)**
proof $-$
have $?rhs$ **if** $?lhs$
using *that* **by** *blast*
moreover **have** $P \ (- \ (- \ S))$ **if** $P \ S$ **for** S
proof $-$
have $S = - \ (- \ S)$ **by** *simp*
with *that* **show** $?thesis$ **by** *metis*
qed
ultimately **show** $?thesis$ **by** *metis*
qed

lemma *connected_clopen*: $\text{connected } S \longleftrightarrow$

```

(∀ T. openin (top_of_set S) T ∧
  closedin (top_of_set S) T ⟶ T = {} ∨ T = S) (is ?lhs ⟷ ?rhs)
proof –
  have ¬ connected S ⟷
    (∃ e1 e2. open e1 ∧ open (¬ e2) ∧ S ⊆ e1 ∪ (¬ e2) ∧ e1 ∩ (¬ e2) ∩ S = {}
  ∧ e1 ∩ S ≠ {} ∧ (¬ e2) ∩ S ≠ {})
  unfolding connected_def openin_open closedin_closed
  by (metis double_complement)
  then have th0: connected S ⟷
    ¬ (∃ e2 e1. closed e2 ∧ open e1 ∧ S ⊆ e1 ∪ (¬ e2) ∧ e1 ∩ (¬ e2) ∩ S = {}
  ∧ e1 ∩ S ≠ {} ∧ (¬ e2) ∩ S ≠ {})
  (is _ ⟷ ¬ (∃ e2 e1. ?P e2 e1))
  by (simp add: closed_def) metis
  have th1: ?rhs ⟷ ¬ (∃ t' t. closed t' ∧ t = S ∩ t' ∧ t ≠ {} ∧ t ≠ S ∧ (∃ t'. open t'
  ∧ t = S ∩ t'))
  (is _ ⟷ ¬ (∃ t' t. ?Q t' t))
  unfolding connected_def openin_open closedin_closed by auto
  have (∃ e1. ?P e2 e1) ⟷ (∃ t. ?Q e2 t) for e2
  proof –
    have ?P e2 e1 ⟷ (∃ t. closed e2 ∧ t = S ∩ e2 ∧ open e1 ∧ t = S ∩ e1 ∧ t ≠ {})
  ∧ t ≠ S) for e1
    by auto
    then show ?thesis
    by metis
  qed
  then have ∀ e2. (∃ e1. ?P e2 e1) ⟷ (∃ t. ?Q e2 t)
  by blast
  then show ?thesis
  by (simp add: th0 th1)
qed

```

2.4.2 Connected components, considered as a connectedness relation or a set

definition *connected_component* S x y ≡ ∃ T. connected T ∧ T ⊆ S ∧ x ∈ T ∧ y ∈ T

abbreviation *connected_component_set* S x ≡ Collect (connected_component S x)

lemma *connected_componentI*:

connected T ⟹ T ⊆ S ⟹ x ∈ T ⟹ y ∈ T ⟹ connected_component S x y
by (auto simp: connected_component_def)

lemma *connected_component_in*: connected_component S x y ⟹ x ∈ S ∧ y ∈ S

by (auto simp: connected_component_def)

lemma *connected_component_refl*: x ∈ S ⟹ connected_component S x x

```

by (auto simp: connected_component_def) (use connected_sing in blast)

lemma connected_component_refl_eq [simp]: connected_component S x x  $\longleftrightarrow$  x
 $\in$  S
by (auto simp: connected_component_refl) (auto simp: connected_component_def)

lemma connected_component_sym: connected_component S x y  $\implies$  connected_component
S y x
by (auto simp: connected_component_def)

lemma connected_component_trans:
  connected_component S x y  $\implies$  connected_component S y z  $\implies$  connected_component
S x z
unfolding connected_component_def
by (metis Int_iff Un_iff Un_subset_iff equals0D connected_Un)

lemma connected_component_of_subset:
  connected_component S x y  $\implies$  S  $\subseteq$  T  $\implies$  connected_component T x y
by (auto simp: connected_component_def)

lemma connected_component_Union: connected_component_set S x =  $\bigcup \{T. \text{connected } T \wedge x \in T \wedge T \subseteq S\}$ 
by (auto simp: connected_component_def)

lemma connected_connected_component [iff]: connected (connected_component_set
S x)
by (auto simp: connected_component_Union intro: connected_Union)

lemma connected_iff_eq_connected_component_set:
  connected S  $\longleftrightarrow$  ( $\forall x \in S. \text{connected\_component\_set } S x = S$ )
proof (cases S = {})
case True
then show ?thesis by simp
next
case False
then obtain x where x  $\in$  S by auto
show ?thesis
proof
assume connected S
then show  $\forall x \in S. \text{connected\_component\_set } S x = S$ 
by (force simp: connected_component_def)
next
assume  $\forall x \in S. \text{connected\_component\_set } S x = S$ 
then show connected S
by (metis  $\langle x \in S \rangle$  connected_connected_component)
qed
qed

```

lemma connected_component_subset: connected_component_set S x $\subseteq$ S

using *connected_component_in* **by** *blast*

lemma *connected_component_eq_self*: *connected* $S \implies x \in S \implies \text{connected\_component\_set } S \ x = S$

by (*simp add: connected_iff_eq_connected_component_set*)

lemma *connected_iff_connected_component*:

connected $S \iff (\forall x \in S. \forall y \in S. \text{connected\_component } S \ x \ y)$

using *connected_component_in* **by** (*auto simp: connected_iff_eq_connected_component_set*)

lemma *connected_component_maximal*:

$x \in T \implies \text{connected } T \implies T \subseteq S \implies T \subseteq (\text{connected\_component\_set } S \ x)$

using *connected_component_eq_self* *connected_component_of_subset* **by** *blast*

lemma *connected_component_mono*:

$S \subseteq T \implies \text{connected\_component\_set } S \ x \subseteq \text{connected\_component\_set } T \ x$

by (*simp add: Collect_mono connected_component_of_subset*)

lemma *connected_component_eq_empty* [*simp*]: *connected_component_set* $S \ x = \{\}$ $\iff x \notin S$

using *connected_component_refl* **by** (*fastforce simp: connected_component_in*)

lemma *connected_component_set_empty* [*simp*]: *connected_component_set* $\{\} \ x = \{\}$

using *connected_component_eq_empty* **by** *blast*

lemma *connected_component_eq*:

$y \in \text{connected\_component\_set } S \ x \implies (\text{connected\_component\_set } S \ y = \text{connected\_component\_set } S \ x)$

by (*metis (no_types, lifting)*)

Collect_cong *connected_component_sym* *connected_component_trans* *mem_Collect_eq*)

lemma *closed_connected_component*:

assumes S : *closed* S

shows *closed* (*connected_component_set* $S \ x$)

proof (*cases* $x \in S$)

case *False*

then show *?thesis*

by (*metis connected_component_eq_empty closed_empty*)

next

case *True*

show *?thesis*

unfolding *closure_eq* [*symmetric*]

proof

show *closure* (*connected_component_set* $S \ x$) $\subseteq$ *connected_component_set* S

x

apply (*rule connected_component_maximal*)

apply (*simp add: closure_def True*)

apply (*simp add: connected_imp_connected_closure*)

```

    apply (simp add: S closure_minimal connected_component_subset)
  done
next
  show connected_component_set S x  $\subseteq$  closure (connected_component_set S x)
    by (simp add: closure_subset)
qed
qed

lemma connected_component_disjoint:
  connected_component_set S a  $\cap$  connected_component_set S b = {}  $\longleftrightarrow$ 
  a  $\notin$  connected_component_set S b
  apply (auto simp: connected_component_eq)
  using connected_component_eq connected_component_sym
  apply blast
  done

lemma connected_component_nonoverlap:
  connected_component_set S a  $\cap$  connected_component_set S b = {}  $\longleftrightarrow$ 
  a  $\notin$  S  $\vee$  b  $\notin$  S  $\vee$  connected_component_set S a  $\neq$  connected_component_set
  S b
  apply (auto simp: connected_component_in)
  using connected_component_refl_eq
  apply blast
  apply (metis connected_component_eq mem_Collect_eq)
  apply (metis connected_component_eq mem_Collect_eq)
  done

lemma connected_component_overlap:
  connected_component_set S a  $\cap$  connected_component_set S b  $\neq$  {}  $\longleftrightarrow$ 
  a  $\in$  S  $\wedge$  b  $\in$  S  $\wedge$  connected_component_set S a = connected_component_set
  S b
  by (auto simp: connected_component_nonoverlap)

lemma connected_component_sym_eq: connected_component S x y  $\longleftrightarrow$  connected_component
  S y x
  using connected_component_sym by blast

lemma connected_component_eq_eq:
  connected_component_set S x = connected_component_set S y  $\longleftrightarrow$ 
  x  $\notin$  S  $\wedge$  y  $\notin$  S  $\vee$  x  $\in$  S  $\wedge$  y  $\in$  S  $\wedge$  connected_component S x y
  apply (cases y  $\in$  S, simp)
  apply (metis connected_component_eq connected_component_eq_empty con-
  nected_component_refl_eq mem_Collect_eq)
  apply (cases x  $\in$  S, simp)
  apply (metis connected_component_eq_empty)
  using connected_component_eq_empty
  apply blast
  done

```

lemma *connected_iff_connected_component_eq*:
 $\text{connected } S \iff (\forall x \in S. \forall y \in S. \text{connected\_component\_set } S \ x = \text{connected\_component\_set } S \ y)$
by (*simp add: connected_component_eq_eq connected_iff_connected_component*)

lemma *connected_component_idemp*:
 $\text{connected\_component\_set } (\text{connected\_component\_set } S \ x) \ x = \text{connected\_component\_set } S \ x$
apply (*rule subset_antisym*)
apply (*simp add: connected_component_subset*)
apply (*metis connected_component_eq_empty connected_component_maximal connected_component_refl_eq connected_connected_component mem_Collect_eq set_eq_subset*)
done

lemma *connected_component_unique*:
 $\llbracket x \in c; c \subseteq S; \text{connected } c; \bigwedge c'. \llbracket x \in c'; c' \subseteq S; \text{connected } c' \rrbracket \implies c' \subseteq c \rrbracket \implies \text{connected\_component\_set } S \ x = c$
apply (*rule subset_antisym*)
apply (*meson connected_component_maximal connected_component_subset connected_connected_component contra_subsetD*)
by (*simp add: connected_component_maximal*)

lemma *joinable_connected_component_eq*:
 $\llbracket \text{connected } T; T \subseteq S; \text{connected\_component\_set } S \ x \cap T \neq \{\}; \text{connected\_component\_set } S \ y \cap T \neq \{\} \rrbracket \implies \text{connected\_component\_set } S \ x = \text{connected\_component\_set } S \ y$
apply (*simp add: ex_in_conv [symmetric]*)
apply (*rule connected_component_eq*)
by (*metis (no_types, opaque_lifting) connected_component_eq_eq connected_component_in connected_component_maximal subsetD mem_Collect_eq*)

lemma *Union_connected_component*: $\bigcup (\text{connected\_component\_set } S \text{ ' } S) = S$
apply (*rule subset_antisym*)
apply (*simp add: SUP_least connected_component_subset*)
using *connected_component_refl_eq*
by *force*

lemma *complement_connected_component_unions*:
 $S - \text{connected\_component\_set } S \ x = \bigcup (\text{connected\_component\_set } S \text{ ' } S - \{\text{connected\_component\_set } S \ x\})$
apply (*subst Union_connected_component [symmetric], auto*)
apply (*metis connected_component_eq_eq connected_component_in*)
by (*metis connected_component_eq mem_Collect_eq*)

```

lemma connected_component_intermediate_subset:
   $\llbracket \text{connected\_component\_set } U \ a \subseteq T; T \subseteq U \rrbracket$ 
   $\implies \text{connected\_component\_set } T \ a = \text{connected\_component\_set } U \ a$ 
  apply (case_tac a  $\in U$ )
  apply (simp add: connected_component_maximal connected_component_mono
subset_antisym)
  using connected_component_eq_empty by blast

```

2.4.3 The set of connected components of a set

```

definition components:: 'a::topological_space set  $\Rightarrow$  'a set set
  where components  $S \equiv \text{connected\_component\_set } S \ 'S$ 

```

```

lemma components_iff:  $S \in \text{components } U \longleftrightarrow (\exists x. x \in U \wedge S = \text{connected\_component\_set } U \ x)$ 
  by (auto simp: components_def)

```

```

lemma componentsI:  $x \in U \implies \text{connected\_component\_set } U \ x \in \text{components } U$ 
  by (auto simp: components_def)

```

```

lemma componentsE:
  assumes  $S \in \text{components } U$ 
  obtains  $x$  where  $x \in U \wedge S = \text{connected\_component\_set } U \ x$ 
  using assms by (auto simp: components_def)

```

```

lemma Union_components [simp]:  $\bigcup (\text{components } u) = u$ 
  apply (rule subset_antisym)
  using Union_connected_component components_def apply fastforce
  apply (metis Union_connected_component components_def set_eq_subset)
  done

```

```

lemma pairwise_disjoint_components:  $\text{pairwise } (\lambda X Y. X \cap Y = \{\}) (\text{components } u)$ 
  apply (simp add: pairwise_def)
  apply (auto simp: components_iff)
  apply (metis connected_component_eq_eq connected_component_in)
  done

```

```

lemma in_components_nonempty:  $c \in \text{components } s \implies c \neq \{\}$ 
  by (metis components_iff connected_component_eq_empty)

```

```

lemma in_components_subset:  $c \in \text{components } s \implies c \subseteq s$ 
  using Union_components by blast

```

```

lemma in_components_connected:  $c \in \text{components } s \implies \text{connected } c$ 
  by (metis components_iff connected_connected_component)

```

```

lemma in_components_maximal:
   $c \in \text{components } s \longleftrightarrow$ 

```

```

       $c \neq \{\} \wedge c \subseteq s \wedge \text{connected } c \wedge (\forall d. d \neq \{\} \wedge c \subseteq d \wedge d \subseteq s \wedge \text{connected } d$ 
 $\longrightarrow d = c)$ 
    apply (rule iffI)
    apply (simp add: in_components_nonempty in_components_connected)
    apply (metis (full_types) components_iff connected_component_eq_self con-
      nected_component_intermediate_subset connected_component_refl in_components_subset
      mem_Collect_eq rev_subsetD)
    apply (metis bot.extremum_uniqueI components_iff connected_component_eq_empty
      connected_component_maximal connected_component_subset connected_connected_component
      subset_emptyI)
    done

lemma joinable_components_eq:
   $\text{connected } t \wedge t \subseteq s \wedge c1 \in \text{components } s \wedge c2 \in \text{components } s \wedge c1 \cap t \neq \{\}$ 
 $\wedge c2 \cap t \neq \{\} \implies c1 = c2$ 
  by (metis (full_types) components_iff joinable_connected_component_eq)

lemma closed_components:  $\llbracket \text{closed } s; c \in \text{components } s \rrbracket \implies \text{closed } c$ 
  by (metis closed_connected_component components_iff)

lemma components_nonoverlap:
   $\llbracket c \in \text{components } s; c' \in \text{components } s \rrbracket \implies (c \cap c' = \{\}) \longleftrightarrow (c \neq c')$ 
  apply (auto simp: in_components_nonempty components_iff)
  using connected_component_refl apply blast
  apply (metis connected_component_eq_eq connected_component_in)
  by (metis connected_component_eq mem_Collect_eq)

lemma components_eq:  $\llbracket c \in \text{components } s; c' \in \text{components } s \rrbracket \implies (c = c' \longleftrightarrow$ 
 $c \cap c' \neq \{\})$ 
  by (metis components_nonoverlap)

lemma components_eq_empty [simp]:  $\text{components } s = \{\} \longleftrightarrow s = \{\}$ 
  by (simp add: components_def)

lemma components_empty [simp]:  $\text{components } \{\} = \{\}$ 
  by simp

lemma connected_eq_connected_components_eq:  $\text{connected } s \longleftrightarrow (\forall c \in \text{components } s. \forall c' \in \text{components } s. c = c')$ 
  by (metis (no_types, opaque_lifting) components_iff connected_component_eq_eq
    connected_iff_connected_component)

lemma components_eq_sing_iff:  $\text{components } s = \{s\} \longleftrightarrow \text{connected } s \wedge s \neq \{\}$ 
  apply (rule iffI)
  using in_components_connected apply fastforce
  apply safe
  using Union_components apply fastforce
  apply (metis components_iff connected_component_eq_self)
  using in_components_maximal

```


apply *auto*
done

lemma *components_eq_sing_exists*: $(\exists a. \text{components } s = \{a\}) \longleftrightarrow \text{connected } s \wedge s \neq \{\}$
apply (*rule iffI*)
using *connected_eq_connected_components_eq* **apply** *fastforce*
apply (*metis components_eq_sing_iff*)
done

lemma *connected_eq_components_subset_sing*: $\text{connected } s \longleftrightarrow \text{components } s \subseteq \{s\}$
by (*metis Union_components components_empty components_eq_sing_iff connected_empty insert_subset order_refl subset_singletonD*)

lemma *connected_eq_components_subset_sing_exists*: $\text{connected } s \longleftrightarrow (\exists a. \text{components } s \subseteq \{a\})$
by (*metis components_eq_sing_exists connected_eq_components_subset_sing empty_iff subset_iff subset_singletonD*)

lemma *in_components_self*: $s \in \text{components } s \longleftrightarrow \text{connected } s \wedge s \neq \{\}$
by (*metis components_empty components_eq_sing_iff empty_iff in_components_connected insertI1*)

lemma *components_maximal*: $\llbracket c \in \text{components } s; \text{connected } t; t \subseteq s; c \cap t \neq \{\} \rrbracket \implies t \subseteq c$
apply (*simp add: components_def ex_in_conv [symmetric], clarify*)
by (*meson connected_component_def connected_component_trans*)

lemma *exists_component_superset*: $\llbracket t \subseteq s; s \neq \{\}; \text{connected } t \rrbracket \implies \exists c. c \in \text{components } s \wedge t \subseteq c$
apply (*cases t = \{\}, force*)
apply (*metis components_def ex_in_conv connected_component_maximal contra_subsetD image_eqI*)
done

lemma *components_intermediate_subset*: $\llbracket s \in \text{components } u; s \subseteq t; t \subseteq u \rrbracket \implies s \in \text{components } t$
apply (*auto simp: components_iff*)
apply (*metis connected_component_eq_empty connected_component_intermediate_subset*)
done

lemma *in_components_unions_complement*: $c \in \text{components } s \implies s - c = \bigcup (\text{components } s - \{c\})$
by (*metis complement_connected_component_unions components_def components_iff*)

lemma *connected_intermediate_closure*:
assumes *cs*: $\text{connected } s$ **and** *st*: $s \subseteq t$ **and** *ts*: $t \subseteq \text{closure } s$

```

  shows connected t
proof (rule connectedI)
  fix A B
  assume A: open A and B: open B and Alap: A ∩ t ≠ {} and Blap: B ∩ t ≠ {}
    and disj: A ∩ B ∩ t = {} and cover: t ⊆ A ∪ B
  have disjs: A ∩ B ∩ s = {}
    using disj st by auto
  have A ∩ closure s ≠ {}
    using Alap Int_absorb1 ts by blast
  then have Alaps: A ∩ s ≠ {}
    by (simp add: A open_Int_closure_eq_empty)
  have B ∩ closure s ≠ {}
    using Blap Int_absorb1 ts by blast
  then have Blaps: B ∩ s ≠ {}
    by (simp add: B open_Int_closure_eq_empty)
  then show False
    using cs [unfolded connected_def] A B disjs Alaps Blaps cover st
    by blast
qed

```

```

lemma closedin_connected_component: closedin (top_of_set s) (connected_component_set s x)
proof (cases connected_component_set s x = {})
  case True
  then show ?thesis
    by (metis closedin_empty)
next
  case False
  then obtain y where y: connected_component s x y
    by blast
  have *: connected_component_set s x ⊆ s ∩ closure (connected_component_set s x)
    by (auto simp: closure_def connected_component_in)
  have connected_component s x y ⇒ s ∩ closure (connected_component_set s x) ⊆ connected_component_set s x
    apply (rule connected_component_maximal, simp)
    using closure_subset connected_component_in apply fastforce
    using * connected_intermediate_closure apply blast+
    done
  with y * show ?thesis
    by (auto simp: closedin_closed)
qed

```

```

lemma closedin_component:
  C ∈ components s ⇒ closedin (top_of_set s) C
  using closedin_connected_component componentsE by blast

```

2.4.4 Proving a function is constant on a connected set by proving that a level set is open

```

lemma continuous_levelset_openin_cases:
  fixes f ::  $\_ \Rightarrow 'b::t1\_space$ 
  shows connected s  $\implies$  continuous_on s f  $\implies$ 
    openin (top_of_set s) {x  $\in$  s. f x = a}
     $\implies$  ( $\forall x \in s. f x \neq a$ )  $\vee$  ( $\forall x \in s. f x = a$ )
  unfolding connected_clopen
  using continuous_closedin_preimage_constant by auto

```

```

lemma continuous_levelset_openin:
  fixes f ::  $\_ \Rightarrow 'b::t1\_space$ 
  shows connected s  $\implies$  continuous_on s f  $\implies$ 
    openin (top_of_set s) {x  $\in$  s. f x = a}  $\implies$ 
    ( $\exists x \in s. f x = a$ )  $\implies$  ( $\forall x \in s. f x = a$ )
  using continuous_levelset_openin_cases[of s f]
  by meson

```

```

lemma continuous_levelset_open:
  fixes f ::  $\_ \Rightarrow 'b::t1\_space$ 
  assumes connected s
    and continuous_on s f
    and open {x  $\in$  s. f x = a}
    and  $\exists x \in s. f x = a$ 
  shows  $\forall x \in s. f x = a$ 
  using continuous_levelset_openin[OF assms(1,2), of a, unfolded openin_open]
  using assms (3,4)
  by fast

```

2.4.5 Preservation of Connectedness

```

lemma homeomorphic_connectedness:
  assumes s homeomorphic t
  shows connected s  $\longleftrightarrow$  connected t
using assms unfolding homeomorphic_def homeomorphism_def by (metis connected_continuous_image)

```

```

lemma connected_monotone_quotient_preimage:
  assumes connected T
    and conf: continuous_on S f and fim: f ' $S = T$ 
    and opT:  $\bigwedge U. U \subseteq T$ 
     $\implies$  openin (top_of_set S) (S  $\cap$  f -' U)  $\longleftrightarrow$ 
    openin (top_of_set T) U
    and connT:  $\bigwedge y. y \in T \implies$  connected (S  $\cap$  f -' {y})
  shows connected S
proof (rule connectedI)
  fix U V
  assume open U and open V and U  $\cap$  S  $\neq \{\}$  and V  $\cap$  S  $\neq \{\}$ 
    and U  $\cap$  V  $\cap$  S =  $\{\}$  and S  $\subseteq$  U  $\cup$  V

```

```

moreover
have disjoint:  $f^{-1}(S \cap U) \cap f^{-1}(S \cap V) = \{\}$ 
proof -
  have False if  $y \in f^{-1}(S \cap U) \cap f^{-1}(S \cap V)$  for  $y$ 
  proof -
    have  $y \in T$ 
    using fim that by blast
    show ?thesis
    using connectedD [OF connT [OF ⟨y ∈ T⟩] ⟨open U⟩ ⟨open V⟩]
       $\langle S \subseteq U \cup V \rangle \langle U \cap V \cap S = \{\} \rangle$  that by fastforce
  qed
then show ?thesis by blast
qed
ultimately have UU:  $(S \cap f^{-1}f^{-1}(S \cap U)) = S \cap U$  and VV:  $(S \cap f^{-1}f^{-1}(S \cap V)) = S \cap V$ 
by auto
have opeU:  $\text{openin } (\text{top\_of\_set } T) (f^{-1}(S \cap U))$ 
by (metis UU ⟨open U⟩ fim image_Int_subset le_inf_iff opT openin_open_Int)
have opeV:  $\text{openin } (\text{top\_of\_set } T) (f^{-1}(S \cap V))$ 
by (metis opT fim VV ⟨open V⟩ openin_open_Int image_Int_subset inf.bounded_iff)
have  $T \subseteq f^{-1}(S \cap U) \cup f^{-1}(S \cap V)$ 
using  $\langle S \subseteq U \cup V \rangle$  fim by auto
then show False
using  $\langle \text{connected } T \rangle$  disjoint opeU opeV ⟨U ∩ S ≠ {}⟩ ⟨V ∩ S ≠ {}⟩
by (auto simp: connected_openin)
qed

lemma connected_open_monotone_preimage:
assumes contf: continuous_on  $S$   $f$  and fim:  $f^{-1}S = T$ 
and ST:  $\bigwedge C. \text{openin } (\text{top\_of\_set } S) C \implies \text{openin } (\text{top\_of\_set } T) (f^{-1}C)$ 
and connT:  $\bigwedge y. y \in T \implies \text{connected } (S \cap f^{-1}\{y\})$ 
and connected C  $C \subseteq T$ 
shows connected  $(S \cap f^{-1}C)$ 
proof -
have contf': continuous_on  $(S \cap f^{-1}C)$   $f$ 
by (meson contf continuous_on_subset inf_le1)
have eqC:  $f^{-1}(S \cap f^{-1}C) = C$ 
using  $\langle C \subseteq T \rangle$  fim by blast
show ?thesis
proof (rule connected_monotone_quotient_preimage [OF ⟨connected C⟩ contf' eqC])
show connected  $(S \cap f^{-1}C \cap f^{-1}\{y\})$  if  $y \in C$  for  $y$ 
proof -
have  $S \cap f^{-1}C \cap f^{-1}\{y\} = S \cap f^{-1}\{y\}$ 
using that by blast
moreover have connected  $(S \cap f^{-1}\{y\})$ 
using  $\langle C \subseteq T \rangle$  connT that by blast
ultimately show ?thesis
by metis

```

```

qed
have  $\bigwedge U. \text{openin } (\text{top\_of\_set } (S \cap f^{-1} C)) U$ 
   $\implies \text{openin } (\text{top\_of\_set } C) (f^{-1} U)$ 
  using open_map_restrict [OF ST  $\langle C \subseteq T \rangle$ ] by metis
then show  $\bigwedge D. D \subseteq C$ 
   $\implies \text{openin } (\text{top\_of\_set } (S \cap f^{-1} C)) (S \cap f^{-1} C \cap f^{-1} D) =$ 
     $\text{openin } (\text{top\_of\_set } C) D$ 
  using open_map_imp_quotient_map [of  $(S \cap f^{-1} C) f$ ] contf' by (simp
add: eqC)
qed
qed

```

```

lemma connected_closed_monotone_preimage:
  assumes contf': continuous_on  $S f$  and fm:  $f^{-1} S = T$ 
  and ST:  $\bigwedge C. \text{closedin } (\text{top\_of\_set } S) C \implies \text{closedin } (\text{top\_of\_set } T) (f^{-1} C)$ 
  and connT:  $\bigwedge y. y \in T \implies \text{connected } (S \cap f^{-1} \{y\})$ 
  and connected C  $C \subseteq T$ 
  shows connected  $(S \cap f^{-1} C)$ 
proof -
  have contf': continuous_on  $(S \cap f^{-1} C) f$ 
    by (meson contf' continuous_on_subset inf_le1)
  have eqC:  $f^{-1} (S \cap f^{-1} C) = C$ 
    using  $\langle C \subseteq T \rangle$  fm by blast
  show ?thesis
  proof (rule connected_monotone_quotient_preimage [OF  $\langle \text{connected } C \rangle$  contf'
eqC])
    show connected  $(S \cap f^{-1} C \cap f^{-1} \{y\})$  if  $y \in C$  for  $y$ 
    proof -
      have  $S \cap f^{-1} C \cap f^{-1} \{y\} = S \cap f^{-1} \{y\}$ 
        using that by blast
      moreover have connected  $(S \cap f^{-1} \{y\})$ 
        using  $\langle C \subseteq T \rangle$  connT that by blast
      ultimately show ?thesis
        by metis
    qed
  have  $\bigwedge U. \text{closedin } (\text{top\_of\_set } (S \cap f^{-1} C)) U$ 
     $\implies \text{closedin } (\text{top\_of\_set } C) (f^{-1} U)$ 
    using closed_map_restrict [OF ST  $\langle C \subseteq T \rangle$ ] by metis
  then show  $\bigwedge D. D \subseteq C$ 
     $\implies \text{openin } (\text{top\_of\_set } (S \cap f^{-1} C)) (S \cap f^{-1} C \cap f^{-1} D) =$ 
       $\text{openin } (\text{top\_of\_set } C) D$ 
    using closed_map_imp_quotient_map [of  $(S \cap f^{-1} C) f$ ] contf' by (simp
add: eqC)
  qed
qed

```

2.4.6 Lemmas about components

See Newman IV, 3.3 and 3.4.

```

lemma connected_Un_clopen_in_complement:
  fixes  $S\ U :: 'a::metric\_space\ set$ 
  assumes connected S connected U  $S \subseteq U$ 
    and opeT: openin (top_of_set (U - S)) T
    and cloT: closedin (top_of_set (U - S)) T
  shows connected (S ∪ T)
proof -
  have *:  $\llbracket \bigwedge x\ y. P\ x\ y \longleftrightarrow P\ y\ x; \bigwedge x\ y. P\ x\ y \implies S \subseteq x \vee S \subseteq y; \bigwedge x\ y. \llbracket P\ x\ y; S \subseteq x \rrbracket \implies False \rrbracket \implies \neg(\exists x\ y. (P\ x\ y))$  for  $P$ 
    by metis
  show ?thesis
    unfolding connected_closedin_eq
  proof (rule *)
    fix  $H1\ H2$ 
    assume  $H$ : closedin (top_of_set (S ∪ T)) H1  $\wedge$ 
      closedin (top_of_set (S ∪ T)) H2  $\wedge$ 
       $H1 \cup H2 = S \cup T \wedge H1 \cap H2 = \{\} \wedge H1 \neq \{\} \wedge H2 \neq \{\}$ 
    then have clo: closedin (top_of_set S) (S ∩ H1)
      closedin (top_of_set S) (S ∩ H2)
      by (metis Un_upper1 closedin_closed_subset inf_commute)
    have Seq:  $S \cap (H1 \cup H2) = S$ 
      by (simp add: H)
    have  $S \cap ((S \cup T) \cap H1) \cup S \cap ((S \cup T) \cap H2) = S$ 
      using Seq by auto
    moreover have  $H1 \cap (S \cap ((S \cup T) \cap H2)) = \{\}$ 
      using  $H$  by blast
    ultimately have  $S \cap H1 = \{\} \vee S \cap H2 = \{\}$ 
      by (metis (no_types) H Int_assoc  $\langle S \cap (H1 \cup H2) = S \rangle$   $\langle \text{connected } S \rangle$ 
        clo Seq connected_closedin_inf_bot_right inf_le1)
    then show  $S \subseteq H1 \vee S \subseteq H2$ 
      using  $H$   $\langle \text{connected } S \rangle$  unfolding connected_closedin by blast
  next
    fix  $H1\ H2$ 
    assume  $H$ : closedin (top_of_set (S ∪ T)) H1  $\wedge$ 
      closedin (top_of_set (S ∪ T)) H2  $\wedge$ 
       $H1 \cup H2 = S \cup T \wedge H1 \cap H2 = \{\} \wedge H1 \neq \{\} \wedge H2 \neq \{\}$ 
    and  $S \subseteq H1$ 
    then have H2T:  $H2 \subseteq T$ 
      by auto
    have  $T \subseteq U$ 
      using Diff_iff opeT openin_imp_subset by auto
    with  $\langle S \subseteq U \rangle$  have Ueq:  $U = (U - S) \cup (S \cup T)$ 
      by auto
    have openin (top_of_set ((U - S) ∪ (S ∪ T))) H2
    proof (rule openin_subtopology_Un)
      show openin (top_of_set (S ∪ T)) H2

```

```

    using  $\langle H2 \subseteq T \rangle$  apply (auto simp: openin_closedin_eq)
  by (metis Diff_Diff_Int Diff_disjoint Diff_partition Diff_subset H Int_absorb1
    Un_Diff)
    then show openin (top_of_set (U - S)) H2
    by (meson H2T Un_upper2 opeT openin_subset_trans openin_trans)
qed
moreover have closedin (top_of_set ((U - S)  $\cup$  (S  $\cup$  T))) H2
proof (rule closedin_subtopology_Un)
  show closedin (top_of_set (U - S)) H2
    using H H2T cloT closedin_subset_trans
    by (blast intro: closedin_subtopology_Un closedin_trans)
qed (simp add: H)
ultimately
have H2:  $H2 = \{\}$   $\vee$   $H2 = U$ 
  using Ueq  $\langle$ connected U $\rangle$  unfolding connected_clopen by metis
then have  $H2 \subseteq S$ 
by (metis Diff_partition H Un_Diff_cancel Un_subset_iff  $\langle H2 \subseteq T \rangle$  assms(3)
  inf.orderE opeT openin_imp_subset)
moreover have  $T \subseteq H2 - S$ 
by (metis (no_types) H2 H opeT openin_closedin_eq topspace_euclidean_subtopology)
ultimately show False
  using H  $\langle S \subseteq H1 \rangle$  by blast
qed blast
qed

```

proposition *component_diff_connected:*

```

fixes S :: 'a::metric_space set
assumes connected S connected U  $S \subseteq U$  and C:  $C \in \text{components } (U - S)$ 
shows connected(U - C)
using  $\langle$ connected S $\rangle$  unfolding connected_closedin_eq not_ex de_Morgan_conj
proof clarify
  fix H3 H4
  assume clo3: closedin (top_of_set (U - C)) H3
    and clo4: closedin (top_of_set (U - C)) H4
    and  $H3 \cup H4 = U - C$  and  $H3 \cap H4 = \{\}$  and  $H3 \neq \{\}$  and  $H4 \neq \{\}$ 
    and * [rule_format]:
       $\forall H1 H2. \neg \text{closedin } (\text{top\_of\_set } S) H1 \vee$ 
         $\neg \text{closedin } (\text{top\_of\_set } S) H2 \vee$ 
         $H1 \cup H2 \neq S \vee H1 \cap H2 \neq \{\} \vee \neg H1 \neq \{\} \vee \neg H2 \neq \{\}$ 
  then have  $H3 \subseteq U - C$  and ope3: openin (top_of_set (U - C)) (U - C -
    H3)
    and  $H4 \subseteq U - C$  and ope4: openin (top_of_set (U - C)) (U - C - H4)
    by (auto simp: closedin_def)
  have  $C \neq \{\}$   $C \subseteq U - S$  connected C
  using C in_components_nonempty in_components_subset in_components_maximal
  by blast+
  have cCH3: connected (C  $\cup$  H3)
  proof (rule connected_Un_clopen_in_complement [OF  $\langle$ connected C $\rangle$   $\langle$ con-

```

```

nected U⟩ _ _ clo3]])
  show openin (top_of_set (U - C)) H3
  apply (simp add: openin_closedin_eq ⟨H3 ⊆ U - C⟩)
  apply (simp add: closedin_subtopology)
  by (metis Diff_cancel Diff_triv Un_Diff clo4 ⟨H3 ∩ H4 = {}⟩ ⟨H3 ∪ H4 =
U - C⟩ closedin_closed inf_commute sup_bot.left_neutral)
qed (use clo3 ⟨C ⊆ U - S⟩ in auto)
have cCH4: connected (C ∪ H4)
proof (rule connected_Un_clopen_in_complement [OF ⟨connected C⟩ ⟨con-
nected U⟩ _ _ clo4]])
  show openin (top_of_set (U - C)) H4
  apply (simp add: openin_closedin_eq ⟨H4 ⊆ U - C⟩)
  apply (simp add: closedin_subtopology)
  by (metis Diff_cancel Int_commute Un_Diff Un_Diff_Int ⟨H3 ∩ H4 = {}⟩
⟨H3 ∪ H4 = U - C⟩ clo3 closedin_closed)
qed (use clo4 ⟨C ⊆ U - S⟩ in auto)
have closedin (top_of_set S) (S ∩ H3) closedin (top_of_set S) (S ∩ H4)
  using clo3 clo4 ⟨S ⊆ U⟩ ⟨C ⊆ U - S⟩ by (auto simp: closedin_closed)
moreover have S ∩ H3 ≠ {}
  using components_maximal [OF C cCH3] ⟨C ≠ {}⟩ ⟨C ⊆ U - S⟩ ⟨H3 ≠ {}⟩
⟨H3 ⊆ U - C⟩ by auto
moreover have S ∩ H4 ≠ {}
  using components_maximal [OF C cCH4] ⟨C ≠ {}⟩ ⟨C ⊆ U - S⟩ ⟨H4 ≠ {}⟩
⟨H4 ⊆ U - C⟩ by auto
ultimately show False
  using * [of S ∩ H3 S ∩ H4] ⟨H3 ∩ H4 = {}⟩ ⟨C ⊆ U - S⟩ ⟨H3 ∪ H4 = U
- C⟩ ⟨S ⊆ U⟩
  by auto
qed

```

2.4.7 Constancy of a function from a connected set into a finite, disconnected or discrete set

Still missing: versions for a set that is smaller than $\mathbb{R}$, or countable.

lemma *continuous_disconnected_range_constant*:

```

assumes S: connected S
  and conf: continuous_on S f
  and fim: f ' S ⊆ t
  and cct: ⋀y. y ∈ t ⟹ connected_component_set t y = {y}
shows f constant_on S
proof (cases S = {})
case True then show ?thesis
  by (simp add: constant_on_def)
next
case False
  { fix x assume x ∈ S
    then have f ' S ⊆ {f x}
      by (metis connected_continuous_image conf connected_component_maximal
fim image_subset_iff rev_image_eqI S cct)

```



```

}
with False show ?thesis
  unfolding constant_on_def by blast
qed

```

This proof requires the existence of two separate values of the range type.

```

lemma finite_range_constant_imp_connected:
  assumes  $\wedge f::'a::topological\_space \Rightarrow 'b::real\_normed\_algebra\_1.$ 
     $\llbracket \text{continuous\_on } S f; \text{finite}(f \text{ ` } S) \rrbracket \Longrightarrow f \text{ constant\_on } S$ 
  shows connected S
proof -
  { fix t u
    assume clt: closedin (top_of_set S) t
      and clu: closedin (top_of_set S) u
      and tue:  $t \cap u = \{\}$  and tus:  $t \cup u = S$ 
    have conif: continuous_on S ( $\lambda x. \text{if } x \in t \text{ then } 0 \text{ else } 1$ )
      apply (subst tus [symmetric])
      apply (rule continuous_on_cases_local)
      using clt clu tue
      apply (auto simp: tus)
      done
    have fi: finite (( $\lambda x. \text{if } x \in t \text{ then } 0 \text{ else } 1$ ) ` S)
      by (rule finite_subset [of _ {0,1}]) auto
    have t = {}  $\vee$  u = {}
      using assms [OF conif fi] tus [symmetric]
      by (auto simp: Ball_def constant_on_def) (metis IntI empty_iff one_neq_zero tue)
    }
  then show ?thesis
    by (simp add: connected_closedin_eq)
qed

end
theory Abstract_Limits
  imports
    Abstract_Topology
begin

```

2.4.8 nhdsin and atin

```

definition nhdsin :: 'a topology  $\Rightarrow$  'a  $\Rightarrow$  'a filter
  where nhdsin X a =
    (if  $a \in \text{topspace } X$  then  $(\text{INF } S \in \{S. \text{openin } X S \wedge a \in S\}. \text{principal } S)$ 
    else bot)

```

```

definition atin :: 'a topology  $\Rightarrow$  'a  $\Rightarrow$  'a filter
  where atin X a  $\equiv \text{inf } (\text{nhdsin } X a) (\text{principal } (\text{topspace } X - \{a\}))$ 

```

lemma *nhdsin_degenerate* [simp]: $a \notin \text{topspace } X \implies \text{nhdsin } X \ a = \text{bot}$
and *atin_degenerate* [simp]: $a \notin \text{topspace } X \implies \text{atin } X \ a = \text{bot}$
by (*simp_all add: nhdsin_def atin_def*)

lemma *eventually_nhdsin*:

eventually P (*nhdsin* $X \ a$) $\longleftrightarrow a \notin \text{topspace } X \vee (\exists S. \text{openin } X \ S \wedge a \in S \wedge (\forall x \in S. P \ x))$

proof (*cases* $a \in \text{topspace } X$)

case *True*

hence *nhdsin* $X \ a = (\text{INF } S \in \{S. \text{openin } X \ S \wedge a \in S\}. \text{principal } S)$

by (*simp add: nhdsin_def*)

also have *eventually* $P \dots \longleftrightarrow (\exists S. \text{openin } X \ S \wedge a \in S \wedge (\forall x \in S. P \ x))$

using *True* **by** (*subst eventually_INF_base*) (*auto simp: eventually_principal*)

finally show *?thesis* **using** *True* **by** *simp*

qed *auto*

lemma *eventually_atin*:

eventually P (*atin* $X \ a$) $\longleftrightarrow a \notin \text{topspace } X \vee$

$(\exists U. \text{openin } X \ U \wedge a \in U \wedge (\forall x \in U - \{a\}. P \ x))$

proof (*cases* $a \in \text{topspace } X$)

case *True*

hence *eventually* P (*atin* $X \ a$) $\longleftrightarrow (\exists S. \text{openin } X \ S \wedge$

$a \in S \wedge (\forall x \in S. x \in \text{topspace } X \wedge x \neq a \longrightarrow P \ x))$

by (*simp add: atin_def eventually_inf_principal eventually_nhdsin*)

also have $\dots \longleftrightarrow (\exists U. \text{openin } X \ U \wedge a \in U \wedge (\forall x \in U - \{a\}. P \ x))$

using *openin_subset* **by** (*intro ex_cong*) *auto*

finally show *?thesis* **by** (*simp add: True*)

qed *auto*

2.4.9 Limits in a topological space

definition *limitin* :: $'a \text{ topology} \Rightarrow ('b \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'b \text{ filter} \Rightarrow \text{bool}$ **where**

limitin $X \ f \ l \ F \equiv l \in \text{topspace } X \wedge (\forall U. \text{openin } X \ U \wedge l \in U \longrightarrow \text{eventually } (\lambda x. f \ x \in U) \ F)$

lemma *limitin_canonical_iff* [simp]: *limitin* *euclidean* $f \ l \ F \longleftrightarrow (f \longrightarrow l) \ F$

by (*auto simp: limitin_def tendsto_def*)

lemma *limitin_topspace*: *limitin* $X \ f \ l \ F \implies l \in \text{topspace } X$

by (*simp add: limitin_def*)

lemma *limitin_const_iff* [simp]: *limitin* $X \ (\lambda a. l) \ l \ F \longleftrightarrow l \in \text{topspace } X$

by (*simp add: limitin_def*)

lemma *limitin_const*: *limitin* *euclidean* $(\lambda a. l) \ l \ F$

by *simp*

lemma *limitin_eventually*:

$\llbracket l \in \text{topspace } X; \text{eventually } (\lambda x. f \ x = l) \ F \rrbracket \implies \text{limitin } X \ f \ l \ F$

by (auto simp: limitin_def eventually_mono)

lemma *limitin_subsequence*:

$\llbracket \text{strict\_mono } r; \text{limitin } X f l \text{ sequentially} \rrbracket \implies \text{limitin } X (f \circ r) l \text{ sequentially}$
unfolding *limitin_def* **using** *eventually_subseq* **by** *fastforce*

lemma *limitin_subtopology*:

limitin (*subtopology* *X S*) *f l F*
 $\longleftrightarrow l \in S \wedge \text{eventually } (\lambda a. f a \in S) F \wedge \text{limitin } X f l F$ (**is** *?lhs* = *?rhs*)

proof (cases $l \in S \cap \text{topspace } X$)

case *True*

show *?thesis*

proof

assume *L*: *?lhs*

with *True*

have $\forall_F b \text{ in } F. f b \in \text{topspace } X \cap S$

by (*metis* (*no_types*) *limitin_def* *openin_topspace* *topspace_subtopology*)

with *L* **show** *?rhs*

apply (*clarsimp* *simp* *add*: *limitin_def* *eventually_mono* *openin_subtopology_alt*)

apply (*drule_tac* *x*= $S \cap U$ **in** *spec*, *force* *simp*: *elim*: *eventually_mono*)

done

next

assume *?rhs*

then show *?lhs*

using *eventually_elim2*

by (*fastforce* *simp* *add*: *limitin_def* *openin_subtopology_alt*)

qed

qed (auto *simp*: *limitin_def*)

lemma *limitin_canonical_iff_gen* [*simp*]:

assumes *open S*

shows *limitin* (*top_of_set S*) *f l F* $\longleftrightarrow (f \longrightarrow l) F \wedge l \in S$

using *assms* **by** (auto *simp*: *limitin_subtopology_tendsto_def*)

lemma *limitin_sequentially*:

limitin *X S l sequentially* $\longleftrightarrow$

$l \in \text{topspace } X \wedge (\forall U. \text{openin } X U \wedge l \in U \longrightarrow (\exists N. \forall n. N \leq n \longrightarrow S n \in U))$

by (*simp* *add*: *limitin_def* *eventually_sequentially*)

lemma *limitin_sequentially_offset*:

limitin *X f l sequentially* $\implies \text{limitin } X (\lambda i. f (i + k)) l \text{ sequentially}$

unfolding *limitin_sequentially*

by (*metis* *add commute* *le_add2* *order_trans*)

lemma *limitin_sequentially_offset_rev*:

assumes *limitin* *X* $(\lambda i. f (i + k)) l \text{ sequentially}$

shows *limitin* *X f l sequentially*

```

proof –
  have  $\exists N. \forall n \geq N. f\ n \in U$  if  $U$ : openin  $X$   $U$   $l \in U$  for  $U$ 
  proof –
    obtain  $N$  where  $\bigwedge n. n \geq N \implies f\ (n + k) \in U$ 
    using assms  $U$  unfolding limitin_sequentially by blast
    then have  $\forall n \geq N+k. f\ n \in U$ 
    by (metis add_leD2 le_add_diff_inverse ordered_cancel_comm_monoid_diff_class.le_diff_conv2
add.commute)
    then show ?thesis ..
  qed
  with assms show ?thesis
    unfolding limitin_sequentially
    by simp
qed

```

```

lemma limitin_atin:
  limitin  $Y$   $f$   $y$  (atin  $X$   $x$ )  $\longleftrightarrow$ 
     $y \in \text{topspace } Y \wedge$ 
     $(x \in \text{topspace } X$ 
     $\longrightarrow (\forall V. \text{openin } Y\ V \wedge y \in V$ 
     $\longrightarrow (\exists U. \text{openin } X\ U \wedge x \in U \wedge f\ ' (U - \{x\}) \subseteq V)))$ 
  by (auto simp: limitin_def eventually_atin image_subset_iff)

```

```

lemma limitin_atin_self:
  limitin  $Y$   $f$  ( $f\ a$ ) (atin  $X$   $a$ )  $\longleftrightarrow$ 
     $f\ a \in \text{topspace } Y \wedge$ 
     $(a \in \text{topspace } X$ 
     $\longrightarrow (\forall V. \text{openin } Y\ V \wedge f\ a \in V$ 
     $\longrightarrow (\exists U. \text{openin } X\ U \wedge a \in U \wedge f\ ' U \subseteq V)))$ 
  unfolding limitin_atin by fastforce

```

```

lemma limitin_trivial:
   $\llbracket \text{trivial\_limit } F; y \in \text{topspace } X \rrbracket \implies \text{limitin } X\ f\ y\ F$ 
  by (simp add: limitin_def)

```

```

lemma limitin_transform_eventually:
   $\llbracket \text{eventually } (\lambda x. f\ x = g\ x)\ F; \text{limitin } X\ f\ l\ F \rrbracket \implies \text{limitin } X\ g\ l\ F$ 
  unfolding limitin_def using eventually_elim2 by fastforce

```

```

lemma continuous_map_limit:
  assumes continuous_map  $X$   $Y$   $g$  and  $f$ : limitin  $X$   $f$   $l$   $F$ 
  shows limitin  $Y$  ( $g \circ f$ ) ( $g\ l$ )  $F$ 
proof –
  have  $g\ l \in \text{topspace } Y$ 
    by (meson assms continuous_map_def limitin_topspace)
  moreover
  have  $\bigwedge U. \llbracket \forall V. \text{openin } X\ V \wedge l \in V \longrightarrow (\forall_F x \text{ in } F. f\ x \in V); \text{openin } Y\ U; g$ 
   $l \in U \rrbracket$ 
     $\implies \forall_F x \text{ in } F. g\ (f\ x) \in U$ 

```

```

    using assms eventually_mono
  by (fastforce simp: limitin_def dest!: openin_continuous_map_preimage)
ultimately show ?thesis
  using f by (fastforce simp add: limitin_def)
qed

```

2.4.10 Pointwise continuity in topological spaces

definition *topcontinuous_at where*

```

topcontinuous_at X Y f x  $\longleftrightarrow$ 
  x  $\in$  topspace X  $\wedge$ 
  ( $\forall x \in$  topspace X. f x  $\in$  topspace Y)  $\wedge$ 
  ( $\forall V$ . openin Y V  $\wedge$  f x  $\in$  V
     $\longrightarrow$  ( $\exists U$ . openin X U  $\wedge$  x  $\in$  U  $\wedge$  ( $\forall y \in$  U. f y  $\in$  V)))

```

lemma *topcontinuous_at_atin:*

```

topcontinuous_at X Y f x  $\longleftrightarrow$ 
  x  $\in$  topspace X  $\wedge$ 
  ( $\forall x \in$  topspace X. f x  $\in$  topspace Y)  $\wedge$ 
  limitin Y f (f x) (atin X x)

```

unfolding *topcontinuous_at_def*

by (fastforce simp add: limitin_atin)+

lemma *continuous_map_eq_topcontinuous_at:*

```

continuous_map X Y f  $\longleftrightarrow$  ( $\forall x \in$  topspace X. topcontinuous_at X Y f x)
(is ?lhs = ?rhs)

```

proof

assume ?lhs

then show ?rhs

by (auto simp: continuous_map_def topcontinuous_at_def)

next

assume R: ?rhs

then show ?lhs

apply (auto simp: continuous_map_def topcontinuous_at_def)

apply (subst openin_subopen, safe)

apply (drule bspec, assumption)

using openin_subset[of X] **apply** (auto simp: subset_iff dest!: spec)

done

qed

lemma *continuous_map_atin:*

```

continuous_map X Y f  $\longleftrightarrow$  ( $\forall x \in$  topspace X. limitin Y f (f x) (atin X x))

```

by (auto simp: limitin_def topcontinuous_at_atin continuous_map_eq_topcontinuous_at)

lemma *limitin_continuous_map:*

```

 $\llbracket$ continuous_map X Y f; a  $\in$  topspace X; f a = b $\rrbracket \implies$  limitin Y f b (atin X a)

```

by (auto simp: continuous_map_atin)

2.4.11 Combining theorems for continuous functions into the reals

lemma *continuous_map_canonical_const* [*continuous_intros*]:
 continuous_map X euclidean ($\lambda x. c$)
 by *simp*

lemma *continuous_map_real_mult* [*continuous_intros*]:
 $\llbracket \text{continuous\_map } X \text{ euclideanreal } f; \text{ continuous\_map } X \text{ euclideanreal } g \rrbracket$
 $\implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. f \ x * g \ x)$
 by (*simp add: continuous_map_atin tendsto_mult*)

lemma *continuous_map_real_pow* [*continuous_intros*]:
 continuous_map X euclideanreal f $\implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. f \ x ^ n)$
 by (*induction n (auto simp: continuous_map_real_mult)*)

lemma *continuous_map_real_mult_left*:
 continuous_map X euclideanreal f $\implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. c * f \ x)$
 by (*simp add: continuous_map_atin tendsto_mult*)

lemma *continuous_map_real_mult_left_eq*:
 continuous_map X euclideanreal ($\lambda x. c * f \ x$) $\longleftrightarrow c = 0 \vee \text{continuous\_map } X \text{ euclideanreal } f$

proof (*cases c = 0*)

case *False*

have *continuous_map X euclideanreal* ($\lambda x. c * f \ x$) $\implies \text{continuous\_map } X \text{ euclideanreal } f$

apply (*frule continuous_map_real_mult_left [where c=inverse c]*)

apply (*simp add: field_simps False*)

done

with *False show ?thesis*

using *continuous_map_real_mult_left* **by** *blast*

qed *simp*

lemma *continuous_map_real_mult_right*:
 continuous_map X euclideanreal f $\implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. f \ x * c)$
 by (*simp add: continuous_map_atin tendsto_mult*)

lemma *continuous_map_real_mult_right_eq*:
 continuous_map X euclideanreal ($\lambda x. f \ x * c$) $\longleftrightarrow c = 0 \vee \text{continuous\_map } X \text{ euclideanreal } f$
 by (*simp add: mult.commute flip: continuous_map_real_mult_left_eq*)

lemma *continuous_map_minus* [*continuous_intros*]:
 fixes $f :: 'a \Rightarrow 'b :: \text{real\_normed\_vector}$
 shows *continuous_map X euclidean f* $\implies \text{continuous\_map } X \text{ euclidean } (\lambda x. - f \ x)$

by (simp add: continuous_map_atin tendsto_minus)

lemma continuous_map_minus_eq [simp]:

fixes $f :: 'a \Rightarrow 'b :: \text{real\_normed\_vector}$

shows $\text{continuous\_map } X \text{ euclidean } (\lambda x. - f x) \longleftrightarrow \text{continuous\_map } X \text{ euclidean } f$

using continuous_map_minus add.inverse_inverse continuous_map_eq **by** fast-force

lemma continuous_map_add [continuous_intros]:

fixes $f :: 'a \Rightarrow 'b :: \text{real\_normed\_vector}$

shows $\llbracket \text{continuous\_map } X \text{ euclidean } f; \text{continuous\_map } X \text{ euclidean } g \rrbracket \implies \text{continuous\_map } X \text{ euclidean } (\lambda x. f x + g x)$

by (simp add: continuous_map_atin tendsto_add)

lemma continuous_map_diff [continuous_intros]:

fixes $f :: 'a \Rightarrow 'b :: \text{real\_normed\_vector}$

shows $\llbracket \text{continuous\_map } X \text{ euclidean } f; \text{continuous\_map } X \text{ euclidean } g \rrbracket \implies \text{continuous\_map } X \text{ euclidean } (\lambda x. f x - g x)$

by (simp add: continuous_map_atin tendsto_diff)

lemma continuous_map_norm [continuous_intros]:

fixes $f :: 'a \Rightarrow 'b :: \text{real\_normed\_vector}$

shows $\text{continuous\_map } X \text{ euclidean } f \implies \text{continuous\_map } X \text{ euclidean } (\lambda x. \text{norm}(f x))$

by (simp add: continuous_map_atin tendsto_norm)

lemma continuous_map_real_abs [continuous_intros]:

$\text{continuous\_map } X \text{ euclideanreal } f \implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. \text{abs}(f x))$

by (simp add: continuous_map_atin tendsto_rabs)

lemma continuous_map_real_max [continuous_intros]:

$\llbracket \text{continuous\_map } X \text{ euclideanreal } f; \text{continuous\_map } X \text{ euclideanreal } g \rrbracket$

$\implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. \text{max } (f x) (g x))$

by (simp add: continuous_map_atin tendsto_max)

lemma continuous_map_real_min [continuous_intros]:

$\llbracket \text{continuous\_map } X \text{ euclideanreal } f; \text{continuous\_map } X \text{ euclideanreal } g \rrbracket$

$\implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. \text{min } (f x) (g x))$

by (simp add: continuous_map_atin tendsto_min)

lemma continuous_map_sum [continuous_intros]:

fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \text{real\_normed\_vector}$

shows $\llbracket \text{finite } I; \bigwedge i. i \in I \implies \text{continuous\_map } X \text{ euclidean } (\lambda x. f x i) \rrbracket$

$\implies \text{continuous\_map } X \text{ euclidean } (\lambda x. \text{sum } (f x) I)$

by (simp add: continuous_map_atin tendsto_sum)

lemma continuous_map_prod [continuous_intros]:

```

    [[finite I;
       $\bigwedge i. i \in I \implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. f\ x\ i)$ 
       $\implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. \text{prod } (f\ x)\ I)$ 
    by (simp add: continuous_map_atin tendsto_prod)

lemma continuous_map_real_inverse [continuous_intros]:
  [[continuous_map X euclideanreal f;  $\bigwedge x. x \in \text{topspace } X \implies f\ x \neq 0$ ]
    $\implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. \text{inverse}(f\ x))$ 
  by (simp add: continuous_map_atin tendsto_inverse)

lemma continuous_map_real_divide [continuous_intros]:
  [[continuous_map X euclideanreal f; continuous_map X euclideanreal g;  $\bigwedge x. x$ 
 $\in \text{topspace } X \implies g\ x \neq 0$ ]
    $\implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. f\ x / g\ x)$ 
  by (simp add: continuous_map_atin tendsto_divide)

end

```


Chapter 3

Functional Analysis

3.1 A decision procedure for metric spaces

```
theory Metric_Arith
  imports HOL.Real_Vector_Spaces
begin
```

A port of the decision procedure “Formalization of metric spaces in HOL Light” [4] for the type class *metric_space*, with the *Argo* solver as backend.

```
named_theorems metric_prenex
named_theorems metric_nnf
named_theorems metric_unfold
named_theorems metric_pre_arith

lemmas pre_arith_simps =
  max.bounded_iff max_less_iff_conj
  le_max_iff_disj less_max_iff_disj
  simp_thms HOL.eq_commute
declare pre_arith_simps [metric_pre_arith]

lemmas unfold_simps =
  Un_iff subset_iff disjoint_iff_not_equal
  Ball_def Bex_def
declare unfold_simps [metric_unfold]

declare HOL.nnf_simps(4) [metric_prenex]

lemma imp_prenex [metric_prenex]:

$$\bigwedge P Q. (\exists x. P x) \longrightarrow Q \equiv \forall x. (P x \longrightarrow Q)$$


$$\bigwedge P Q. P \longrightarrow (\exists x. Q x) \equiv \exists x. (P \longrightarrow Q x)$$


$$\bigwedge P Q. (\forall x. P x) \longrightarrow Q \equiv \exists x. (P x \longrightarrow Q)$$


$$\bigwedge P Q. P \longrightarrow (\forall x. Q x) \equiv \forall x. (P \longrightarrow Q x)$$

  by simp_all

lemma ex_prenex [metric_prenex]:
```

$\bigwedge P Q. (\exists x. P x) \wedge Q \equiv \exists x. (P x \wedge Q)$
 $\bigwedge P Q. P \wedge (\exists x. Q x) \equiv \exists x. (P \wedge Q x)$
 $\bigwedge P Q. (\exists x. P x) \vee Q \equiv \exists x. (P x \vee Q)$
 $\bigwedge P Q. P \vee (\exists x. Q x) \equiv \exists x. (P \vee Q x)$
 $\bigwedge P. \neg(\exists x. P x) \equiv \forall x. \neg P x$
by *simp_all*

lemma *all_prenex* [*metric_prenex*]:
 $\bigwedge P Q. (\forall x. P x) \wedge Q \equiv \forall x. (P x \wedge Q)$
 $\bigwedge P Q. P \wedge (\forall x. Q x) \equiv \forall x. (P \wedge Q x)$
 $\bigwedge P Q. (\forall x. P x) \vee Q \equiv \forall x. (P x \vee Q)$
 $\bigwedge P Q. P \vee (\forall x. Q x) \equiv \forall x. (P \vee Q x)$
 $\bigwedge P. \neg(\forall x. P x) \equiv \exists x. \neg P x$
by *simp_all*

lemma *nnf_thms* [*metric_nnf*]:
 $(\neg (P \wedge Q)) = (\neg P \vee \neg Q)$
 $(\neg (P \vee Q)) = (\neg P \wedge \neg Q)$
 $(P \longrightarrow Q) = (\neg P \vee Q)$
 $(P = Q) \longleftrightarrow (P \vee \neg Q) \wedge (\neg P \vee Q)$
 $(\neg (P = Q)) \longleftrightarrow (\neg P \vee \neg Q) \wedge (P \vee Q)$
 $(\neg \neg P) = P$
by *blast+*

lemmas *nnf_simps* = *nnf_thms* *linorder_not_less* *linorder_not_le*
declare *nnf_simps* [*metric_nnf*]

lemma *ball_insert*: $(\forall x \in \text{insert } a B. P x) = (P a \wedge (\forall x \in B. P x))$
by *blast*

lemma *Sup_insert_insert*:
fixes *a::real*
shows *Sup* (*insert* *a* (*insert* *b* *s*)) = *Sup* (*insert* (*max* *a* *b*) *s*)
by (*simp* *add*: *Sup_real_def*)

lemma *real_abs_dist*: $|\text{dist } x y| = \text{dist } x y$
by *simp*

lemma *maxdist_thm* [*THEN HOL.eq_reflection*]:
assumes *finite* *s* $x \in s$ $y \in s$
defines $\bigwedge a. f a \equiv |\text{dist } x a - \text{dist } a y|$
shows $\text{dist } x y = \text{Sup } (f \text{ ` } s)$
proof –
have $\text{dist } x y \leq \text{Sup } (f \text{ ` } s)$
proof –
have *finite* (*f* ` *s*)
by (*simp* *add*: *finite_s*)
moreover **have** $|\text{dist } x y - \text{dist } y y| \in f \text{ ` } s$
by (*metis* *y* $\in s$ *f_def* *imageI*)

```

    ultimately show ?thesis
      using le_cSup_finite by simp
  qed
  also have  $\text{Sup } (f \text{ ` } s) \leq \text{dist } x \ y$ 
    using  $\langle x \in s \rangle \text{ cSUP\_least[of } s \ f] \text{ abs\_dist\_diff\_le}$ 
    unfolding f_def
    by blast
  finally show ?thesis .
qed

theorem metric_eq_thm [THEN HOL.eq_reflection]:
   $x \in s \implies y \in s \implies x = y \longleftrightarrow (\forall a \in s. \text{dist } x \ a = \text{dist } y \ a)$ 
  by auto

ML_file metric_arith.ML

method_setup metric = <
  Scan.succeed (SIMPLE_METHOD' o Metric_Arith.metric_arith_tac)
> prove simple linear statements in metric spaces ( $\forall \exists_p$  fragment)

end

```

3.2 Elementary Metric Spaces

```

theory Elementary_Metric_Spaces
  imports
    Abstract_Topology_2
    Metric_Arith
begin

```

3.2.1 Open and closed balls

```

definition ball :: 'a::metric_space  $\Rightarrow$  real  $\Rightarrow$  'a set
  where ball x e = {y. dist x y < e}

```

```

definition cball :: 'a::metric_space  $\Rightarrow$  real  $\Rightarrow$  'a set
  where cball x e = {y. dist x y  $\leq$  e}

```

```

definition sphere :: 'a::metric_space  $\Rightarrow$  real  $\Rightarrow$  'a set
  where sphere x e = {y. dist x y = e}

```

```

lemma mem_ball [simp, metric_unfold]:  $y \in \text{ball } x \ e \longleftrightarrow \text{dist } x \ y < e$ 
  by (simp add: ball_def)

```

```

lemma mem_cball [simp, metric_unfold]:  $y \in \text{cball } x \ e \longleftrightarrow \text{dist } x \ y \leq e$ 
  by (simp add: cball_def)

```

```

lemma mem_sphere [simp]:  $y \in \text{sphere } x \ e \longleftrightarrow \text{dist } x \ y = e$ 
  by (simp add: sphere_def)

```

lemma *ball_trivial* [*simp*]: $\text{ball } x \ 0 = \{\}$
by (*simp add: ball_def*)

lemma *cball_trivial* [*simp*]: $\text{cball } x \ 0 = \{x\}$
by (*simp add: cball_def*)

lemma *sphere_trivial* [*simp*]: $\text{sphere } x \ 0 = \{x\}$
by (*simp add: sphere_def*)

lemma *disjoint_ballI*: $\text{dist } x \ y \geq r + s \implies \text{ball } x \ r \cap \text{ball } y \ s = \{\}$
using *dist_triangle_less_add not_le* **by** *fastforce*

lemma *disjoint_cballI*: $\text{dist } x \ y > r + s \implies \text{cball } x \ r \cap \text{cball } y \ s = \{\}$
by (*metis add_mono disjoint_iff_not_equal dist_triangle2 dual_order.trans leD mem_cball*)

lemma *sphere_empty* [*simp*]: $r < 0 \implies \text{sphere } a \ r = \{\}$
for $a :: 'a::\text{metric\_space}$
by *auto*

lemma *centre_in_ball* [*simp*]: $x \in \text{ball } x \ e \longleftrightarrow 0 < e$
by *simp*

lemma *centre_in_cball* [*simp*]: $x \in \text{cball } x \ e \longleftrightarrow 0 \leq e$
by *simp*

lemma *ball_subset_cball* [*simp, intro*]: $\text{ball } x \ e \subseteq \text{cball } x \ e$
by (*simp add: subset_eq*)

lemma *mem_ball_imp_mem_cball*: $x \in \text{ball } y \ e \implies x \in \text{cball } y \ e$
by *auto*

lemma *sphere_cball* [*simp, intro*]: $\text{sphere } z \ r \subseteq \text{cball } z \ r$
by *force*

lemma *cball_diff_sphere*: $\text{cball } a \ r - \text{sphere } a \ r = \text{ball } a \ r$
by *auto*

lemma *subset_ball*[*intro*]: $d \leq e \implies \text{ball } x \ d \subseteq \text{ball } x \ e$
by *auto*

lemma *subset_cball*[*intro*]: $d \leq e \implies \text{cball } x \ d \subseteq \text{cball } x \ e$
by *auto*

lemma *mem_ball_leI*: $x \in \text{ball } y \ e \implies e \leq f \implies x \in \text{ball } y \ f$
by *auto*

lemma *mem_cball_leI*: $x \in \text{cball } y \ e \implies e \leq f \implies x \in \text{cball } y \ f$

by *auto*

lemma *cball_trans*: $y \in \text{cball } z \ b \implies x \in \text{cball } y \ a \implies x \in \text{cball } z \ (b + a)$
by *metric*

lemma *ball_max_Un*: $\text{ball } a \ (\max r \ s) = \text{ball } a \ r \cup \text{ball } a \ s$
by *auto*

lemma *ball_min_Int*: $\text{ball } a \ (\min r \ s) = \text{ball } a \ r \cap \text{ball } a \ s$
by *auto*

lemma *cball_max_Un*: $\text{cball } a \ (\max r \ s) = \text{cball } a \ r \cup \text{cball } a \ s$
by *auto*

lemma *cball_min_Int*: $\text{cball } a \ (\min r \ s) = \text{cball } a \ r \cap \text{cball } a \ s$
by *auto*

lemma *cball_diff_eq_sphere*: $\text{cball } a \ r - \text{ball } a \ r = \text{sphere } a \ r$
by *auto*

lemma *open_ball* [*intro*, *simp*]: $\text{open } (\text{ball } x \ e)$
proof –
have $\text{open } (\text{dist } x - \{..<e\})$
 by (*intro open_vimage open_lessThan continuous_intros*)
also have $\text{dist } x - \{..<e\} = \text{ball } x \ e$
 by *auto*
finally show *?thesis* .
qed

lemma *open_contains_ball*: $\text{open } S \longleftrightarrow (\forall x \in S. \exists e > 0. \text{ball } x \ e \subseteq S)$
by (*simp add: open_dist subset_eq Ball_def dist_commute*)

lemma *openI* [*intro?*]: $(\bigwedge x. x \in S \implies \exists e > 0. \text{ball } x \ e \subseteq S) \implies \text{open } S$
by (*auto simp: open_contains_ball*)

lemma *openE*[*elim?*]:
assumes $\text{open } S \ x \in S$
obtains e **where** $e > 0 \ \text{ball } x \ e \subseteq S$
using *assms unfolding open_contains_ball* **by** *auto*

lemma *open_contains_ball_eq*: $\text{open } S \implies x \in S \longleftrightarrow (\exists e > 0. \text{ball } x \ e \subseteq S)$
by (*metis open_contains_ball subset_eq centre_in_ball*)

lemma *ball_eq_empty*[*simp*]: $\text{ball } x \ e = \{\} \longleftrightarrow e \leq 0$
unfolding *mem_ball set_eq_iff*
by (*simp add: not_less*) *metric*

lemma *ball_empty*: $e \leq 0 \implies \text{ball } x \ e = \{\}$
by *simp*

```

lemma closed_cball [iff]: closed (cball x e)
proof –
  have closed (dist x – ‘{..e}’)
    by (intro closed_vimage closed_atMost continuous_intros)
  also have dist x – ‘{..e}’ = cball x e
    by auto
  finally show ?thesis .
qed

lemma open_contains_cball: open S  $\longleftrightarrow$  ( $\forall x \in S. \exists e > 0. \text{ cball } x \ e \subseteq S$ )
proof –
  {
    fix x and e::real
    assume  $x \in S \ e > 0 \ \text{ball } x \ e \subseteq S$ 
    then have  $\exists d > 0. \text{ cball } x \ d \subseteq S$ 
      unfolding subset_eq by (rule_tac  $x = e/2$  in exI, auto)
  }
moreover
  {
    fix x and e::real
    assume  $x \in S \ e > 0 \ \text{cball } x \ e \subseteq S$ 
    then have  $\exists d > 0. \text{ ball } x \ d \subseteq S$ 
      using mem_ball_imp_mem_cball by blast
  }
ultimately show ?thesis
  unfolding open_contains_ball by auto
qed

lemma open_contains_cball_eq: open S  $\implies$  ( $\forall x. x \in S \longleftrightarrow (\exists e > 0. \text{ cball } x \ e \subseteq S)$ )
  by (metis open_contains_cball subset_eq order_less_imp_le centre_in_cball)

lemma eventually_nhds_ball:  $d > 0 \implies \text{eventually } (\lambda x. x \in \text{ball } z \ d) \ (\text{nhds } z)$ 
  by (rule eventually_nhds_in_open) simp_all

lemma eventually_at_ball:  $d > 0 \implies \text{eventually } (\lambda t. t \in \text{ball } z \ d \wedge t \in A) \ (\text{at } z \text{ within } A)$ 
  unfolding eventually_at by (intro exI[of _ d]) (simp_all add: dist_commute)

lemma eventually_at_ball':  $d > 0 \implies \text{eventually } (\lambda t. t \in \text{ball } z \ d \wedge t \neq z \wedge t \in A) \ (\text{at } z \text{ within } A)$ 
  unfolding eventually_at by (intro exI[of _ d]) (simp_all add: dist_commute)

lemma at_within_ball:  $e > 0 \implies \text{dist } x \ y < e \implies \text{at } y \text{ within ball } x \ e = \text{at } y$ 
  by (subst at_within_open) auto

lemma atLeastAtMost_eq_cball:
  fixes a b::real

```

shows $\{a \dots b\} = \text{cball } ((a + b)/2) ((b - a)/2)$
by (*auto simp: dist_real_def field_simps*)

lemma *cball_eq_atLeastAtMost*:
fixes $a b :: \text{real}$
shows $\text{cball } a b = \{a - b \dots a + b\}$
by (*auto simp: dist_real_def*)

lemma *greaterThanLessThan_eq_ball*:
fixes $a b :: \text{real}$
shows $\{a < \dots < b\} = \text{ball } ((a + b)/2) ((b - a)/2)$
by (*auto simp: dist_real_def field_simps*)

lemma *ball_eq_greaterThanLessThan*:
fixes $a b :: \text{real}$
shows $\text{ball } a b = \{a - b < \dots < a + b\}$
by (*auto simp: dist_real_def*)

lemma *interior_ball [simp]*: $\text{interior } (\text{ball } x e) = \text{ball } x e$
by (*simp add: interior_open*)

lemma *cball_eq_empty [simp]*: $\text{cball } x e = \{\} \longleftrightarrow e < 0$
apply (*simp add: set_eq_iff not_le*)
apply (*metis zero_le_dist dist_self order_less_le_trans*)
done

lemma *cball_empty [simp]*: $e < 0 \implies \text{cball } x e = \{\}$
by *simp*

lemma *cball_singleton*:
fixes $x :: 'a :: \text{metric\_space}$
shows $e = 0 \implies \text{cball } x e = \{x\}$
by *simp*

lemma *ball_divide_subset*: $d \geq 1 \implies \text{ball } x (e/d) \subseteq \text{ball } x e$
by (*metis ball_eq_empty div_by_1 frac_le linear subset_ball zero_less_one*)

lemma *ball_divide_subset_numeral*: $\text{ball } x (e / \text{numeral } w) \subseteq \text{ball } x e$
using *ball_divide_subset one_le_numerical* **by** *blast*

lemma *cball_divide_subset*: $d \geq 1 \implies \text{cball } x (e/d) \subseteq \text{cball } x e$
apply (*cases e < 0, simp add: field_split_simps*)
by (*metis div_by_1 frac_le less_numerical_extra(1) not_le order_refl subset_cball*)

lemma *cball_divide_subset_numeral*: $\text{cball } x (e / \text{numeral } w) \subseteq \text{cball } x e$
using *cball_divide_subset one_le_numerical* **by** *blast*

lemma *cball_scale*:
assumes $a \neq 0$

```

shows  $(\lambda x. a *_R x) \text{ ' } cball\ c\ r = cball\ (a *_R\ c :: 'a :: real\_normed\_vector)\ (|a|$ 
 $*\ r)$ 
proof –
  have  $1: (\lambda x. a *_R x) \text{ ' } cball\ c\ r \subseteq cball\ (a *_R\ c)\ (|a| * r)$  if  $a \neq 0$  for  $a\ r$  and
 $c :: 'a$ 
  proof safe
    fix  $x$ 
    assume  $x: x \in cball\ c\ r$ 
    have  $dist\ (a *_R\ c)\ (a *_R\ x) = norm\ (a *_R\ c - a *_R\ x)$ 
      by (auto simp: dist_norm)
    also have  $a *_R\ c - a *_R\ x = a *_R\ (c - x)$ 
      by (simp add: algebra_simps)
    finally show  $a *_R\ x \in cball\ (a *_R\ c)\ (|a| * r)$ 
      using that x by (auto simp: dist_norm)
  qed

  have  $cball\ (a *_R\ c)\ (|a| * r) = (\lambda x. a *_R\ x) \text{ ' } (\lambda x. inverse\ a *_R\ x) \text{ ' } cball\ (a *_R$ 
 $c)\ (|a| * r)$ 
    unfolding image_image using assms by simp
    also have  $\dots \subseteq (\lambda x. a *_R\ x) \text{ ' } cball\ (inverse\ a *_R\ (a *_R\ c))\ (|inverse\ a| * (|a|$ 
 $*\ r))$ 
      using assms by (intro image_mono 1) auto
    also have  $\dots = (\lambda x. a *_R\ x) \text{ ' } cball\ c\ r$ 
      using assms by (simp add: algebra_simps)
    finally have  $cball\ (a *_R\ c)\ (|a| * r) \subseteq (\lambda x. a *_R\ x) \text{ ' } cball\ c\ r$  .
    moreover from assms have  $(\lambda x. a *_R\ x) \text{ ' } cball\ c\ r \subseteq cball\ (a *_R\ c)\ (|a| * r)$ 
      by (intro 1) auto
    ultimately show ?thesis by blast
  qed

lemma ball_scale:
  assumes  $a \neq 0$ 
  shows  $(\lambda x. a *_R\ x) \text{ ' } ball\ c\ r = ball\ (a *_R\ c :: 'a :: real\_normed\_vector)\ (|a|$ 
 $*\ r)$ 
proof –
  have  $1: (\lambda x. a *_R\ x) \text{ ' } ball\ c\ r \subseteq ball\ (a *_R\ c)\ (|a| * r)$  if  $a \neq 0$  for  $a\ r$  and  $c$ 
 $:: 'a$ 
  proof safe
    fix  $x$ 
    assume  $x: x \in ball\ c\ r$ 
    have  $dist\ (a *_R\ c)\ (a *_R\ x) = norm\ (a *_R\ c - a *_R\ x)$ 
      by (auto simp: dist_norm)
    also have  $a *_R\ c - a *_R\ x = a *_R\ (c - x)$ 
      by (simp add: algebra_simps)
    finally show  $a *_R\ x \in ball\ (a *_R\ c)\ (|a| * r)$ 
      using that x by (auto simp: dist_norm)
  qed

  have  $ball\ (a *_R\ c)\ (|a| * r) = (\lambda x. a *_R\ x) \text{ ' } (\lambda x. inverse\ a *_R\ x) \text{ ' } ball\ (a *_R\ c)$ 

```



```

(|a| * r)
  unfolding image_image using assms by simp
  also have ...  $\subseteq (\lambda x. a *_R x) \text{ ` ball (inverse a *_R (a *_R c)) (|inverse a| * (|a| * r)) }$ 
  using assms by (intro image_mono 1) auto
  also have ... =  $(\lambda x. a *_R x) \text{ ` ball c r}$ 
  using assms by (simp add: algebra_simps)
  finally have ball (a *_R c) (|a| * r)  $\subseteq (\lambda x. a *_R x) \text{ ` ball c r .}$ 
  moreover from assms have  $(\lambda x. a *_R x) \text{ ` ball c r} \subseteq \text{ball (a *_R c) (|a| * r)}$ 
  by (intro 1) auto
  ultimately show ?thesis by blast
qed

```

3.2.2 Limit Points

lemma islimpt_approachable:

fixes $x :: 'a::\text{metric\_space}$

shows $x \text{ islimpt } S \longleftrightarrow (\forall e>0. \exists x' \in S. x' \neq x \wedge \text{dist } x' x < e)$

unfolding islimpt_iff_eventually_eventually_at by fast

lemma islimpt_approachable_le: $x \text{ islimpt } S \longleftrightarrow (\forall e>0. \exists x' \in S. x' \neq x \wedge \text{dist } x' x \leq e)$

for $x :: 'a::\text{metric\_space}$

unfolding islimpt_approachable

using approachable_lt_le2 [where $f=\lambda y. \text{dist } y x$ and $P=\lambda y. y \notin S \vee y = x$ and $Q=\lambda x. \text{True}$]

by auto

lemma limpt_of_limpts: $x \text{ islimpt } \{y. y \text{ islimpt } S\} \implies x \text{ islimpt } S$

for $x :: 'a::\text{metric\_space}$

apply (clarsimp simp add: islimpt_approachable)

apply (drule_tac $x=e/2$ in spec)

apply (auto simp: simp del: less_divide_eq numeral1)

apply (drule_tac $x=\text{dist } x' x$ in spec)

apply (auto simp del: less_divide_eq numeral1)

apply metric

done

lemma closed_limpts: $\text{closed } \{x::'a::\text{metric\_space}. x \text{ islimpt } S\}$

using closed_limpt limpt_of_limpts by blast

lemma limpt_of_closure: $x \text{ islimpt closure } S \longleftrightarrow x \text{ islimpt } S$

for $x :: 'a::\text{metric\_space}$

by (auto simp: closure_def islimpt_Un dest: limpt_of_limpts)

lemma islimpt_eq_infinite_ball: $x \text{ islimpt } S \longleftrightarrow (\forall e>0. \text{infinite}(S \cap \text{ball } x e))$

apply (simp add: islimpt_eq_acc_point, safe)

apply (metis Int_commute open_ball centre_in_ball)

by (metis open_contains_ball Int_mono finite_subset inf_commute subset_refl)

```

lemma islimpt_eq_infinite_cball:  $x \text{ islimpt } S \longleftrightarrow (\forall e > 0. \text{infinite}(S \cap \text{cball } x \ e))$ 
  apply (simp add: islimpt_eq_infinite_ball, safe)
  apply (meson Int_mono ball_subset_cball finite_subset order_refl)
  by (metis open_ball centre_in_ball finite_Int inf.absorb_iff2 inf_assoc open_contains_cball_eq)

```

3.2.3 Perfect Metric Spaces

```

lemma perfect_choose_dist:  $0 < r \implies \exists a. a \neq x \wedge \text{dist } a \ x < r$ 
  for  $x :: 'a :: \{\text{perfect\_space}, \text{metric\_space}\}$ 
  using islimpt_UNIV [of  $x$ ] by (simp add: islimpt_approachable)

```

```

lemma cball_eq_sing:
  fixes  $x :: 'a :: \{\text{metric\_space}, \text{perfect\_space}\}$ 
  shows  $\text{cball } x \ e = \{x\} \longleftrightarrow e = 0$ 
proof (rule linorder_cases)
  assume  $e: 0 < e$ 
  obtain  $a$  where  $a \neq x \wedge \text{dist } a \ x < e$ 
  using perfect_choose_dist [OF  $e$ ] by auto
  then have  $a \neq x \wedge \text{dist } x \ a \leq e$ 
  by (auto simp: dist_commute)
  with  $e$  show ?thesis by (auto simp: set_eq_iff)
qed auto

```

3.2.4 ?

```

lemma finite_ball_include:
  fixes  $a :: 'a :: \text{metric\_space}$ 
  assumes finite S
  shows  $\exists e > 0. S \subseteq \text{ball } a \ e$ 
  using assms
proof induction
  case (insert x S)
  then obtain  $e0$  where  $e0 > 0$  and  $e0 : S \subseteq \text{ball } a \ e0$  by auto
  define  $e$  where  $e = \max e0 \ (2 * \text{dist } a \ x)$ 
  have  $e > 0$  unfolding e_def using  $\langle e0 > 0 \rangle$  by auto
  moreover have  $\text{insert } x \ S \subseteq \text{ball } a \ e$ 
  using  $e0 \ \langle e > 0 \rangle$  unfolding e_def by auto
  ultimately show ?case by auto
qed (auto intro: zero_less_one)

```

```

lemma finite_set_avoid:
  fixes  $a :: 'a :: \text{metric\_space}$ 
  assumes finite S
  shows  $\exists d > 0. \forall x \in S. x \neq a \longrightarrow d \leq \text{dist } a \ x$ 
  using assms
proof induction
  case (insert x S)
  then obtain  $d$  where  $d > 0$  and  $d: \forall x \in S. x \neq a \longrightarrow d \leq \text{dist } a \ x$ 
  by blast

```

```

show ?case
proof (cases x = a)
  case True
  with ‹d > 0› show ?thesis by auto
next
  case False
  let ?d = min d (dist a x)
  from False ‹d > 0› have dp: ?d > 0
  by auto
  from d have d':  $\forall x \in S. x \neq a \longrightarrow ?d \leq \text{dist } a \ x$ 
  by auto
  with dp False show ?thesis
  by (metis insert_iff le_less min_less_iff_conj not_less)
qed
qed (auto intro: zero_less_one)

lemma discrete_imp_closed:
  fixes S :: 'a::metric_space set
  assumes e: 0 < e
  and d:  $\forall x \in S. \forall y \in S. \text{dist } y \ x < e \longrightarrow y = x$ 
  shows closed S
proof -
  have False if C:  $\bigwedge e. e > 0 \implies \exists x' \in S. x' \neq x \wedge \text{dist } x' \ x < e$  for x
  proof -
    from e have e2: e/2 > 0 by arith
    from C[rule_format, OF e2] obtain y where y: y ∈ S y ≠ x dist y x < e/2
    by blast
    from e2 y(2) have mp: min (e/2) (dist x y) > 0
    by simp
    from d y C[OF mp] show ?thesis
    by metric
  qed
  then show ?thesis
  by (metis islimpt_approachable closed_limpt [where 'a='a])
qed

```

3.2.5 Interior

```

lemma mem_interior:  $x \in \text{interior } S \longleftrightarrow (\exists e > 0. \text{ball } x \ e \subseteq S)$ 
  using open_contains_ball_eq [where S=interior S]
  by (simp add: open_subset_interior)

lemma mem_interior_cball:  $x \in \text{interior } S \longleftrightarrow (\exists e > 0. \text{cball } x \ e \subseteq S)$ 
  by (meson ball_subset_cball interior_subset mem_interior open_contains_cball
    open_interior subset_trans)

```

3.2.6 Frontier

```

lemma frontier_straddle:

```

```

fixes a :: 'a::metric_space
shows a ∈ frontier S ⟷ (∀ e>0. (∃ x∈S. dist a x < e) ∧ (∃ x. x ∉ S ∧ dist a
x < e))
unfolding frontier_def closure_interior
by (auto simp: mem_interior subset_eq ball_def)

```

3.2.7 Limits

proposition *Lim*: $(f \longrightarrow l) \text{ net} \longleftrightarrow \text{trivial\_limit net} \vee (\forall e>0. \text{eventually } (\lambda x. \text{dist } (f x) l < e) \text{ net})$

by (auto simp: tendsto_iff trivial_limit_eq)

Show that they yield usual definitions in the various cases.

proposition *Lim_within_le*: $(f \longrightarrow l) (\text{at } a \text{ within } S) \longleftrightarrow (\forall e>0. \exists d>0. \forall x \in S. 0 < \text{dist } x a \wedge \text{dist } x a \leq d \longrightarrow \text{dist } (f x) l < e)$

by (auto simp: tendsto_iff eventually_at_le)

proposition *Lim_within*: $(f \longrightarrow l) (\text{at } a \text{ within } S) \longleftrightarrow (\forall e>0. \exists d>0. \forall x \in S. 0 < \text{dist } x a \wedge \text{dist } x a < d \longrightarrow \text{dist } (f x) l < e)$

by (auto simp: tendsto_iff eventually_at)

corollary *Lim_withinI* [intro?]:

```

assumes ⋀e. e > 0 ⟹ ∃ d>0. ∀ x ∈ S. 0 < dist x a ∧ dist x a < d ⟹ dist
(f x) l ≤ e
shows (f ⟶ l) (at a within S)
apply (simp add: Lim_within, clarify)
apply (rule ex_forward [OF assms [OF half_gt_zero]], auto)
done

```

proposition *Lim_at*: $(f \longrightarrow l) (\text{at } a) \longleftrightarrow (\forall e>0. \exists d>0. \forall x. 0 < \text{dist } x a \wedge \text{dist } x a < d \longrightarrow \text{dist } (f x) l < e)$

by (auto simp: tendsto_iff eventually_at)

lemma *Lim_transform_within_set*:

```

fixes a :: 'a::metric_space and l :: 'b::metric_space
shows [(f ⟶ l) (at a within S); eventually (λx. x ∈ S ⟷ x ∈ T) (at a)]
  ⟹ (f ⟶ l) (at a within T)
apply (clarsimp simp: eventually_at Lim_within)
apply (drule_tac x=e in spec, clarify)
apply (rename_tac k)
apply (rule_tac x=min d k in exI, simp)
done

```

Another limit point characterization.

lemma *limpt_sequential_inj*:

```

fixes x :: 'a::metric_space
shows x islimpt S ⟷
  (∃ f. (∀ n::nat. f n ∈ S - {x}) ∧ inj f ∧ (f ⟶ x) sequentially)
(is ?lhs = ?rhs)

```

```

proof
  assume ?lhs
  then have  $\forall e > 0. \exists x' \in S. x' \neq x \wedge \text{dist } x' x < e$ 
    by (force simp: islimpt_approachable)
  then obtain y where  $y: \bigwedge e. e > 0 \implies y \in S \wedge y \neq x \wedge \text{dist } (y \ e) x < e$ 
    by metis
  define f where  $f \equiv \text{rec\_nat } (y \ 1) (\lambda n \text{ fn. } y \ (\min (\text{inverse}(2 \wedge (\text{Suc } n))) (\text{dist } \text{fn } x)))$ 
  have [simp]:  $f \ 0 = y \ 1$ 
    by (simp)
  have  $f(\text{Suc } n) = y \ (\min (\text{inverse}(2 \wedge (\text{Suc } n))) (\text{dist } (f \ n) x))$  for n
    by (simp_all add: f_def)
  have  $f: f \ n \in S \wedge (f \ n \neq x) \wedge \text{dist } (f \ n) x < \text{inverse}(2 \wedge n)$  for n
  proof (induction n)
    case 0 show ?case
      by (simp add: y)
    next
      case (Suc n) then show ?case
        apply (auto simp: y)
        by (metis half_gt_zero_iff inverse_positive_iff positive_less_divide_eq numeral1(1)
          min_less_iff_conj y zero_less_dist_iff zero_less_numeral zero_less_power)
  qed
  show ?rhs
  proof (rule_tac x=f in exI, intro conjI allI)
    show  $\bigwedge n. f \ n \in S - \{x\}$ 
      using f by blast
    have  $\text{dist } (f \ n) x < \text{dist } (f \ m) x$  if  $m < n$  for m n
      using that
    proof (induction n)
      case 0 then show ?case by simp
    next
      case (Suc n)
      then consider  $m < n \mid m = n$  using less_Suc_eq by blast
      then show ?case
        proof cases
          assume  $m < n$ 
          have  $\text{dist } (f(\text{Suc } n)) x = \text{dist } (y \ (\min (\text{inverse}(2 \wedge (\text{Suc } n))) (\text{dist } (f \ n) x)))$ 
            x
            by simp
          also have  $\dots < \text{dist } (f \ n) x$ 
            by (metis dist_pos_lt f min.strict_order_iff min_less_iff_conj y)
          also have  $\dots < \text{dist } (f \ m) x$ 
            using Suc.IH  $\langle m < n \rangle$  by blast
          finally show ?thesis .
        next
          assume  $m = n$  then show ?case
            by simp (metis dist_pos_lt f half_gt_zero_iff inverse_positive_iff positive
              min_less_iff_conj y zero_less_numeral zero_less_power)
        qed
    qed
  qed

```

```

then show inj f
  by (metis less_irrefl linorder_injI)
show f  $\longrightarrow$  x
  apply (rule tendstoI)
  apply (rule_tac c=nat (ceiling(1/e)) in eventually_sequentiallyI)
  apply (rule less_trans [OF f [THEN conjunct2, THEN conjunct2]])
  apply (simp add: field_simps)
  by (meson le_less_trans mult_less_cancel_left not_le of_nat_less_two_power)
qed
next
  assume ?rhs
  then show ?lhs
    by (fastforce simp add: islimpt_approachable lim_sequentially)
qed

```

```

lemma Lim_dist_ubound:
  assumes  $\neg$ (trivial_limit net)
  and  $f \longrightarrow l$  net
  and eventually  $(\lambda x. \text{dist } a (f x) \leq e)$  net
  shows  $\text{dist } a l \leq e$ 
  using assms by (fast intro: tendsto_le tendsto_intros)

```

3.2.8 Continuity

Derive the epsilon-delta forms, which we often use as "definitions"

```

proposition continuous_within_eps_delta:
  continuous (at x within s) f  $\longleftrightarrow$   $(\forall e > 0. \exists d > 0. \forall x' \in s. \text{dist } x' x < d \longrightarrow \text{dist } (f x') (f x) < e)$ 
  unfolding continuous_within and Lim_within by fastforce

```

```

corollary continuous_at_eps_delta:
  continuous (at x) f  $\longleftrightarrow$   $(\forall e > 0. \exists d > 0. \forall x'. \text{dist } x' x < d \longrightarrow \text{dist } (f x') (f x) < e)$ 
  using continuous_within_eps_delta [of x UNIV f] by simp

```

```

lemma continuous_at_right_real_increasing:
  fixes f :: real  $\Rightarrow$  real
  assumes nondecF:  $\bigwedge x y. x \leq y \implies f x \leq f y$ 
  shows continuous (at_right a) f  $\longleftrightarrow$   $(\forall e > 0. \exists d > 0. f (a + d) - f a < e)$ 
  apply (simp add: greaterThan_def dist_real_def continuous_within Lim_within_le)
  apply (intro all_cong ex_cong, safe)
  apply (erule_tac x=a + d in allE, simp)
  apply (simp add: nondecF field_simps)
  apply (drule nondecF, simp)
  done

```

```

lemma continuous_at_left_real_increasing:
  assumes nondecF:  $\bigwedge x y. x \leq y \implies f x \leq (f y) :: \text{real}$ 
  shows (continuous (at_left (a :: real)) f) =  $(\forall e > 0. \exists \text{delta} > 0. f a - f (a - \text{delta}) < e)$ 

```

```

delta) < e)
apply (simp add: lessThan_def dist_real_def continuous_within Lim_within_le)
apply (intro all_cong ex_cong, safe)
apply (erule_tac x=a - d in allE, simp)
apply (simp add: nondecF field_simps)
apply (cut_tac x=a - d and y=x in nondecF, simp_all)
done

```

Versions in terms of open balls.

```

lemma continuous_within_ball:
  continuous (at x within s) f  $\longleftrightarrow$ 
    ( $\forall e > 0. \exists d > 0. f \text{' } (ball\ x\ d \cap s) \subseteq ball\ (f\ x)\ e$ )
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  {
    fix e :: real
    assume e > 0
    then obtain d where d: d>0  $\forall xa \in s. 0 < dist\ xa\ x \wedge dist\ xa\ x < d \longrightarrow dist$ 
    (f xa) (f x) < e
    using <?lhs>[unfolded continuous_within Lim_within] by auto
    {
      fix y
      assume y  $\in f \text{' } (ball\ x\ d \cap s)$ 
      then have y  $\in ball\ (f\ x)\ e$ 
      using d(2)
      using <e > 0>
      by (auto simp: dist_commute)
    }
    then have  $\exists d > 0. f \text{' } (ball\ x\ d \cap s) \subseteq ball\ (f\ x)\ e$ 
    using <d > 0>
    unfolding subset_eq ball_def by (auto simp: dist_commute)
  }
  then show ?rhs by auto
next
  assume ?rhs
  then show ?lhs
    unfolding continuous_within Lim_within ball_def subset_eq
    apply (auto simp: dist_commute)
    apply (erule_tac x=e in allE, auto)
  done
qed

```

```

lemma continuous_at_ball:
  continuous (at x) f  $\longleftrightarrow$  ( $\forall e > 0. \exists d > 0. f \text{' } (ball\ x\ d) \subseteq ball\ (f\ x)\ e$ ) (is ?lhs =
  ?rhs)
proof
  assume ?lhs
  then show ?rhs

```

```

unfolding continuous_at Lim_at subset_eq Ball_def Bex_def image_iff mem_ball
  by (metis dist_commute dist_pos_lt dist_self)
next
  assume ?rhs
  then show ?lhs
  unfolding continuous_at Lim_at subset_eq Ball_def Bex_def image_iff mem_ball
  by (metis dist_commute)
qed

```

Define setwise continuity in terms of limits within the set.

```

lemma continuous_on_iff:
  continuous_on s f  $\longleftrightarrow$ 
  ( $\forall x \in s. \forall e > 0. \exists d > 0. \forall x' \in s. \text{dist } x' x < d \longrightarrow \text{dist } (f x') (f x) < e$ )
unfolding continuous_on_def Lim_within
by (metis dist_pos_lt dist_self)

lemma continuous_within_E:
  assumes continuous (at x within s) f e > 0
  obtains d where d > 0  $\wedge x'. \llbracket x' \in s; \text{dist } x' x \leq d \rrbracket \implies \text{dist } (f x') (f x) < e$ 
using assms apply (simp add: continuous_within_eps_delta)
apply (drule spec [of _ e], clarify)
apply (rule_tac d=d/2 in that, auto)
done

```

```

lemma continuous_onI [intro?]:
  assumes  $\bigwedge x e. \llbracket e > 0; x \in s \rrbracket \implies \exists d > 0. \forall x' \in s. \text{dist } x' x < d \longrightarrow \text{dist } (f x') (f x) \leq e$ 
  shows continuous_on s f
apply (simp add: continuous_on_iff, clarify)
apply (rule ex_forward [OF assms [OF half_gt_zero]], auto)
done

```

Some simple consequential lemmas.

```

lemma continuous_onE:
  assumes continuous_on s f x ∈ s e > 0
  obtains d where d > 0  $\wedge x'. \llbracket x' \in s; \text{dist } x' x \leq d \rrbracket \implies \text{dist } (f x') (f x) < e$ 
using assms
apply (simp add: continuous_on_iff)
apply (elim ballE allE)
apply (auto intro: that [where d=d/2 for d])
done

```

The usual transformation theorems.

```

lemma continuous_transform_within:
  fixes f g :: 'a::metric_space  $\Rightarrow$  'b::topological_space
  assumes continuous (at x within s) f
  and 0 < d
  and x ∈ s
  and  $\bigwedge x'. \llbracket x' \in s; \text{dist } x' x < d \rrbracket \implies f x' = g x'$ 

```



```

shows continuous (at x within s) g
using assms
unfolding continuous_within
by (force intro: Lim_transform_within)

```

3.2.9 Closure and Limit Characterization

```

lemma closure_approachable:
  fixes S :: 'a::metric_space set
  shows  $x \in \text{closure } S \longleftrightarrow (\forall e>0. \exists y \in S. \text{dist } y \ x < e)$ 
  apply (auto simp: closure_def islimpt_approachable)
  apply (metis dist_self)
  done

```

```

lemma closure_approachable_le:
  fixes S :: 'a::metric_space set
  shows  $x \in \text{closure } S \longleftrightarrow (\forall e>0. \exists y \in S. \text{dist } y \ x \leq e)$ 
  unfolding closure_approachable
  using dense by force

```

```

lemma closure_approachableD:
  assumes  $x \in \text{closure } S$   $e>0$ 
  shows  $\exists y \in S. \text{dist } x \ y < e$ 
  using assms unfolding closure_approachable by (auto simp: dist_commute)

```

```

lemma closed_approachable:
  fixes S :: 'a::metric_space set
  shows  $\text{closed } S \implies (\forall e>0. \exists y \in S. \text{dist } y \ x < e) \longleftrightarrow x \in S$ 
  by (metis closure_closed closure_approachable)

```

```

lemma closure_contains_Inf:
  fixes S :: real set
  assumes  $S \neq \{\}$  bdd_below S
  shows  $\text{Inf } S \in \text{closure } S$ 

```

```

proof -
  have *:  $\forall x \in S. \text{Inf } S \leq x$ 
    using cInf_lower[of _ S] assms by metis
  {
    fix e :: real
    assume  $e > 0$ 
    then have  $\text{Inf } S < \text{Inf } S + e$  by simp
    with assms obtain x where  $x \in S$   $x < \text{Inf } S + e$ 
      by (subst (asm) cInf_less_iff) auto
    with * have  $\exists x \in S. \text{dist } x (\text{Inf } S) < e$ 
      by (intro bexI[of _ x]) (auto simp: dist_real_def)
  }
  then show ?thesis unfolding closure_approachable by auto
qed

```

```

lemma closure_contains_Sup:
  fixes S :: real set
  assumes S ≠ {} bdd_above S
  shows Sup S ∈ closure S
proof -
  have *: ∀ x ∈ S. x ≤ Sup S
    using cSup_upper[of _ S] assms by metis
  {
    fix e :: real
    assume e > 0
    then have Sup S - e < Sup S by simp
    with assms obtain x where x ∈ S Sup S - e < x
      by (subst (asm) less_cSup_iff) auto
    with * have ∃ x ∈ S. dist x (Sup S) < e
      by (intro bexI[of _ x]) (auto simp: dist_real_def)
  }
  then show ?thesis unfolding closure_approachable by auto
qed

lemma not_trivial_limit_within_ball:
  ¬ trivial_limit (at x within S) ⟷ (∀ e > 0. S ∩ ball x e - {x} ≠ {})
  (is ?lhs ⟷ ?rhs)
proof
  show ?rhs if ?lhs
  proof -
    {
      fix e :: real
      assume e > 0
      then obtain y where y ∈ S - {x} and dist y x < e
        using ‹?lhs› not_trivial_limit_within[of x S] closure_approachable[of x S
- {x}]
        by auto
      then have y ∈ S ∩ ball x e - {x}
        unfolding ball_def by (simp add: dist_commute)
      then have S ∩ ball x e - {x} ≠ {} by blast
    }
    then show ?thesis by auto
  qed
  show ?lhs if ?rhs
  proof -
    {
      fix e :: real
      assume e > 0
      then obtain y where y ∈ S ∩ ball x e - {x}
        using ‹?rhs› by blast
      then have y ∈ S - {x} and dist y x < e
        unfolding ball_def by (simp_all add: dist_commute)
      then have ∃ y ∈ S - {x}. dist y x < e
        by auto
    }
  qed

```

```

    }
  then show ?thesis
    using not_trivial_limit_within[of x S] closure_approachable[of x S - {x}]
    by auto
qed
qed

```

3.2.10 Boundedness

definition (in *metric_space*) *bounded* :: 'a set $\Rightarrow$ bool
 where *bounded* $S \longleftrightarrow (\exists x \in S. \forall y \in S. \text{dist } x \ y \leq e)$

lemma *bounded_subset_cball*: *bounded* $S \longleftrightarrow (\exists e \ x. S \subseteq \text{cball } x \ e \wedge 0 \leq e)$
unfolding *bounded_def subset_eq* **by** auto (meson order_trans zero_le_dist)

lemma *bounded_any_center*: *bounded* $S \longleftrightarrow (\exists e. \forall y \in S. \text{dist } a \ y \leq e)$
unfolding *bounded_def*
by auto (metis add_commute add_le_cancel_right dist_commute dist_triangle_le)

lemma *bounded_iff*: *bounded* $S \longleftrightarrow (\exists a. \forall x \in S. \text{norm } x \leq a)$
unfolding *bounded_any_center* [where $a=0$]
by (simp add: dist_norm)

lemma *bdd_above_norm*: *bdd_above* (*norm* ' X) $\longleftrightarrow$ *bounded* X
by (simp add: bounded_iff bdd_above_def)

lemma *bounded_norm_comp*: *bounded* $((\lambda x. \text{norm } (f \ x)) ' S) = \text{bounded } (f ' S)$
by (simp add: bounded_iff)

lemma *boundedI*:
assumes $\bigwedge x. x \in S \implies \text{norm } x \leq B$
shows *bounded* S
using *assms bounded_iff* **by** blast

lemma *bounded_empty* [simp]: *bounded* $\{\}$
by (simp add: bounded_def)

lemma *bounded_subset*: *bounded* $T \implies S \subseteq T \implies \text{bounded } S$
by (metis bounded_def subset_eq)

lemma *bounded_interior* [intro]: *bounded* $S \implies \text{bounded}(\text{interior } S)$
by (metis bounded_subset interior_subset)

lemma *bounded_closure* [intro]:
assumes *bounded* S
shows *bounded* (*closure* S)

proof –
from *assms* **obtain** x **and** a **where** $a: \forall y \in S. \text{dist } x \ y \leq a$
unfolding *bounded_def* **by** auto

```

{
  fix y
  assume  $y \in \text{closure } S$ 
  then obtain  $f$  where  $f: \forall n. f\ n \in S \ (f \longrightarrow y)$  sequentially
    unfolding closure_sequential by auto
  have  $\forall n. f\ n \in S \longrightarrow \text{dist } x\ (f\ n) \leq a$  using  $a$  by simp
  then have eventually  $(\lambda n. \text{dist } x\ (f\ n) \leq a)$  sequentially
    by (simp add: f(1))
  then have  $\text{dist } x\ y \leq a$ 
    using Lim_dist_ubound f(2) trivial_limit_sequentially by blast
}
then show ?thesis
  unfolding bounded_def by auto
qed

lemma bounded_closure_image:  $\text{bounded } (f \text{ ` closure } S) \implies \text{bounded } (f \text{ ` } S)$ 
  by (simp add: bounded_subset closure_subset image_mono)

lemma bounded_cball[simp,intro]:  $\text{bounded } (\text{cball } x\ e)$ 
  unfolding bounded_def using mem_cball by blast

lemma bounded_ball[simp,intro]:  $\text{bounded } (\text{ball } x\ e)$ 
  by (metis ball_subset_cball bounded_cball bounded_subset)

lemma bounded_Un[simp]:  $\text{bounded } (S \cup T) \longleftrightarrow \text{bounded } S \wedge \text{bounded } T$ 
  by (auto simp: bounded_def) (metis Un_iff bounded_any_center le_max_iff_disj)

lemma bounded_Union[intro]:  $\text{finite } F \implies \forall S \in F. \text{bounded } S \implies \text{bounded } (\bigcup F)$ 
  by (induct rule: finite_induct[of F]) auto

lemma bounded_UN [intro]:  $\text{finite } A \implies \forall x \in A. \text{bounded } (B\ x) \implies \text{bounded } (\bigcup_{x \in A} B\ x)$ 
  by auto

lemma bounded_insert [simp]:  $\text{bounded } (\text{insert } x\ S) \longleftrightarrow \text{bounded } S$ 
proof -
  have  $\forall y \in \{x\}. \text{dist } x\ y \leq 0$ 
    by simp
  then have  $\text{bounded } \{x\}$ 
    unfolding bounded_def by fast
  then show ?thesis
    by (metis insert_is_Un bounded_Un)
qed

lemma bounded_subset_ballI:  $S \subseteq \text{ball } x\ r \implies \text{bounded } S$ 
  by (meson bounded_ball bounded_subset)

lemma bounded_subset_ballD:
  assumes  $\text{bounded } S$  shows  $\exists r. 0 < r \wedge S \subseteq \text{ball } x\ r$ 

```

```

proof -
  obtain  $e::\text{real}$  and  $y$  where  $S \subseteq \text{cball } y \ e$  and  $0 \leq e$ 
  using assms by (auto simp: bounded_subset_cball)
  then show ?thesis
  by (intro exI [where  $x = \text{dist } x \ y + e + 1$ ]) metric
qed

lemma finite_imp_bounded [intro]: finite  $S \implies$  bounded  $S$ 
  by (induct set: finite) simp_all

lemma bounded_Int [intro]: bounded  $S \vee$  bounded  $T \implies$  bounded  $(S \cap T)$ 
  by (metis Int_lower1 Int_lower2 bounded_subset)

lemma bounded_diff [intro]: bounded  $S \implies$  bounded  $(S - T)$ 
  by (metis Diff_subset bounded_subset)

lemma bounded_dist_comp:
  assumes bounded  $(f \text{ ' } S)$  bounded  $(g \text{ ' } S)$ 
  shows bounded  $((\lambda x. \text{dist } (f \ x) \ (g \ x)) \text{ ' } S)$ 
proof -
  from assms obtain  $M1 \ M2$  where  $\ast: \text{dist } (f \ x) \ \text{undefined} \leq M1$  and  $\text{dist } (g \ x) \ \text{undefined} \leq M2$  if  $x \in S$  for  $x$ 
  by (auto simp: bounded_any_center [of _ undefined] dist_commute)
  have  $\text{dist } (f \ x) \ (g \ x) \leq M1 + M2$  if  $x \in S$  for  $x$ 
  using  $\ast$  [OF that]
  by metric
  then show ?thesis
  by (auto intro!: boundedI)
qed

lemma bounded_Times:
  assumes bounded  $s$  bounded  $t$ 
  shows bounded  $(s \times t)$ 
proof -
  obtain  $x \ y \ a \ b$  where  $\forall z \in s. \text{dist } x \ z \leq a$  and  $\forall z \in t. \text{dist } y \ z \leq b$ 
  using assms [unfolded bounded_def] by auto
  then have  $\forall z \in s \times t. \text{dist } (x, y) \ z \leq \text{sqrt } (a^2 + b^2)$ 
  by (auto simp: dist_Pair_Pair real_sqrt_le_mono add_mono power_mono)
  then show ?thesis unfolding bounded_any_center [where  $a = (x, y)$ ] by auto
qed

```

3.2.11 Compactness

```

lemma compact_imp_bounded:
  assumes compact  $U$ 
  shows bounded  $U$ 
proof -
  have compact  $U \ \forall x \in U. \text{open } (\text{ball } x \ 1) \implies U \subseteq \bigcup_{x \in U} \text{ball } x \ 1$ 
  using assms by auto

```

```

then obtain D where D: D ⊆ U finite D U ⊆ (⋃ x∈D. ball x 1)
  by (metis compactE_image)
from ⟨finite D⟩ have bounded (⋃ x∈D. ball x 1)
  by (simp add: bounded_UN)
then show bounded U using ⟨U ⊆ (⋃ x∈D. ball x 1)⟩
  by (rule bounded_subset)
qed

```

```

lemma closure_Int_ball_not_empty:
  assumes S ⊆ closure T x ∈ S r > 0
  shows T ∩ ball x r ≠ {}
  using assms centre_in_ball closure_iff_nhds_not_empty by blast

```

```

lemma compact_sup_maxdistance:
  fixes S :: 'a::metric_space set
  assumes compact S
  and S ≠ {}
  shows ∃ x∈S. ∃ y∈S. ∀ u∈S. ∀ v∈S. dist u v ≤ dist x y
proof -
  have compact (S × S)
    using ⟨compact S⟩ by (intro compact_Times)
  moreover have S × S ≠ {}
    using ⟨S ≠ {}⟩ by auto
  moreover have continuous_on (S × S) (λx. dist (fst x) (snd x))
    by (intro continuous_at_imp_continuous_on ballI continuous_intros)
  ultimately show ?thesis
    using continuous_attains_sup[of S × S λx. dist (fst x) (snd x)] by auto
qed

```

Totally bounded

```

lemma cauchy_def: Cauchy S ⟷ (∀ e>0. ∃ N. ∀ m n. m ≥ N ∧ n ≥ N ⟶ dist
(S m) (S n) < e)
  unfolding Cauchy_def by metis

```

```

proposition seq_compact_imp_totally_bounded:
  assumes seq_compact S
  shows ∀ e>0. ∃ k. finite k ∧ k ⊆ S ∧ S ⊆ (⋃ x∈k. ball x e)
proof -
  { fix e::real assume e > 0 assume *: ⋀ k. finite k ⟹ k ⊆ S ⟹ ¬ S ⊆ (⋃ x∈k.
ball x e)
    let ?Q = λx n r. r ∈ S ∧ (∀ m < (n::nat). ¬ (dist (x m) r < e))
    have ∃ x. ∀ n::nat. ?Q x n (x n)
    proof (rule dependent_wellorder_choice)
      fix n x assume ⋀ y. y < n ⟹ ?Q x y (x y)
      then have ¬ S ⊆ (⋃ x∈x ' {0..<n}. ball x e)
        using *[of x ' {0..<n}] by (auto simp: subset_eq)
      then obtain z where z: z∈S z ∉ (⋃ x∈x ' {0..<n}. ball x e)
        unfolding subset_eq by auto

```

```

    show  $\exists r. ?Q\ x\ n\ r$ 
    using  $z$  by auto
  qed simp
  then obtain  $x$  where  $\forall n::nat. x\ n \in S$  and  $x:\bigwedge n\ m. m < n \implies \neg (dist\ (x\ m)\ (x\ n) < e)$ 
    by blast
  then obtain  $l\ r$  where  $l \in S$  and  $r:strict\_mono\ r$  and  $((x \circ r) \longrightarrow l)$ 
    sequentially
  using  $assms$  by (metis  $seq\_compact\_def$ )
  then have  $Cauchy\ (x \circ r)$ 
    using  $LIMSEQ\_imp\_Cauchy$  by auto
  then obtain  $N::nat$  where  $\bigwedge m\ n. N \leq m \implies N \leq n \implies dist\ ((x \circ r)\ m)\ ((x \circ r)\ n) < e$ 
    unfolding  $cauchy\_def$  using  $\langle e > 0 \rangle$  by blast
  then have  $False$ 
    using  $x[of\ r\ N\ r\ (N+1)]\ r$  by (auto simp:  $strict\_mono\_def$ ) }
  then show  $?thesis$ 
    by metis
qed

```

Heine-Borel theorem

proposition $seq\_compact\_imp\_Heine\_Borel$:

fixes $S :: 'a :: metric\_space\ set$

assumes $seq\_compact\ S$

shows $compact\ S$

proof –

from $seq\_compact\_imp\_totally\_bounded[OF\ \langle seq\_compact\ S \rangle]$

obtain f where $f: \forall e>0. finite\ (f\ e) \wedge f\ e \subseteq S \wedge S \subseteq (\bigcup x \in f\ e. ball\ x\ e)$

unfolding $choice\_iff'$..

define K where $K = (\lambda(x, r). ball\ x\ r) \cdot ((\bigcup e \in \mathbb{Q} \cap \{0 < ..\}. f\ e) \times \mathbb{Q})$

have $countably\_compact\ S$

using $\langle seq\_compact\ S \rangle$ by (rule $seq\_compact\_imp\_countably\_compact$)

then show $compact\ S$

proof (rule $countably\_compact\_imp\_compact$)

show $countable\ K$

unfolding K_def using f

by (auto intro: $countable\_finite\ countable\_subset\ countable\_rat$
intro!: $countable\_image\ countable\_SIGMA\ countable\_UN$)

show $\forall b \in K. open\ b$ by (auto simp: K_def)

next

fix $T\ x$

assume $T: open\ T\ x \in T$ and $x: x \in S$

from $openE[OF\ T]$ **obtain** e where $0 < e$ and $ball\ x\ e \subseteq T$

by auto

then have $0 < e/2$ and $ball\ x\ (e/2) \subseteq T$

by auto

from $Rats\_dense\_in\_real[OF\ \langle 0 < e/2 \rangle]$ **obtain** r where $r \in \mathbb{Q}$ and $0 < r < e/2$

```

      by auto
    from f[rule_format, of r] ⟨0 < r⟩ ⟨x ∈ S⟩ obtain k where k ∈ f r x ∈ ball k
  r
    by auto
  from ⟨r ∈ Q⟩ ⟨0 < r⟩ ⟨k ∈ f r⟩ have ball k r ∈ K
    by (auto simp: K_def)
  then show ∃ b ∈ K. x ∈ b ∧ b ∩ S ⊆ T
  proof (rule bexI[rotated], safe)
    fix y
    assume y ∈ ball k r
    with ⟨r < e/2⟩ ⟨x ∈ ball k r⟩ have dist x y < e
      by (intro dist_triangle_half_r [of k _ e]) (auto simp: dist_commute)
    with ⟨ball x e ⊆ T⟩ show y ∈ T
      by auto
  next
    show x ∈ ball k r by fact
  qed
qed
qed

```

proposition *compact_eq_seq_compact_metric*:
 $compact (S :: 'a::metric\_space\ set) \longleftrightarrow seq\_compact\ S$
using *compact_imp_seq_compact* *seq_compact_imp_Heine_Borel* **by** *blast*

proposition *compact_def*: — this is the definition of compactness in HOL Light
 $compact (S :: 'a::metric\_space\ set) \longleftrightarrow$
 $(\forall f. (\forall n. f\ n \in S) \longrightarrow (\exists l \in S. \exists r::nat \Rightarrow nat. strict\_mono\ r \wedge (f \circ r) \longrightarrow$
 $l))$
unfolding *compact_eq_seq_compact_metric* *seq_compact_def* **by** *auto*

Complete the chain of compactness variants

proposition *compact_eq_Bolzano_Weierstrass*:
fixes $S :: 'a::metric\_space\ set$
shows $compact\ S \longleftrightarrow (\forall T. infinite\ T \wedge T \subseteq S \longrightarrow (\exists x \in S. x\ islimpt\ T))$
using *Bolzano_Weierstrass_imp_seq_compact* *Heine_Borel_imp_Bolzano_Weierstrass*
compact_eq_seq_compact_metric
by *blast*

proposition *Bolzano_Weierstrass_imp_bounded*:
 $(\bigwedge T. [infinite\ T; T \subseteq S] \Longrightarrow (\exists x \in S. x\ islimpt\ T)) \Longrightarrow bounded\ S$
using *compact_imp_bounded* **unfolding** *compact_eq_Bolzano_Weierstrass* **by**
metis

3.2.12 Banach fixed point theorem

theorem *banach_fix*:— TODO: rename to *Banach_fix*
assumes $s: complete\ s\ s \neq \{\}$
and $c: 0 \leq c < 1$
and $f: f\ 's \subseteq s$


```

    and lipschitz:  $\forall x \in s. \forall y \in s. \text{dist } (f x) (f y) \leq c * \text{dist } x y$ 
  shows  $\exists ! x \in s. f x = x$ 
proof -
  from c have  $1 - c > 0$  by simp

  from s(2) obtain z0 where z0:  $z0 \in s$  by blast
  define z where  $z n = (f \hat{\phantom{x}}^n) z0$  for n
  with f z0 have z_in_s:  $z n \in s$  for  $n :: \text{nat}$ 
    by (induct n) auto
  define d where  $d = \text{dist } (z 0) (z 1)$ 

  have fzn:  $f (z n) = z (Suc n)$  for n
    by (simp add: z_def)
  have cf_z:  $\text{dist } (z n) (z (Suc n)) \leq (c \hat{\phantom{x}}^n) * d$  for  $n :: \text{nat}$ 
  proof (induct n)
    case 0
    then show ?case
      by (simp add: d_def)
  next
    case (Suc m)
    with  $\langle 0 \leq c \rangle$  have  $c * \text{dist } (z m) (z (Suc m)) \leq c \hat{\phantom{x}}^{Suc m} * d$ 
      using mult_left_mono[of  $\text{dist } (z m) (z (Suc m))$   $c \hat{\phantom{x}}^m * d$ ] by simp
    then show ?case
      using lipschitz[THEN bspec[where  $x = z m$ ], OF z_in_s, THEN bspec[where
 $x = z (Suc m)$ ], OF z_in_s]
      by (simp add: fzn mult_le_cancel_left)
  qed

  have cf_z2:  $(1 - c) * \text{dist } (z m) (z (m + n)) \leq (c \hat{\phantom{x}}^m) * d * (1 - c \hat{\phantom{x}}^n)$  for
 $n m :: \text{nat}$ 
  proof (induct n)
    case 0
    show ?case by simp
  next
    case (Suc k)
    from c have  $(1 - c) * \text{dist } (z m) (z (m + Suc k)) \leq$ 
       $(1 - c) * (\text{dist } (z m) (z (m + k)) + \text{dist } (z (m + k)) (z (Suc (m + k))))$ 
      by (simp add: dist_triangle)
    also from c cf_z[of  $m + k$ ] have  $\dots \leq (1 - c) * (\text{dist } (z m) (z (m + k)) +$ 
 $c \hat{\phantom{x}}^{(m + k)} * d)$ 
      by simp
    also from Suc have  $\dots \leq c \hat{\phantom{x}}^m * d * (1 - c \hat{\phantom{x}}^k) + (1 - c) * c \hat{\phantom{x}}^{(m + k)}$ 
 $* d$ 
      by (simp add: field_simps)
    also have  $\dots = (c \hat{\phantom{x}}^m) * (d * (1 - c \hat{\phantom{x}}^k) + (1 - c) * c \hat{\phantom{x}}^k * d)$ 
      by (simp add: power_add field_simps)
    also from c have  $\dots \leq (c \hat{\phantom{x}}^m) * d * (1 - c \hat{\phantom{x}}^{Suc k})$ 
      by (simp add: field_simps)
    finally show ?case by simp

```

qed

```

have  $\exists N. \forall m n. N \leq m \wedge N \leq n \longrightarrow \text{dist } (z m) (z n) < e$  if  $e > 0$  for  $e$ 
proof (cases  $d = 0$ )
  case True
    from  $\langle 1 - c > 0 \rangle$  have  $(1 - c) * x \leq 0 \longleftrightarrow x \leq 0$  for  $x$ 
    by (simp add: mult_le_0_iff)
    with  $c \text{ cf\_z2[of } 0] \text{ True}$  have  $z n = z 0$  for  $n$ 
    by (simp add: z_def)
    with  $\langle e > 0 \rangle$  show ?thesis by simp
  next
    case False
    with  $\text{zero\_le\_dist[of } z 0 z 1]$  have  $d > 0$ 
    by (metis d_def less_le)
    with  $\langle 1 - c > 0 \rangle \langle e > 0 \rangle$  have  $0 < e * (1 - c) / d$ 
    by simp
    with  $c$  obtain  $N$  where  $N: c ^ N < e * (1 - c) / d$ 
    using  $\text{real\_arch\_pow\_inv[of } e * (1 - c) / d c]$  by auto
    have *:  $\text{dist } (z m) (z n) < e$  if  $m > n$  and  $as: m \geq N \ n \geq N$  for  $m n :: nat$ 
    proof -
      from  $c \langle n \geq N \rangle$  have *:  $c ^ n \leq c ^ N$ 
      using  $\text{power\_decreasing[OF } \langle n \geq N \rangle, \text{ of } c]$  by simp
      from  $c \langle m > n \rangle$  have  $1 - c ^ (m - n) > 0$ 
      using  $\text{power\_strict\_mono[of } c 1 m - n]$  by simp
      with  $\langle d > 0 \rangle \langle 0 < 1 - c \rangle$  have **:  $d * (1 - c ^ (m - n)) / (1 - c) > 0$ 
      by simp
      from  $c \text{ cf\_z2[of } n m - n] \langle m > n \rangle$ 
      have  $\text{dist } (z m) (z n) \leq c ^ n * d * (1 - c ^ (m - n)) / (1 - c)$ 
      by (simp add: pos_le_divide_eq[OF  $\langle 1 - c > 0 \rangle$ ] mult.commute dist_commute)
      also have  $\dots \leq c ^ N * d * (1 - c ^ (m - n)) / (1 - c)$ 
      using  $\text{mult\_right\_mono[OF } * \text{ order\_less\_imp\_le[OF } **]]$ 
      by (simp add: mult.assoc)
      also have  $\dots < (e * (1 - c) / d) * d * (1 - c ^ (m - n)) / (1 - c)$ 
      using  $\text{mult\_strict\_right\_mono[OF } N **]$  by (auto simp: mult.assoc)
      also from  $c \langle d > 0 \rangle \langle 1 - c > 0 \rangle$  have  $\dots = e * (1 - c ^ (m - n))$ 
      by simp
      also from  $c \langle 1 - c ^ (m - n) > 0 \rangle \langle e > 0 \rangle$  have  $\dots \leq e$ 
      using  $\text{mult\_right\_le\_one\_le[of } e 1 - c ^ (m - n)]$  by auto
      finally show ?thesis by simp
    qed
  qed
  have  $\text{dist } (z n) (z m) < e$  if  $N \leq m \ N \leq n$  for  $m n :: nat$ 
  proof (cases  $n = m$ )
    case True
      with  $\langle e > 0 \rangle$  show ?thesis by simp
    next
      case False
      with * [of  $n m$ ] * [of  $m n$ ] and that show ?thesis
      by (auto simp: dist_commute nat_neq_iff)
  qed

```

```

    then show ?thesis by auto
qed
then have Cauchy z
  by (simp add: cauchy_def)
then obtain x where x∈s and x:(z ⟶ x) sequentially
  using s(1)[unfolded compact_def complete_def, THEN spec[where x=z]] and
z_in_s by auto

define e where e = dist (f x) x
have e = 0
proof (rule ccontr)
  assume e ≠ 0
  then have e > 0
    unfolding e_def using zero_le_dist[of f x x]
    by (metis dist_eq_0_iff dist_nz e_def)
  then obtain N where N:∀ n≥N. dist (z n) x < e/2
    using x[unfolded lim_sequentially, THEN spec[where x=e/2]] by auto
  then have N':dist (z N) x < e/2 by auto
  have *: c * dist (z N) x ≤ dist (z N) x
    unfolding mult_le_cancel_right2
    using zero_le_dist[of z N x] and c
    by (metis dist_eq_0_iff dist_nz order_less_asym less_le)
  have dist (f (z N)) (f x) ≤ c * dist (z N) x
    using lipschitz[THEN bspec[where x=z N], THEN bspec[where x=x]]
    using z_in_s[of N] ⟨x∈s⟩
    using c
    by auto
  also have ... < e/2
    using N' and c using * by auto
  finally show False
    unfolding fzn
    using N[THEN spec[where x=Suc N]] and dist_triangle_half_r[of z (Suc
N) f x e x]
    unfolding e_def
    by auto
qed
then have f x = x by (auto simp: e_def)
moreover have y = x if f y = y y ∈ s for y
proof -
  from ⟨x ∈ s⟩ ⟨f x = x⟩ that have dist x y ≤ c * dist x y
    using lipschitz[THEN bspec[where x=x], THEN bspec[where x=y]] by simp
  with c and zero_le_dist[of x y] have dist x y = 0
    by (simp add: mult_le_cancel_right1)
  then show ?thesis by simp
qed
ultimately show ?thesis
  using ⟨x∈s⟩ by blast
qed

```

3.2.13 Edelstein fixed point theorem

theorem *Edelstein_fix*:

fixes $S :: 'a::\text{metric\_space set}$

assumes $S: \text{compact } S \ S \neq \{\}$

and $gs: (g \text{ ` } S) \subseteq S$

and $\text{dist}: \forall x \in S. \forall y \in S. x \neq y \longrightarrow \text{dist } (g \ x) \ (g \ y) < \text{dist } x \ y$

shows $\exists! x \in S. g \ x = x$

proof –

let $?D = (\lambda x. (x, x)) \text{ ` } S$

have $D: \text{compact } ?D \ ?D \neq \{\}$

by $(\text{rule compact\_continuous\_image})$

$(\text{auto intro!}: S \text{ continuous\_Pair continuous\_ident simp: continuous\_on\_eq\_continuous\_within})$

have $\bigwedge x \ y \ e. x \in S \implies y \in S \implies 0 < e \implies \text{dist } y \ x < e \implies \text{dist } (g \ y) \ (g \ x) < e$

using dist **by** fastforce

then have $\text{continuous\_on } S \ g$

by $(\text{auto simp: continuous\_on\_iff})$

then have $\text{cont}: \text{continuous\_on } ?D \ (\lambda x. \text{dist } ((g \circ \text{fst}) \ x) \ (\text{snd } x))$

unfolding $\text{continuous\_on\_eq\_continuous\_within}$

by $(\text{intro continuous\_dist ballI continuous\_within\_compose})$

$(\text{auto intro!}: \text{continuous\_fst continuous\_snd continuous\_ident simp: image\_image})$

obtain a **where** $a \in S$ **and** $le: \bigwedge x. x \in S \implies \text{dist } (g \ a) \ a \leq \text{dist } (g \ x) \ x$

using $\text{continuous\_attains\_inf}[OF \ D \ \text{cont}]$ **by** auto

have $g \ a = a$

proof (rule ccontr)

assume $g \ a \neq a$

with $\langle a \in S \rangle \ gs$ **have** $\text{dist } (g \ (g \ a)) \ (g \ a) < \text{dist } (g \ a) \ a$

by $(\text{intro dist}[\text{rule\_format}]) \ \text{auto}$

moreover have $\text{dist } (g \ a) \ a \leq \text{dist } (g \ (g \ a)) \ (g \ a)$

using $\langle a \in S \rangle \ gs$ **by** $(\text{intro le}) \ \text{auto}$

ultimately show False **by** auto

qed

moreover have $\bigwedge x. x \in S \implies g \ x = x \implies x = a$

using $\text{dist}[\text{THEN bspec}[\text{where } x=a]] \ \langle g \ a = a \rangle$ **and** $\langle a \in S \rangle$ **by** auto

ultimately show $\exists! x \in S. g \ x = x$

using $\langle a \in S \rangle$ **by** blast

qed

3.2.14 The diameter of a set

definition $\text{diameter} :: 'a::\text{metric\_space set} \Rightarrow \text{real}$ **where**

$\text{diameter } S = (\text{if } S = \{\} \text{ then } 0 \text{ else } \text{SUP } (x,y) \in S \times S. \text{dist } x \ y)$

lemma $\text{diameter\_empty} \ [\text{simp}]: \text{diameter } \{\} = 0$

by $(\text{auto simp: diameter\_def})$

```
lemma diameter_singleton [simp]: diameter{x} = 0
  by (auto simp: diameter_def)
```

```
lemma diameter_le:
  assumes  $S \neq \{\}$   $\vee 0 \leq d$ 
  and no:  $\bigwedge x y. \llbracket x \in S; y \in S \rrbracket \implies \text{norm}(x - y) \leq d$ 
  shows diameter  $S \leq d$ 
  using assms
  by (auto simp: dist_norm diameter_def intro: cSUP_least)
```

```
lemma diameter_bounded_bound:
  fixes  $S :: 'a :: \text{metric\_space set}$ 
  assumes  $S$ : bounded  $S$   $x \in S$   $y \in S$ 
  shows  $\text{dist } x \ y \leq \text{diameter } S$ 
proof -
  from  $S$  obtain  $z$   $d$  where  $z$ :  $\bigwedge x. x \in S \implies \text{dist } z \ x \leq d$ 
  unfolding bounded_def by auto
  have bdd_above (case_prod dist ' ( $S \times S$ ))
  proof (intro bdd_aboveI, safe)
    fix  $a$   $b$ 
    assume  $a \in S$   $b \in S$ 
    with  $z$ [of  $a$ ]  $z$ [of  $b$ ] dist_triangle[of  $a$   $b$   $z$ ]
    show  $\text{dist } a \ b \leq 2 * d$ 
    by (simp add: dist_commute)
  qed
  moreover have  $(x,y) \in S \times S$  using  $S$  by auto
  ultimately have  $\text{dist } x \ y \leq (\text{SUP } (x,y) \in S \times S. \text{dist } x \ y)$ 
  by (rule cSUP_upper2) simp
  with  $\langle x \in S \rangle$  show ?thesis
  by (auto simp: diameter_def)
qed
```

```
lemma diameter_lower_bounded:
  fixes  $S :: 'a :: \text{metric\_space set}$ 
  assumes  $S$ : bounded  $S$ 
  and  $d$ :  $0 < d$   $d < \text{diameter } S$ 
  shows  $\exists x \in S. \exists y \in S. d < \text{dist } x \ y$ 
proof (rule ccontr)
  assume contr:  $\neg ?thesis$ 
  moreover have  $S \neq \{\}$ 
  using  $d$  by (auto simp: diameter_def)
  ultimately have diameter  $S \leq d$ 
  by (auto simp: not_less diameter_def intro!: cSUP_least)
  with  $\langle d < \text{diameter } S \rangle$  show False by auto
qed
```

```
lemma diameter_bounded:
  assumes bounded  $S$ 
```

shows $\forall x \in S. \forall y \in S. \text{dist } x \ y \leq \text{diameter } S$
and $\forall d > 0. d < \text{diameter } S \longrightarrow (\exists x \in S. \exists y \in S. \text{dist } x \ y > d)$
using *diameter_bounded_bound*[of *S*] *diameter_lower_bounded*[of *S*] *assms*
by *auto*

lemma *bounded_two_points*: $\text{bounded } S \longleftrightarrow (\exists e. \forall x \in S. \forall y \in S. \text{dist } x \ y \leq e)$
by (*meson bounded_def diameter_bounded*(1))

lemma *diameter_compact_attained*:
assumes *compact S*
and $S \neq \{\}$
shows $\exists x \in S. \exists y \in S. \text{dist } x \ y = \text{diameter } S$
proof –
have *b: bounded S* **using** *assms*(1)
by (*rule compact_imp_bounded*)
then obtain *x y* **where** *xy: x ∈ S y ∈ S*
and *xy: $\forall u \in S. \forall v \in S. \text{dist } u \ v \leq \text{dist } x \ y$*
using *compact_sup_maxdistance*[OF *assms*] **by** *auto*
then have $\text{diameter } S \leq \text{dist } x \ y$
unfolding *diameter_def*
apply *clarsimp*
apply (*rule cSUP_least, fast+*)
done
then show *?thesis*
by (*metis b diameter_bounded_bound order_antisym xy*)
qed

lemma *diameter_ge_0*:
assumes *bounded S* **shows** $0 \leq \text{diameter } S$
by (*metis all_not_in_conv assms diameter_bounded_bound diameter_empty dist_self order_refl*)

lemma *diameter_subset*:
assumes $S \subseteq T$ *bounded T*
shows $\text{diameter } S \leq \text{diameter } T$
proof (*cases S = {} ∨ T = {}*)
case *True*
with *assms* **show** *?thesis*
by (*force simp: diameter_ge_0*)
next
case *False*
then have *bdd_above* $((\lambda x. \text{case } x \text{ of } (x, xa) \Rightarrow \text{dist } x \ xa) ' (T \times T))$
using $\langle \text{bounded } T \rangle \text{diameter_bounded_bound}$ **by** (*force simp: bdd_above_def*)
with *False* $\langle S \subseteq T \rangle$ **show** *?thesis*
apply (*simp add: diameter_def*)
apply (*rule cSUP_subset_mono, auto*)
done
qed

```

lemma diameter_closure:
  assumes bounded S
  shows  $\text{diameter}(\text{closure } S) = \text{diameter } S$ 
proof (rule order_antisym)
  have  $\text{False}$  if  $\text{diameter } S < \text{diameter}(\text{closure } S)$ 
  proof –
    define  $d$  where  $d = \text{diameter}(\text{closure } S) - \text{diameter}(S)$ 
    have  $d > 0$ 
    using that by (simp add: d_def)
    then have  $\text{diameter}(\text{closure}(S)) - d / 2 < \text{diameter}(\text{closure}(S))$ 
    by simp
    have  $dd: \text{diameter}(\text{closure } S) - d / 2 = (\text{diameter}(\text{closure}(S)) + \text{diameter}(S))$ 
    / 2
    by (simp add: d_def field_split_simps)
    have bocl: bounded (closure S)
    using assms by blast
    moreover have  $0 \leq \text{diameter } S$ 
    using assms diameter_ge_0 by blast
    ultimately obtain  $x\ y$  where  $x \in \text{closure } S\ y \in \text{closure } S$  and xy:  $\text{diameter}(\text{closure}(S)) - d / 2 < \text{dist } x\ y$ 
    using diameter_bounded(2) [OF bocl, rule_format, of diameter(closure(S))]
    –  $d / 2] \langle d > 0 \rangle d\_def$  by auto
    then obtain  $x'\ y'$  where  $x' y'$ :  $x' \in S\ \text{dist } x'\ x < d/4\ y' \in S\ \text{dist } y'\ y < d/4$ 
    using closure_approachable
    by (metis  $\langle 0 < d \rangle \text{zero\_less\_divide\_iff zero\_less\_numeral}$ )
    then have  $\text{dist } x'\ y' \leq \text{diameter } S$ 
    using assms diameter_bounded_bound by blast
    with  $x' y'$  have  $\text{dist } x\ y \leq d / 4 + \text{diameter } S + d / 4$ 
    by (meson add_mono_thms_linordered_semiring(1) dist_triangle dist_triangle3
    less_eq_real_def order_trans)
    then show ?thesis
    using xy d_def by linarith
  qed
  then show  $\text{diameter}(\text{closure } S) \leq \text{diameter } S$ 
  by fastforce
next
  show  $\text{diameter } S \leq \text{diameter}(\text{closure } S)$ 
  by (simp add: assms bounded_closure closure_subset diameter_subset)
qed

proposition Lebesgue_number_lemma:
  assumes compact S  $C \neq \{\}$   $S \subseteq \bigcup C$  and ope:  $\bigwedge B. B \in C \implies \text{open } B$ 
  obtains  $\delta$  where  $0 < \delta \wedge T. \llbracket T \subseteq S; \text{diameter } T < \delta \rrbracket \implies \exists B \in C. T \subseteq B$ 
proof (cases S = {})
  case True
  then show ?thesis
  by (metis  $\langle C \neq \{\} \rangle \text{zero\_less\_one empty\_subsetI equals0I subset\_trans that}$ )
next
  case False

```

```

{ fix x assume x ∈ S
  then obtain C where C: x ∈ C C ∈ C
    using ⟨S ⊆ ⋃ C⟩ by blast
  then obtain r where r: r > 0 ball x (2*r) ⊆ C
    by (metis mult.commute mult_2_right not_le ope openE field_sum_of_halves
zero_le_numeral zero_less_mult_iff)
  then have ∃ r C. r > 0 ∧ ball x (2*r) ⊆ C ∧ C ∈ C
    using C by blast
}
then obtain r where r: ∧x. x ∈ S ⇒ r x > 0 ∧ (∃ C ∈ C. ball x (2*r x) ⊆
C)
  by metis
then have S ⊆ (⋃ x ∈ S. ball x (r x))
  by auto
then obtain T where finite T S ⊆ ⋃ T and T: T ⊆ (λx. ball x (r x)) ‘ S
  by (rule compactE [OF ⟨compact S⟩]) auto
then obtain S0 where S0 ⊆ S finite S0 and S0: T = (λx. ball x (r x)) ‘ S0
  by (meson finite_subset_image)
then have S0 ≠ {}
  using False ⟨S ⊆ ⋃ T⟩ by auto
define δ where δ = Inf (r ‘ S0)
have δ > 0
  using ⟨finite S0⟩ ⟨S0 ⊆ S⟩ ⟨S0 ≠ {}⟩ r by (auto simp: δ_def finite_less_Inf_iff)
show ?thesis
proof
  show 0 < δ
    by (simp add: ⟨0 < δ⟩)
  show ∃ B ∈ C. T ⊆ B if T ⊆ S and dia: diameter T < δ for T
  proof (cases T = {})
    case True
      then show ?thesis
        using ⟨C ≠ {}⟩ by blast
    next
      case False
        then obtain y where y ∈ T by blast
        then have y ∈ S
          using ⟨T ⊆ S⟩ by auto
        then obtain x where x ∈ S0 and x: y ∈ ball x (r x)
          using ⟨S ⊆ ⋃ T⟩ S0 that by blast
        have ball y δ ⊆ ball y (r x)
          by (metis δ_def ⟨S0 ≠ {}⟩ ⟨finite S0⟩ ⟨x ∈ S0⟩ empty_is_image fi-
nite_imageI finite_less_Inf_iff imageI less_irrefl not_le subset_ball)
        also have ... ⊆ ball x (2*r x)
          using x by metric
        finally obtain C where C ∈ C ball y δ ⊆ C
          by (meson r ⟨S0 ⊆ S⟩ ⟨x ∈ S0⟩ dual_order.trans subsetCE)
        have bounded T
          using ⟨compact S⟩ bounded_subset compact_imp_bounded ⟨T ⊆ S⟩ by blast
        then have T ⊆ ball y δ

```



```

    using ⟨y ∈ T⟩ dia diameter_bounded_bound by fastforce
  then show ?thesis
    apply (rule_tac x=C in bexI)
    using ⟨ball y δ ⊆ C⟩ ⟨C ∈ C⟩ by auto
  qed
qed
qed

```

3.2.15 Metric spaces with the Heine-Borel property

A metric space (or topological vector space) is said to have the Heine-Borel property if every closed and bounded subset is compact.

```

class heine_borel = metric_space +
  assumes bounded_imp_convergent_subsequence:
    bounded (range f) ⟹ ∃ l r. strict_mono (r::nat⟹nat) ∧ ((f ∘ r) ⟶ l)
  sequentially

```

proposition *bounded_closed_imp_seq_compact*:

```

  fixes S::'a::heine_borel set
  assumes bounded S
    and closed S
  shows seq_compact S
proof (unfold seq_compact_def, clarify)
  fix f :: nat ⟹ 'a
  assume f: ∀ n. f n ∈ S
  with ⟨bounded S⟩ have bounded (range f)
    by (auto intro: bounded_subset)
  obtain l r where r: strict_mono (r :: nat ⟹ nat) and l: ((f ∘ r) ⟶ l)
  sequentially
    using bounded_imp_convergent_subsequence [OF ⟨bounded (range f)⟩] by auto
  from f have fr: ∀ n. (f ∘ r) n ∈ S
    by simp
  have l ∈ S using ⟨closed S⟩ fr l
    by (rule closed_sequentially)
  show ∃ l ∈ S. ∃ r. strict_mono r ∧ ((f ∘ r) ⟶ l) sequentially
    using ⟨l ∈ S⟩ r l by blast
qed

```

lemma *compact_eq_bounded_closed*:

```

  fixes S :: 'a::heine_borel set
  shows compact S ⟷ bounded S ∧ closed S
  using bounded_closed_imp_seq_compact compact_eq_seq_compact_metric compact_imp_bounded compact_imp_closed
  by auto

```

lemma *bounded_infinite_imp_islimpt*:

```

  fixes S :: 'a::heine_borel set
  assumes T ⊆ S bounded S infinite T
  obtains x where x islimpt S

```

by (*meson* *assms* *closed_limpt* *compact_eq_Bolzano_Weierstrass* *compact_eq_bounded_closed* *islimpt_subset*)

lemma *compact_Inter*:

fixes $\mathcal{F} :: 'a :: \text{heine\_borel set set}$

assumes *com*: $\bigwedge S. S \in \mathcal{F} \implies \text{compact } S$ **and** $\mathcal{F} \neq \{\}$

shows $\text{compact}(\bigcap \mathcal{F})$

using *assms*

by (*meson* *Inf_lower* *all_not_in_conv* *bounded_subset* *closed_Inter* *compact_eq_bounded_closed*)

lemma *compact_closure* [*simp*]:

fixes $S :: 'a :: \text{heine\_borel set}$

shows $\text{compact}(\text{closure } S) \longleftrightarrow \text{bounded } S$

by (*meson* *bounded_closure* *bounded_subset* *closed_closure* *closure_subset* *compact_eq_bounded_closed*)

instance *real* :: *heine_borel*

proof

fix $f :: \text{nat} \Rightarrow \text{real}$

assume f : *bounded* (*range* f)

obtain $r :: \text{nat} \Rightarrow \text{nat}$ **where** r : *strict_mono* r *monoseq* ($f \circ r$)

unfolding *comp_def* **by** (*metis* *seq_monosub*)

then have *Bseq* ($f \circ r$)

unfolding *Bseq_eq_bounded* **using** f

by (*metis* *BseqI'* *bounded_iff_comp_apply_rangeI*)

with r **show** $\exists l r. \text{strict\_mono } r \wedge (f \circ r) \longrightarrow l$

using *Bseq_monoseq_convergent*[*of* $f \circ r$] **by** (*auto* *simp*: *convergent_def*)

qed

lemma *compact_lemma_general*:

fixes $f :: \text{nat} \Rightarrow 'a$

fixes *proj*:: $'a \Rightarrow 'b \Rightarrow 'c :: \text{heine\_borel}$ (*infixl* *proj* 60)

fixes *unproj*:: $('b \Rightarrow 'c) \Rightarrow 'a$

assumes *finite_basis*: *finite basis*

assumes *bounded_proj*: $\bigwedge k. k \in \text{basis} \implies \text{bounded } ((\lambda x. x \text{ proj } k) \text{ ` } \text{range } f)$

assumes *proj_unproj*: $\bigwedge e k. k \in \text{basis} \implies (\text{unproj } e) \text{ proj } k = e k$

assumes *unproj_proj*: $\bigwedge x. \text{unproj } (\lambda k. x \text{ proj } k) = x$

shows $\forall d \subseteq \text{basis}. \exists l :: 'a. \exists r :: \text{nat} \Rightarrow \text{nat}.$

$\text{strict\_mono } r \wedge (\forall e > 0. \text{eventually } (\lambda n. \forall i \in d. \text{dist } (f (r n) \text{ proj } i) (l \text{ proj } i) < e) \text{ sequentially})$

proof *safe*

fix $d :: 'b \text{ set}$

assume d : $d \subseteq \text{basis}$

with *finite_basis* **have** *finite* d

by (*blast* *intro*: *finite_subset*)

from *this* **show** $\exists l :: 'a. \exists r :: \text{nat} \Rightarrow \text{nat}. \text{strict\_mono } r \wedge$

$(\forall e > 0. \text{eventually } (\lambda n. \forall i \in d. \text{dist } (f (r n) \text{ proj } i) (l \text{ proj } i) < e) \text{ sequentially})$

proof (*induct* d)

case *empty*

```

    then show ?case
      unfolding strict_mono_def by auto
  next
    case (insert k d)
    have k[intro]:  $k \in \text{basis}$ 
      using insert by auto
    have s':  $\text{bounded } ((\lambda x. x \text{ proj } k) \text{ ` } \text{range } f)$ 
      using k
      by (rule bounded_proj)
    obtain l1::'a and r1 where r1:  $\text{strict\_mono } r1$ 
      and lr1:  $\forall e > 0. \text{eventually } (\lambda n. \forall i \in d. \text{dist } (f \text{ (} r1 \text{ } n) \text{ proj } i) (l1 \text{ proj } i) < e)$ 
    sequentially
      using insert(3) using insert(4) by auto
    have f':  $\forall n. f \text{ (} r1 \text{ } n) \text{ proj } k \in (\lambda x. x \text{ proj } k) \text{ ` } \text{range } f$ 
      by simp
    have bounded (range ( $\lambda i. f \text{ (} r1 \text{ } i) \text{ proj } k$ ))
      by (metis (lifting) bounded_subset f' image_subsetI s')
    then obtain l2 r2 where r2:  $\text{strict\_mono } r2$  and lr2:  $((\lambda i. f \text{ (} r1 \text{ (} r2 \text{ } i)) \text{ proj } k) \longrightarrow l2)$  sequentially
      using bounded_imp_convergent_subsequence[of  $\lambda i. f \text{ (} r1 \text{ } i) \text{ proj } k$ ]
      by (auto simp: o_def)
    define r where  $r = r1 \circ r2$ 
    have r:  $\text{strict\_mono } r$ 
      using r1 and r2 unfolding r_def o_def strict_mono_def by auto
    moreover
      define l where  $l = \text{unproj } (\lambda i. \text{if } i = k \text{ then } l2 \text{ else } l1 \text{ proj } i)$ 
      {
        fix e::real
        assume  $e > 0$ 
        from lr1  $\langle e > 0 \rangle$  have N1:  $\text{eventually } (\lambda n. \forall i \in d. \text{dist } (f \text{ (} r1 \text{ } n) \text{ proj } i) (l1 \text{ proj } i) < e)$  sequentially
          by blast
        from lr2  $\langle e > 0 \rangle$  have N2:  $\text{eventually } (\lambda n. \text{dist } (f \text{ (} r1 \text{ (} r2 \text{ } n)) \text{ proj } k) l2 < e)$  sequentially
          by (rule tendstoD)
        from r2 N1 have N1':  $\text{eventually } (\lambda n. \forall i \in d. \text{dist } (f \text{ (} r1 \text{ (} r2 \text{ } n)) \text{ proj } i) (l1 \text{ proj } i) < e)$  sequentially
          by (rule eventually_subseq)
        have eventually  $(\lambda n. \forall i \in (\text{insert } k \text{ } d). \text{dist } (f \text{ (} r \text{ } n) \text{ proj } i) (l \text{ proj } i) < e)$ 
          sequentially
          using N1' N2
          by eventually_elim (insert insert.premis, auto simp: l_def r_def o_def proj_unproj)
      }
    ultimately show ?case by auto
  qed
qed

```

lemma $\text{bounded\_fst}: \text{bounded } s \implies \text{bounded } (\text{fst } ` s)$

```

unfolding bounded_def
by (metis (erased, opaque_lifting) dist_fst_le image_iff order_trans)

lemma bounded_snd: bounded s  $\implies$  bounded (snd ' s)
unfolding bounded_def
by (metis (no_types, opaque_lifting) dist_snd_le image_iff order_trans)

instance prod :: (heine_borel, heine_borel) heine_borel
proof
  fix f :: nat  $\Rightarrow$  'a  $\times$  'b
  assume f: bounded (range f)
  then have bounded (fst ' range f)
    by (rule bounded_fst)
  then have s1: bounded (range (fst  $\circ$  f))
    by (simp add: image_comp)
  obtain l1 r1 where r1: strict_mono r1 and l1: ( $\lambda n$ . fst (f (r1 n)))  $\longrightarrow$  l1
    using bounded_imp_convergent_subsequence [OF s1] unfolding o_def by fast
  from f have s2: bounded (range (snd  $\circ$  f  $\circ$  r1))
    by (auto simp: image_comp intro: bounded_snd bounded_subset)
  obtain l2 r2 where r2: strict_mono r2 and l2: ( $\lambda n$ . snd (f (r1 (r2 n))))  $\longrightarrow$ 
    l2) sequentially
    using bounded_imp_convergent_subsequence [OF s2]
  unfolding o_def by fast
  have l1': ( $\lambda n$ . fst (f (r1 (r2 n))))  $\longrightarrow$  l1 sequentially
    using LIMSEQ_subseq_LIMSEQ [OF l1 r2] unfolding o_def .
  have l: ((f  $\circ$  (r1  $\circ$  r2))  $\longrightarrow$  (l1, l2)) sequentially
    using tendsto_Pair [OF l1' l2] unfolding o_def by simp
  have r: strict_mono (r1  $\circ$  r2)
    using r1 r2 unfolding strict_mono_def by simp
  show  $\exists$  l r. strict_mono r  $\wedge$  ((f  $\circ$  r)  $\longrightarrow$  l) sequentially
    using l r by fast
qed

```

3.2.16 Completeness

proposition (in metric_space) completeI:
assumes $\bigwedge f. \forall n. f\ n \in s \implies \text{Cauchy } f \implies \exists l \in s. f \longrightarrow l$
shows complete s
using assms **unfolding** complete_def **by fast**

proposition (in metric_space) completeE:
assumes complete s **and** $\forall n. f\ n \in s$ **and** Cauchy f
obtains l **where** l $\in$ s **and** f $\longrightarrow$ l
using assms **unfolding** complete_def **by fast**

lemma compact_imp_complete:
fixes s :: 'a::metric_space set
assumes compact s

```

shows complete s
proof -
{
  fix f
  assume as: (∀ n::nat. f n ∈ s) Cauchy f
  from as(1) obtain l r where lr: l ∈ s strict_mono r (f ∘ r) ⟶ l
    using assms unfolding compact_def by blast

  note lr' = seq_suble [OF lr(2)]
  {
    fix e :: real
    assume e > 0
    from as(2) obtain N where N: ∀ m n. N ≤ m ∧ N ≤ n ⟶ dist (f m) (f
n) < e/2
    unfolding cauchy_def
    using ‹e > 0›
    apply (erule_tac x=e/2 in allE, auto)
    done
    from lr(3)[unfolded lim_sequentially, THEN spec[where x=e/2]]
    obtain M where M: ∀ n ≥ M. dist ((f ∘ r) n) l < e/2
    using ‹e > 0› by auto
    {
      fix n :: nat
      assume n: n ≥ max N M
      have dist ((f ∘ r) n) l < e/2
        using n M by auto
      moreover have r n ≥ N
        using lr'[of n] n by auto
      then have dist (f n) ((f ∘ r) n) < e/2
        using N and n by auto
      ultimately have dist (f n) l < e using n M
        by metric
    }
    then have ∃ N. ∀ n ≥ N. dist (f n) l < e by blast
  }
  then have ∃ l ∈ s. (f ⟶ l) sequentially using ‹l ∈ s›
    unfolding lim_sequentially by auto
}
then show ?thesis unfolding complete_def by auto
qed

proposition compact_eq_totally_bounded:
  compact s ⟷ complete s ∧ (∀ e > 0. ∃ k. finite k ∧ s ⊆ (⋃ x ∈ k. ball x e))
  (is _ ⟷ ?rhs)
proof
  assume assms: ?rhs
  then obtain k where k: ∧ e. 0 < e ⟹ finite (k e) ∧ e. 0 < e ⟹ s ⊆ (⋃ x ∈ k
e. ball x e)
    by (auto simp: choice_iff)

```

```

show compact s
proof cases
  assume s = {}
  then show compact s by (simp add: compact_def)
next
  assume s ≠ {}
  show ?thesis
    unfolding compact_def
  proof safe
    fix f :: nat ⇒ 'a
    assume f: ∀ n. f n ∈ s

    define e where e n = 1 / (2 * Suc n) for n
    then have [simp]: ∧ n. 0 < e n by auto
    define B where B n U = (SOME b. infinite {n. f n ∈ b} ∧ (∃ x. b ⊆ ball x
(e n) ∩ U)) for n U
    {
      fix n U
      assume infinite {n. f n ∈ U}
      then have ∃ b ∈ k (e n). infinite {i ∈ {n. f n ∈ U}. f i ∈ ball b (e n)}
        using k f by (intro pigeonhole_infinite_rel) (auto simp: subset_eq)
      then obtain a where
        a ∈ k (e n)
        infinite {i ∈ {n. f n ∈ U}. f i ∈ ball a (e n)} ..
      then have ∃ b. infinite {i. f i ∈ b} ∧ (∃ x. b ⊆ ball x (e n) ∩ U)
        by (intro exI[of _ ball a (e n) ∩ U] exI[of _ a]) (auto simp: ac_simps)
      from someI_ex[OF this]
      have infinite {i. f i ∈ B n U} ∃ x. B n U ⊆ ball x (e n) ∩ U
        unfolding B_def by auto
    }
    note B = this

    define F where F = rec_nat (B 0 UNIV) B
    {
      fix n
      have infinite {i. f i ∈ F n}
        by (induct n) (auto simp: F_def B)
    }
    then have F: ∧ n. ∃ x. F (Suc n) ⊆ ball x (e n) ∩ F n
      using B by (simp add: F_def)
    then have F_dec: ∧ m n. m ≤ n ⇒ F n ⊆ F m
      using decseq_SucI[of F] by (auto simp: decseq_def)

    obtain sel where sel: ∧ k i. i < sel k i ∧ k i. f (sel k i) ∈ F k
    proof (atomize_elim, unfold all_conj_distrib[symmetric], intro choice allI)
      fix k i
      have infinite ({n. f n ∈ F k} - {.. i})
        using ⟨infinite {n. f n ∈ F k}⟩ by auto

```

```

    from infinite_imp_nonempty[OF this]
  show  $\exists x > i. f x \in F k$ 
    by (simp add: set_eq_iff not_le conj_commute)
qed

define t where  $t = \text{rec\_nat } (\text{sel } 0 \ 0) \ (\lambda n \ i. \text{sel } (\text{Suc } n) \ i)$ 
have strict_mono t
  unfolding strict_mono_Suc_iff by (simp add: t_def sel)
moreover have  $\forall i. (f \circ t) \ i \in s$ 
  using f by auto
moreover
have  $t: (f \circ t) \ n \in F \ n$  for  $n$ 
  by (cases n) (simp_all add: t_def sel)

have Cauchy (f  $\circ$  t)
proof (safe intro!: metric_CauchyI exI elim!: nat_approx_posE)
  fix  $r :: \text{real}$  and  $N \ n \ m$ 
  assume  $1 / \text{Suc } N < r$   $\text{Suc } N \leq n$   $\text{Suc } N \leq m$ 
  then have  $(f \circ t) \ n \in F (\text{Suc } N)$   $(f \circ t) \ m \in F (\text{Suc } N)$   $2 * e \ N < r$ 
    using F_dec t by (auto simp: e_def field_simps)
  with F[of N] obtain  $x$  where  $\text{dist } x ((f \circ t) \ n) < e \ N$   $\text{dist } x ((f \circ t) \ m)$ 
    <  $e \ N$ 
    by (auto simp: subset_eq)
  with  $\langle 2 * e \ N < r \rangle$  show  $\text{dist } ((f \circ t) \ m) ((f \circ t) \ n) < r$ 
    by metric
qed

ultimately show  $\exists l \in s. \exists r. \text{strict\_mono } r \wedge (f \circ r) \longrightarrow l$ 
  using assms unfolding complete_def by blast
qed
qed
qed (metis compact_imp_complete compact_imp_seq_compact seq_compact_imp_totally_bounded)

lemma cauchy_imp_bounded:
  assumes Cauchy s
  shows bounded (range s)
proof -
  from assms obtain  $N :: \text{nat}$  where  $\forall m \ n. N \leq m \wedge N \leq n \longrightarrow \text{dist } (s \ m) (s \ n) < 1$ 
    unfolding cauchy_def by force
  then have  $N: \forall n. N \leq n \longrightarrow \text{dist } (s \ N) (s \ n) < 1$  by auto
  moreover
  have bounded (s '  $\{0..N\}$ )
    using finite_imp_bounded[of s '  $\{1..N\}$ ] by auto
  then obtain  $a$  where  $a: \forall x \in s ' \{0..N\}. \text{dist } (s \ N) \ x \leq a$ 
    unfolding bounded_any_center [where  $a = s \ N$ ] by auto
  ultimately show ?thesis
    unfolding bounded_any_center [where  $a = s \ N$ ]
    apply (rule_tac  $x = \max a \ 1$  in exI, auto)

```

```

    apply (erule_tac x=y in allE)
    apply (erule_tac x=y in ballE, auto)
  done
qed

```

```

instance heine_borel < complete_space
proof
  fix f :: nat  $\Rightarrow$  'a assume Cauchy f
  then have bounded (range f)
    by (rule cauchy_imp_bounded)
  then have compact (closure (range f))
    unfolding compact_eq_bounded_closed by auto
  then have complete (closure (range f))
    by (rule compact_imp_complete)
  moreover have  $\forall n. f\ n \in \text{closure (range f)}$ 
    using closure_subset [of range f] by auto
  ultimately have  $\exists l \in \text{closure (range f)}. (f \longrightarrow l)$  sequentially
    using <Cauchy f> unfolding complete_def by auto
  then show convergent f
    unfolding convergent_def by auto
qed

```

```

lemma complete_UNIV: complete (UNIV :: ('a::complete_space) set)
proof (rule completeI)
  fix f :: nat  $\Rightarrow$  'a assume Cauchy f
  then have convergent f by (rule Cauchy_convergent)
  then show  $\exists l \in \text{UNIV}. f \longrightarrow l$  unfolding convergent_def by simp
qed

```

```

lemma complete_imp_closed:
  fixes S :: 'a::metric_space set
  assumes complete S
  shows closed S
proof (unfold closed_sequential_limits, clarify)
  fix f x assume  $\forall n. f\ n \in S$  and  $f \longrightarrow x$ 
  from <f  $\longrightarrow x$ > have Cauchy f
    by (rule LIMSEQ_imp_Cauchy)
  with <complete S> and < $\forall n. f\ n \in S$ > obtain l where  $l \in S$  and  $f \longrightarrow l$ 
    by (rule completeE)
  from <f  $\longrightarrow x$ > and <f  $\longrightarrow l$ > have  $x = l$ 
    by (rule LIMSEQ_unique)
  with <l  $\in S$ > show  $x \in S$ 
    by simp
qed

```

```

lemma complete_Int_closed:
  fixes S :: 'a::metric_space set
  assumes complete S and closed t
  shows complete (S  $\cap$  t)

```



```

proof (rule completeI)
  fix  $f$  assume  $\forall n. f\ n \in S \cap t$  and  $\text{Cauchy } f$ 
  then have  $\forall n. f\ n \in S$  and  $\forall n. f\ n \in t$ 
    by simp_all
  from  $\langle \text{complete } S \rangle$  obtain  $l$  where  $l \in S$  and  $f \longrightarrow l$ 
    using  $\langle \forall n. f\ n \in S \rangle$  and  $\langle \text{Cauchy } f \rangle$  by (rule completeE)
  from  $\langle \text{closed } t \rangle$  and  $\langle \forall n. f\ n \in t \rangle$  and  $\langle f \longrightarrow l \rangle$  have  $l \in t$ 
    by (rule closed_sequentially)
  with  $\langle l \in S \rangle$  and  $\langle f \longrightarrow l \rangle$  show  $\exists l \in S \cap t. f \longrightarrow l$ 
    by fast
qed

lemma complete_closed_subset:
  fixes  $S :: 'a::\text{metric\_space}$  set
  assumes  $\text{closed } S$  and  $S \subseteq t$  and  $\text{complete } t$ 
  shows  $\text{complete } S$ 
  using assms complete_Int_closed [of  $t\ S$ ] by (simp add: Int_absorb1)

lemma complete_eq_closed:
  fixes  $S :: ('a::\text{complete\_space})$  set
  shows  $\text{complete } S \longleftrightarrow \text{closed } S$ 
proof
  assume  $\text{closed } S$  then show  $\text{complete } S$ 
    using subset_UNIV complete_UNIV by (rule complete_closed_subset)
next
  assume  $\text{complete } S$  then show  $\text{closed } S$ 
    by (rule complete_imp_closed)
qed

lemma convergent_eq_Cauchy:
  fixes  $S :: \text{nat} \Rightarrow 'a::\text{complete\_space}$ 
  shows  $(\exists l. (S \longrightarrow l) \text{ sequentially}) \longleftrightarrow \text{Cauchy } S$ 
  unfolding Cauchy_convergent_iff convergent_def ..

lemma convergent_imp_bounded:
  fixes  $S :: \text{nat} \Rightarrow 'a::\text{metric\_space}$ 
  shows  $(S \longrightarrow l) \text{ sequentially} \implies \text{bounded } (\text{range } S)$ 
  by (intro cauchy_imp_bounded LIMSEQ_imp_Cauchy)

lemma frontier_subset_compact:
  fixes  $S :: 'a::\text{heine\_borel}$  set
  shows  $\text{compact } S \implies \text{frontier } S \subseteq S$ 
  using frontier_subset_closed compact_eq_bounded_closed
  by blast

lemma continuous_closed_imp_Cauchy_continuous:
  fixes  $S :: ('a::\text{complete\_space})$  set
  shows  $\llbracket \text{continuous\_on } S\ f; \text{closed } S; \text{Cauchy } \sigma; \bigwedge n. (\sigma\ n) \in S \rrbracket \implies \text{Cauchy}(f \circ \sigma)$ 

```

```

apply (simp add: complete_eq_closed [symmetric] continuous_on_sequentially)
by (meson LIMSEQ_imp_Cauchy complete_def)

```

```

lemma banach_fix_type:
  fixes f :: 'a :: complete_space  $\Rightarrow$  'a
  assumes c:  $0 \leq c < 1$ 
    and lipschitz:  $\forall x. \forall y. \text{dist } (f\ x) (f\ y) \leq c * \text{dist } x\ y$ 
  shows  $\exists! x. (f\ x = x)$ 
  using assms banach_fix[OF complete_UNIV UNIV_not_empty assms(1,2) subset_UNIV, of f]
  by auto

```

3.2.17 Finite intersection property

Also developed in HOL's topological spaces theory, but the Heine-Borel type class isn't available there.

```

lemma closed_imp_fip:
  fixes S :: 'a :: heine_borel set
  assumes closed S
    and T:  $T \in \mathcal{F}$  bounded T
    and clof:  $\bigwedge T. T \in \mathcal{F} \Rightarrow \text{closed } T$ 
    and none:  $\bigwedge \mathcal{F}'. [\text{finite } \mathcal{F}'; \mathcal{F}' \subseteq \mathcal{F}] \Rightarrow S \cap \bigcap \mathcal{F}' \neq \{\}$ 
  shows  $S \cap \bigcap \mathcal{F} \neq \{\}$ 
proof -
  have compact (S  $\cap$  T)
    using <closed S> clof compact_eq_bounded_closed T by blast
  then have  $(S \cap T) \cap \bigcap \mathcal{F} \neq \{\}$ 
    apply (rule compact_imp_fip)
    apply (simp add: clof)
    by (metis Int_assoc complete_lattice_class.Inf_insert finite_insert insert_subset
  none <T  $\in$   $\mathcal{F}$ >)
  then show ?thesis by blast
qed

```

```

lemma closed_imp_fip_compact:
  fixes S :: 'a :: heine_borel set
  shows
     $[\text{closed } S; \bigwedge T. T \in \mathcal{F} \Rightarrow \text{compact } T;$ 
     $\bigwedge \mathcal{F}'. [\text{finite } \mathcal{F}'; \mathcal{F}' \subseteq \mathcal{F}] \Rightarrow S \cap \bigcap \mathcal{F}' \neq \{\}]$ 
     $\Rightarrow S \cap \bigcap \mathcal{F} \neq \{\}$ 
  by (metis Inf_greatest closed_imp_fip compact_eq_bounded_closed empty_subsetI
  finite.emptyI inf.orderE)

```

```

lemma closed_fip_Heine_Borel:
  fixes  $\mathcal{F}$  :: 'a :: heine_borel set set
  assumes closed S  $T \in \mathcal{F}$  bounded T
    and  $\bigwedge T. T \in \mathcal{F} \Rightarrow \text{closed } T$ 
    and  $\bigwedge \mathcal{F}'. [\text{finite } \mathcal{F}'; \mathcal{F}' \subseteq \mathcal{F}] \Rightarrow \bigcap \mathcal{F}' \neq \{\}$ 
  shows  $\bigcap \mathcal{F} \neq \{\}$ 

```

```

proof –
  have  $UNIV \cap \bigcap \mathcal{F} \neq \{\}$ 
    using assms closed_imp_fip [OF closed_UNIV] by auto
  then show ?thesis by simp
qed

lemma compact_fip_Heine_Borel:
  fixes  $\mathcal{F} :: 'a::heine\_borel\ set\ set$ 
  assumes clof:  $\bigwedge T. T \in \mathcal{F} \implies compact\ T$ 
    and none:  $\bigwedge \mathcal{F}'. \llbracket finite\ \mathcal{F}'; \mathcal{F}' \subseteq \mathcal{F} \rrbracket \implies \bigcap \mathcal{F}' \neq \{\}$ 
  shows  $\bigcap \mathcal{F} \neq \{\}$ 
by (metis InterI all_not_in_conv clof closed_fip_Heine_Borel compact_eq_bounded_closed none)

lemma compact_sequence_with_limit:
  fixes  $f :: nat \Rightarrow 'a::heine\_borel$ 
  shows  $(f \longrightarrow l)\ sequentially \implies compact\ (insert\ l\ (range\ f))$ 
apply (simp add: compact_eq_bounded_closed, auto)
apply (simp add: convergent_imp_bounded)
by (simp add: closed_limpt islimpt_insert sequence_unique_limpt)

```

3.2.18 Properties of Balls and Spheres

```

lemma compact_cball[simp]:
  fixes  $x :: 'a::heine\_borel$ 
  shows  $compact\ (cball\ x\ e)$ 
  using compact_eq_bounded_closed bounded_cball closed_cball
  by blast

lemma compact_frontier_bounded[intro]:
  fixes  $S :: 'a::heine\_borel\ set$ 
  shows  $bounded\ S \implies compact\ (frontier\ S)$ 
  unfolding frontier_def
  using compact_eq_bounded_closed
  by blast

lemma compact_frontier[intro]:
  fixes  $S :: 'a::heine\_borel\ set$ 
  shows  $compact\ S \implies compact\ (frontier\ S)$ 
  using compact_eq_bounded_closed compact_frontier_bounded
  by blast

```

3.2.19 Distance from a Set

```

lemma distance_attains_sup:
  assumes  $compact\ s\ s \neq \{\}$ 
  shows  $\exists x \in s. \forall y \in s. dist\ a\ y \leq dist\ a\ x$ 
proof (rule continuous_attains_sup [OF assms])
  {
    fix  $x$ 

```

```

    assume  $x \in s$ 
    have ( $\text{dist } a \longrightarrow \text{dist } a \ x$ ) (at  $x$  within  $s$ )
      by (intro tendsto_dist tendsto_const tendsto_ident_at)
  }
  then show continuous_on  $s$  ( $\text{dist } a$ )
    unfolding continuous_on ..
qed

```

For *minimal* distance, we only need closure, not compactness.

```

lemma distance_attains_inf:
  fixes  $a :: 'a::\text{heine\_borel}$ 
  assumes closed  $s$  and  $s \neq \{\}$ 
  obtains  $x$  where  $x \in s \wedge y. y \in s \implies \text{dist } a \ x \leq \text{dist } a \ y$ 
proof -
  from assms obtain  $b$  where  $b \in s$  by auto
  let  $?B = s \cap \text{cball } a (\text{dist } b \ a)$ 
  have  $?B \neq \{\}$  using  $\langle b \in s \rangle$ 
    by (auto simp: dist_commute)
  moreover have continuous_on  $?B$  ( $\text{dist } a$ )
    by (auto intro!: continuous_at_imp_continuous_on continuous_dist continuous_ident continuous_const)
  moreover have compact  $?B$ 
    by (intro closed_Int_compact  $\langle \text{closed } s \rangle$  compact_cball)
  ultimately obtain  $x$  where  $x \in ?B \ \forall y \in ?B. \text{dist } a \ x \leq \text{dist } a \ y$ 
    by (metis continuous_attains_inf)
  with that show ?thesis by fastforce
qed

```

3.2.20 Infimum Distance

definition $\text{infdist } x \ A = (\text{if } A = \{\} \text{ then } 0 \text{ else } \text{INF } a \in A. \text{dist } x \ a)$

```

lemma bdd_below_image_dist[intro, simp]: bdd_below ( $\text{dist } x \ ^\cdot A$ )
  by (auto intro!: zero_le_dist)

```

```

lemma infdist_notempty:  $A \neq \{\} \implies \text{infdist } x \ A = (\text{INF } a \in A. \text{dist } x \ a)$ 
  by (simp add: infdist_def)

```

```

lemma infdist_nonneg:  $0 \leq \text{infdist } x \ A$ 
  by (auto simp: infdist_def intro: cINF_greatest)

```

```

lemma infdist_le:  $a \in A \implies \text{infdist } x \ A \leq \text{dist } x \ a$ 
  by (auto intro: cINF_lower simp add: infdist_def)

```

```

lemma infdist_le2:  $a \in A \implies \text{dist } x \ a \leq d \implies \text{infdist } x \ A \leq d$ 
  by (auto intro!: cINF_lower2 simp add: infdist_def)

```

```

lemma infdist_zero[simp]:  $a \in A \implies \text{infdist } a \ A = 0$ 
  by (auto intro!: antisym infdist_nonneg infdist_le2)

```

```

lemma infdist_Un_min:
  assumes  $A \neq \{\}$   $B \neq \{\}$ 
  shows  $\text{infdist } x (A \cup B) = \min (\text{infdist } x A) (\text{infdist } x B)$ 
using assms by (simp add: infdist_def cINF_union inf_real_def)

lemma infdist_triangle:  $\text{infdist } x A \leq \text{infdist } y A + \text{dist } x y$ 
proof (cases  $A = \{\}$ )
  case True
    then show ?thesis by (simp add: infdist_def)
  next
  case False
    then obtain a where  $a \in A$  by auto
    have  $\text{infdist } x A \leq \text{Inf } \{\text{dist } x y + \text{dist } y a \mid a. a \in A\}$ 
    proof (rule cInf_greatest)
      from  $\langle A \neq \{\} \rangle$  show  $\{\text{dist } x y + \text{dist } y a \mid a. a \in A\} \neq \{\}$ 
      by simp
    fix d
    assume  $d \in \{\text{dist } x y + \text{dist } y a \mid a. a \in A\}$ 
    then obtain a where  $d = \text{dist } x y + \text{dist } y a$   $a \in A$ 
    by auto
    show  $\text{infdist } x A \leq d$ 
    unfolding infdist_notempty[OF  $\langle A \neq \{\} \rangle$ ]
    proof (rule cINF_lower2)
      show  $a \in A$  by fact
      show  $\text{dist } x a \leq d$ 
      unfolding d by (rule dist_triangle)
    qed simp
  qed
  also have  $\dots = \text{dist } x y + \text{infdist } y A$ 
  proof (rule cInf_eq, safe)
    fix a
    assume  $a \in A$ 
    then show  $\text{dist } x y + \text{infdist } y A \leq \text{dist } x y + \text{dist } y a$ 
    by (auto intro: infdist_le)
  next
  fix i
  assume inf:  $\bigwedge d. d \in \{\text{dist } x y + \text{dist } y a \mid a. a \in A\} \implies i \leq d$ 
  then have  $i - \text{dist } x y \leq \text{infdist } y A$ 
  unfolding infdist_notempty[OF  $\langle A \neq \{\} \rangle$ ] using  $\langle a \in A \rangle$ 
  by (intro cINF_greatest) (auto simp: field_simps)
  then show  $i \leq \text{dist } x y + \text{infdist } y A$ 
  by simp
  qed
  finally show ?thesis by simp
qed

lemma infdist_triangle_abs:  $|\text{infdist } x A - \text{infdist } y A| \leq \text{dist } x y$ 
by (metis (full_types) abs_diff_le_iff diff_le_eq dist_commute infdist_triangle)

```

```

lemma in_closure_iff_infdist_zero:
  assumes  $A \neq \{\}$ 
  shows  $x \in \text{closure } A \longleftrightarrow \text{infdist } x \ A = 0$ 
proof
  assume  $x \in \text{closure } A$ 
  show  $\text{infdist } x \ A = 0$ 
  proof (rule ccontr)
    assume  $\text{infdist } x \ A \neq 0$ 
    with infdist_nonneg[of  $x \ A$ ] have  $\text{infdist } x \ A > 0$ 
    by auto
    then have ball  $x \ (\text{infdist } x \ A) \cap \text{closure } A = \{\}$ 
    apply auto
    apply (metis  $\langle x \in \text{closure } A \rangle \text{closure\_approachable dist\_commute infdist\_le not\_less}$ )
    done
    then have  $x \notin \text{closure } A$ 
    by (metis  $\langle 0 < \text{infdist } x \ A \rangle \text{centre\_in\_ball disjoint\_iff\_not\_equal}$ )
    then show False using  $\langle x \in \text{closure } A \rangle$  by simp
  qed
next
  assume  $x: \text{infdist } x \ A = 0$ 
  then obtain  $a$  where  $a \in A$ 
  by atomize_elim (metis all_not_in_conv assms)
  show  $x \in \text{closure } A$ 
  unfolding closure_approachable
  apply safe
  proof (rule ccontr)
    fix  $e :: \text{real}$ 
    assume  $e > 0$ 
    assume  $\neg (\exists y \in A. \text{dist } y \ x < e)$ 
    then have  $\text{infdist } x \ A \geq e$  using  $\langle a \in A \rangle$ 
    unfolding infdist_def
    by (force simp: dist_commute intro: cINF_greatest)
    with  $x \ \langle e > 0 \rangle$  show False by auto
  qed
qed

```



```

lemma in_closed_iff_infdist_zero:
  assumes closed  $A \ A \neq \{\}$ 
  shows  $x \in A \longleftrightarrow \text{infdist } x \ A = 0$ 
proof -
  have  $x \in \text{closure } A \longleftrightarrow \text{infdist } x \ A = 0$ 
  by (rule in_closure_iff_infdist_zero) fact
  with assms show ?thesis by simp
qed

```



```

lemma infdist_pos_not_in_closed:
  assumes closed  $S \ S \neq \{\} \ x \notin S$ 

```

```

shows infdist  $x\ S > 0$ 
using in_closed_iff_infdist_zero[OF assms(1) assms(2), of  $x$ ] assms(3) infdist_nonneg
le_less by fastforce

```

lemma

```

  infdist_attains_inf:
  fixes  $X::'a::heine\_borel\ set$ 
  assumes closed  $X$ 
  assumes  $X \neq \{\}$ 
  obtains  $x$  where  $x \in X$  infdist  $y\ X = dist\ y\ x$ 
proof -
  have bdd_below ( $dist\ y\ `X$ )
    by auto
  from distance_attains_inf[OF assms, of  $y$ ]
  obtain  $x$  where INF:  $x \in X \wedge z. z \in X \implies dist\ y\ x \leq dist\ y\ z$  by auto
  have infdist  $y\ X = dist\ y\ x$ 
    by (auto simp: infdist_def assms
      intro!: antisym cINF_lower[OF _  $\langle x \in X \rangle$ ] cINF_greatest[OF assms(2)
        INF(2)])
  with  $\langle x \in X \rangle$  show ?thesis ..
qed

```

Every metric space is a T4 space:

```

instance metric_space  $\subseteq$  t4_space

```

proof

```

  fix  $S\ T::'a\ set$  assume  $H: closed\ S\ closed\ T\ S \cap T = \{\}$ 
  consider  $S = \{\} \mid T = \{\} \mid S \neq \{\} \wedge T \neq \{\}$  by auto
  then show  $\exists U\ V. open\ U \wedge open\ V \wedge S \subseteq U \wedge T \subseteq V \wedge U \cap V = \{\}$ 
  proof (cases)
    case 1
    show ?thesis
    apply (rule exI[of _  $\{\}$ ], rule exI[of _ UNIV]) using 1 by auto
  next
    case 2
    show ?thesis
    apply (rule exI[of _ UNIV], rule exI[of _  $\{\}$ ]) using 2 by auto
  next
    case 3
    define  $U$  where  $U = (\bigcup_{x \in S}. ball\ x\ ((infdist\ x\ T)/2))$ 
    have  $A: open\ U$  unfolding  $U\_def$  by auto
    have infdist  $x\ T > 0$  if  $x \in S$  for  $x$ 
      using  $H$  that 3 by (auto intro!: infdist_pos_not_in_closed)
    then have  $B: S \subseteq U$  unfolding  $U\_def$  by auto
    define  $V$  where  $V = (\bigcup_{x \in T}. ball\ x\ ((infdist\ x\ S)/2))$ 
    have  $C: open\ V$  unfolding  $V\_def$  by auto
    have infdist  $x\ S > 0$  if  $x \in T$  for  $x$ 
      using  $H$  that 3 by (auto intro!: infdist_pos_not_in_closed)
    then have  $D: T \subseteq V$  unfolding  $V\_def$  by auto

```

```

    have (ball x ((infdist x T)/2)) ∩ (ball y ((infdist y S)/2)) = {} if x ∈ S y ∈
T for x y
  proof auto
    fix z assume H: dist x z * 2 < infdist x T dist y z * 2 < infdist y S
    have 2 * dist x y ≤ 2 * dist x z + 2 * dist y z
      by metric
    also have ... < infdist x T + infdist y S
      using H by auto
    finally have dist x y < infdist x T ∨ dist x y < infdist y S
      by auto
    then show False
      using infdist_le[OF ⟨x ∈ S⟩, of y] infdist_le[OF ⟨y ∈ T⟩, of x] by (auto
simp add: dist_commute)
  qed
  then have E: U ∩ V = {}
    unfolding U_def V_def by auto
  show ?thesis
    apply (rule exI[of _ U], rule exI[of _ V]) using A B C D E by auto
  qed
qed

```

```

lemma tendsto_infdist [tendsto_intros]:
  assumes f: (f ⟶ l) F
  shows ((λx. infdist (f x) A) ⟶ infdist l A) F
proof (rule tendstoI)
  fix e :: real
  assume e > 0
  from tendstoD[OF f this]
  show eventually (λx. dist (infdist (f x) A) (infdist l A) < e) F
  proof (eventually_elim)
    fix x
    from infdist_triangle[of l A f x] infdist_triangle[of f x A l]
    have dist (infdist (f x) A) (infdist l A) ≤ dist (f x) l
      by (simp add: dist_commute dist_real_def)
    also assume dist (f x) l < e
    finally show dist (infdist (f x) A) (infdist l A) < e .
  qed
qed

```

```

lemma continuous_infdist [continuous_intros]:
  assumes continuous F f
  shows continuous F (λx. infdist (f x) A)
  using assms unfolding continuous_def by (rule tendsto_infdist)

```

```

lemma continuous_on_infdist [continuous_intros]:
  assumes continuous_on S f
  shows continuous_on S (λx. infdist (f x) A)
  using assms unfolding continuous_on by (auto intro: tendsto_infdist)

```



```

lemma compact_infdist_le:
  fixes A::'a::heine_borel set
  assumes A  $\neq$  {}
  assumes compact A
  assumes e > 0
  shows compact {x. infdist x A  $\leq$  e}
proof -
  from continuous_closed_vimage[of {0..e}  $\lambda$ x. infdist x A]
    continuous_infdist[OF continuous_ident, of _ UNIV A]
  have closed {x. infdist x A  $\leq$  e} by (auto simp: vimage_def infdist_nonneg)
  moreover
  from assms obtain x0 b where b:  $\bigwedge$ x. x  $\in$  A  $\implies$  dist x0 x  $\leq$  b closed A
    by (auto simp: compact_eq_bounded_closed bounded_def)
  {
    fix y
    assume infdist y A  $\leq$  e
    moreover
    from infdist_attains_inf[OF  $\langle$ closed A $\rangle$   $\langle$ A  $\neq$  {} $\rangle$ , of y]
    obtain z where z  $\in$  A infdist y A = dist y z by blast
    ultimately
    have dist x0 y  $\leq$  b + e using b by metric
  } then
  have bounded {x. infdist x A  $\leq$  e}
    by (auto simp: bounded_any_center[where a=x0] intro!: exI[where x=b +
e])
  ultimately show compact {x. infdist x A  $\leq$  e}
    by (simp add: compact_eq_bounded_closed)
qed

```

3.2.21 Separation between Points and Sets

```

proposition separate_point_closed:
  fixes s :: 'a::heine_borel set
  assumes closed s and a  $\notin$  s
  shows  $\exists d > 0. \forall x \in s. d \leq \text{dist } a \ x$ 
proof (cases s = {})
  case True
  then show ?thesis by (auto intro!: exI[where x=1])
next
  case False
  from assms obtain x where x  $\in$  s  $\forall y \in s. \text{dist } a \ x \leq \text{dist } a \ y$ 
    using  $\langle s \neq \{\} \rangle$  by (blast intro: distance_attains_inf [of s a])
  with  $\langle x \in s \rangle$  show ?thesis using dist_pos_lt[of a x] and  $\langle a \notin s \rangle$ 
    by blast
qed

```

```

proposition separate_compact_closed:
  fixes s t :: 'a::heine_borel set
  assumes compact s

```

```

    and  $t$ :  $\text{closed } t \ s \cap t = \{\}$ 
  shows  $\exists d > 0. \forall x \in s. \forall y \in t. d \leq \text{dist } x \ y$ 
proof cases
  assume  $s \neq \{\} \wedge t \neq \{\}$ 
  then have  $s \neq \{\} \ t \neq \{\}$  by auto
  let  $?inf = \lambda x. \text{infdist } x \ t$ 
  have  $\text{continuous\_on } s \ ?inf$ 
  by (auto intro!:  $\text{continuous\_at\_imp\_continuous\_on continuous\_infdist continuous\_ident}$ )
  then obtain  $x$  where  $x: x \in s \ \forall y \in s. ?inf \ x \leq ?inf \ y$ 
  using  $\text{continuous\_attains\_inf}[OF \ \langle \text{compact } s \rangle \ \langle s \neq \{\} \rangle]$  by auto
  then have  $0 < ?inf \ x$ 
  using  $t \ \langle t \neq \{\} \rangle$  in  $\text{closed\_iff\_infdist\_zero}$  by (auto simp:  $\text{less\_le infdist\_nonneg}$ )
  moreover have  $\forall x' \in s. \forall y \in t. ?inf \ x \leq \text{dist } x' \ y$ 
  using  $x$  by (auto intro:  $\text{order\_trans infdist\_le}$ )
  ultimately show  $?thesis$  by auto
qed (auto intro!:  $\text{exI}[of \ _ \ 1]$ )

```

```

proposition  $\text{separate\_closed\_compact}$ :
  fixes  $s \ t :: 'a::\text{heine\_borel set}$ 
  assumes  $\text{closed } s$ 
    and  $\text{compact } t$ 
    and  $s \cap t = \{\}$ 
  shows  $\exists d > 0. \forall x \in s. \forall y \in t. d \leq \text{dist } x \ y$ 
proof -
  have  $*$ :  $t \cap s = \{\}$ 
  using  $\text{assms}(3)$  by auto
  show  $?thesis$ 
  using  $\text{separate\_compact\_closed}[OF \ \text{assms}(2,1) \ *]$  by (force simp:  $\text{dist\_commute}$ )
qed

```

```

proposition  $\text{compact\_in\_open\_separated}$ :
  fixes  $A :: 'a::\text{heine\_borel set}$ 
  assumes  $A \neq \{\}$ 
  assumes  $\text{compact } A$ 
  assumes  $\text{open } B$ 
  assumes  $A \subseteq B$ 
  obtains  $e > 0 \ \{x. \text{infdist } x \ A \leq e\} \subseteq B$ 
proof atomize_elim
  have  $\text{closed } (- \ B) \ \text{compact } A - B \cap A = \{\}$ 
  using  $\text{assms}$  by (auto simp:  $\text{open\_Diff compact\_eq\_bounded\_closed}$ )
  from  $\text{separate\_closed\_compact}[OF \ \text{this}]$ 
  obtain  $d'::\text{real}$  where  $d': d' > 0 \ \bigwedge x \ y. x \notin B \implies y \in A \implies d' \leq \text{dist } x \ y$ 
  by auto
  define  $d$  where  $d = d' / 2$ 
  hence  $d > 0 \ d < d'$  using  $d'$  by auto
  with  $d'$  have  $d: \bigwedge x \ y. x \notin B \implies y \in A \implies d < \text{dist } x \ y$ 
  by force
  show  $\exists e > 0. \{x. \text{infdist } x \ A \leq e\} \subseteq B$ 

```

```

proof (rule ccontr)
  assume  $\nexists e. 0 < e \wedge \{x. \text{infdist } x \ A \leq e\} \subseteq B$ 
  with  $\langle d > 0 \rangle$  obtain  $x$  where  $x: \text{infdist } x \ A \leq d \wedge x \notin B$ 
  by auto
  from assms have  $\text{closed } A \wedge A \neq \{\}$  by (auto simp: compact_eq_bounded_closed)
  from infdist_attains_inf [OF this]
  obtain  $y$  where  $y: y \in A \wedge \text{infdist } x \ A = \text{dist } x \ y$ 
  by auto
  have  $\text{dist } x \ y \leq d$  using  $x \ y$  by simp
  also have  $\dots < \text{dist } x \ y$  using  $y \ d \ x$  by auto
  finally show False by simp
qed
qed

```

3.2.22 Uniform Continuity

```

lemma uniformly_continuous_onE:
  assumes uniformly_continuous_on  $s \ f \ 0 < e$ 
  obtains  $d$  where  $d > 0 \wedge x \ x'. \llbracket x \in s; x' \in s; \text{dist } x' \ x < d \rrbracket \implies \text{dist } (f \ x') \ (f \ x) < e$ 
using assms
by (auto simp: uniformly_continuous_on_def)

```

```

lemma uniformly_continuous_on_sequentially:
  uniformly_continuous_on  $s \ f \longleftrightarrow (\forall x \ y. (\forall n. x \ n \in s) \wedge (\forall n. y \ n \in s) \wedge$ 
     $(\lambda n. \text{dist } (x \ n) \ (y \ n)) \longrightarrow 0 \longrightarrow (\lambda n. \text{dist } (f \ (x \ n)) \ (f \ (y \ n))) \longrightarrow 0)$  (is
     $?lhs = ?rhs)$ 

```

```

proof
  assume ?lhs
  {
    fix  $x \ y$ 
    assume  $x: \forall n. x \ n \in s$ 
    and  $y: \forall n. y \ n \in s$ 
    and  $xy: ((\lambda n. \text{dist } (x \ n) \ (y \ n)) \longrightarrow 0) \text{ sequentially}$ 
    {
      fix  $e :: \text{real}$ 
      assume  $e > 0$ 
      then obtain  $d$  where  $d > 0$  and  $d: \forall x \in s. \forall x' \in s. \text{dist } x' \ x < d \longrightarrow \text{dist } (f \ x') \ (f \ x) < e$ 
      using  $\langle ?lhs \rangle$  [unfolded uniformly_continuous_on_def, THEN spec[where  $x=e$ ]] by auto
      obtain  $N$  where  $N: \forall n \geq N. \text{dist } (x \ n) \ (y \ n) < d$ 
      using  $xy$  [unfolded lim_sequentially_dist_norm] and  $\langle d > 0 \rangle$  by auto
      {
        fix  $n$ 
        assume  $n \geq N$ 
        then have  $\text{dist } (f \ (x \ n)) \ (f \ (y \ n)) < e$ 
        using  $N$  [THEN spec[where  $x=n$ ]]
        using  $d$  [THEN bspec[where  $x=x \ n$ ], THEN bspec[where  $x=y \ n$ ]]
        using  $x$  and  $y$ 

```

```

      by (simp add: dist_commute)
    }
    then have  $\exists N. \forall n \geq N. \text{dist } (f(x\ n)) (f(y\ n)) < e$ 
      by auto
    }
    then have  $((\lambda n. \text{dist } (f(x\ n)) (f(y\ n))) \longrightarrow 0)$  sequentially
      unfolding lim_sequentially and dist_real_def by auto
    }
    then show ?rhs by auto
next
  assume ?rhs
  {
    assume  $\neg ?lhs$ 
    then obtain  $e$  where  $e > 0 \ \forall d > 0. \exists x \in s. \exists x' \in s. \text{dist } x' x < d \wedge \neg \text{dist } (f\ x') (f\ x) < e$ 
      unfolding uniformly_continuous_on_def by auto
    then obtain  $fa$  where  $fa$ :
       $\forall x. 0 < x \longrightarrow \text{fst } (fa\ x) \in s \wedge \text{snd } (fa\ x) \in s \wedge \text{dist } (\text{fst } (fa\ x)) (\text{snd } (fa\ x)) < x \wedge \neg \text{dist } (f\ (\text{fst } (fa\ x))) (f\ (\text{snd } (fa\ x))) < e$ 
      using choice[of  $\lambda d\ x. d > 0 \longrightarrow \text{fst } x \in s \wedge \text{snd } x \in s \wedge \text{dist } (\text{snd } x) (\text{fst } x) < d \wedge \neg \text{dist } (f\ (\text{snd } x)) (f\ (\text{fst } x)) < e$ ]
      unfolding Bex_def
      by (auto simp: dist_commute)
    define  $x$  where  $x\ n = \text{fst } (fa\ (\text{inverse } (\text{real } n + 1)))$  for  $n$ 
    define  $y$  where  $y\ n = \text{snd } (fa\ (\text{inverse } (\text{real } n + 1)))$  for  $n$ 
    have  $xy0: \forall n. x\ n \in s \wedge y\ n \in s$ 
      and  $xy0: \forall n. \text{dist } (x\ n) (y\ n) < \text{inverse } (\text{real } n + 1)$ 
      and  $fxy: \forall n. \neg \text{dist } (f\ (x\ n)) (f\ (y\ n)) < e$ 
      unfolding  $x\_def$  and  $y\_def$  using  $fa$ 
      by auto
    {
      fix  $e :: \text{real}$ 
      assume  $e > 0$ 
      then obtain  $N :: \text{nat}$  where  $N \neq 0$  and  $N: 0 < \text{inverse } (\text{real } N) \wedge \text{inverse } (\text{real } N) < e$ 
        unfolding real_arch_inverse[of  $e$ ] by auto
      {
        fix  $n :: \text{nat}$ 
        assume  $n \geq N$ 
        then have  $\text{inverse } (\text{real } n + 1) < \text{inverse } (\text{real } N)$ 
          using of_nat_0_le_iff and  $\langle N \neq 0 \rangle$  by auto
        also have  $\dots < e$  using  $N$  by auto
        finally have  $\text{inverse } (\text{real } n + 1) < e$  by auto
        then have  $\text{dist } (x\ n) (y\ n) < e$ 
          using  $xy0$  [THEN spec[where  $x=n$ ]] by auto
      }
      then have  $\exists N. \forall n \geq N. \text{dist } (x\ n) (y\ n) < e$  by auto
    }
    then have  $\forall e > 0. \exists N. \forall n \geq N. \text{dist } (f\ (x\ n)) (f\ (y\ n)) < e$ 

```

```

    using ‹?rhs›[THEN spec[where x=x], THEN spec[where x=y]] and xyn
    unfolding lim_sequentially dist_real_def by auto
    then have False using fxy and ‹e>0› by auto
  }
  then show ?lhs
    unfolding uniformly_continuous_on_def by blast
qed

```

3.2.23 Continuity on a Compact Domain Implies Uniform Continuity

From the proof of the Heine-Borel theorem: Lemma 2 in section 3.7, page 69 of J. C. Burkill and H. Burkill. A Second Course in Mathematical Analysis (CUP, 2002)

lemma *Heine_Borel_lemma*:

```

  assumes compact S and Ssub:  $S \subseteq \bigcup \mathcal{G}$  and opn:  $\bigwedge G. G \in \mathcal{G} \implies \text{open } G$ 
  obtains e where  $0 < e \wedge x \in S \implies \exists G \in \mathcal{G}. \text{ball } x \ e \subseteq G$ 
proof -
  have False if neg:  $\bigwedge e. 0 < e \implies \exists x \in S. \forall G \in \mathcal{G}. \neg \text{ball } x \ e \subseteq G$ 
  proof -
    have  $\exists x \in S. \forall G \in \mathcal{G}. \neg \text{ball } x \ (1 / \text{Suc } n) \subseteq G$  for n
      using neg by simp
    then obtain f where  $\bigwedge n. f \ n \in S$  and  $fG: \bigwedge G n. G \in \mathcal{G} \implies \neg \text{ball } (f \ n) \ (1 / \text{Suc } n) \subseteq G$ 
      by metis
    then obtain l r where  $l \in S$  strict_mono r and to_l:  $(f \circ r) \longrightarrow l$ 
      using ‹compact S› compact_def that by metis
    then obtain G where  $l \in G$   $G \in \mathcal{G}$ 
      using Ssub by auto
    then obtain e where  $0 < e$  and e:  $\bigwedge z. \text{dist } z \ l < e \implies z \in G$ 
      using opn open_dist by blast
    obtain N1 where  $N1: \bigwedge n. n \geq N1 \implies \text{dist } (f \ (r \ n)) \ l < e/2$ 
      using to_l apply (simp add: lim_sequentially)
      using ‹0 < e› half_gt_zero that by blast
    obtain N2 where  $N2: \text{of\_nat } N2 > 2/e$ 
      using reals_Archimedean2 by blast
    obtain x where  $x \in \text{ball } (f \ (r \ (\max N1 N2))) \ (1 / \text{real } (\text{Suc } (r \ (\max N1 N2))))$  and  $x \notin G$ 
      using fG [OF ‹G ∈ G›, of r (max N1 N2)] by blast
    then have  $\text{dist } (f \ (r \ (\max N1 N2))) \ x < 1 / \text{real } (\text{Suc } (r \ (\max N1 N2)))$ 
      by simp
    also have  $\dots \leq 1 / \text{real } (\text{Suc } (\max N1 N2))$ 
      apply (simp add: field_split_simps del: max.bounded_iff)
      using ‹strict_mono r› seq_suble by blast
    also have  $\dots \leq 1 / \text{real } (\text{Suc } N2)$ 
      by (simp add: field_simps)
    also have  $\dots < e/2$ 
      using N2 ‹0 < e› by (simp add: field_simps)

```

```

    finally have  $\text{dist } (f \ (r \ (\max N1 \ N2))) \ x < e/2$  .
    moreover have  $\text{dist } (f \ (r \ (\max N1 \ N2))) \ l < e/2$ 
      using  $N1 \ \text{max.cobounded1}$  by blast
    ultimately have  $\text{dist } x \ l < e$ 
      by metric
    then show ?thesis
      using  $e \ \langle x \notin G \rangle$  by blast
  qed
  then show ?thesis
    by (meson that)
  qed

lemma compact_uniformly_equicontinuous:
  assumes compact  $S$ 
    and cont:  $\bigwedge x \ e. \llbracket x \in S; \ 0 < e \rrbracket \implies \exists d. \ 0 < d \wedge$ 
       $(\forall f \in \mathcal{F}. \forall x' \in S. \text{dist } x' \ x < d \longrightarrow \text{dist } (f \ x') \ (f \ x) < e)$ 
    and  $0 < e$ 
  obtains  $d$  where  $0 < d$ 
     $\bigwedge f \ x \ x'. \llbracket f \in \mathcal{F}; \ x \in S; \ x' \in S; \ \text{dist } x' \ x < d \rrbracket \implies \text{dist } (f \ x') \ (f \ x)$ 
  <  $e$ 
  proof -
    obtain  $d$  where  $d\_pos: \bigwedge x \ e. \llbracket x \in S; \ 0 < e \rrbracket \implies 0 < d \ x \ e$ 
      and  $d\_dist: \bigwedge x \ x' \ e \ f. \llbracket \text{dist } x' \ x < d \ x \ e; \ x \in S; \ x' \in S; \ 0 < e; \ f \in \mathcal{F} \rrbracket \implies$ 
 $\text{dist } (f \ x') \ (f \ x) < e$ 
      using cont by metis
    let ? $\mathcal{G} = ((\lambda x. \text{ball } x \ (d \ x \ (e/2)))) \ ' S$ 
    have  $S_{sub}: S \subseteq \bigcup \ ?\mathcal{G}$ 
      by clarsimp (metis  $d\_pos \ \langle 0 < e \rangle \ \text{dist\_self half\_gt\_zero\_iff}$ )
    then obtain  $k$  where  $0 < k$  and  $k: \bigwedge x. x \in S \implies \exists G \in \ ?\mathcal{G}. \text{ball } x \ k \subseteq G$ 
      by (rule Heine_Borel_lemma [OF  $\langle \text{compact } S \rangle$ ]) auto
    moreover have  $\text{dist } (f \ v) \ (f \ u) < e$  if  $f \in \mathcal{F} \ u \in S \ v \in S \ \text{dist } v \ u < k$  for  $f \ u \ v$ 
      proof -
        obtain  $G$  where  $G \in \ ?\mathcal{G} \ u \in G \ v \in G$ 
          using  $k$  that
        by (metis  $\langle \text{dist } v \ u < k \rangle \ \langle u \in S \rangle \ \langle 0 < k \rangle \ \text{centre\_in\_ball subsetD dist\_commute mem\_ball}$ )
        then obtain  $w$  where  $w: \text{dist } w \ u < d \ w \ (e/2) \ \text{dist } w \ v < d \ w \ (e/2) \ w \in S$ 
          by auto
        with that  $d\_dist$  have  $\text{dist } (f \ w) \ (f \ v) < e/2$ 
          by (metis  $\langle 0 < e \rangle \ \text{dist\_commute half\_gt\_zero}$ )
        moreover
          have  $\text{dist } (f \ w) \ (f \ u) < e/2$ 
            using that  $d\_dist \ w$  by (metis  $\langle 0 < e \rangle \ \text{dist\_commute divide\_pos\_pos zero\_less\_numeral}$ )
          ultimately show ?thesis
            using  $\text{dist\_triangle\_half\_r}$  by blast
      qed
    ultimately show ?thesis using that by blast
  qed

```

qed

corollary *compact_uniformly_continuous*:
 fixes $f :: 'a :: \text{metric\_space} \Rightarrow 'b :: \text{metric\_space}$
 assumes $f: \text{continuous\_on } S \text{ f}$ and $S: \text{compact } S$
 shows $\text{uniformly\_continuous\_on } S \text{ f}$
 using f
 unfolding continuous_on_iff $\text{uniformly\_continuous\_on\_def}$
 by (force intro: $\text{compact\_uniformly\_equicontinuous [OF } S, \text{ of } \{f\}]$)

3.2.24 Theorems relating continuity and uniform continuity to closures

lemma *continuous_on_closure*:
 $\text{continuous\_on } (\text{closure } S) \text{ f} \longleftrightarrow$
 $(\forall x \ e. x \in \text{closure } S \wedge 0 < e \longrightarrow (\exists d. 0 < d \wedge (\forall y. y \in S \wedge \text{dist } y \ x < d \longrightarrow \text{dist } (f \ y) \ (f \ x) < e)))$
 (is ?lhs = ?rhs)
proof
 assume ?lhs then show ?rhs
 unfolding continuous_on_iff by (metis Un_iff closure_def)
next
 assume R [rule_format]: ?rhs
 show ?lhs
proof
 fix x and $e::\text{real}$
 assume $0 < e$ and $x: x \in \text{closure } S$
 obtain $\delta::\text{real}$ where $\delta > 0$
 and $\delta: \bigwedge y. [y \in S; \text{dist } y \ x < \delta] \Longrightarrow \text{dist } (f \ y) \ (f \ x) < e/2$
 using R [of $x \ e/2$] $\langle 0 < e \rangle x$ by auto
 have $\text{dist } (f \ y) \ (f \ x) \leq e$ if $y: y \in \text{closure } S$ and $dyx: \text{dist } y \ x < \delta/2$ for y
proof –
 obtain $\delta':\text{real}$ where $\delta' > 0$
 and $\delta': \bigwedge z. [z \in S; \text{dist } z \ y < \delta'] \Longrightarrow \text{dist } (f \ z) \ (f \ y) < e/2$
 using R [of $y \ e/2$] $\langle 0 < e \rangle y$ by auto
 obtain z where $z \in S$ and $z: \text{dist } z \ y < \min \delta' \delta / 2$
 using $\text{closure\_approachable } y$
 by (metis $\langle 0 < \delta' \rangle \langle 0 < \delta \rangle \text{divide\_pos\_pos}$ min_less_iff_conj zero_less_numeral)
 have $\text{dist } (f \ z) \ (f \ y) < e/2$
 using $\delta' [OF \langle z \in S \rangle] z \langle 0 < \delta' \rangle$ by metric
 moreover have $\text{dist } (f \ z) \ (f \ x) < e/2$
 using $\delta [OF \langle z \in S \rangle] z \ dyx$ by metric
 ultimately show ?thesis
 by metric
qed
 then show $\exists d > 0. \forall x' \in \text{closure } S. \text{dist } x' \ x < d \longrightarrow \text{dist } (f \ x') \ (f \ x) \leq e$
 by (rule_tac $x = \delta/2$ in exI) (simp add: $\langle \delta > 0 \rangle$)
qed
qed

lemma *continuous_on_closure_sequentially*:

fixes $f :: 'a::metric\_space \Rightarrow 'b::metric\_space$

shows

$continuous\_on\ (closure\ S)\ f \longleftrightarrow$

$(\forall x\ a.\ a \in closure\ S \wedge (\forall n.\ x\ n \in S) \wedge x \longrightarrow a \longrightarrow (f \circ x) \longrightarrow f\ a)$

(is ?lhs = ?rhs)

proof $-$

have $continuous\_on\ (closure\ S)\ f \longleftrightarrow$

$(\forall x \in closure\ S.\ continuous\ (at\ x\ within\ S)\ f)$

by $(force\ simp:\ continuous\_on\_closure\ continuous\_within\_eps\_delta)$

also have $\dots = ?rhs$

by $(force\ simp:\ continuous\_within\_sequentially)$

finally show $?thesis\ .$

qed

lemma *uniformly_continuous_on_closure*:

fixes $f :: 'a::metric\_space \Rightarrow 'b::metric\_space$

assumes $ucont:\ uniformly\_continuous\_on\ S\ f$

and $cont:\ continuous\_on\ (closure\ S)\ f$

shows $uniformly\_continuous\_on\ (closure\ S)\ f$

unfolding *uniformly_continuous_on_def*

proof $(intro\ allI\ impI)$

fix $e::real$

assume $0 < e$

then obtain $d::real$

where $d > 0$

and $d:\ \bigwedge x\ x'.\ [x \in S;\ x' \in S;\ dist\ x'\ x < d] \implies dist\ (f\ x')\ (f\ x) < e/3$

using $ucont\ [unfolded\ uniformly\_continuous\_on\_def,\ rule\_format,\ of\ e/3]$ **by**

auto

show $\exists d > 0.\ \forall x \in closure\ S.\ \forall x' \in closure\ S.\ dist\ x'\ x < d \longrightarrow dist\ (f\ x')\ (f\ x) < e$

e

proof $(rule\ exI\ [where\ x=d/3],\ clarsimp\ simp:\ \langle d > 0 \rangle)$

fix $x\ y$

assume $x:\ x \in closure\ S$ **and** $y:\ y \in closure\ S$ **and** $dyx:\ dist\ y\ x * 3 < d$

obtain $d1::real$ **where** $d1 > 0$

and $d1:\ \bigwedge w.\ [w \in closure\ S;\ dist\ w\ x < d1] \implies dist\ (f\ w)\ (f\ x) < e/3$

using $cont\ [unfolded\ continuous\_on\_iff,\ rule\_format,\ of\ x\ e/3]\ \langle 0 < e \rangle\ x$

by *auto*

obtain x' **where** $x' \in S$ **and** $x':\ dist\ x'\ x < \min\ d1\ (d / 3)$

using *closure_approachable* $[of\ x\ S]$

by $(metis\ \langle 0 < d1 \rangle\ \langle 0 < d \rangle\ divide\_pos\_pos\ min\_less\_iff\ conj\ x\ zero\_less\_numeral)$

obtain $d2::real$ **where** $d2 > 0$

and $d2:\ \forall w \in closure\ S.\ dist\ w\ y < d2 \longrightarrow dist\ (f\ w)\ (f\ y) < e/3$

using $cont\ [unfolded\ continuous\_on\_iff,\ rule\_format,\ of\ y\ e/3]\ \langle 0 < e \rangle\ y$

by *auto*

obtain y' **where** $y' \in S$ **and** $y':\ dist\ y'\ y < \min\ d2\ (d / 3)$

using *closure_approachable* $[of\ y\ S]$

by $(metis\ \langle 0 < d2 \rangle\ \langle 0 < d \rangle\ divide\_pos\_pos\ min\_less\_iff\ conj\ y\ zero\_less\_numeral)$


```

have  $\text{dist } x' x < d/3$  using  $x'$  by auto
then have  $\text{dist } x' y' < d$ 
  using  $\text{dys } y'$  by metric
then have  $\text{dist } (f x') (f y') < e/3$ 
  by (rule d [OF  $\langle y' \in S \rangle \langle x' \in S \rangle$ ])
moreover have  $\text{dist } (f x') (f x) < e/3$  using  $\langle x' \in S \rangle$  closure_subset  $x'$  d1
  by (simp add: closure_def)
moreover have  $\text{dist } (f y') (f y) < e/3$  using  $\langle y' \in S \rangle$  closure_subset  $y'$  d2
  by (simp add: closure_def)
ultimately show  $\text{dist } (f y) (f x) < e$  by metric
qed
qed

```

lemma *uniformly_continuous_on_extension_at_closure*:

```

fixes  $f::'a::\text{metric\_space} \Rightarrow 'b::\text{complete\_space}$ 
assumes  $\text{uc: uniformly\_continuous\_on } X f$ 
assumes  $x \in \text{closure } X$ 
obtains  $l$  where  $(f \longrightarrow l)$  (at  $x$  within  $X$ )
proof -
  from assms obtain  $xs$  where  $xs: xs \longrightarrow x \wedge n. xs\ n \in X$ 
  by (auto simp: closure_sequential)

```

```

from uniformly_continuous_on_Cauchy[OF uc LIMSEQ_imp_Cauchy, OF xs]
obtain  $l$  where  $l: (\lambda n. f (xs\ n)) \longrightarrow l$ 
  by atomize_elim (simp only: convergent_eq_Cauchy)

```

```

have  $(f \longrightarrow l)$  (at  $x$  within  $X$ )

```

```

proof (safe intro!: Lim_within_LIMSEQ)

```

```

  fix  $xs'$ 

```

```

  assume  $\forall n. xs'\ n \neq x \wedge xs'\ n \in X$ 

```

```

  and  $xs': xs' \longrightarrow x$ 

```

```

  then have  $xs'\ n \neq x \wedge xs'\ n \in X$  for  $n$  by auto

```

```

  from uniformly_continuous_on_Cauchy[OF uc LIMSEQ_imp_Cauchy, OF
 $\langle xs' \longrightarrow x \rangle \langle xs' \_ \in X \rangle$ ]

```

```

  obtain  $l'$  where  $l': (\lambda n. f (xs'\ n)) \longrightarrow l'$ 

```

```

  by atomize_elim (simp only: convergent_eq_Cauchy)

```

```

show  $(\lambda n. f (xs'\ n)) \longrightarrow l$ 

```

```

proof (rule tendstoI)

```

```

  fix  $e::\text{real}$  assume  $e > 0$ 

```

```

  define  $e'$  where  $e' \equiv e/2$ 

```

```

  have  $e' > 0$  using  $\langle e > 0 \rangle$  by (simp add:  $e'_\text{def}$ )

```

```

  have  $\forall_F n$  in sequentially.  $\text{dist } (f (xs\ n))\ l < e'$ 

```

```

    by (simp add:  $\langle 0 < e' \rangle$   $l$  tendstoD)

```

```

  moreover

```

```

  from uc[unfolded uniformly_continuous_on_def, rule_format, OF  $\langle e' > 0 \rangle$ ]

```

```

  obtain  $d$  where  $d: d > 0 \wedge x x'. x \in X \Longrightarrow x' \in X \Longrightarrow \text{dist } x x' < d \Longrightarrow$ 

```

```

dist (f x) (f x') < e'
  by auto
  have  $\forall_F n$  in sequentially. dist (xs n) (xs' n) < d
    by (auto intro!:  $\langle 0 < d \rangle$  order_tendstoD tendsto_eq_intros xs xs')
  ultimately
  show  $\forall_F n$  in sequentially. dist (f (xs' n)) l < e
  proof eventually_elim
    case (elim n)
    have dist (f (xs' n)) l  $\leq$  dist (f (xs n)) (f (xs' n)) + dist (f (xs n)) l
      by metric
    also have dist (f (xs n)) (f (xs' n)) < e'
      by (auto intro!: d xs  $\langle xs' \_ \in \_ \rangle$  elim)
    also note  $\langle \text{dist (f (xs n)) l} < e' \rangle$ 
    also have  $e' + e' = e$  by (simp add: e'_def)
    finally show ?case by simp
  qed
qed
qed
thus ?thesis ..
qed

lemma uniformly_continuous_on_extension_on_closure:
  fixes f::'a::metric_space  $\Rightarrow$  'b::complete_space
  assumes uc: uniformly_continuous_on X f
  obtains g where uniformly_continuous_on (closure X) g  $\wedge$   $x \in X \Rightarrow f x =$ 
 $g x$ 
 $\wedge Y h x. X \subseteq Y \Rightarrow Y \subseteq \text{closure } X \Rightarrow \text{continuous\_on } Y h \Rightarrow (\wedge x. x \in X$ 
 $\Rightarrow f x = h x) \Rightarrow x \in Y \Rightarrow h x = g x$ 
proof -
  from uc have cont_f: continuous_on X f
    by (simp add: uniformly_continuous_imp_continuous)
  obtain y where y: (f  $\longrightarrow$  y x) (at x within X) if  $x \in \text{closure } X$  for x
    apply atomize_elim
    apply (rule choice)
    using uniformly_continuous_on_extension_at_closure[OF assms]
    by metis
  let ?g =  $\lambda x. \text{if } x \in X \text{ then } f x \text{ else } y x$ 

  have uniformly_continuous_on (closure X) ?g
    unfolding uniformly_continuous_on_def
  proof safe
    fix e::real assume e > 0
    define e' where e'  $\equiv e / 3$ 
    have e' > 0 using  $\langle e > 0 \rangle$  by (simp add: e'_def)
    from uc[unfolded uniformly_continuous_on_def, rule_format, OF  $\langle 0 < e' \rangle$ ]
    obtain d where d > 0 and d:  $\wedge x x'. x \in X \Rightarrow x' \in X \Rightarrow \text{dist } x' x < d$ 
 $\Rightarrow \text{dist (f x') (f x)} < e'$ 
    by auto
    define d' where d' = d / 3

```

```

have  $d' > 0$  using  $\langle d > 0 \rangle$  by (simp add:  $d\_def$ )
show  $\exists d > 0. \forall x \in \text{closure } X. \forall x' \in \text{closure } X. \text{dist } x' x < d \longrightarrow \text{dist } (?g x') (?g x) < e$ 
proof (safe intro!: exI[where  $x=d'$ ]  $\langle d' > 0 \rangle$ )
  fix  $x x'$  assume  $x: x \in \text{closure } X$  and  $x': x' \in \text{closure } X$  and  $\text{dist}: \text{dist } x' x < d'$ 
  then obtain  $xs xs'$  where  $xs: xs \longrightarrow x \bigwedge n. xs\ n \in X$ 
    and  $xs': xs' \longrightarrow x' \bigwedge n. xs'\ n \in X$ 
    by (auto simp: closure_sequential)
  have  $\forall_F n \text{ in sequentially. dist } (xs'\ n) x' < d'$ 
    and  $\forall_F n \text{ in sequentially. dist } (xs\ n) x < d'$ 
    by (auto intro!:  $\langle 0 < d' \rangle$  order_tendstoD tendsto_eq_intros  $xs\ xs'$ )
  moreover
  have  $(\lambda x. f (xs\ x)) \longrightarrow y$  if  $x \in \text{closure } X$   $x \notin X$   $xs \longrightarrow x \bigwedge n. xs\ n \in X$ 
  for  $xs\ x$ 
    using that not_eventuallyD
    by (force intro!: filterlim_compose[OF  $y$ ][OF  $\langle x \in \text{closure } X \rangle$ ] simp: filterlim_at)
  then have  $(\lambda x. f (xs'\ x)) \longrightarrow ?g\ x' (\lambda x. f (xs\ x)) \longrightarrow ?g\ x$ 
    using  $x\ x'$ 
    by (auto intro!: continuous_on_tendsto_compose[OF cont_f] simp:  $xs'\ xs$ )
  then have  $\forall_F n \text{ in sequentially. dist } (f (xs'\ n)) (?g\ x') < e'$ 
    and  $\forall_F n \text{ in sequentially. dist } (f (xs\ n)) (?g\ x) < e'$ 
    by (auto intro!:  $\langle 0 < e' \rangle$  order_tendstoD tendsto_eq_intros)
  ultimately
  have  $\forall_F n \text{ in sequentially. dist } (?g\ x') (?g\ x) < e$ 
  proof eventually_elim
    case (elim  $n$ )
    have  $\text{dist } (?g\ x') (?g\ x) \leq$ 
       $\text{dist } (f (xs'\ n)) (?g\ x') + \text{dist } (f (xs'\ n)) (f (xs\ n)) + \text{dist } (f (xs\ n)) (?g\ x)$ 
      by (metis add.commute add_le_cancel_left dist_commute dist_triangle)
    also
    from  $\langle \text{dist } (xs'\ n) x' < d' \rangle \langle \text{dist } x' x < d' \rangle \langle \text{dist } (xs\ n) x < d' \rangle$ 
    have  $\text{dist } (xs'\ n) (xs\ n) < d$  unfolding  $d\_def$  by metric
    with  $\langle xs\_ \in X \rangle \langle xs'\_ \in X \rangle$  have  $\text{dist } (f (xs'\ n)) (f (xs\ n)) < e'$ 
      by (rule d)
    also note  $\langle \text{dist } (f (xs'\ n)) (?g\ x') < e' \rangle$ 
    also note  $\langle \text{dist } (f (xs\ n)) (?g\ x) < e' \rangle$ 
    finally show ?case by (simp add:  $e\_def$ )
  qed
  qed
  then show  $\text{dist } (?g\ x') (?g\ x) < e$  by simp
qed
qed
moreover have  $f\ x = ?g\ x$  if  $x \in X$  for  $x$  using that by simp
moreover
{
  fix  $Y\ h\ x$ 
  assume  $Y: x \in Y\ X \subseteq Y\ Y \subseteq \text{closure } X$  and  $\text{cont\_h}: \text{continuous\_on } Y\ h$ 

```

```

    and extension: ( $\bigwedge x. x \in X \implies f x = h x$ )
  {
    assume  $x \notin X$ 
    have  $x \in \text{closure } X$  using  $Y$  by auto
    then obtain  $xs$  where  $xs: xs \longrightarrow x \wedge n. xs\ n \in X$ 
      by (auto simp: closure_sequential)
    from continuous_on_tendsto_compose[OF cont_h xs(1)] xs(2)  $Y$ 
    have  $hx: (\lambda x. f (xs\ x)) \longrightarrow h\ x$ 
      by (auto simp: subsetD extension)
    then have  $(\lambda x. f (xs\ x)) \longrightarrow y\ x$ 
      using  $\langle x \notin X \rangle$  not_eventuallyD xs(2)
      by (force intro!: filterlim_compose[OF y[OF  $\langle x \in \text{closure } X \rangle$ ]] simp: filter-
lim_at xs)
    with  $hx$  have  $h\ x = y\ x$  by (rule LIMSEQ_unique)
  } then
  have  $h\ x = ?g\ x$ 
    using extension by auto
  }
  ultimately show ?thesis ..
qed

```

```

lemma bounded_uniformly_continuous_image:
  fixes  $f :: 'a :: \text{heine\_borel} \Rightarrow 'b :: \text{heine\_borel}$ 
  assumes uniformly_continuous_on  $S$   $f$  bounded  $S$ 
  shows bounded( $f\ ` S$ )
  by (metis (no_types, lifting) assms bounded_closure_image compact_closure
compact_continuous_image compact_eq_bounded_closed image_cong uniformly_continuous_imp_cont
uniformly_continuous_on_extension_on_closure)

```

3.2.25 With Abstract Topology (TODO: move and remove dependency?)

```

lemma openin_contains_ball:
  openin (top_of_set  $T$ )  $S \longleftrightarrow$ 
   $S \subseteq T \wedge (\forall x \in S. \exists e. 0 < e \wedge \text{ball } x\ e \cap T \subseteq S)$ 
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    apply (simp add: openin_open)
    apply (metis Int_commute Int_mono inf.cobounded2 open_contains_ball or-
der_refl subsetCE)
  done
next
  assume ?rhs
  then show ?lhs
    apply (simp add: openin_euclidean_subtopology_iff)
    by (metis (no_types) Int_iff dist_commute inf.absorb_iff2 mem_ball)
qed

```

```

lemma openin_contains_cball:
  openin (top_of_set T) S  $\longleftrightarrow$ 
     $S \subseteq T \wedge (\forall x \in S. \exists e. 0 < e \wedge \text{cball } x \ e \cap T \subseteq S)$ 
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (force simp add: openin_contains_ball intro: exI [where x=_/2])
next
  assume ?rhs
  then show ?lhs
    by (force simp add: openin_contains_ball)
qed

```

3.2.26 Closed Nest

Bounded closed nest property (proof does not use Heine-Borel)

```

lemma bounded_closed_nest:
  fixes S :: nat  $\Rightarrow$  ('a::heine_borel) set
  assumes  $\bigwedge n. \text{closed } (S \ n)$ 
    and  $\bigwedge n. S \ n \neq \{\}$ 
    and  $\bigwedge m \ n. m \leq n \implies S \ n \subseteq S \ m$ 
    and bounded (S 0)
  obtains a where  $\bigwedge n. a \in S \ n$ 
proof -
  from assms(2) obtain x where  $x: \forall n. x \ n \in S \ n$ 
    using choice[of  $\lambda n \ x. x \ n$ ] by auto
  from assms(4,1) have seq_compact (S 0)
    by (simp add: bounded_closed_imp_seq_compact)
  then obtain l r where  $lr: l \in S \ 0 \ \text{strict\_mono } r \ (x \circ r) \longrightarrow l$ 
    using x and assms(3) unfolding seq_compact_def by blast
  have  $\forall n. l \in S \ n$ 
proof
  fix n :: nat
  have closed (S n)
    using assms(1) by simp
  moreover have  $\forall i. (x \circ r) \ i \in S \ i$ 
    using x and assms(3) and lr(2) [THEN seq_suble] by auto
  then have  $\forall i. (x \circ r) \ (i + n) \in S \ n$ 
    using assms(3) by (fast intro!: le_add2)
  moreover have  $(\lambda i. (x \circ r) \ (i + n)) \longrightarrow l$ 
    using lr(3) by (rule LIMSEQ_ignore_initial_segment)
  ultimately show  $l \in S \ n$ 
    by (rule closed_sequentially)
qed
  then show ?thesis
    using that by blast
qed

```

Decreasing case does not even need compactness, just completeness.

```

lemma decreasing_closed_nest:
  fixes  $S :: \text{nat} \Rightarrow ('a::\text{complete\_space}) \text{ set}$ 
  assumes  $\bigwedge n. \text{closed } (S\ n)$ 
     $\bigwedge n. S\ n \neq \{\}$ 
     $\bigwedge m\ n. m \leq n \implies S\ n \subseteq S\ m$ 
     $\bigwedge e. e > 0 \implies \exists n. \forall x \in S\ n. \forall y \in S\ n. \text{dist } x\ y < e$ 
  obtains  $a$  where  $\bigwedge n. a \in S\ n$ 
proof -
  have  $\forall n. \exists x. x \in S\ n$ 
    using assms(2) by auto
  then have  $\exists t. \forall n. t\ n \in S\ n$ 
    using choice[of  $\lambda n\ x. x \in S\ n$ ] by auto
  then obtain  $t$  where  $t: \forall n. t\ n \in S\ n$  by auto
  {
    fix  $e :: \text{real}$ 
    assume  $e > 0$ 
    then obtain  $N$  where  $N: \forall x \in S\ N. \forall y \in S\ N. \text{dist } x\ y < e$ 
      using assms(4) by blast
    {
      fix  $m\ n :: \text{nat}$ 
      assume  $N \leq m \wedge N \leq n$ 
      then have  $t\ m \in S\ N \wedge t\ n \in S\ N$ 
        using assms(3)  $t$  unfolding subset_eq  $t$  by blast+
      then have  $\text{dist } (t\ m)\ (t\ n) < e$ 
        using  $N$  by auto
    }
    then have  $\exists N. \forall m\ n. N \leq m \wedge N \leq n \longrightarrow \text{dist } (t\ m)\ (t\ n) < e$ 
      by auto
  }
  then have Cauchy  $t$ 
    unfolding cauchy_def by auto
  then obtain  $l$  where  $l: (t \longrightarrow l)$  sequentially
    using complete_UNIV unfolding complete_def by auto
  { fix  $n :: \text{nat}$ 
    { fix  $e :: \text{real}$ 
      assume  $e > 0$ 
      then obtain  $N :: \text{nat}$  where  $N: \forall n \geq N. \text{dist } (t\ n)\ l < e$ 
        using  $l$ [unfolded lim_sequentially] by auto
      have  $t\ (\max\ n\ N) \in S\ n$ 
        by (meson assms(3) contra_subsetD max.cobounded1  $t$ )
      then have  $\exists y \in S\ n. \text{dist } y\ l < e$ 
        using  $N$  max.cobounded2 by blast
    }
  }
  then have  $l \in S\ n$ 
    using closed_approachable[of  $S\ n\ l$ ] assms(1) by auto
  }
then show ?thesis
  using that by blast

```

qed

Strengthen it to the intersection actually being a singleton.

```

lemma decreasing_closed_nest_sing:
  fixes  $S :: \text{nat} \Rightarrow 'a::\text{complete\_space set}$ 
  assumes  $\bigwedge n. \text{closed}(S\ n)$ 
            $\bigwedge n. S\ n \neq \{\}$ 
            $\bigwedge m\ n. m \leq n \implies S\ n \subseteq S\ m$ 
            $\bigwedge e. e > 0 \implies \exists n. \forall x \in (S\ n). \forall y \in (S\ n). \text{dist } x\ y < e$ 
  shows  $\exists a. \bigcap (\text{range } S) = \{a\}$ 
proof -
  obtain  $a$  where  $a: \forall n. a \in S\ n$ 
    using decreasing_closed_nest[of  $S$ ] using assms by auto
  { fix  $b$ 
    assume  $b: b \in \bigcap (\text{range } S)$ 
    { fix  $e :: \text{real}$ 
      assume  $e > 0$ 
      then have  $\text{dist } a\ b < e$ 
        using assms(4) and  $b$  and  $a$  by blast
    }
    then have  $\text{dist } a\ b = 0$ 
      by (metis dist_eq_0_iff dist_nz_less_le)
  }
  with  $a$  have  $\bigcap (\text{range } S) = \{a\}$ 
    unfolding image_def by auto
  then show ?thesis ..
qed

```

3.2.27 Making a continuous function avoid some value in a neighbourhood

```

lemma continuous_within_avoid:
  fixes  $f :: 'a::\text{metric\_space} \Rightarrow 'b::t1\_space$ 
  assumes continuous (at  $x$  within  $s$ )  $f$ 
           and  $f\ x \neq a$ 
  shows  $\exists e > 0. \forall y \in s. \text{dist } x\ y < e \implies f\ y \neq a$ 
proof -
  obtain  $U$  where open  $U$  and  $f\ x \in U$  and  $a \notin U$ 
    using t1_space [OF  $\langle f\ x \neq a \rangle$ ] by fast
  have  $(f \longrightarrow f\ x)$  (at  $x$  within  $s$ )
    using assms(1) by (simp add: continuous_within)
  then have eventually  $(\lambda y. f\ y \in U)$  (at  $x$  within  $s$ )
    using  $\langle \text{open } U \rangle$  and  $\langle f\ x \in U \rangle$ 
    unfolding tendsto_def by fast
  then have eventually  $(\lambda y. f\ y \neq a)$  (at  $x$  within  $s$ )
    using  $\langle a \notin U \rangle$  by (fast elim: eventually_mono)
  then show ?thesis
    using  $\langle f\ x \neq a \rangle$  by (auto simp: dist_commute eventually_at)
qed

```

```

lemma continuous_at_avoid:
  fixes  $f :: 'a::metric\_space \Rightarrow 'b::t1\_space$ 
  assumes continuous (at  $x$ )  $f$ 
  and  $f\ x \neq a$ 
  shows  $\exists e>0. \forall y. dist\ x\ y < e \longrightarrow f\ y \neq a$ 
  using assms continuous_within_avoid[of  $x\ UNIV\ f\ a$ ] by simp

lemma continuous_on_avoid:
  fixes  $f :: 'a::metric\_space \Rightarrow 'b::t1\_space$ 
  assumes continuous_on  $s\ f$ 
  and  $x \in s$ 
  and  $f\ x \neq a$ 
  shows  $\exists e>0. \forall y \in s. dist\ x\ y < e \longrightarrow f\ y \neq a$ 
  using assms(1)[unfolded continuous_on_eq_continuous_within, THEN bspec][where
 $x=x$ ],
  OF assms(2)] continuous_within_avoid[of  $x\ s\ f\ a$ ]
  using assms(3)
  by auto

lemma continuous_on_open_avoid:
  fixes  $f :: 'a::metric\_space \Rightarrow 'b::t1\_space$ 
  assumes continuous_on  $s\ f$ 
  and open  $s$ 
  and  $x \in s$ 
  and  $f\ x \neq a$ 
  shows  $\exists e>0. \forall y. dist\ x\ y < e \longrightarrow f\ y \neq a$ 
  using assms(1)[unfolded continuous_on_eq_continuous_at [OF assms(2)], THEN
bspec][where  $x=x$ ], OF assms(3)]
  using continuous_at_avoid[of  $x\ f\ a$ ] assms(4)
  by auto

```

3.2.28 Consequences for Real Numbers

```

lemma closed_contains_Inf:
  fixes  $S :: real\ set$ 
  shows  $S \neq \{\} \Longrightarrow bdd\_below\ S \Longrightarrow closed\ S \Longrightarrow Inf\ S \in S$ 
  by (metis closure_contains_Inf closure_closed)

lemma closed_subset_contains_Inf:
  fixes  $A\ C :: real\ set$ 
  shows  $closed\ C \Longrightarrow A \subseteq C \Longrightarrow A \neq \{\} \Longrightarrow bdd\_below\ A \Longrightarrow Inf\ A \in C$ 
  by (metis closure_contains_Inf closure_minimal subset_eq)

lemma closed_contains_Sup:
  fixes  $S :: real\ set$ 
  shows  $S \neq \{\} \Longrightarrow bdd\_above\ S \Longrightarrow closed\ S \Longrightarrow Sup\ S \in S$ 
  by (subst closure_closed[symmetric], assumption, rule closure_contains_Sup)

```


lemma *closed_subset_contains_Sup*:

fixes $A\ C :: \text{real set}$

shows $\text{closed } C \implies A \subseteq C \implies A \neq \{\} \implies \text{bdd\_above } A \implies \text{Sup } A \in C$

by (metis *closure_contains_Sup closure_minimal subset_eq*)

lemma *atLeastAtMost_subset_contains_Inf*:

fixes $A :: \text{real set}$ **and** $a\ b :: \text{real}$

shows $A \neq \{\} \implies a \leq b \implies A \subseteq \{a..b\} \implies \text{Inf } A \in \{a..b\}$

by (rule *closed_subset_contains_Inf*)

(auto intro: *closed_real_atLeastAtMost intro!*: *bdd_belowI*[of $A\ a$])

lemma *bounded_real*: $\text{bounded } (S :: \text{real set}) \longleftrightarrow (\exists a. \forall x \in S. |x| \leq a)$

by (simp add: *bounded_iff*)

lemma *bounded_imp_bdd_above*: $\text{bounded } S \implies \text{bdd\_above } (S :: \text{real set})$

by (auto simp: *bounded_def bdd_above_def dist_real_def*)

(metis *abs_le_D1 abs_minus_commute diff_le_eq*)

lemma *bounded_imp_bdd_below*: $\text{bounded } S \implies \text{bdd\_below } (S :: \text{real set})$

by (auto simp: *bounded_def bdd_below_def dist_real_def*)

(metis *abs_le_D1 add.commute diff_le_eq*)

lemma *bounded_has_Sup*:

fixes $S :: \text{real set}$

assumes *bounded* S

and $S \neq \{\}$

shows $\forall x \in S. x \leq \text{Sup } S$

and $\forall b. (\forall x \in S. x \leq b) \longrightarrow \text{Sup } S \leq b$

proof

show $\forall b. (\forall x \in S. x \leq b) \longrightarrow \text{Sup } S \leq b$

using *assms* **by** (metis *cSup_least*)

qed (metis *cSup_upper assms(1) bounded_imp_bdd_above*)

lemma *Sup_insert*:

fixes $S :: \text{real set}$

shows $\text{bounded } S \implies \text{Sup } (\text{insert } x\ S) = (\text{if } S = \{\} \text{ then } x \text{ else } \max\ x\ (\text{Sup } S))$

by (auto simp: *bounded_imp_bdd_above sup_max cSup_insert_If*)

lemma *bounded_has_Inf*:

fixes $S :: \text{real set}$

assumes *bounded* S

and $S \neq \{\}$

shows $\forall x \in S. x \geq \text{Inf } S$

and $\forall b. (\forall x \in S. x \geq b) \longrightarrow \text{Inf } S \geq b$

proof

show $\forall b. (\forall x \in S. x \geq b) \longrightarrow \text{Inf } S \geq b$

using *assms* **by** (metis *cInf_greatest*)

qed (metis *cInf_lower assms(1) bounded_imp_bdd_below*)

lemma *Inf_insert*:
fixes $S :: \text{real set}$
shows $\text{bounded } S \implies \text{Inf } (\text{insert } x \ S) = (\text{if } S = \{\} \text{ then } x \text{ else } \min x \ (\text{Inf } S))$
by (*auto simp: bounded_imp_bdd_below inf_min cInf_insert_If*)

lemma *open_real*:
fixes $s :: \text{real set}$
shows $\text{open } s \longleftrightarrow (\forall x \in s. \exists e > 0. \forall x'. |x' - x| < e \longrightarrow x' \in s)$
unfolding *open_dist dist_norm* **by** *simp*

lemma *islimpt_approachable_real*:
fixes $s :: \text{real set}$
shows $x \text{ islimpt } s \longleftrightarrow (\forall e > 0. \exists x' \in s. x' \neq x \wedge |x' - x| < e)$
unfolding *islimpt_approachable dist_norm* **by** *simp*

lemma *closed_real*:
fixes $s :: \text{real set}$
shows $\text{closed } s \longleftrightarrow (\forall x. (\forall e > 0. \exists x' \in s. x' \neq x \wedge |x' - x| < e) \longrightarrow x \in s)$
unfolding *closed_limpt islimpt_approachable dist_norm* **by** *simp*

lemma *continuous_at_real_range*:
fixes $f :: 'a :: \text{real\_normed\_vector} \Rightarrow \text{real}$
shows $\text{continuous } (\text{at } x) \ f \longleftrightarrow (\forall e > 0. \exists d > 0. \forall x'. \text{norm}(x' - x) < d \longrightarrow |f \ x' - f \ x| < e)$
unfolding *continuous_at*
unfolding *Lim_at*
unfolding *dist_norm*
apply *auto*
apply (*erule_tac x=e in allE, auto*)
apply (*rule_tac x=d in exI, auto*)
apply (*erule_tac x=x' in allE, auto*)
apply (*erule_tac x=e in allE, auto*)
done

lemma *continuous_on_real_range*:
fixes $f :: 'a :: \text{real\_normed\_vector} \Rightarrow \text{real}$
shows $\text{continuous\_on } s \ f \longleftrightarrow$
 $(\forall x \in s. \forall e > 0. \exists d > 0. (\forall x' \in s. \text{norm}(x' - x) < d \longrightarrow |f \ x' - f \ x| < e))$
unfolding *continuous_on_iff dist_norm* **by** *simp*

lemma *continuous_on_closed_Collect_le*:
fixes $f \ g :: 'a :: \text{topological\_space} \Rightarrow \text{real}$
assumes $f: \text{continuous\_on } s \ f$ **and** $g: \text{continuous\_on } s \ g$ **and** $s: \text{closed } s$
shows $\text{closed } \{x \in s. f \ x \leq g \ x\}$

proof –
have $\text{closed } ((\lambda x. g \ x - f \ x) - \{0.. \} \cap s)$
using *closed_real_atLeast continuous_on_diff [OF g f]*
by (*simp add: continuous_on_closed_vimage [OF s]*)
also have $((\lambda x. g \ x - f \ x) - \{0.. \} \cap s) = \{x \in s. f \ x \leq g \ x\}$

```

    by auto
  finally show ?thesis .
qed

```

```

lemma continuous_le_on_closure:
  fixes a::real
  assumes f: continuous_on (closure s) f
    and x: x ∈ closure(s)
    and xlo:  $\bigwedge x. x \in s \implies f(x) \leq a$ 
  shows f(x) ≤ a
  using image_closure_subset [OF f, where T = {x. x ≤ a}] assms
    continuous_on_closed_Collect_le[of UNIV λx. x λx. a]
  by auto

```

```

lemma continuous_ge_on_closure:
  fixes a::real
  assumes f: continuous_on (closure s) f
    and x: x ∈ closure(s)
    and xlo:  $\bigwedge x. x \in s \implies f(x) \geq a$ 
  shows f(x) ≥ a
  using image_closure_subset [OF f, where T = {x. a ≤ x}] assms
    continuous_on_closed_Collect_le[of UNIV λx. a λx. x]
  by auto

```

3.2.29 The infimum of the distance between two sets

definition *setdist* :: 'a::metric_space set $\Rightarrow$ 'a set $\Rightarrow$ real **where**

```

  setdist s t  $\equiv$ 
    (if s = {}  $\vee$  t = {} then 0
     else Inf {dist x y | x y. x ∈ s  $\wedge$  y ∈ t})

```

```

lemma setdist_empty1 [simp]: setdist {} t = 0
  by (simp add: setdist_def)

```

```

lemma setdist_empty2 [simp]: setdist t {} = 0
  by (simp add: setdist_def)

```

```

lemma setdist_pos_le [simp]: 0 ≤ setdist s t
  by (auto simp: setdist_def ex_in_conv [symmetric] intro: cInf_greatest)

```

```

lemma le_setdistI:
  assumes s ≠ {} t ≠ {}  $\wedge$  x y.  $\llbracket x \in s; y \in t \rrbracket \implies d \leq \text{dist } x \ y$ 
  shows d ≤ setdist s t
  using assms
  by (auto simp: setdist_def Set.ex_in_conv [symmetric] intro: cInf_greatest)

```

```

lemma setdist_le_dist:  $\llbracket x \in s; y \in t \rrbracket \implies \text{setdist } s \ t \leq \text{dist } x \ y$ 
  unfolding setdist_def
  by (auto intro!: bdd_belowI [where m=0] cInf_lower)

```

```

lemma le_setdist_iff:
   $d \leq \text{setdist } S \ T \iff$ 
   $(\forall x \in S. \forall y \in T. d \leq \text{dist } x \ y) \wedge (S = \{\} \vee T = \{\} \implies d \leq 0)$ 
  apply (cases  $S = \{\} \vee T = \{\}$ )
  apply (force simp add: setdist_def)
  apply (intro iffI conjI)
  using setdist_le_dist apply fastforce
  apply (auto simp: intro: le_setdistI)
  done

lemma setdist_ltE:
  assumes  $\text{setdist } S \ T < b \ S \neq \{\} \ T \neq \{\}$ 
  obtains  $x \ y$  where  $x \in S \ y \in T \ \text{dist } x \ y < b$ 
using assms
by (auto simp: not_le [symmetric] le_setdist_iff)

lemma setdist_refl:  $\text{setdist } S \ S = 0$ 
  apply (cases  $S = \{\}$ )
  apply (force simp add: setdist_def)
  apply (rule antisym [OF _ setdist_pos_le])
  apply (metis all_not_in_conv dist_self setdist_le_dist)
  done

lemma setdist_sym:  $\text{setdist } S \ T = \text{setdist } T \ S$ 
  by (force simp: setdist_def dist_commute intro!: arg_cong [where f=Inf])

lemma setdist_triangle:  $\text{setdist } S \ T \leq \text{setdist } S \ \{a\} + \text{setdist } \{a\} \ T$ 
proof (cases  $S = \{\} \vee T = \{\}$ )
  case True then show ?thesis
    using setdist_pos_le by fastforce
next
  case False
  then have  $\bigwedge x. x \in S \implies \text{setdist } S \ T - \text{dist } x \ a \leq \text{setdist } \{a\} \ T$ 
    apply (intro le_setdistI)
    apply (simp_all add: algebra_simps)
    apply (metis dist_commute dist_triangle3 order_trans [OF setdist_le_dist])
    done
  then have  $\text{setdist } S \ T - \text{setdist } \{a\} \ T \leq \text{setdist } S \ \{a\}$ 
    using False by (fastforce intro: le_setdistI)
  then show ?thesis
    by (simp add: algebra_simps)
qed

lemma setdist_singletons [simp]:  $\text{setdist } \{x\} \ \{y\} = \text{dist } x \ y$ 
  by (simp add: setdist_def)

lemma setdist_Lipschitz:  $|\text{setdist } \{x\} \ S - \text{setdist } \{y\} \ S| \leq \text{dist } x \ y$ 
  apply (subst setdist_singletons [symmetric])

```

```

by (metis abs_diff_le_iff diff_le_eq setdist_triangle setdist_sym)

lemma continuous_at_setdist [continuous_intros]: continuous (at x) ( $\lambda y. (setdist \{y\} S)$ )
  by (force simp: continuous_at_eps_delta dist_real_def intro: le_less_trans [OF setdist_Lipschitz])

lemma continuous_on_setdist [continuous_intros]: continuous_on T ( $\lambda y. (setdist \{y\} S)$ )
  by (metis continuous_at_setdist continuous_at_imp_continuous_on)

lemma uniformly_continuous_on_setdist: uniformly_continuous_on T ( $\lambda y. (setdist \{y\} S)$ )
  by (force simp: uniformly_continuous_on_def dist_real_def intro: le_less_trans [OF setdist_Lipschitz])

lemma setdist_subset_right:  $\llbracket T \neq \{\}; T \subseteq u \rrbracket \implies setdist S u \leq setdist S T$ 
  apply (cases  $S = \{\} \vee u = \{\}$ , force)
  apply (auto simp: setdist_def intro!: bdd_belowI [where m=0] cInf_superset_mono)
  done

lemma setdist_subset_left:  $\llbracket S \neq \{\}; S \subseteq T \rrbracket \implies setdist T u \leq setdist S u$ 
  by (metis setdist_subset_right setdist_sym)

lemma setdist_closure_1 [simp]:  $setdist (closure S) T = setdist S T$ 
proof (cases  $S = \{\} \vee T = \{\}$ )
  case True then show ?thesis by force
next
  case False
  { fix y
    assume  $y \in T$ 
    have continuous_on (closure S) ( $\lambda a. dist a y$ )
      by (auto simp: continuous_intros dist_norm)
    then have *:  $\bigwedge x. x \in closure S \implies setdist S T \leq dist x y$ 
      by (fast intro: setdist_le_dist ‹ $y \in T$ › continuous_ge_on_closure)
  } note * = this
show ?thesis
  apply (rule antisym)
  using False closure_subset apply (blast intro: setdist_subset_left)
  using False * apply (force intro!: le_setdistI)
  done
qed

lemma setdist_closure_2 [simp]:  $setdist T (closure S) = setdist T S$ 
  by (metis setdist_closure_1 setdist_sym)

lemma setdist_eq_0I:  $\llbracket x \in S; x \in T \rrbracket \implies setdist S T = 0$ 
  by (metis antisym dist_self setdist_le_dist setdist_pos_le)

```

lemma *setdist_unique*:

$\llbracket a \in S; b \in T; \bigwedge x y. x \in S \wedge y \in T \implies \text{dist } a \ b \leq \text{dist } x \ y \rrbracket$
 $\implies \text{setdist } S \ T = \text{dist } a \ b$
by (force simp add: setdist_le_dist le_setdist_iff intro: antisym)

lemma *setdist_le_sing*: $x \in S \implies \text{setdist } S \ T \leq \text{setdist } \{x\} \ T$
using setdist_subset_left **by** auto

lemma *infdist_eq_setdist*: $\text{infdist } x \ A = \text{setdist } \{x\} \ A$
by (simp add: infdist_def setdist_def Setcompr_eq_image)

lemma *setdist_eq_infdist*: $\text{setdist } A \ B = (\text{if } A = \{\} \text{ then } 0 \text{ else } \text{INF } a \in A. \text{ infdist } a \ B)$

proof –

have $\text{Inf } \{\text{dist } x \ y \mid x y. x \in A \wedge y \in B\} = (\text{INF } x \in A. \text{ Inf } (\text{dist } x \text{ ‘ } B))$

if $b \in B$ $a \in A$ **for** $a \ b$

proof (rule order_antisym)

have $\text{Inf } \{\text{dist } x \ y \mid x y. x \in A \wedge y \in B\} \leq \text{Inf } (\text{dist } x \text{ ‘ } B)$

if $b \in B$ $a \in A$ $x \in A$ **for** x

proof –

have *: $\bigwedge b'. b' \in B \implies \text{Inf } \{\text{dist } x \ y \mid x y. x \in A \wedge y \in B\} \leq \text{dist } x \ b'$

by (metis (mono_tags, lifting) ex_in_conv setdist_def setdist_le_dist

that(3))

show ?thesis

using that **by** (subst conditionally_complete_lattice_class.le_cInf_iff) (auto

simp: *)+

qed

then show $\text{Inf } \{\text{dist } x \ y \mid x y. x \in A \wedge y \in B\} \leq (\text{INF } x \in A. \text{ Inf } (\text{dist } x \text{ ‘ } B))$

using that

by (subst conditionally_complete_lattice_class.le_cInf_iff) (auto simp: bdd_below_def)

next

have *: $\bigwedge x y. \llbracket b \in B; a \in A; x \in A; y \in B \rrbracket \implies \exists a \in A. \text{ Inf } (\text{dist } a \text{ ‘ } B) \leq \text{dist } x \ y$

by (meson bdd_below_image_dist cINF_lower)

show $(\text{INF } x \in A. \text{ Inf } (\text{dist } x \text{ ‘ } B)) \leq \text{Inf } \{\text{dist } x \ y \mid x y. x \in A \wedge y \in B\}$

proof (rule conditionally_complete_lattice_class.cInf_mono)

show bdd_below $((\lambda x. \text{ Inf } (\text{dist } x \text{ ‘ } B)) \text{ ‘ } A)$

by (metis (no_types, lifting) bdd_belowI2 ex_in_conv infdist_def infdist_nonneg that(1))

qed (use that in <auto simp: *>)

qed

then show ?thesis

by (auto simp: setdist_def infdist_def)

qed

lemma *infdist_mono*:

assumes $A \subseteq B$ $A \neq \{\}$

shows $\text{infdist } x \ B \leq \text{infdist } x \ A$

by (simp add: asms infdist_eq_setdist setdist_subset_right)

```

lemma infdist_singleton [simp]:
  infdist  $x$   $\{y\} = \text{dist } x \ y$ 
  by (simp add: infdist_eq_setdist)

proposition setdist_attains_inf:
  assumes compact  $B \neq \{\}$ 
  obtains  $y$  where  $y \in B$  setdist  $A \ B = \text{infdist } y \ A$ 
proof (cases  $A = \{\}$ )
  case True
  then show thesis
    by (metis assms diameter_compact_attained infdist_def setdist_def that)
next
  case False
  obtain  $y$  where  $y \in B$  and min:  $\bigwedge y'. y' \in B \implies \text{infdist } y \ A \leq \text{infdist } y' \ A$ 
  by (metis continuous_attains_inf [OF assms continuous_on_infdist] continuous_on_id)
  show thesis
  proof
    have setdist  $A \ B = (\text{INF } y \in B. \text{infdist } y \ A)$ 
    by (metis  $\langle B \neq \{\} \rangle$  setdist_eq_infdist setdist_sym)
    also have  $\dots = \text{infdist } y \ A$ 
    proof (rule order_antisym)
      show  $(\text{INF } y \in B. \text{infdist } y \ A) \leq \text{infdist } y \ A$ 
      proof (rule cInf_lower)
        show  $\text{infdist } y \ A \in (\lambda y. \text{infdist } y \ A) \ ` \ B$ 
        using  $\langle y \in B \rangle$  by blast
        show bdd_below  $((\lambda y. \text{infdist } y \ A) \ ` \ B)$ 
        by (meson bdd_belowI2 infdist_nonneg)
      qed
    next
      show  $\text{infdist } y \ A \leq (\text{INF } y \in B. \text{infdist } y \ A)$ 
      by (simp add:  $\langle B \neq \{\} \rangle$  cINF_greatest min)
    qed
    finally show setdist  $A \ B = \text{infdist } y \ A$  .
  qed (fact  $\langle y \in B \rangle$ )
qed

end

```

3.3 Elementary Normed Vector Spaces

```

theory Elementary_Normed_Spaces
  imports
    HOL-Library.FuncSet
    Elementary_Metric_Spaces Cartesian_Space
    Connected
begin

```

3.3.1 Orthogonal Transformation of Balls

3.3.2 Various Lemmas Combining Imports

```

lemma open_sums:
  fixes T :: ('b::real_normed_vector) set
  assumes open S  $\vee$  open T
  shows open ( $\bigcup x \in S. \bigcup y \in T. \{x + y\}$ )
  using assms
proof
  assume S: open S
  show ?thesis
  proof (clarsimp simp: open_dist)
    fix x y
    assume x  $\in$  S y  $\in$  T
    with S obtain e where e > 0 and e:  $\bigwedge x'. \text{dist } x' x < e \implies x' \in S$ 
    by (auto simp: open_dist)
    then have  $\bigwedge z. \text{dist } z (x + y) < e \implies \exists x \in S. \exists y \in T. z = x + y$ 
    by (metis  $\langle y \in T \rangle \text{diff\_add\_cancel dist\_add\_cancel2}$ )
    then show  $\exists e > 0. \forall z. \text{dist } z (x + y) < e \longrightarrow (\exists x \in S. \exists y \in T. z = x + y)$ 
    using  $\langle 0 < e \rangle \langle x \in S \rangle$  by blast
  qed
next
  assume T: open T
  show ?thesis
  proof (clarsimp simp: open_dist)
    fix x y
    assume x  $\in$  S y  $\in$  T
    with T obtain e where e > 0 and e:  $\bigwedge x'. \text{dist } x' y < e \implies x' \in T$ 
    by (auto simp: open_dist)
    then have  $\bigwedge z. \text{dist } z (x + y) < e \implies \exists x \in S. \exists y \in T. z = x + y$ 
    by (metis  $\langle x \in S \rangle \text{add\_diff\_cancel left' add\_diff\_eq diff\_diff\_add dist\_norm}$ )
    then show  $\exists e > 0. \forall z. \text{dist } z (x + y) < e \longrightarrow (\exists x \in S. \exists y \in T. z = x + y)$ 
    using  $\langle 0 < e \rangle \langle y \in T \rangle$  by blast
  qed
qed

lemma image_orthogonal_transformation_ball:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'a
  assumes orthogonal_transformation f
  shows f ' ball x r = ball (f x) r
proof (intro equalityI subsetI)
  fix y assume y  $\in$  f ' ball x r
  with assms show y  $\in$  ball (f x) r
  by (auto simp: orthogonal_transformation_isometry)
next
  fix y assume y: y  $\in$  ball (f x) r
  then obtain z where z: y = f z
  using assms orthogonal_transformation_surj by blast
  with y assms show y  $\in$  f ' ball x r

```


by (auto simp: orthogonal_transformation_isometry)
qed

lemma *image_orthogonal_transformation_cball*:
 fixes $f :: 'a::euclidean\_space \Rightarrow 'a$
 assumes *orthogonal_transformation* f
 shows $f \text{ ` } cball\ x\ r = cball\ (f\ x)\ r$
proof (*intro equalityI subsetI*)
 fix y assume $y \in f \text{ ` } cball\ x\ r$
 with *assms* show $y \in cball\ (f\ x)\ r$
 by (auto simp: orthogonal_transformation_isometry)
next
 fix y assume $y: y \in cball\ (f\ x)\ r$
 then obtain z where $z: y = f\ z$
 using *assms* *orthogonal_transformation_surj* by blast
 with *y assms* show $y \in f \text{ ` } cball\ x\ r$
 by (auto simp: orthogonal_transformation_isometry)
qed

3.3.3 Support

definition (*in monoid_add*) *support_on* :: $'b\ set \Rightarrow ('b \Rightarrow 'a) \Rightarrow 'b\ set$
 where *support_on* $S\ f = \{x \in S. f\ x \neq 0\}$

lemma *in_support_on*: $x \in support\_on\ S\ f \longleftrightarrow x \in S \wedge f\ x \neq 0$
 by (*simp add: support_on_def*)

lemma *support_on_simps*[*simp*]:
 $support\_on\ \{\}\ f = \{\}$
 $support\_on\ (insert\ x\ S)\ f =$
 (*if* $f\ x = 0$ *then* $support\_on\ S\ f$ *else* $insert\ x\ (support\_on\ S\ f)$)
 $support\_on\ (S \cup T)\ f = support\_on\ S\ f \cup support\_on\ T\ f$
 $support\_on\ (S \cap T)\ f = support\_on\ S\ f \cap support\_on\ T\ f$
 $support\_on\ (S - T)\ f = support\_on\ S\ f - support\_on\ T\ f$
 $support\_on\ (f \text{ ` } S)\ g = f \text{ ` } (support\_on\ S\ (g \circ f))$
unfolding *support_on_def* **by** *auto*

lemma *support_on_cong*:
 $(\bigwedge x. x \in S \Longrightarrow f\ x = 0 \longleftrightarrow g\ x = 0) \Longrightarrow support\_on\ S\ f = support\_on\ S\ g$
by (*auto simp: support_on_def*)

lemma *support_on_if*: $a \neq 0 \Longrightarrow support\_on\ A\ (\lambda x. \text{if } P\ x \text{ then } a \text{ else } 0) =$
 $\{x \in A. P\ x\}$
by (*auto simp: support_on_def*)

lemma *support_on_if_subset*: $support\_on\ A\ (\lambda x. \text{if } P\ x \text{ then } a \text{ else } 0) \subseteq \{x \in A. P\ x\}$
by (*auto simp: support_on_def*)

lemma *finite_support*[intro]: *finite* $S \implies \text{finite } (\text{support\_on } S f)$
unfolding *support_on_def* **by** *auto*

definition (**in** *comm_monoid_add*) *supp_sum* :: $('b \Rightarrow 'a) \Rightarrow 'b \text{ set} \Rightarrow 'a$
where *supp_sum* $f S = (\sum_{x \in \text{support\_on } S f} f x)$

lemma *supp_sum_empty*[simp]: *supp_sum* $f \{\} = 0$
unfolding *supp_sum_def* **by** *auto*

lemma *supp_sum_insert*[simp]:
finite (support_on S f) $\implies$
supp_sum f (insert x S) = (if $x \in S$ then supp_sum f S else $f x + \text{supp\_sum } f$
S)
by (*simp add: supp_sum_def in_support_on insert_absorb*)

lemma *supp_sum_divide_distrib*: *supp_sum* $f A / (r :: 'a :: \text{field}) = \text{supp\_sum } (\lambda n.$
 $f n / r) A$
by (*cases r = 0*)
(auto simp: supp_sum_def sum_divide_distrib intro!: sum.cong support_on_cong)

3.3.4 Intervals

lemma *image_affinity_interval*:
fixes $c :: 'a :: \text{ordered\_real\_vector}$
shows $((\lambda x. m *_R x + c) ' \{a..b\}) =$
 $(\text{if } \{a..b\} = \{\} \text{ then } \{\}$
 $\text{else if } 0 \leq m \text{ then } \{m *_R a + c .. m *_R b + c\}$
 $\text{else } \{m *_R b + c .. m *_R a + c\})$
 $(\text{is } ?lhs = ?rhs)$
proof (*cases m=0*)
case *True*
then show *?thesis*
by *force*
next
case *False*
show *?thesis*
proof
show $?lhs \subseteq ?rhs$
by (*auto simp: scaleR_left_mono scaleR_left_mono_neg*)
show $?rhs \subseteq ?lhs$
proof (*clarsimp, intro conjI impI subsetI*)
show $\llbracket 0 \leq m; a \leq b; x \in \{m *_R a + c .. m *_R b + c\} \rrbracket$
 $\implies x \in (\lambda x. m *_R x + c) ' \{a..b\}$ **for** x
using *False*
by (*rule_tac x=inverse m *_R (x-c) in image_eqI*)
(auto simp: pos_le_divideR_eq pos_divideR_le_eq le_diff_eq diff_le_eq)
show $\llbracket \neg 0 \leq m; a \leq b; x \in \{m *_R b + c .. m *_R a + c\} \rrbracket$
 $\implies x \in (\lambda x. m *_R x + c) ' \{a..b\}$ **for** x

```

    by (rule_tac x=inverse m *R (x-c) in image_eqI)
      (auto simp add: neg_le_divideR_eq neg_divideR_le_eq le_diff_eq
diff_le_eq)
  qed
qed
qed

```

3.3.5 Limit Points

```

lemma islimpt_ball:
  fixes x y :: 'a::{real_normed_vector,perfect_space}
  shows y islimpt ball x e  $\longleftrightarrow$  0 < e  $\wedge$  y  $\in$  cball x e
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  show ?rhs if ?lhs
  proof
    {
      assume e  $\leq$  0
      then have *: ball x e = {}
      using ball_eq_empty[of x e] by auto
      have False using <?lhs>
      unfolding * using islimpt_EMPTY[of y] by auto
    }
    then show e > 0 by (metis not_less)
  show y  $\in$  cball x e
    using closed_cball[of x e] islimpt_subset[of y ball x e cball x e]
      ball_subset_cball[of x e] <?lhs>
    unfolding closed_limpt by auto
  qed
  show ?lhs if ?rhs
  proof -
    from that have e > 0 by auto
    {
      fix d :: real
      assume d > 0
      have  $\exists x' \in \text{ball } x \text{ e. } x' \neq y \wedge \text{dist } x' y < d$ 
      proof (cases d  $\leq$  dist x y)
      case True
      then show ?thesis
      proof (cases x = y)
      case True
      then have False
        using <d  $\leq$  dist x y> <d > 0> by auto
      then show ?thesis
        by auto
      next
      case False
      have dist x (y - (d / (2 * dist y x)) *R (y - x)) =
        norm (x - y + (d / (2 * norm (y - x))) *R (y - x))
    }
  end

```

```

unfolding mem_cball mem_ball dist_norm diff_diff_eq2 diff_add_eq[symmetric]
  by auto
also have ... = |- 1 + d / (2 * norm (x - y))| * norm (x - y)
  using scaleR_left_distrib[of - 1 d / (2 * norm (y - x)), symmetric, of
y - x]
  unfolding scaleR_minus_left scaleR_one
  by (auto simp: norm_minus_commute)
also have ... = |- norm (x - y) + d / 2|
  unfolding abs_mult_pos[of norm (x - y), OF norm_ge_zero[of x - y]]
  unfolding distrib_right using ⟨x≠y⟩ by auto
also have ... ≤ e - d/2 using ⟨d ≤ dist x y⟩ and ⟨d>0⟩ and ⟨?rhs⟩
  by (auto simp: dist_norm)
finally have y - (d / (2 * dist y x)) *R (y - x) ∈ ball x e using ⟨d>0⟩
  by auto
moreover
have (d / (2*dist y x)) *R (y - x) ≠ 0
  using ⟨x≠y⟩[unfolded dist_nz] ⟨d>0⟩ unfolding scaleR_eq_0_iff
  by (auto simp: dist_commute)
moreover
have dist (y - (d / (2 * dist y x)) *R (y - x)) y < d
  using ⟨0 < d⟩ by (fastforce simp: dist_norm)
ultimately show ?thesis
  by (rule_tac x = y - (d / (2*dist y x)) *R (y - x) in bexI) auto
qed
next
case False
then have d > dist x y by auto
show ∃ x' ∈ ball x e. x' ≠ y ∧ dist x' y < d
proof (cases x = y)
case True
obtain z where z: z ≠ y dist z y < min e d
  using perfect_choose_dist[of min e d y]
  using ⟨d > 0⟩ ⟨e>0⟩ by auto
show ?thesis
  by (metis True z dist_commute mem_ball min_less_iff_conj)
next
case False
then show ?thesis
  using ⟨d>0⟩ ⟨d > dist x y⟩ ⟨?rhs⟩ by force
qed
qed
}
then show ?thesis
  unfolding mem_cball islimpt_approachable mem_ball by auto
qed
qed

lemma closure_ball_lemma:
  fixes x y :: 'a::real_normed_vector

```

```

    assumes  $x \neq y$ 
    shows  $y \text{ islimpt ball } x \text{ (dist } x \text{ } y)$ 
  proof (rule islimptI)
    fix  $T$ 
    assume  $y \in T$  open  $T$ 
    then obtain  $r$  where  $0 < r \ \forall z. \text{dist } z \ y < r \longrightarrow z \in T$ 
      unfolding open_dist by fast
    — choose point between  $x$  and  $y$ , within distance  $r$  of  $y$ .
    define  $k$  where  $k = \min 1 \ (r / (2 * \text{dist } x \ y))$ 
    define  $z$  where  $z = y + \text{scaleR } k \ (x - y)$ 
    have  $z\_def2: z = x + \text{scaleR } (1 - k) \ (y - x)$ 
      unfolding  $z\_def$  by (simp add: algebra_simps)
    have  $\text{dist } z \ y < r$ 
      unfolding  $z\_def \ k\_def$  using  $\langle 0 < r \rangle$ 
      by (simp add: dist_norm min_def)
    then have  $z \in T$ 
      using  $\langle \forall z. \text{dist } z \ y < r \longrightarrow z \in T \rangle$  by simp
    have  $\text{dist } x \ z < \text{dist } x \ y$ 
      using  $\langle 0 < r \rangle$  assms by (simp add:  $z\_def2 \ k\_def \text{dist\_norm norm\_minus\_commute}$ )

    then have  $z \in \text{ball } x \text{ (dist } x \text{ } y)$ 
      by simp
    have  $z \neq y$ 
      unfolding  $z\_def \ k\_def$  using  $\langle x \neq y \rangle \langle 0 < r \rangle$ 
      by (simp add: min_def)
    show  $\exists z \in \text{ball } x \text{ (dist } x \text{ } y). z \in T \wedge z \neq y$ 
      using  $\langle z \in \text{ball } x \text{ (dist } x \text{ } y) \rangle \langle z \in T \rangle \langle z \neq y \rangle$ 
      by fast
  qed

```

3.3.6 Balls and Spheres in Normed Spaces

```

lemma mem_ball_0 [simp]:  $x \in \text{ball } 0 \ e \longleftrightarrow \text{norm } x < e$ 
  for  $x :: 'a::\text{real\_normed\_vector}$ 
  by simp

```

```

lemma mem_cball_0 [simp]:  $x \in \text{cball } 0 \ e \longleftrightarrow \text{norm } x \leq e$ 
  for  $x :: 'a::\text{real\_normed\_vector}$ 
  by simp

```

```

lemma closure_ball [simp]:
  fixes  $x :: 'a::\text{real\_normed\_vector}$ 
  assumes  $0 < e$ 
  shows  $\text{closure } (\text{ball } x \ e) = \text{cball } x \ e$ 
proof
  show  $\text{closure } (\text{ball } x \ e) \subseteq \text{cball } x \ e$ 
    using closed_cball closure_minimal by blast
  have  $\bigwedge y. \text{dist } x \ y < e \vee \text{dist } x \ y = e \implies y \in \text{closure } (\text{ball } x \ e)$ 
    by (metis Un_iff assms closure_ball_lemma closure_def dist_eq_0_iff mem_Collect_eq)

```

```

mem_ball)
  then show cball x e  $\subseteq$  closure (ball x e)
    by force
qed

```

```

lemma mem_sphere_0 [simp]:  $x \in \text{sphere } 0 \ e \longleftrightarrow \text{norm } x = e$ 
  for  $x :: 'a :: \text{real\_normed\_vector}$ 
  by simp

```

```

lemma interior_cball [simp]:
  fixes  $x :: 'a :: \{\text{real\_normed\_vector}, \text{perfect\_space}\}$ 
  shows interior (cball x e) = ball x e
proof (cases  $e \geq 0$ )
  case False note cs = this
  from cs have null: ball x e = {}
    using ball_empty[of e x] by auto
  moreover
  have cball x e = {}
  proof (rule equals0I)
    fix y
    assume  $y \in \text{cball } x \ e$ 
    then show False
  by (metis ball_eq_empty null cs dist_eq_0_iff dist_le_zero_iff empty_subsetI
mem_cball
subset_antisym subset_ball)
  qed
  then have interior (cball x e) = {}
    using interior_empty by auto
  ultimately show ?thesis by blast
next
  case True note cs = this
  have ball x e  $\subseteq$  cball x e
    using ball_subset_cball by auto
  moreover
  {
    fix  $S \ y$ 
    assume as:  $S \subseteq \text{cball } x \ e$  open S  $y \in S$ 
    then obtain d where  $d > 0$  and  $d: \forall x'. \text{dist } x' \ y < d \longrightarrow x' \in S$ 
      unfolding open_dist by blast
    then obtain xa where  $xa\_y: xa \neq y$  and  $xa: \text{dist } xa \ y < d$ 
      using perfect_choose_dist [of d] by auto
    have  $xa \in S$ 
      using d[THEN spec[where  $x = xa$ ]]
      using xa by (auto simp: dist_commute)
    then have  $xa\_cball: xa \in \text{cball } x \ e$ 
      using as(1) by auto
    then have  $y \in \text{ball } x \ e$ 
    proof (cases  $x = y$ )

```

```

    case True
    then have  $e > 0$  using cs order.order_iff_strict xa_cball xa_y by fastforce
    then show  $y \in \text{ball } x \ e$ 
      using  $\langle x = y \rangle$  by simp
    next
    case False
    have  $\text{dist } (y + (d / 2 / \text{dist } y \ x) *_R (y - x)) \ y < d$ 
      unfolding dist_norm
      using  $\langle d > 0 \rangle$  norm_ge_zero[of  $y - x$ ]  $\langle x \neq y \rangle$  by auto
    then have  $*: y + (d / 2 / \text{dist } y \ x) *_R (y - x) \in \text{cball } x \ e$ 
      using  $d \text{ as } (1)[\text{unfolded subset\_eq}]$  by blast
    have  $y - x \neq 0$  using  $\langle x \neq y \rangle$  by auto
    hence  $**: d / (2 * \text{norm } (y - x)) > 0$ 
      unfolding zero_less_norm_iff[symmetric] using  $\langle d > 0 \rangle$  by auto
    have  $\text{dist } (y + (d / 2 / \text{dist } y \ x) *_R (y - x)) \ x =$ 
       $\text{norm } (y + (d / (2 * \text{norm } (y - x))) *_R y - (d / (2 * \text{norm } (y - x))) *_R$ 
 $x - x)$ 
      by (auto simp: dist_norm algebra_simps)
    also have  $\dots = \text{norm } ((1 + d / (2 * \text{norm } (y - x))) *_R (y - x))$ 
      by (auto simp: algebra_simps)
    also have  $\dots = |1 + d / (2 * \text{norm } (y - x))| * \text{norm } (y - x)$ 
      using  $**$  by auto
    also have  $\dots = (\text{dist } y \ x) + d/2$ 
      using  $**$  by (auto simp: distrib_right dist_norm)
    finally have  $e \geq \text{dist } x \ y + d/2$ 
      using  $*[\text{unfolded mem\_cball}]$  by (auto simp: dist_commute)
    then show  $y \in \text{ball } x \ e$ 
      unfolding mem_ball using  $\langle d > 0 \rangle$  by auto
  qed
}
then have  $\forall S \subseteq \text{cball } x \ e. \text{open } S \longrightarrow S \subseteq \text{ball } x \ e$ 
  by auto
ultimately show ?thesis
  using interior_unique[of  $\text{ball } x \ e \ \text{cball } x \ e$ ]
  using open_ball[of  $x \ e$ ]
  by auto
qed

```

```

lemma frontier_ball [simp]:
  fixes  $a :: 'a::\text{real\_normed\_vector}$ 
  shows  $0 < e \implies \text{frontier } (\text{ball } a \ e) = \text{sphere } a \ e$ 
  by (force simp: frontier_def)

```

```

lemma frontier_cball [simp]:
  fixes  $a :: 'a::\{\text{real\_normed\_vector}, \text{perfect\_space}\}$ 
  shows  $\text{frontier } (\text{cball } a \ e) = \text{sphere } a \ e$ 
  by (force simp: frontier_def)

```

```

corollary compact_sphere [simp]:

```

```

    fixes a :: 'a::{real_normed_vector,perfect_space,heine_borel}
    shows compact (sphere a r)
using compact_frontier [of cball a r] by simp

```

```

corollary bounded_sphere [simp]:
  fixes a :: 'a::{real_normed_vector,perfect_space,heine_borel}
  shows bounded (sphere a r)
by (simp add: compact_imp_bounded)

```

```

corollary closed_sphere [simp]:
  fixes a :: 'a::{real_normed_vector,perfect_space,heine_borel}
  shows closed (sphere a r)
by (simp add: compact_imp_closed)

```

```

lemma image_add_ball [simp]:
  fixes a :: 'a::real_normed_vector
  shows (+) b ' ball a r = ball (a+b) r
proof -
  { fix x :: 'a
    assume dist (a + b) x < r
    moreover
    have b + (x - b) = x
    by simp
    ultimately have x ∈ (+) b ' ball a r
    by (metis add_commute dist_add_cancel image_eqI mem_ball) }
  then show ?thesis
  by (auto simp: add_commute)
qed

```

```

lemma image_add_cball [simp]:
  fixes a :: 'a::real_normed_vector
  shows (+) b ' cball a r = cball (a+b) r
proof -
  have  $\bigwedge x. \text{dist } (a + b) x \leq r \implies \exists y \in \text{cball } a r. x = b + y$ 
  by (metis (no_types) add_commute diff_add_cancel dist_add_cancel2 mem_cball)
  then show ?thesis
  by (force simp: add_commute)
qed

```

3.3.7 Various Lemmas on Normed Algebras

```

lemma closed_of_nat_image: closed (of_nat ' A :: 'a::real_normed_algebra_1
set)
  by (rule discrete_imp_closed[of 1]) (auto simp: dist_of_nat)

```

```

lemma closed_of_int_image: closed (of_int ' A :: 'a::real_normed_algebra_1
set)
  by (rule discrete_imp_closed[of 1]) (auto simp: dist_of_int)

```


lemma *closed_Nats* [simp]: *closed* ($\mathbb{N} :: 'a :: \text{real\_normed\_algebra\_1}$ set)
unfolding *Nats_def* **by** (rule *closed_of_nat_image*)

lemma *closed_Ints* [simp]: *closed* ($\mathbb{Z} :: 'a :: \text{real\_normed\_algebra\_1}$ set)
unfolding *Ints_def* **by** (rule *closed_of_int_image*)

lemma *closed_subset_Ints*:
fixes $A :: 'a :: \text{real\_normed\_algebra\_1}$ set
assumes $A \subseteq \mathbb{Z}$
shows *closed* A
proof (intro *discrete_imp_closed*[OF *zero_less_one*] *ballI* *impI*, *goal_cases*)
case (1 $x\ y$)
with *assms* **have** $x \in \mathbb{Z}$ **and** $y \in \mathbb{Z}$ **by** *auto*
with $\langle \text{dist } y\ x < 1 \rangle$ **show** $y = x$
by (*auto elim!*: *Ints_cases* *simp*: *dist_of_int*)
qed

3.3.8 Filters

definition *indirection* :: $'a :: \text{real\_normed\_vector} \Rightarrow 'a \Rightarrow 'a$ filter (**infixr** *indirection* 70)
where a *indirection* $v = \text{at } a$ *within* $\{b. \exists c \geq 0. b - a = \text{scaleR } c\ v\}$

3.3.9 Trivial Limits

lemma *trivial_limit_at_infinity*:
 $\neg$ *trivial_limit* (*at_infinity* :: $('a :: \{\text{real\_normed\_vector}, \text{perfect\_space}\})$ filter)
proof –
obtain $x :: 'a$ **where** $x \neq 0$
by (*meson* *perfect_choose_dist* *zero_less_one*)
then have $b \leq \text{norm } ((b / \text{norm } x) *_{\mathbb{R}} x)$ **for** b
by *simp*
then show ?thesis
unfolding *trivial_limit_def* *eventually_at_infinity*
by *blast*
qed

lemma *at_within_ball_bot_iff*:
fixes $x\ y :: 'a :: \{\text{real\_normed\_vector}, \text{perfect\_space}\}$
shows *at* x *within* *ball* $y\ r = \text{bot} \iff (r = 0 \vee x \notin \text{cball } y\ r)$
unfolding *trivial_limit_within*
by (*metis* (*no_types*) *cball_empty_equals0D* *islimpt_ball* *less_linear*)

3.3.10 Limits

proposition *Lim_at_infinity*: $(f \longrightarrow l)$ *at_infinity* $\iff (\forall e > 0. \exists b. \forall x. \text{norm } x \geq b \implies \text{dist } (f\ x)\ l < e)$
by (*auto simp*: *tendsto_iff_eventually_at_infinity*)

corollary *Lim_at_infinityI* [*intro?*]:

```

    assumes  $\bigwedge e. e > 0 \implies \exists B. \forall x. \text{norm } x \geq B \longrightarrow \text{dist } (f \ x) \ l \leq e$ 
    shows  $(f \longrightarrow l) \text{ at\_infinity}$ 
  proof -
    have  $\bigwedge e. e > 0 \implies \exists B. \forall x. \text{norm } x \geq B \longrightarrow \text{dist } (f \ x) \ l < e$ 
      by (meson assms dense le_less_trans)
    then show ?thesis
      using Lim_at_infinity by blast
  qed

lemma Lim_transform_within_set_eq:
  fixes  $a :: 'a::\text{metric\_space}$  and  $l :: 'b::\text{metric\_space}$ 
  shows eventually  $(\lambda x. x \in S \longleftrightarrow x \in T) \text{ (at } a\text{)}$ 
     $\implies ((f \longrightarrow l) \text{ (at } a \text{ within } S) \longleftrightarrow (f \longrightarrow l) \text{ (at } a \text{ within } T))$ 
  by (force intro: Lim_transform_within_set elim: eventually_mono)

lemma Lim_null:
  fixes  $f :: 'a \Rightarrow 'b::\text{real\_normed\_vector}$ 
  shows  $(f \longrightarrow l) \text{ net} \longleftrightarrow ((\lambda x. f(x) - l) \longrightarrow 0) \text{ net}$ 
  by (simp add: Lim_dist_norm)

lemma Lim_null_comparison:
  fixes  $f :: 'a \Rightarrow 'b::\text{real\_normed\_vector}$ 
  assumes eventually  $(\lambda x. \text{norm } (f \ x) \leq g \ x) \text{ net } (g \longrightarrow 0) \text{ net}$ 
  shows  $(f \longrightarrow 0) \text{ net}$ 
  using assms(2)
proof (rule metric_tendsto_imp_tendsto)
  show eventually  $(\lambda x. \text{dist } (f \ x) \ 0 \leq \text{dist } (g \ x) \ 0) \text{ net}$ 
    using assms(1) by (rule eventually_mono) (simp add: dist_norm)
qed

lemma Lim_transform_bound:
  fixes  $f :: 'a \Rightarrow 'b::\text{real\_normed\_vector}$ 
  and  $g :: 'a \Rightarrow 'c::\text{real\_normed\_vector}$ 
  assumes eventually  $(\lambda n. \text{norm } (f \ n) \leq \text{norm } (g \ n)) \text{ net}$ 
  and  $(g \longrightarrow 0) \text{ net}$ 
  shows  $(f \longrightarrow 0) \text{ net}$ 
  using assms(1) tendsto_norm_zero [OF assms(2)]
  by (rule Lim_null_comparison)

lemma lim_null_mult_right_bounded:
  fixes  $f :: 'a \Rightarrow 'b::\text{real\_normed\_div\_algebra}$ 
  assumes  $f: (f \longrightarrow 0) \ F$  and  $g: \text{eventually } (\lambda x. \text{norm}(g \ x) \leq B) \ F$ 
  shows  $((\lambda z. f \ z * g \ z) \longrightarrow 0) \ F$ 
proof -
  have  $((\lambda x. \text{norm } (f \ x) * \text{norm } (g \ x)) \longrightarrow 0) \ F$ 
proof (rule Lim_null_comparison)
  show  $\forall_F x \text{ in } F. \text{norm } (\text{norm } (f \ x) * \text{norm } (g \ x)) \leq \text{norm } (f \ x) * B$ 
    by (simp add: eventually_mono [OF g] mult_left_mono)
  show  $((\lambda x. \text{norm } (f \ x) * B) \longrightarrow 0) \ F$ 

```

```

    by (simp add: f tendsto_mult_left_zero tendsto_norm_zero)
  qed
  then show ?thesis
    by (subst tendsto_norm_zero_iff [symmetric]) (simp add: norm_mult)
  qed

```

```

lemma lim_null_mult_left_bounded:
  fixes f :: 'a  $\Rightarrow$  'b::real_normed_div_algebra
  assumes g: eventually ( $\lambda x. \text{norm}(g\ x) \leq B$ ) F and f: ( $f \longrightarrow 0$ ) F
  shows (( $\lambda z. g\ z * f\ z \longrightarrow 0$ ) F)
proof -
  have (( $\lambda x. \text{norm}(g\ x) * \text{norm}(f\ x) \longrightarrow 0$ ) F)
  proof (rule Lim_null_comparison)
    show  $\forall_F x \text{ in } F. \text{norm}(\text{norm}(g\ x) * \text{norm}(f\ x)) \leq B * \text{norm}(f\ x)$ 
      by (simp add: eventually_mono [OF g] mult_right_mono)
    show (( $\lambda x. B * \text{norm}(f\ x) \longrightarrow 0$ ) F)
      by (simp add: f tendsto_mult_right_zero tendsto_norm_zero)
  qed
  then show ?thesis
    by (subst tendsto_norm_zero_iff [symmetric]) (simp add: norm_mult)
  qed

```

```

lemma lim_null_scaleR_bounded:
  assumes f: ( $f \longrightarrow 0$ ) net and gB: eventually ( $\lambda a. f\ a = 0 \vee \text{norm}(g\ a) \leq B$ )
  net
  shows (( $\lambda n. f\ n *_{\mathbb{R}} g\ n \longrightarrow 0$ ) net)
proof
  fix  $\varepsilon::\text{real}$ 
  assume  $0 < \varepsilon$ 
  then have B:  $0 < \varepsilon / (|B| + 1)$  by simp
  have *:  $|f\ x| * \text{norm}(g\ x) < \varepsilon$  if f:  $|f\ x| * (|B| + 1) < \varepsilon$  and g:  $\text{norm}(g\ x) \leq B$  for x
  proof -
    have  $|f\ x| * \text{norm}(g\ x) \leq |f\ x| * B$ 
      by (simp add: mult_left_mono g)
    also have  $\dots \leq |f\ x| * (|B| + 1)$ 
      by (simp add: mult_left_mono)
    also have  $\dots < \varepsilon$ 
      by (rule f)
    finally show ?thesis .
  qed
  have  $\bigwedge x. [|f\ x| < \varepsilon / (|B| + 1); \text{norm}(g\ x) \leq B] \implies |f\ x| * \text{norm}(g\ x) < \varepsilon$ 
    by (simp add: * pos_less_divide_eq)
  then show  $\forall_F x \text{ in } \text{net}. \text{dist}(f\ x *_{\mathbb{R}} g\ x)\ 0 < \varepsilon$ 
    using  $\langle 0 < \varepsilon \rangle$  by (auto intro: eventually_mono [OF eventually_conj [OF tendstoD [OF f B] gB]])
  qed

```

```

lemma Lim_norm_ubound:

```

```

fixes f :: 'a  $\Rightarrow$  'b::real_normed_vector
assumes  $\neg(\text{trivial\_limit\_net})$  (f  $\longrightarrow$  l) net eventually ( $\lambda x. \text{norm}(f\ x) \leq e$ ) net
shows  $\text{norm}(l) \leq e$ 
using assms by (fast intro: tendsto_le tendsto_intros)

```

```

lemma Lim_norm_lbound:
  fixes f :: 'a  $\Rightarrow$  'b::real_normed_vector
  assumes  $\neg \text{trivial\_limit\_net}$ 
    and (f  $\longrightarrow$  l) net
    and eventually ( $\lambda x. e \leq \text{norm}(f\ x)$ ) net
  shows  $e \leq \text{norm}\ l$ 
  using assms by (fast intro: tendsto_le tendsto_intros)

```

Limit under bilinear function

```

lemma Lim_bilinear:
  assumes (f  $\longrightarrow$  l) net
    and (g  $\longrightarrow$  m) net
    and bounded_bilinear h
  shows (( $\lambda x. h\ (f\ x)\ (g\ x)$ )  $\longrightarrow$  (h l m)) net
  using <bounded_bilinear h> <(f  $\longrightarrow$  l) net> <(g  $\longrightarrow$  m) net>
  by (rule bounded_bilinear.tendsto)

```

```

lemma Lim_at_zero:
  fixes a :: 'a::real_normed_vector
    and l :: 'b::topological_space
  shows (f  $\longrightarrow$  l) (at a)  $\longleftrightarrow$  (( $\lambda x. f(a + x)$ )  $\longrightarrow$  l) (at 0)
  using LIM_offset_zero LIM_offset_zero_cancel ..

```

3.3.11 Limit Point of Filter

```

lemma netlimit_at_vector:
  fixes a :: 'a::real_normed_vector
  shows netlimit (at a) = a
proof (cases  $\exists x. x \neq a$ )
  case True then obtain x where x:  $x \neq a$  ..
  have  $\bigwedge d. 0 < d \implies \exists x. x \neq a \wedge \text{norm}(x - a) < d$ 
    by (rule_tac x=a + scaleR (d / 2) (sgn (x - a)) in exI) (simp add: norm_sgn
sgn_zero_iff x)
  then have  $\neg \text{trivial\_limit}$  (at a)
    by (auto simp: trivial_limit_def eventually_at dist_norm)
  then show ?thesis
    by (rule Lim_ident_at [of a UNIV])
qed simp

```

3.3.12 Boundedness

```

lemma continuous_on_closure_norm_le:
  fixes f :: 'a::metric_space  $\Rightarrow$  'b::real_normed_vector
  assumes continuous_on (closure s) f

```

```

    and  $\forall y \in s. \text{norm}(f y) \leq b$ 
    and  $x \in (\text{closure } s)$ 
  shows  $\text{norm } (f x) \leq b$ 
proof -
  have *:  $f ' s \subseteq \text{cball } 0 b$ 
    using  $\text{assms}(2)[\text{unfolded mem\_cball\_0}[\text{symmetric}]]$  by auto
  show ?thesis
    by (meson *  $\text{assms}(1)$   $\text{assms}(3)$   $\text{closed\_cball image\_closure\_subset image\_subset\_iff mem\_cball\_0}$ )
qed

```

```

lemma bounded_pos:  $\text{bounded } S \longleftrightarrow (\exists b > 0. \forall x \in S. \text{norm } x \leq b)$ 
  unfolding bounded_iff
  by (meson less_imp_le not_le order_trans zero_less_one)

```

```

lemma bounded_pos_less:  $\text{bounded } S \longleftrightarrow (\exists b > 0. \forall x \in S. \text{norm } x < b)$ 
  by (metis bounded_pos le_less_trans less_imp_le linordered_field_no_ub)

```

```

lemma Bseq_eq_bounded:
  fixes  $f :: \text{nat} \Rightarrow 'a::\text{real\_normed\_vector}$ 
  shows  $Bseq f \longleftrightarrow \text{bounded } (\text{range } f)$ 
  unfolding Bseq_def bounded_pos by auto

```

```

lemma bounded_linear_image:
  assumes bounded S
    and bounded_linear f
  shows bounded (f ' S)
proof -
  from  $\text{assms}(1)$  obtain  $b$  where  $b > 0$  and  $b: \forall x \in S. \text{norm } x \leq b$ 
    unfolding bounded_pos by auto
  from  $\text{assms}(2)$  obtain  $B$  where  $B: B > 0 \ \forall x. \text{norm } (f x) \leq B * \text{norm } x$ 
    using bounded_linear.pos_bounded by (auto simp: ac_simps)
  show ?thesis
    unfolding bounded_pos
  proof (intro exI, safe)
    show  $\text{norm } (f x) \leq B * b$  if  $x \in S$  for  $x$ 
      by (meson B b less_imp_le mult_left_mono order_trans that)
    qed (use  $\langle b > 0 \rangle \langle B > 0 \rangle$  in auto)
  qed

```

```

lemma bounded_scaling:
  fixes  $S :: 'a::\text{real\_normed\_vector}$  set
  shows  $\text{bounded } S \implies \text{bounded } ((\lambda x. c *_{\text{R}} x) ' S)$ 
  by (simp add: bounded_linear_image bounded_linear_scaleR_right)

```

```

lemma bounded_scaleR_comp:
  fixes  $f :: 'a \Rightarrow 'b::\text{real\_normed\_vector}$ 
  assumes bounded (f ' S)
  shows  $\text{bounded } ((\lambda x. r *_{\text{R}} f x) ' S)$ 

```

using *bounded_scaling*[*of f ' S r*] *assms*
by (*auto simp: image_image*)

lemma *bounded_translation*:

fixes *S* :: '*a*::*real_normed_vector* set

assumes *bounded S*

shows *bounded (($\lambda x. a + x$) ' *S*)*

proof –

from *assms* **obtain** *b* **where** *b* : *b* > 0 $\forall x \in S. \text{norm } x \leq b$

unfolding *bounded_pos* **by** *auto*

{

fix *x*

assume *x* $\in S$

then have *norm (a + x) $\leq b + \text{norm } a$*

using *norm_triangle_ineq*[*of a x*] *b* **by** *auto*

}

then show *?thesis*

unfolding *bounded_pos*

using *norm_ge_zero*[*of a*] *b(1)* **and** *add_strict_increasing*[*of b 0 norm a*]

by (*auto intro!: exI*[*of _ b + norm a*])

qed

lemma *bounded_translation_minus*:

fixes *S* :: '*a*::*real_normed_vector* set

shows *bounded S $\implies$ bounded (($\lambda x. x - a$) ' *S*)*

using *bounded_translation* [*of S -a*] **by** *simp*

lemma *bounded_uminus* [*simp*]:

fixes *X* :: '*a*::*real_normed_vector* set

shows *bounded (uminus ' X) $\longleftrightarrow$ bounded X*

by (*auto simp: bounded_def dist_norm; rule_tac x = -x in exI; force simp: add commute norm_minus_commute*)

lemma *uminus_bounded_comp* [*simp*]:

fixes *f* :: '*a* $\Rightarrow$ '*b*::*real_normed_vector*

shows *bounded (($\lambda x. - f x$) ' *S*) $\longleftrightarrow$ bounded (f ' *S*)*

using *bounded_uminus*[*of f ' S*]

by (*auto simp: image_image*)

lemma *bounded_plus_comp*:

fixes *f g* :: '*a* $\Rightarrow$ '*b*::*real_normed_vector*

assumes *bounded (f ' S)*

assumes *bounded (g ' S)*

shows *bounded (($\lambda x. f x + g x$) ' *S*)*

proof –

{

fix *B C*

assume $\bigwedge x. x \in S \implies \text{norm } (f x) \leq B$ $\bigwedge x. x \in S \implies \text{norm } (g x) \leq C$

then have $\bigwedge x. x \in S \implies \text{norm } (f x + g x) \leq B + C$

```

      by (auto intro!: norm_triangle_le add_mono)
    } then show ?thesis
      using assms by (fastforce simp: bounded_iff)
qed

```

```

lemma bounded_plus:
  fixes S :: 'a::real_normed_vector set
  assumes bounded S bounded T
  shows bounded (( $\lambda(x,y). x + y$ ) ' ( $S \times T$ ))
  using bounded_plus_comp [of fst S  $\times$  T snd] assms
  by (auto simp: split_def split: if_split_asm)

```

```

lemma bounded_minus_comp:
  bounded (f ' S)  $\implies$  bounded (g ' S)  $\implies$  bounded (( $\lambda x. f x - g x$ ) ' S)
  for f g :: 'a  $\Rightarrow$  'b::real_normed_vector
  using bounded_plus_comp [of f S  $\lambda x. - g x$ ]
  by auto

```

```

lemma bounded_minus:
  fixes S :: 'a::real_normed_vector set
  assumes bounded S bounded T
  shows bounded (( $\lambda(x,y). x - y$ ) ' ( $S \times T$ ))
  using bounded_minus_comp [of fst S  $\times$  T snd] assms
  by (auto simp: split_def split: if_split_asm)

```

```

lemma bounded_sums:
  fixes S :: 'a::real_normed_vector set
  assumes bounded S and bounded T
  shows bounded ( $\bigcup x \in S. \bigcup y \in T. \{x + y\}$ )
  using assms by (simp add: bounded_iff) (meson norm_triangle_mono)

```

```

lemma bounded_differences:
  fixes S :: 'a::real_normed_vector set
  assumes bounded S and bounded T
  shows bounded ( $\bigcup x \in S. \bigcup y \in T. \{x - y\}$ )
  using assms by (simp add: bounded_iff) (meson add_mono norm_triangle_le_diff)

```

```

lemma not_bounded_UNIV[simp]:
   $\neg$  bounded (UNIV :: 'a::{real_normed_vector, perfect_space} set)

```

```

proof (auto simp: bounded_pos not_le)
  obtain x :: 'a where  $x \neq 0$ 
    using perfect_choose_dist [OF zero_less_one] by fast
  fix b :: real
  assume b:  $b > 0$ 
  have b1:  $b + 1 \geq 0$ 
    using b by simp
  with  $\langle x \neq 0 \rangle$  have  $b < \text{norm} (\text{scaleR } (b + 1) (\text{sgn } x))$ 
    by (simp add: norm_sgn)
  then show  $\exists x :: 'a. b < \text{norm } x$  ..

```

qed

corollary *cobounded_imp_unbounded*:
 fixes $S :: 'a::\{\text{real\_normed\_vector}, \text{perfect\_space}\}$ set
 shows $\text{bounded } (-S) \implies \neg \text{bounded } S$
 using *bounded_Un [of S -S]* **by** (*simp*)

3.3.13 Relations among convergence and absolute convergence for power series

lemma *summable_imp_bounded*:
 fixes $f :: \text{nat} \Rightarrow 'a::\text{real\_normed\_vector}$
 shows $\text{summable } f \implies \text{bounded } (\text{range } f)$
by (*frule summable_LIMSEQ_zero*) (*simp add: convergent_imp_bounded*)

lemma *summable_imp_sums_bounded*:
 $\text{summable } f \implies \text{bounded } (\text{range } (\lambda n. \text{sum } f \{..<n\}))$
by (*auto simp: summable_def sums_def dest: convergent_imp_bounded*)

lemma *power_series_conv_imp_absconv_weak*:
 fixes $a :: \text{nat} \Rightarrow 'a::\{\text{real\_normed\_div\_algebra}, \text{banach}\}$ **and** $w :: 'a$
 assumes $\text{sum: summable } (\lambda n. a\ n * z^{\wedge} n)$ **and** $\text{no: norm } w < \text{norm } z$
 shows $\text{summable } (\lambda n. \text{of\_real}(\text{norm}(a\ n)) * w^{\wedge} n)$
proof –
 obtain M **where** $M: \bigwedge x. \text{norm } (a\ x * z^{\wedge} x) \leq M$
 using *summable_imp_bounded [OF sum]* **by** (*force simp: bounded_iff*)
 show ?thesis
proof (*rule series_comparison_complex*)
 have $\bigwedge n. \text{norm } (a\ n) * \text{norm } z^{\wedge} n \leq M$
by (*metis (no_types) M norm_mult norm_power*)
 then show $\text{summable } (\lambda n. \text{complex\_of\_real } (\text{norm } (a\ n) * \text{norm } w^{\wedge} n))$
 using *Abel_lemma no_norm_ge_zero summable_of_real* **by** *blast*
 qed (*auto simp: norm_mult norm_power*)
 qed

3.3.14 Normed spaces with the Heine-Borel property

lemma *not_compact_UNIV[simp]*:
 fixes $s :: 'a::\{\text{real\_normed\_vector}, \text{perfect\_space}, \text{heine\_borel}\}$ set
 shows $\neg \text{compact } (\text{UNIV}::'a \text{ set})$
by (*simp add: compact_eq_bounded_closed*)

lemma *not_compact_space_euclideanreal [simp]*: $\neg \text{compact\_space euclideanreal}$
by (*simp add: compact_space_def*)

Representing sets as the union of a chain of compact sets.

lemma *closed_Union_compact_subsets*:
 fixes $S :: 'a::\{\text{heine\_borel}, \text{real\_normed\_vector}\}$ set
 assumes *closed S*


```

obtains  $F$  where  $\bigwedge n. \text{compact}(F\ n) \wedge n. F\ n \subseteq S \wedge n. F\ n \subseteq F(\text{Suc } n)$ 
 $(\bigcup n. F\ n) = S \wedge K. [\text{compact } K; K \subseteq S] \implies \exists N. \forall n \geq N. K \subseteq$ 
 $F\ n$ 
proof
  show  $\text{compact } (S \cap \text{cball } 0\ (\text{of\_nat } n))$  for  $n$ 
    using assms compact_eq_bounded_closed by auto
next
  show  $(\bigcup n. S \cap \text{cball } 0\ (\text{real } n)) = S$ 
    by (auto simp: real_arch_simple)
next
  fix  $K :: 'a\ \text{set}$ 
  assume  $\text{compact } K\ K \subseteq S$ 
  then obtain  $N$  where  $K \subseteq \text{cball } 0\ N$ 
    by (meson bounded_pos mem_cball_0 compact_imp_bounded subsetI)
  then show  $\exists N. \forall n \geq N. K \subseteq S \cap \text{cball } 0\ (\text{real } n)$ 
    by (metis of_nat_le_iff Int_subset_iff <K ⊆ S> real_arch_simple subset_cball
subset_trans)
qed auto

```

3.3.15 Intersecting chains of compact sets and the Baire property

```

proposition bounded_closed_chain:
  fixes  $\mathcal{F} :: 'a::\text{heine\_borel}\ \text{set}\ \text{set}$ 
  assumes  $B \in \mathcal{F}$  bounded  $B$  and  $\mathcal{F}: \bigwedge S. S \in \mathcal{F} \implies \text{closed } S$  and  $\{\} \notin \mathcal{F}$ 
    and chain:  $\bigwedge S\ T. S \in \mathcal{F} \wedge T \in \mathcal{F} \implies S \subseteq T \vee T \subseteq S$ 
  shows  $\bigcap \mathcal{F} \neq \{\}$ 
proof –
  have  $B \cap \bigcap \mathcal{F} \neq \{\}$ 
  proof (rule compact_imp_fip)
    show  $\text{compact } B \wedge T. T \in \mathcal{F} \implies \text{closed } T$ 
      by (simp_all add: assms compact_eq_bounded_closed)
    show  $[\text{finite } \mathcal{G}; \mathcal{G} \subseteq \mathcal{F}] \implies B \cap \bigcap \mathcal{G} \neq \{\}$  for  $\mathcal{G}$ 
    proof (induction  $\mathcal{G}$  rule: finite_induct)
      case empty
        with assms show ?case by force
      next
        case (insert  $U\ \mathcal{G}$ )
        then have  $U \in \mathcal{F}$  and ne:  $B \cap \bigcap \mathcal{G} \neq \{\}$  by auto
        then consider  $B \subseteq U \mid U \subseteq B$ 
          using  $\langle B \in \mathcal{F} \rangle$  chain by blast
        then show ?case
        proof cases
          case 1
            then show ?thesis
              using Int_left_commute ne by auto
          next
            case 2
            have  $U \neq \{\}$ 

```

```

      using  $\langle U \in \mathcal{F} \rangle \langle \{\} \notin \mathcal{F} \rangle$  by blast
    moreover
    have False if  $\bigwedge x. x \in U \implies \exists Y \in \mathcal{G}. x \notin Y$ 
    proof -
      have  $\bigwedge x. x \in U \implies \exists Y \in \mathcal{G}. Y \subseteq U$ 
      by (metis chain contra_subsetD insert.premis insert_subset that)
    then obtain Y where  $Y \in \mathcal{G} \ Y \subseteq U$ 
    by (metis all_not_in_conv  $\langle U \neq \{\} \rangle$ )
    moreover obtain x where  $x \in \bigcap \mathcal{G}$ 
    by (metis Int_emptyI ne)
    ultimately show ?thesis
    by (metis Inf_lower subset_eq that)
  qed
with 2 show ?thesis
by blast
qed
qed
qed
then show ?thesis by blast
qed

corollary compact_chain:
  fixes  $\mathcal{F} :: 'a::heine\_borel \text{ set set}$ 
  assumes  $\bigwedge S. S \in \mathcal{F} \implies \text{compact } S \ \{\} \notin \mathcal{F}$ 
     $\bigwedge S \ T. S \in \mathcal{F} \wedge T \in \mathcal{F} \implies S \subseteq T \vee T \subseteq S$ 
  shows  $\bigcap \mathcal{F} \neq \{\}$ 
proof (cases  $\mathcal{F} = \{\}$ )
  case True
  then show ?thesis by auto
next
  case False
  show ?thesis
  by (metis False all_not_in_conv assms compact_imp_bounded compact_imp_closed
    bounded_closed_chain)
qed

lemma compact_nest:
  fixes  $F :: 'a::linorder \Rightarrow 'b::heine\_borel \text{ set}$ 
  assumes  $F: \bigwedge n. \text{compact}(F \ n) \ \bigwedge n. F \ n \neq \{\}$  and mono:  $\bigwedge m \ n. m \leq n \implies F \ n \subseteq F \ m$ 
  shows  $\bigcap (\text{range } F) \neq \{\}$ 
proof -
  have *:  $\bigwedge S \ T. S \in \text{range } F \wedge T \in \text{range } F \implies S \subseteq T \vee T \subseteq S$ 
  by (metis mono image_iff le_cases)
  show ?thesis
  using F by (intro compact_chain [OF _ _ *]; blast dest: *)
qed

```

The Baire property of dense sets

```

theorem Baire:
  fixes  $S::'a::\{\text{real\_normed\_vector, heine\_borel}\}$  set
  assumes  $\text{closed } S$   $\text{countable } \mathcal{G}$ 
    and  $\text{ope}: \bigwedge T. T \in \mathcal{G} \implies \text{openin } (\text{top\_of\_set } S) \ T \wedge S \subseteq \text{closure } T$ 
  shows  $S \subseteq \text{closure}(\bigcap \mathcal{G})$ 
proof (cases  $\mathcal{G} = \{\}$ )
  case True
    then show ?thesis
      using  $\text{closure\_subset}$  by auto
  next
    let ?g =  $\text{from\_nat\_into } \mathcal{G}$ 
    case False
      then have  $\text{gin}: ?g \ n \in \mathcal{G}$  for  $n$ 
        by ( $\text{simp add: from\_nat\_into}$ )
      show ?thesis
        proof ( $\text{clarsimp simp: closure\_approachable}$ )
          fix  $x$  and  $e::\text{real}$ 
          assume  $x \in S$   $0 < e$ 
          obtain  $TF$  where  $\text{opeF}: \bigwedge n. \text{openin } (\text{top\_of\_set } S) \ (TF \ n)$ 
            and  $\text{ne}: \bigwedge n. TF \ n \neq \{\}$ 
            and  $\text{subg}: \bigwedge n. S \cap \text{closure}(TF \ n) \subseteq ?g \ n$ 
            and  $\text{subball}: \bigwedge n. \text{closure}(TF \ n) \subseteq \text{ball } x \ e$ 
            and  $\text{decr}: \bigwedge n. TF(Suc \ n) \subseteq TF \ n$ 
          proof –
            have *:  $\exists Y. (\text{openin } (\text{top\_of\_set } S) \ Y \wedge Y \neq \{\} \wedge$ 
               $S \cap \text{closure } Y \subseteq ?g \ n \wedge \text{closure } Y \subseteq \text{ball } x \ e) \wedge Y \subseteq U$ 
            if  $\text{opeU}: \text{openin } (\text{top\_of\_set } S) \ U$  and  $U \neq \{\}$  and  $\text{cloU}: \text{closure } U \subseteq \text{ball}$ 
               $x \ e$  for  $U \ n$ 
              proof –
                obtain  $T$  where  $T: \text{open } T \ U = T \cap S$ 
                  using  $\langle \text{openin } (\text{top\_of\_set } S) \ U \rangle$  by ( $\text{auto simp: openin\_subtopology}$ )
                with  $\langle U \neq \{\} \rangle$  have  $T \cap \text{closure } (?g \ n) \neq \{\}$ 
                  using  $\text{gin ope}$  by  $\text{fastforce}$ 
                then have  $T \cap ?g \ n \neq \{\}$ 
                  using  $\langle \text{open } T \rangle \text{ open\_Int\_closure\_eq\_empty}$  by  $\text{blast}$ 
                then obtain  $y$  where  $y \in U$   $y \in ?g \ n$ 
                  using  $T \text{ ope } [\text{of } ?g \ n, OF \ \text{gin}]$  by ( $\text{blast dest: openin\_imp\_subset}$ )
                moreover have  $\text{openin } (\text{top\_of\_set } S) \ (U \cap ?g \ n)$ 
                  using  $\text{gin ope opeU}$  by  $\text{blast}$ 
                ultimately obtain  $d$  where  $U: U \cap ?g \ n \subseteq S$  and  $d > 0$  and  $d: \text{ball } y \ d$ 
                   $\cap S \subseteq U \cap ?g \ n$ 
                  by ( $\text{force simp: openin\_contains\_ball}$ )
                show ?thesis
                  proof ( $\text{intro exI conjI}$ )
                    show  $\text{openin } (\text{top\_of\_set } S) \ (S \cap \text{ball } y \ (d/2))$ 
                      by ( $\text{simp add: openin\_open\_Int}$ )
                    show  $S \cap \text{ball } y \ (d/2) \neq \{\}$ 
                      using  $\langle 0 < d \rangle \langle y \in U \rangle \text{ opeU openin\_imp\_subset}$  by  $\text{fastforce}$ 
                    have  $S \cap \text{closure } (S \cap \text{ball } y \ (d/2)) \subseteq S \cap \text{closure } (\text{ball } y \ (d/2))$ 

```

```

    using closure_mono by blast
  also have ...  $\subseteq$  ?g n
    using <d > 0> d by force
  finally show  $S \cap \text{closure } (S \cap \text{ball } y \ (d/2)) \subseteq$  ?g n .
  have closure (S  $\cap$  ball y (d/2))  $\subseteq$  S  $\cap$  ball y d
  proof -
    have closure (ball y (d/2))  $\subseteq$  ball y d
    using <d > 0> by auto
    then have closure (S  $\cap$  ball y (d/2))  $\subseteq$  ball y d
    by (meson closure_mono inf.cobounded2 subset_trans)
    then show ?thesis
    by (simp add: <closed S> closure_minimal)
  qed
  also have ...  $\subseteq$  ball x e
    using cloU closure_subset d by blast
  finally show closure (S  $\cap$  ball y (d/2))  $\subseteq$  ball x e .
  show S  $\cap$  ball y (d/2)  $\subseteq$  U
    using ball_divide_subset_numeral d by blast
  qed
  qed
  let ? $\Phi$  =  $\lambda n \ X. \text{openin } (\text{top\_of\_set } S) \ X \wedge X \neq \{\} \wedge$ 
     $S \cap \text{closure } X \subseteq$  ?g n  $\wedge \text{closure } X \subseteq$  ball x e
  have closure (S  $\cap$  ball x (e/2))  $\subseteq$  closure(ball x (e/2))
    by (simp add: closure_mono)
  also have ...  $\subseteq$  ball x e
    using <e > 0> by auto
  finally have closure (S  $\cap$  ball x (e/2))  $\subseteq$  ball x e .
  moreover have openin (top_of_set S) (S  $\cap$  ball x (e/2)) S  $\cap$  ball x (e/2)  $\neq$ 
{}
    using <0 < e> <x  $\in$  S> by auto
  ultimately obtain Y where Y: ? $\Phi$  0 Y  $\wedge$  Y  $\subseteq$  S  $\cap$  ball x (e/2)
    using * [of S  $\cap$  ball x (e/2) 0] by metis
  show thesis
  proof (rule exE [OF dependent_nat_choice])
    show  $\exists x. ?\Phi$  0 x
      using Y by auto
    show  $\exists Y. ?\Phi$  (Suc n) Y  $\wedge$  Y  $\subseteq$  X if ? $\Phi$  n X for X n
      using that by (blast intro: *)
    qed (use that in metis)
  qed
  have ( $\bigcap n. S \cap \text{closure } (TF \ n)$ )  $\neq$  {}
  proof (rule compact_nest)
    show  $\bigwedge n. \text{compact } (S \cap \text{closure } (TF \ n))$ 
    by (metis closed_closure subball bounded_subset_ballI compact_eq_bounded_closed
closed_Int_compact [OF <closed S>])
    show  $\bigwedge n. S \cap \text{closure } (TF \ n) \neq \{\}$ 
    by (metis Int_absorb1 opeF <closed S> closure_eq_empty closure_minimal
ne openin_imp_subset)
    show  $\bigwedge m \ n. m \leq n \implies S \cap \text{closure } (TF \ n) \subseteq S \cap \text{closure } (TF \ m)$ 

```

```

    by (meson closure_mono decr dual_order.refl inf_mono lift_Suc_antimono_le)
  qed
  moreover have  $(\bigcap n. S \cap \text{closure } (TF\ n)) \subseteq \{y \in \bigcap \mathcal{G}. \text{dist } y\ x < e\}$ 
  proof (clarsimp, intro conjI)
    fix y
    assume y ∈ S and y:  $\forall n. y \in \text{closure } (TF\ n)$ 
    then show  $\forall T \in \mathcal{G}. y \in T$ 
      by (metis Int_iff from_nat_into_surj [OF ‹countable  $\mathcal{G}$ ›] subsetD subg)
    show  $\text{dist } y\ x < e$ 
      by (metis y dist_commute mem_ball subball subsetCE)
  qed
  ultimately show  $\exists y \in \bigcap \mathcal{G}. \text{dist } y\ x < e$ 
    by auto
  qed
qed

```

3.3.16 Continuity

Structural rules for uniform continuity

lemma (in bounded_linear) uniformly_continuous_on[continuous_intros]:

```

  fixes g ::  $\_ :: \text{metric\_space} \Rightarrow \_$ 
  assumes uniformly_continuous_on s g
  shows uniformly_continuous_on s  $(\lambda x. f\ (g\ x))$ 
  using assms unfolding uniformly_continuous_on_sequentially
  unfolding dist_norm tendsto_norm_zero_iff diff[symmetric]
  by (auto intro: tendsto_zero)

```

lemma uniformly_continuous_on_dist[continuous_intros]:

```

  fixes f g ::  $'a :: \text{metric\_space} \Rightarrow 'b :: \text{metric\_space}$ 
  assumes uniformly_continuous_on s f
    and uniformly_continuous_on s g
  shows uniformly_continuous_on s  $(\lambda x. \text{dist } (f\ x)\ (g\ x))$ 

```

proof –

```

{
  fix a b c d :: 'b
  have  $|\text{dist } a\ b - \text{dist } c\ d| \leq \text{dist } a\ c + \text{dist } b\ d$ 
    using dist_triangle2 [of a b c] dist_triangle2 [of b c d]
    using dist_triangle3 [of c d a] dist_triangle [of a d b]
    by arith
}
note le = this
{
  fix x y
  assume f:  $(\lambda n. \text{dist } (f\ (x\ n))\ (f\ (y\ n))) \longrightarrow 0$ 
  assume g:  $(\lambda n. \text{dist } (g\ (x\ n))\ (g\ (y\ n))) \longrightarrow 0$ 
  have  $(\lambda n. |\text{dist } (f\ (x\ n))\ (g\ (x\ n)) - \text{dist } (f\ (y\ n))\ (g\ (y\ n))|) \longrightarrow 0$ 
    by (rule Lim_transform_bound [OF _ tendsto_add_zero [OF f g]],
        simp add: le)
}

```

```

}
then show ?thesis

```

```

    using assms unfolding uniformly_continuous_on_sequentially
    unfolding dist_real_def by simp
qed

```

```

lemma uniformly_continuous_on_cmul_right [continuous_intros]:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_algebra
  shows uniformly_continuous_on s f  $\implies$  uniformly_continuous_on s ( $\lambda x. f x * c$ )
  using bounded_linear.uniformly_continuous_on[OF bounded_linear_mult_left]
.

```

```

lemma uniformly_continuous_on_cmul_left [continuous_intros]:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_algebra
  assumes uniformly_continuous_on s f
  shows uniformly_continuous_on s ( $\lambda x. c * f x$ )
by (metis assms bounded_linear.uniformly_continuous_on bounded_linear_mult_right)

```

```

lemma uniformly_continuous_on_norm [continuous_intros]:
  fixes f :: 'a::metric_space  $\Rightarrow$  'b::real_normed_vector
  assumes uniformly_continuous_on s f
  shows uniformly_continuous_on s ( $\lambda x. \text{norm } (f x)$ )
  unfolding norm_conv_dist using assms
  by (intro uniformly_continuous_on_dist uniformly_continuous_on_const)

```

```

lemma uniformly_continuous_on_cmul [continuous_intros]:
  fixes f :: 'a::metric_space  $\Rightarrow$  'b::real_normed_vector
  assumes uniformly_continuous_on s f
  shows uniformly_continuous_on s ( $\lambda x. c *_R f(x)$ )
  using bounded_linear_scaleR_right assms
  by (rule bounded_linear.uniformly_continuous_on)

```

```

lemma dist_minus:
  fixes x y :: 'a::real_normed_vector
  shows dist (- x) (- y) = dist x y
  unfolding dist_norm minus_diff_minus norm_minus_cancel ..

```

```

lemma uniformly_continuous_on_minus [continuous_intros]:
  fixes f :: 'a::metric_space  $\Rightarrow$  'b::real_normed_vector
  shows uniformly_continuous_on s f  $\implies$  uniformly_continuous_on s ( $\lambda x. - f x$ )
  unfolding uniformly_continuous_on_def dist_minus .

```

```

lemma uniformly_continuous_on_add [continuous_intros]:
  fixes f g :: 'a::metric_space  $\Rightarrow$  'b::real_normed_vector
  assumes uniformly_continuous_on s f
  and uniformly_continuous_on s g
  shows uniformly_continuous_on s ( $\lambda x. f x + g x$ )
  using assms
  unfolding uniformly_continuous_on_sequentially

```

unfolding *dist_norm_tendsto_norm_zero_iff_add_diff_add*
by (*auto intro: tendsto_add_zero*)

lemma *uniformly_continuous_on_diff* [*continuous_intros*]:
fixes $f :: 'a::metric\_space \Rightarrow 'b::real\_normed\_vector$
assumes *uniformly_continuous_on s f*
and *uniformly_continuous_on s g*
shows *uniformly_continuous_on s* $(\lambda x. f\ x - g\ x)$
using *assms uniformly_continuous_on_add* [*of s f - g*]
by (*simp add: fun_Compl_def uniformly_continuous_on_minus*)

3.3.17 Arithmetic Preserves Topological Properties

lemma *open_scaling* [*intro*]:
fixes $s :: 'a::real\_normed\_vector\ set$
assumes $c \neq 0$
and *open s*
shows *open* $((\lambda x. c *_{\mathbb{R}} x) \text{ ` } s)$
proof –
{
fix x
assume $x \in s$
then obtain e **where** $e > 0$
and $e : \forall x'. dist\ x'\ x < e \longrightarrow x' \in s$ **using** *assms(2)* [*unfolded open_dist,*
THEN bspec [*where x=x*]]
by *auto*
have $e * |c| > 0$
using *assms(1)* [*unfolded zero_less_abs_iff* [*symmetric*]] $\langle e > 0 \rangle$ **by** *auto*
moreover
{
fix y
assume $dist\ y\ (c *_{\mathbb{R}} x) < e * |c|$
then have $norm\ (c *_{\mathbb{R}} ((1 / c) *_{\mathbb{R}} y - x)) < e * norm\ c$
by (*simp add: $\langle c \neq 0 \rangle dist\_norm\ scale\_right\_diff\_distrib$*)
then have $norm\ ((1 / c) *_{\mathbb{R}} y - x) < e$
by (*simp add: $\langle c \neq 0 \rangle$*)
then have $y \in (*_{\mathbb{R}})\ c \text{ ` } s$
using *rev_image_eqI* [*of (1 / c) *_{\mathbb{R}} y s y (*_{\mathbb{R}})\ c*]
by (*simp add: $\langle c \neq 0 \rangle dist\_norm\ e$*)
}
ultimately have $\exists e > 0. \forall x'. dist\ x'\ (c *_{\mathbb{R}} x) < e \longrightarrow x' \in (*_{\mathbb{R}})\ c \text{ ` } s$
by (*rule_tac x=e * |c| in exI, auto*)
}
then show *?thesis* **unfolding** *open_dist* **by** *auto*
qed

lemma *minus_image_eq_vimage*:
fixes $A :: 'a::ab\_group\_add\ set$
shows $(\lambda x. -\ x) \text{ ` } A = (\lambda x. -\ x) - \text{ ` } A$

by (auto intro!: image_eqI [where f= $\lambda x. - x$])

lemma open_negations:

fixes $S :: 'a::\text{real\_normed\_vector\_set}$
 shows $\text{open } S \implies \text{open } ((\lambda x. - x) ' S)$
 using open_scaling [of $- 1$ S] by simp

lemma open_translation:

fixes $S :: 'a::\text{real\_normed\_vector\_set}$
 assumes open S
 shows $\text{open}((\lambda x. a + x) ' S)$

proof –

{
 fix x
 have continuous (at x) ($\lambda x. x - a$)
 by (intro continuous_diff continuous_ident continuous_const)
 }
 moreover have $\{x. x - a \in S\} = (+) a ' S$
 by force
 ultimately show ?thesis
 by (metis assms continuous_open_vimage vimage_def)

qed

lemma open_translation_subtract:

fixes $S :: 'a::\text{real\_normed\_vector\_set}$
 assumes open S
 shows $\text{open } ((\lambda x. x - a) ' S)$
 using assms open_translation [of $S - a$] by (simp cong: image_cong_simp)

lemma open_neg_translation:

fixes $S :: 'a::\text{real\_normed\_vector\_set}$
 assumes open S
 shows $\text{open}((\lambda x. a - x) ' S)$
 using open_translation[OF open_negations[OF assms], of a]
 by (auto simp: image_image)

lemma open_affinity:

fixes $S :: 'a::\text{real\_normed\_vector\_set}$
 assumes open S $c \neq 0$
 shows $\text{open } ((\lambda x. a + c *_{\mathbb{R}} x) ' S)$

proof –

have *: $(\lambda x. a + c *_{\mathbb{R}} x) = (\lambda x. a + x) \circ (\lambda x. c *_{\mathbb{R}} x)$
 unfolding o_def ..
 have $(+) a ' (*_{\mathbb{R}}) c ' S = ((+) a \circ (*_{\mathbb{R}}) c) ' S$
 by auto
 then show ?thesis
 using assms open_translation[of $(*_{\mathbb{R}}) c ' S a$]
 unfolding *
 by auto

qed

lemma *interior_translation*:

interior $((+) a \text{ ' } S) = (+) a \text{ ' } (\text{interior } S)$ **for** $S :: 'a::\text{real\_normed\_vector set}$

proof (rule *set_eqI*, rule)

fix x

assume $x \in \text{interior } ((+) a \text{ ' } S)$

then obtain e **where** $e > 0$ **and** $e: \text{ball } x \ e \subseteq (+) a \text{ ' } S$

unfolding *mem_interior* **by** *auto*

then have $\text{ball } (x - a) \ e \subseteq S$

unfolding *subset_eq Ball_def mem_ball dist_norm*

by (*auto simp: diff_diff_eq*)

then show $x \in (+) a \text{ ' } \text{interior } S$

unfolding *image_iff*

by (*metis* $\langle 0 < e \rangle$ *add.commute diff_add_cancel mem_interior*)

next

fix x

assume $x \in (+) a \text{ ' } \text{interior } S$

then obtain $y \ e$ **where** $e > 0$ **and** $e: \text{ball } y \ e \subseteq S$ **and** $y: x = a + y$

unfolding *image_iff Bex_def mem_interior* **by** *auto*

{

fix z

have $*: a + y - z = y + a - z$ **by** *auto*

assume $z \in \text{ball } x \ e$

then have $z - a \in S$

using $e[\text{unfolded } \text{subset\_eq}, \text{ THEN } \text{bspec}[\text{where } x=z - a]]$

unfolding *mem_ball dist_norm y group_add_class.diff_diff_eq2* *

by *auto*

then have $z \in (+) a \text{ ' } S$

unfolding *image_iff* **by** (*auto intro!: bexI* $[\text{where } x=z - a]$)

}

then have $\text{ball } x \ e \subseteq (+) a \text{ ' } S$

unfolding *subset_eq* **by** *auto*

then show $x \in \text{interior } ((+) a \text{ ' } S)$

unfolding *mem_interior* **using** $\langle e > 0 \rangle$ **by** *auto*

qed

lemma *interior_translation_subtract*:

interior $((\lambda x. x - a) \text{ ' } S) = (\lambda x. x - a) \text{ ' } \text{interior } S$ **for** $S :: 'a::\text{real\_normed\_vector set}$

using *interior_translation* $[\text{of } - a]$ **by** (*simp cong: image_cong_simp*)

lemma *compact_scaling*:

fixes $s :: 'a::\text{real\_normed\_vector set}$

assumes *compact s*

shows *compact* $((\lambda x. c *_R x) \text{ ' } s)$

proof —

let $?f = \lambda x. \text{scaleR } c \ x$

```

have *: bounded_linear ?f by (rule bounded_linear_scaleR_right)
show ?thesis
  using compact_continuous_image[of s ?f] continuous_at_imp_continuous_on[of
s ?f]
  using linear_continuous_at[OF *] assms
  by auto
qed

```

```

lemma compact_negations:
  fixes s :: 'a::real_normed_vector set
  assumes compact s
  shows compact (( $\lambda x. - x$ ) ' s)
  using compact_scaling [OF assms, of - 1] by auto

```

```

lemma compact_sums:
  fixes s t :: 'a::real_normed_vector set
  assumes compact s
    and compact t
  shows compact { $x + y \mid x y. x \in s \wedge y \in t$ }
proof -
  have *: { $x + y \mid x y. x \in s \wedge y \in t$ } = ( $\lambda z. \text{fst } z + \text{snd } z$ ) ' (s  $\times$  t)
  by (fastforce simp: image_iff)
  have continuous_on (s  $\times$  t) ( $\lambda z. \text{fst } z + \text{snd } z$ )
  unfolding continuous_on by (rule ballI) (intro tendsto_intros)
  then show ?thesis
    unfolding * using compact_continuous_image compact_Times [OF assms]
  by auto
qed

```

```

lemma compact_differences:
  fixes s t :: 'a::real_normed_vector set
  assumes compact s
    and compact t
  shows compact { $x - y \mid x y. x \in s \wedge y \in t$ }
proof -
  have { $x - y \mid x y. x \in s \wedge y \in t$ } = { $x + y \mid x y. x \in s \wedge y \in (\text{uminus ' } t)$ }
  using diff_conv_add_uminus by force
  then show ?thesis
    using compact_sums[OF assms(1) compact_negations[OF assms(2)]] by auto
qed

```

```

lemma compact_sums':
  fixes S :: 'a::real_normed_vector set
  assumes compact S and compact T
  shows compact ( $\bigcup_{x \in S. \bigcup_{y \in T. \{x + y\}}$ )
proof -
  have ( $\bigcup_{x \in S. \bigcup_{y \in T. \{x + y\}}$ ) = { $x + y \mid x y. x \in S \wedge y \in T$ }
  by blast
  then show ?thesis

```

```

    using compact_sums [OF assms] by simp
qed

lemma compact_differences':
  fixes S :: 'a::real_normed_vector set
  assumes compact S and compact T
  shows compact ( $\bigcup_{x \in S. \bigcup_{y \in T. \{x - y\}}$ )
proof -
  have ( $\bigcup_{x \in S. \bigcup_{y \in T. \{x - y\}}$ ) =  $\{x - y \mid x \in S \wedge y \in T\}$ 
    by blast
  then show ?thesis
    using compact_differences [OF assms] by simp
qed

lemma compact_translation:
  compact ((+) a ' s) if compact s for s :: 'a::real_normed_vector set
proof -
  have  $\{x + y \mid x \in s \wedge y \in \{a\}\} = (\lambda x. a + x) ' s$ 
    by auto
  then show ?thesis
    using compact_sums [OF that compact_sing [of a]] by auto
qed

lemma compact_translation_subtract:
  compact (( $\lambda x. x - a$ ) ' s) if compact s for s :: 'a::real_normed_vector set
  using that compact_translation [of s - a] by (simp cong: image_cong_simp)

lemma compact_affinity:
  fixes s :: 'a::real_normed_vector set
  assumes compact s
  shows compact (( $\lambda x. a + c *_{\mathbb{R}} x$ ) ' s)
proof -
  have  $(+) a ' (*_{\mathbb{R}}) c ' s = (\lambda x. a + c *_{\mathbb{R}} x) ' s$ 
    by auto
  then show ?thesis
    using compact_translation [OF compact_scaling [OF assms], of a c] by auto
qed

lemma closed_scaling:
  fixes S :: 'a::real_normed_vector set
  assumes closed S
  shows closed (( $\lambda x. c *_{\mathbb{R}} x$ ) ' S)
proof (cases c = 0)
  case True then show ?thesis
    by (auto simp: image_constant_conv)
next
  case False
  from assms have closed (( $\lambda x. \text{inverse } c *_{\mathbb{R}} x$ ) - ' S)
    by (simp add: continuous_closed_vimage)

```

```

    also have  $(\lambda x. \text{inverse } c *_{\mathcal{R}} x) - ' S = (\lambda x. c *_{\mathcal{R}} x) ' S$ 
    using  $\langle c \neq 0 \rangle$  by (auto elim: image_eqI [rotated])
    finally show ?thesis .
qed

```

```

lemma closed_negations:
  fixes  $S :: 'a::\text{real\_normed\_vector\_set}$ 
  assumes closed  $S$ 
  shows closed  $((\lambda x. -x) ' S)$ 
  using closed_scaling[OF assms, of  $- 1$ ] by simp

```

```

lemma compact_closed_sums:
  fixes  $S :: 'a::\text{real\_normed\_vector\_set}$ 
  assumes compact  $S$  and closed  $T$ 
  shows closed  $(\bigcup_{x \in S} \bigcup_{y \in T} \{x + y\})$ 
proof -
  let ?S =  $\{x + y \mid x \in S \wedge y \in T\}$ 
  {
    fix  $x \ l$ 
    assume as:  $\forall n. x \ n \in ?S \ (x \longrightarrow l) \text{ sequentially}$ 
    from as(1) obtain  $f$  where  $f: \forall n. x \ n = \text{fst } (f \ n) + \text{snd } (f \ n) \ \forall n. \text{fst } (f \ n) \in S \ \forall n. \text{snd } (f \ n) \in T$ 
    using choice[of  $\lambda n y. x \ n = (\text{fst } y) + (\text{snd } y) \wedge \text{fst } y \in S \wedge \text{snd } y \in T$ ] by auto
    obtain  $l' \ r$  where  $l' \in S$  and  $r: \text{strict\_mono } r$  and  $lr: ((\lambda n. \text{fst } (f \ n)) \circ r) \longrightarrow l'$  sequentially
    using assms(1)[unfolded compact_def, THEN spec[where  $x = \lambda n. \text{fst } (f \ n)$ ]]
  using  $f(2)$  by auto
    have  $((\lambda n. \text{snd } (f \ (r \ n))) \longrightarrow l - l') \text{ sequentially}$ 
    using tendsto_diff[OF LIMSEQ_subseq_LIMSEQ[OF as(2)  $r$ ]  $lr$ ] and  $f(1)$ 
    unfolding o_def
    by auto
    then have  $l - l' \in T$ 
    using assms(2)[unfolded closed_sequential_limits,
      THEN spec[where  $x = \lambda n. \text{snd } (f \ (r \ n))$ ],
      THEN spec[where  $x = l - l'$ ]]
    using  $f(3)$ 
    by auto
    then have  $l \in ?S$ 
    using  $\langle l' \in S \rangle$  by force
  }
  moreover have  $?S = (\bigcup_{x \in S} \bigcup_{y \in T} \{x + y\})$ 
  by force
  ultimately show ?thesis
  unfolding closed_sequential_limits
  by (metis (no_types, lifting))
qed

```

```

lemma closed_compact_sums:

```

```

fixes  $S\ T :: 'a::real\_normed\_vector\ set$ 
assumes  $closed\ S\ compact\ T$ 
shows  $closed\ (\bigcup_{x \in S} \bigcup_{y \in T} \{x + y\})$ 
proof -
  have  $(\bigcup_{x \in T} \bigcup_{y \in S} \{x + y\}) = (\bigcup_{x \in S} \bigcup_{y \in T} \{x + y\})$ 
    by auto
  then show ?thesis
    using  $compact\_closed\_sums[OF\ assms(2,1)]$  by simp
qed

lemma  $compact\_closed\_differences:$ 
  fixes  $S\ T :: 'a::real\_normed\_vector\ set$ 
assumes  $compact\ S\ closed\ T$ 
shows  $closed\ (\bigcup_{x \in S} \bigcup_{y \in T} \{x - y\})$ 
proof -
  have  $(\bigcup_{x \in S} \bigcup_{y \in uminus\ 'T} \{x + y\}) = (\bigcup_{x \in S} \bigcup_{y \in T} \{x - y\})$ 
    by force
  then show ?thesis
    by  $(metis\ assms\ closed\_negations\ compact\_closed\_sums)$ 
qed

lemma  $closed\_compact\_differences:$ 
  fixes  $S\ T :: 'a::real\_normed\_vector\ set$ 
assumes  $closed\ S\ compact\ T$ 
shows  $closed\ (\bigcup_{x \in S} \bigcup_{y \in T} \{x - y\})$ 
proof -
  have  $(\bigcup_{x \in S} \bigcup_{y \in uminus\ 'T} \{x + y\}) = \{x - y \mid x \in S \wedge y \in T\}$ 
    by auto
  then show ?thesis
    using  $closed\_compact\_sums[OF\ assms(1)\ compact\_negations[OF\ assms(2)]]$  by
simp
qed

lemma  $closed\_translation:$ 
   $closed\ ((+) a\ 'S)$  if  $closed\ S$  for  $a :: 'a::real\_normed\_vector$ 
proof -
  have  $(\bigcup_{x \in \{a\}} \bigcup_{y \in S} \{x + y\}) = ((+) a\ 'S)$  by auto
  then show ?thesis
    using  $compact\_closed\_sums\ [OF\ compact\_sing\ [of\ a]\ that]$  by auto
qed

lemma  $closed\_translation\_subtract:$ 
   $closed\ ((\lambda x. x - a)\ 'S)$  if  $closed\ S$  for  $a :: 'a::real\_normed\_vector$ 
  using  $that\ closed\_translation\ [of\ S - a]$  by  $(simp\ cong:\ image\_cong\_simp)$ 

lemma  $closure\_translation:$ 
   $closure\ ((+) a\ 's) = (+) a\ 'closure\ s$  for  $a :: 'a::real\_normed\_vector$ 
proof -
  have  $*(+)\ a\ '(-\ s) = -\ (+)\ a\ 's$ 

```

```

    by (auto intro!: image_eqI [where x = x - a for x])
  show ?thesis
    using interior_translation [of a - s, symmetric]
    by (simp add: closure_interior_translation_Cmpl *)
qed

```

```

lemma closure_translation_subtract:
  closure ((λx. x - a) 's) = (λx. x - a) ' closure s for a :: 'a::real_normed_vector
  using closure_translation [of - a s] by (simp cong: image_cong_simp)

```

```

lemma frontier_translation:
  frontier ((+) a 's) = (+) a ' frontier s for a :: 'a::real_normed_vector
  by (auto simp add: frontier_def translation_diff interior_translation closure_translation)

```

```

lemma frontier_translation_subtract:
  frontier ((+) a 's) = (+) a ' frontier s for a :: 'a::real_normed_vector
  by (auto simp add: frontier_def translation_diff interior_translation closure_translation)

```

```

lemma sphere_translation:
  sphere (a + c) r = (+) a ' sphere c r for a :: 'n::real_normed_vector
  by (auto simp: dist_norm algebra_simps intro!: image_eqI [where x = x - a
for x])

```

```

lemma sphere_translation_subtract:
  sphere (c - a) r = (λx. x - a) ' sphere c r for a :: 'n::real_normed_vector
  using sphere_translation [of - a c] by (simp cong: image_cong_simp)

```

```

lemma cball_translation:
  cball (a + c) r = (+) a ' cball c r for a :: 'n::real_normed_vector
  by (auto simp: dist_norm algebra_simps intro!: image_eqI [where x = x - a
for x])

```

```

lemma cball_translation_subtract:
  cball (c - a) r = (λx. x - a) ' cball c r for a :: 'n::real_normed_vector
  using cball_translation [of - a c] by (simp cong: image_cong_simp)

```

```

lemma ball_translation:
  ball (a + c) r = (+) a ' ball c r for a :: 'n::real_normed_vector
  by (auto simp: dist_norm algebra_simps intro!: image_eqI [where x = x - a
for x])

```

```

lemma ball_translation_subtract:
  ball (c - a) r = (λx. x - a) ' ball c r for a :: 'n::real_normed_vector
  using ball_translation [of - a c] by (simp cong: image_cong_simp)

```

3.3.18 Homeomorphisms

```

lemma homeomorphic_scaling:
  fixes S :: 'a::real_normed_vector set

```

```

assumes  $c \neq 0$ 
shows  $S$  homeomorphic  $((\lambda x. c *_{\mathbb{R}} x) ' S)$ 
unfolding homeomorphic_minimal
apply (rule_tac  $x = \lambda x. c *_{\mathbb{R}} x$  in exI)
apply (rule_tac  $x = \lambda x. (1 / c) *_{\mathbb{R}} x$  in exI)
using assms by (auto simp: continuous_intros)

```

```

lemma homeomorphic_translation:
  fixes  $S :: 'a :: \text{real\_normed\_vector\_set}$ 
  shows  $S$  homeomorphic  $((\lambda x. a + x) ' S)$ 
  unfolding homeomorphic_minimal
  apply (rule_tac  $x = \lambda x. a + x$  in exI)
  apply (rule_tac  $x = \lambda x. -a + x$  in exI)
  by (auto simp: continuous_intros)

```

```

lemma homeomorphic_affinity:
  fixes  $S :: 'a :: \text{real\_normed\_vector\_set}$ 
  assumes  $c \neq 0$ 
  shows  $S$  homeomorphic  $((\lambda x. a + c *_{\mathbb{R}} x) ' S)$ 
proof -
  have *:  $(+) a ' (*_{\mathbb{R}}) c ' S = (\lambda x. a + c *_{\mathbb{R}} x) ' S$  by auto
  show ?thesis
    by (metis * assms homeomorphic_scaling homeomorphic_trans homeomorphic_translation)
qed

```

```

lemma homeomorphic_balls:
  fixes  $a b :: 'a :: \text{real\_normed\_vector}$ 
  assumes  $0 < d \ 0 < e$ 
  shows (ball  $a d$ ) homeomorphic (ball  $b e$ ) (is ?th)
    and (cball  $a d$ ) homeomorphic (cball  $b e$ ) (is ?cth)
proof -
  show ?th unfolding homeomorphic_minimal
    apply (rule_tac  $x = \lambda x. b + (e/d) *_{\mathbb{R}} (x - a)$  in exI)
    apply (rule_tac  $x = \lambda x. a + (d/e) *_{\mathbb{R}} (x - b)$  in exI)
    using assms
    by (auto intro!: continuous_intros simp: dist_commute dist_norm pos_divide_less_eq)
  show ?cth unfolding homeomorphic_minimal
    apply (rule_tac  $x = \lambda x. b + (e/d) *_{\mathbb{R}} (x - a)$  in exI)
    apply (rule_tac  $x = \lambda x. a + (d/e) *_{\mathbb{R}} (x - b)$  in exI)
    using assms
    by (auto intro!: continuous_intros simp: dist_commute dist_norm pos_divide_le_eq)
qed

```

```

lemma homeomorphic_spheres:
  fixes  $a b :: 'a :: \text{real\_normed\_vector}$ 
  assumes  $0 < d \ 0 < e$ 
  shows (sphere  $a d$ ) homeomorphic (sphere  $b e$ )
unfolding homeomorphic_minimal

```

```

    apply(rule_tac x= $\lambda x. b + (e/d) *_{\mathbb{R}} (x - a)$  in exI)
    apply(rule_tac x= $\lambda x. a + (d/e) *_{\mathbb{R}} (x - b)$  in exI)
    using assms
    by (auto intro!: continuous_intros simp: dist_commute dist_norm pos_divide_less_eq)

lemma homeomorphic_ball01_UNIV:
  ball (0::'a::real_normed_vector) 1 homeomorphic (UNIV::'a set)
  (is ?B homeomorphic ?U)
proof
  have  $x \in (\lambda z. z /_{\mathbb{R}} (1 - \text{norm } z)) \text{ ' ball } 0 \ 1$  for  $x::'a$ 
  apply (rule_tac x= $x /_{\mathbb{R}} (1 + \text{norm } x)$  in image_eqI)
  apply (auto simp: field_split_simps)
  using norm_ge_zero [of x] apply linarith+
  done
  then show  $(\lambda z::'a. z /_{\mathbb{R}} (1 - \text{norm } z)) \text{ ' ?B} = ?U$ 
  by blast
  have  $x \in \text{range } (\lambda z. (1 / (1 + \text{norm } z)) *_{\mathbb{R}} z)$  if  $\text{norm } x < 1$  for  $x::'a$ 
  using that
  by (rule_tac x= $x /_{\mathbb{R}} (1 - \text{norm } x)$  in image_eqI) (auto simp: field_split_simps)
  then show  $(\lambda z::'a. z /_{\mathbb{R}} (1 + \text{norm } z)) \text{ ' ?U} = ?B$ 
  by (force simp: field_split_simps dest: add_less_zeroD)
  show continuous_on (ball 0 1)  $(\lambda z. z /_{\mathbb{R}} (1 - \text{norm } z))$ 
  by (rule continuous_intros | force)+
  have  $0: \bigwedge z. 1 + \text{norm } z \neq 0$ 
  by (metis (no_types) le_add_same_cancel1 norm_ge_zero not_one_le_zero)
  then show continuous_on UNIV  $(\lambda z. z /_{\mathbb{R}} (1 + \text{norm } z))$ 
  by (auto intro!: continuous_intros)
  show  $\bigwedge x. x \in \text{ball } 0 \ 1 \implies$ 
     $x /_{\mathbb{R}} (1 - \text{norm } x) /_{\mathbb{R}} (1 + \text{norm } (x /_{\mathbb{R}} (1 - \text{norm } x))) = x$ 
  by (auto simp: field_split_simps)
  show  $\bigwedge y. y /_{\mathbb{R}} (1 + \text{norm } y) /_{\mathbb{R}} (1 - \text{norm } (y /_{\mathbb{R}} (1 + \text{norm } y))) = y$ 
  using 0 by (auto simp: field_split_simps)
qed

proposition homeomorphic_ball_UNIV:
  fixes  $a :: 'a::\text{real\_normed\_vector}$ 
  assumes  $0 < r$  shows ball a r homeomorphic (UNIV::'a set)
  using assms homeomorphic_ball01_UNIV homeomorphic_balls(1) homeomor-
  phic_trans zero_less_one by blast

```

3.3.19 Discrete

```

lemma finite_implies_discrete:
  fixes  $S :: 'a::\text{topological\_space set}$ 
  assumes finite (f ' S)
  shows  $(\forall x \in S. \exists e > 0. \forall y. y \in S \wedge f y \neq f x \implies e \leq \text{norm } (f y - f x))$ 
proof -
  have  $\exists e > 0. \forall y. y \in S \wedge f y \neq f x \implies e \leq \text{norm } (f y - f x)$  if  $x \in S$  for  $x$ 
  proof (cases f ' S - {f x} = {})

```



```

    case True
    with zero_less_numeral show ?thesis
    by (fastforce simp add: Set.image_subset_iff cong: conj_cong)
next
case False
then obtain z where  $z \in S$   $f z \neq f x$ 
by blast
moreover have finn: finite {norm (z - f x) | z. z ∈ f ' S - {f x}}
using assms by simp
ultimately have *:  $0 < \text{Inf}\{\text{norm}(z - f x) \mid z. z \in f ' S - \{f x\}\}$ 
by (force intro: finite_imp_less_Inf)
show ?thesis
by (force intro!: * cInf_le_finite [OF finn])
qed
with assms show ?thesis
by blast
qed

```

3.3.20 Completeness of "Isometry" (up to constant bounds)

lemma *cauchy_isometric*:— TODO: rename lemma to *Cauchy_isometric*

```

assumes e:  $e > 0$ 
and s: subspace s
and f: bounded_linear f
and normf:  $\forall x \in s. \text{norm}(f x) \geq e * \text{norm } x$ 
and xs:  $\forall n. x n \in s$ 
and cf: Cauchy (f ∘ x)
shows Cauchy x
proof -
interpret f: bounded_linear f by fact
have  $\exists N. \forall n \geq N. \text{norm}(x n - x N) < d$  if  $d > 0$  for  $d :: \text{real}$ 
proof -
from that obtain N where  $N: \forall n \geq N. \text{norm}(f(x n) - f(x N)) < e * d$ 
using cf[unfolded Cauchy_def o_def dist_norm, THEN spec[where x=e*d]]
e
by auto
have  $\text{norm}(x n - x N) < d$  if  $n \geq N$  for n
proof -
have  $e * \text{norm}(x n - x N) \leq \text{norm}(f(x n - x N))$ 
using subspace_diff[OF s, of x n x N]
using xs[THEN spec[where x=N]] and xs[THEN spec[where x=n]]
using normf[THEN bspec[where x=x n - x N]]
by auto
also have  $\text{norm}(f(x n - x N)) < e * d$ 
using  $\langle N \leq n \rangle$  N unfolding f.diff[symmetric] by auto
finally show ?thesis
using  $\langle e > 0 \rangle$  by simp
qed
then show ?thesis by auto

```

```

qed
then show ?thesis
  by (simp add: Cauchy_altdf2 dist_norm)
qed

lemma complete_isometric_image:
  assumes  $0 < e$ 
  and  $s$ : subspace  $s$ 
  and  $f$ : bounded_linear  $f$ 
  and  $normf$ :  $\forall x \in s. norm(f\ x) \geq e * norm(x)$ 
  and  $cs$ : complete  $s$ 
  shows complete  $(f\ ' s)$ 
proof -
  have  $\exists l \in f\ ' s. (g \longrightarrow l)$  sequentially
  if  $as: \forall n::nat. g\ n \in f\ ' s$  and  $cfg: Cauchy\ g$  for  $g$ 
  proof -
    from that obtain  $x$  where  $\forall n. x\ n \in s \wedge g\ n = f\ (x\ n)$ 
    using choice[of  $\lambda n\ xa. xa \in s \wedge g\ n = f\ xa$ ] by auto
    then have  $x: \forall n. x\ n \in s \wedge g\ n = f\ (x\ n)$  by auto
    then have  $f \circ x = g$  by (simp add: fun_eq_iff)
    then obtain  $l$  where  $l \in s$  and  $l: (x \longrightarrow l)$  sequentially
    using  $cs[unfolded\ complete\_def, THEN\ spec[where\ x=x]]$ 
    using  $cauchy\_isometric[OF\ \langle 0 < e \rangle\ s\ f\ normf]$  and  $cfg$  and  $x(1)$ 
    by auto
    then show ?thesis
      using linear_continuous_at[OF  $f$ , unfolded continuous_at_sequentially,
        THEN spec[where  $x=x$ ], of  $l$ ]
      by (auto simp:  $\langle f \circ x = g \rangle$ )
  qed
  then show ?thesis
    unfolding complete_def by auto
qed

```

3.3.21 Connected Normed Spaces

```

lemma compact_components:
  fixes  $s :: 'a::heine\_borel\ set$ 
  shows  $\llbracket compact\ s; c \in components\ s \rrbracket \implies compact\ c$ 
by (meson bounded_subset closed_components_in_components_subset compact_eq_bounded_closed)

lemma discrete_subset_disconnected:
  fixes  $S :: 'a::topological\_space\ set$ 
  fixes  $t :: 'b::real\_normed\_vector\ set$ 
  assumes  $conf: continuous\_on\ S\ f$ 
  and  $no: \bigwedge x. x \in S \implies \exists e>0. \forall y. y \in S \wedge f\ y \neq f\ x \longrightarrow e \leq norm\ (f\ y - f\ x)$ 
  shows  $f\ ' S \subseteq \{y. connected\_component\_set\ (f\ ' S)\ y = \{y\}\}$ 
proof -
  { fix  $x$  assume  $x: x \in S$ 

```

```

    then obtain e where e>0 and ele:  $\bigwedge y. \llbracket y \in S; f y \neq f x \rrbracket \implies e \leq \text{norm } (f y - f x)$ 
    using conf no [OF x] by auto
    then have e2:  $0 \leq e/2$ 
    by simp
    define F where  $F \equiv \text{connected\_component\_set } (f \text{ ` } S) (f x)$ 
    have False if  $y \in S$  and ccs:  $f y \in F$  and not:  $f y \neq f x$  for y
    proof -
      define C where  $C \equiv \text{cball } (f x) (e/2)$ 
      define D where  $D \equiv - \text{ball } (f x) e$ 
      have disj:  $C \cap D = \{\}$ 
        unfolding C_def D_def using  $\langle 0 < e \rangle$  by fastforce
      moreover have FCD:  $F \subseteq C \cup D$ 
      proof -
        have  $t \in C \vee t \in D$  if  $t \in F$  for t
        proof -
          obtain y where  $y \in S$   $t = f y$ 
          using F_def  $\langle t \in F \rangle$  connected_component_in by blast
          then show ?thesis
            by (metis C_def ComplI D_def centre_in_cball dist_norm e2 ele
mem_ball norm_minus_commute not_le)
        qed
        then show ?thesis
          by auto
      qed
      ultimately have  $C \cap F = \{\} \vee D \cap F = \{\}$ 
        using connected_closed [of F]  $\langle e > 0 \rangle$  not
        unfolding C_def D_def
        by (metis Elementary_Metric_Spaces.open_ball F_def closed_cball connected_connected_component_inf_bot_left open_closed)
      moreover have  $C \cap F \neq \{\}$ 
        unfolding disjoint_iff
        by (metis FCD ComplD image_eqI mem_Collect_eq subsetD x D_def F_def Un_iff  $\langle 0 < e \rangle$  centre_in_ball connected_component_refl_eq)
      moreover have  $D \cap F \neq \{\}$ 
        unfolding disjoint_iff
        by (metis ComplI D_def ccs dist_norm ele mem_ball norm_minus_commute not not_le that(1))
      ultimately show ?thesis by metis
    qed
    moreover have  $\text{connected\_component\_set } (f \text{ ` } S) (f x) \subseteq f \text{ ` } S$ 
      by (auto simp: connected_component_in)
    ultimately have  $\text{connected\_component\_set } (f \text{ ` } S) (f x) = \{f x\}$ 
      by (auto simp: x F_def)
  }
  with assms show ?thesis
    by blast
qed

```

```

lemma continuous_disconnected_range_constant_eq:
  (connected S  $\longleftrightarrow$ 
    ( $\forall f::'a::\text{topological\_space} \Rightarrow 'b::\text{real\_normed\_algebra\_1}.$ 
       $\forall t. \text{continuous\_on } S f \wedge f ' S \subseteq t \wedge (\forall y \in t. \text{connected\_component\_set}$ 
 $t\ y = \{y\})$ 
       $\longrightarrow f \text{ constant\_on } S$ )) (is ?thesis1)
and continuous_discrete_range_constant_eq:
  (connected S  $\longleftrightarrow$ 
    ( $\forall f::'a::\text{topological\_space} \Rightarrow 'b::\text{real\_normed\_algebra\_1}.$ 
      continuous_on S f  $\wedge$ 
      ( $\forall x \in S. \exists e. 0 < e \wedge (\forall y. y \in S \wedge (f\ y \neq f\ x) \longrightarrow e \leq \text{norm}(f\ y - f\ x))$ 
       $\longrightarrow f \text{ constant\_on } S$ )) (is ?thesis2)
and continuous_finite_range_constant_eq:
  (connected S  $\longleftrightarrow$ 
    ( $\forall f::'a::\text{topological\_space} \Rightarrow 'b::\text{real\_normed\_algebra\_1}.$ 
      continuous_on S f  $\wedge$  finite (f ' S)
       $\longrightarrow f \text{ constant\_on } S$ )) (is ?thesis3)
proof -
  have *:  $\bigwedge s\ t\ u\ v. [s \Longrightarrow t; t \Longrightarrow u; u \Longrightarrow v; v \Longrightarrow s]$ 
     $\Longrightarrow (s \longleftrightarrow t) \wedge (s \longleftrightarrow u) \wedge (s \longleftrightarrow v)$ 
  by blast
  have ?thesis1  $\wedge$  ?thesis2  $\wedge$  ?thesis3
  apply (rule *)
  using continuous_disconnected_range_constant apply metis
  apply clarify
  apply (frule discrete_subset_disconnected; blast)
  apply (blast dest: finite_implies_discrete)
  apply (blast intro!: finite_range_constant_imp_connected)
  done
  then show ?thesis1 ?thesis2 ?thesis3
  by blast+
qed

```

```

lemma continuous_discrete_range_constant:
  fixes f :: 'a::topological_space  $\Rightarrow$  'b::real_normed_algebra_1
  assumes S: connected S
  and continuous_on S f
  and  $\bigwedge x. x \in S \Longrightarrow \exists e > 0. \forall y. y \in S \wedge f\ y \neq f\ x \longrightarrow e \leq \text{norm}(f\ y - f\ x)$ 
  shows f constant_on S
  using continuous_discrete_range_constant_eq [THEN iffD1, OF S] assms by
  blast

```

```

lemma continuous_finite_range_constant:
  fixes f :: 'a::topological_space  $\Rightarrow$  'b::real_normed_algebra_1
  assumes connected S
  and continuous_on S f
  and finite (f ' S)
  shows f constant_on S
  using assms continuous_finite_range_constant_eq by blast

```

end

3.4 Linear Decision Procedure for Normed Spaces

```
theory Norm_Arith
imports HOL-Library.Sum_of_Squares
begin
```

```
lemma sum_sqs_eq:
  fixes x::'a::idom shows  $x * x + y * y = x * (y * 2) \implies y = x$ 
  by algebra
```

```
lemma norm_cmul_rule_thm:
  fixes x :: 'a::real_normed_vector
  shows  $b \geq \text{norm } x \implies |c| * b \geq \text{norm } (\text{scaleR } c \ x)$ 
  unfolding norm_scaleR
  apply (erule mult_left_mono)
  apply simp
  done
```

```
lemma norm_add_rule_thm:
  fixes x1 x2 :: 'a::real_normed_vector
  shows  $\text{norm } x1 \leq b1 \implies \text{norm } x2 \leq b2 \implies \text{norm } (x1 + x2) \leq b1 + b2$ 
  by (rule order_trans [OF norm_triangle_ineq add_mono])
```

```
lemma ge_iff_diff_ge_0:
  fixes a :: 'a::linordered_ring
  shows  $a \geq b \equiv a - b \geq 0$ 
  by (simp add: field_simps)
```

```
lemma pth_1:
  fixes x :: 'a::real_normed_vector
  shows  $x \equiv \text{scaleR } 1 \ x$  by simp
```

```
lemma pth_2:
  fixes x :: 'a::real_normed_vector
  shows  $x - y \equiv x + -y$ 
  by (atomize (full)) simp
```

```
lemma pth_3:
  fixes x :: 'a::real_normed_vector
  shows  $-x \equiv \text{scaleR } (-1) \ x$ 
  by simp
```

```
lemma pth_4:
  fixes x :: 'a::real_normed_vector
```

```

shows scaleR 0 x  $\equiv$  0
and scaleR c 0 = (0::'a)
by simp_all

```

```

lemma pth_5:
  fixes x :: 'a::real_normed_vector
  shows scaleR c (scaleR d x)  $\equiv$  scaleR (c * d) x
  by simp

```

```

lemma pth_6:
  fixes x :: 'a::real_normed_vector
  shows scaleR c (x + y)  $\equiv$  scaleR c x + scaleR c y
  by (simp add: scaleR_right_distrib)

```

```

lemma pth_7:
  fixes x :: 'a::real_normed_vector
  shows 0 + x  $\equiv$  x
  and x + 0  $\equiv$  x
  by simp_all

```

```

lemma pth_8:
  fixes x :: 'a::real_normed_vector
  shows scaleR c x + scaleR d x  $\equiv$  scaleR (c + d) x
  by (simp add: scaleR_left_distrib)

```

```

lemma pth_9:
  fixes x :: 'a::real_normed_vector
  shows (scaleR c x + z) + scaleR d x  $\equiv$  scaleR (c + d) x + z
  and scaleR c x + (scaleR d x + z)  $\equiv$  scaleR (c + d) x + z
  and (scaleR c x + w) + (scaleR d x + z)  $\equiv$  scaleR (c + d) x + (w + z)
  by (simp_all add: algebra_simps)

```

```

lemma pth_a:
  fixes x :: 'a::real_normed_vector
  shows scaleR 0 x + y  $\equiv$  y
  by simp

```

```

lemma pth_b:
  fixes x :: 'a::real_normed_vector
  shows scaleR c x + scaleR d y  $\equiv$  scaleR c x + scaleR d y
  and (scaleR c x + z) + scaleR d y  $\equiv$  scaleR c x + (z + scaleR d y)
  and scaleR c x + (scaleR d y + z)  $\equiv$  scaleR c x + (scaleR d y + z)
  and (scaleR c x + w) + (scaleR d y + z)  $\equiv$  scaleR c x + (w + (scaleR d y +
z))
  by (simp_all add: algebra_simps)

```

```

lemma pth_c:
  fixes x :: 'a::real_normed_vector
  shows scaleR c x + scaleR d y  $\equiv$  scaleR d y + scaleR c x

```

```

    and (scaleR c x + z) + scaleR d y  $\equiv$  scaleR d y + (scaleR c x + z)
    and scaleR c x + (scaleR d y + z)  $\equiv$  scaleR d y + (scaleR c x + z)
    and (scaleR c x + w) + (scaleR d y + z)  $\equiv$  scaleR d y + ((scaleR c x + w) +
z)
  by (simp_all add: algebra_simps)

```

```

lemma pth_d:
  fixes x :: 'a::real_normed_vector
  shows x + 0  $\equiv$  x
  by simp

```

```

lemma norm_imp_pos_and_ge:
  fixes x :: 'a::real_normed_vector
  shows norm x  $\equiv$  n  $\implies$  norm x  $\geq$  0  $\wedge$  n  $\geq$  norm x
  by atomize auto

```

```

lemma real_eq_0_iff_le_ge_0:
  fixes x :: real
  shows x = 0  $\equiv$  x  $\geq$  0  $\wedge$  - x  $\geq$  0
  by arith

```

```

lemma norm_pths:
  fixes x :: 'a::real_normed_vector
  shows x = y  $\longleftrightarrow$  norm (x - y)  $\leq$  0
    and x  $\neq$  y  $\longleftrightarrow$   $\neg$  (norm (x - y)  $\leq$  0)
  using norm_ge_zero[of x - y] by auto

```

```

lemmas arithmetic_simps =
  arith_simps
  add_numeral_special
  add_neg_numeral_special
  mult_1_left
  mult_1_right

```

```
ML_file <normarith.ML>
```

```

method_setup norm = <
  Scan.succeed (SIMPLE_METHOD' o NormArith.norm_arith_tac)
> prove simple linear statements about vector norms

```

Hence more metric properties.

```

proposition dist_triangle_add:
  fixes x y x' y' :: 'a::real_normed_vector
  shows dist (x + y) (x' + y')  $\leq$  dist x x' + dist y y'
  by norm

```

```

lemma dist_triangle_add_half:
  fixes x x' y y' :: 'a::real_normed_vector
  shows dist x x' < e / 2  $\implies$  dist y y' < e / 2  $\implies$  dist(x + y) (x' + y') < e

```

596

by *norm*

end

Chapter 4

Vector Analysis

```
theory Topology_Euclidean_Space
imports
  Elementary_Normed_Spaces
  Linear_Algebra
  Norm_Arith
begin
```

4.1 Elementary Topology in Euclidean Space

```
lemma euclidean_dist_l2:
  fixes  $x\ y :: 'a :: euclidean\_space$ 
  shows  $\text{dist } x\ y = L2\_set\ (\lambda i. \text{dist } (x \cdot i)\ (y \cdot i))\ Basis$ 
  unfolding  $\text{dist\_norm}\ \text{norm\_eq\_sqrt\_inner}\ L2\_set\_def$ 
  by ( $\text{subst euclidean\_inner}$ ) ( $\text{simp add: power2\_eq\_square inner\_diff\_left}$ )
```

```
lemma norm_nth_le:  $\text{norm } (x \cdot i) \leq \text{norm } x$  if  $i \in Basis$ 
proof –
  have  $(x \cdot i)^2 = (\sum_{i \in \{i\}} (x \cdot i)^2)$ 
  by simp
  also have  $\dots \leq (\sum_{i \in Basis} (x \cdot i)^2)$ 
  by ( $\text{intro sum\_mono2}$ ) ( $\text{auto simp: that}$ )
  finally show ?thesis
  unfolding  $\text{norm\_conv\_dist euclidean\_dist\_l2[of } x]\ L2\_set\_def$ 
  by ( $\text{auto intro!: real\_le\_rsqrt}$ )
qed
```

4.1.1 Continuity of the representation WRT an orthogonal basis

```
lemma orthogonal_Basis: pairwise orthogonal Basis
  by ( $\text{simp add: inner\_not\_same\_Basis orthogonal\_def pairwise\_def}$ )
```

```
lemma representation_bound:
  fixes  $B :: 'N :: \text{real\_inner set}$ 
```

assumes *finite B independent B b ∈ B and orth: pairwise orthogonal B*
obtains *m where m > 0 ∧ x. x ∈ span B ⇒ |representation B x b| ≤ m * norm x*
proof
fix *x*
assume *x: x ∈ span B*
have *b ≠ 0*
using *⟨independent B⟩ ⟨b ∈ B⟩ dependent_zero by blast*
have *[simp]: b · b' = (if b' = b then (norm b)² else 0)*
if *b ∈ B b' ∈ B for b b'*
using *orth by (simp add: orthogonal_def pairwise_def norm_eq_sqrt_inner that)*
have *norm x = norm (∑ b∈B. representation B x b *_R b)*
using *real_vector.sum_representation_eq [OF ⟨independent B⟩ x ⟨finite B⟩ order_refl]*
by *simp*
also have *... = sqrt ((∑ b∈B. representation B x b *_R b) · (∑ b∈B. representation B x b *_R b))*
by *(simp add: norm_eq_sqrt_inner)*
also have *... = sqrt (∑ b∈B. (representation B x b *_R b) · (representation B x b *_R b))*
using *⟨finite B⟩*
by *(simp add: inner_sum_left inner_sum_right if_distrib [of λx. _ * x] cong: if_cong sum.cong_simp)*
also have *... = sqrt (∑ b∈B. (norm (representation B x b *_R b))²)*
by *(simp add: mult.commute mult.left_commute power2_eq_square)*
also have *... = sqrt (∑ b∈B. (representation B x b)² * (norm b)²)*
by *(simp add: norm_mult power_mult_distrib)*
finally have *norm x = sqrt (∑ b∈B. (representation B x b)² * (norm b)²) .*
moreover
have *sqrt ((representation B x b)² * (norm b)²) ≤ sqrt (∑ b∈B. (representation B x b)² * (norm b)²)*
using *⟨b ∈ B⟩ ⟨finite B⟩ by (auto intro: member_le_sum)*
then have *|representation B x b| ≤ (1 / norm b) * sqrt (∑ b∈B. (representation B x b)² * (norm b)²)*
using *⟨b ≠ 0⟩ by (simp add: field_split_simps real_sqrt_mult del: real_sqrt_le_iff)*
ultimately show *|representation B x b| ≤ (1 / norm b) * norm x*
by *simp*
next
show *0 < 1 / norm b*
using *⟨independent B⟩ ⟨b ∈ B⟩ dependent_zero by auto*
qed

lemma *continuous_on_representation:*

fixes *B :: 'N::euclidean_space set*

assumes *finite B independent B b ∈ B pairwise orthogonal B*

shows *continuous_on (span B) (λx. representation B x b)*

proof

show *∃ d > 0. ∀ x' ∈ span B. dist x' x < d ⟶ dist (representation B x' b) (representation B x b)*

```

 $B\ x\ b) \leq e$ 
  if  $e > 0$   $x \in \text{span } B$  for  $x\ e$ 
proof -
  obtain  $m$  where  $m > 0$  and  $m$ :  $\bigwedge x. x \in \text{span } B \implies |\text{representation } B\ x\ b| \leq$ 
 $m * \text{norm } x$ 
  using assms representation_bound by blast
  show ?thesis
  unfolding dist_norm
  proof (intro exI conjI ballI impI)
    show  $e/m > 0$ 
    by (simp add:  $\langle e > 0 \rangle \langle m > 0 \rangle$ )
    show  $\text{norm } (\text{representation } B\ x'\ b - \text{representation } B\ x\ b) \leq e$ 
    if  $x': x' \in \text{span } B$  and less:  $\text{norm } (x' - x) < e/m$  for  $x'$ 
    proof -
      have  $|\text{representation } B\ (x' - x)\ b| \leq m * \text{norm } (x' - x)$ 
      using  $m$  [of  $x' - x$ ]  $\langle x \in \text{span } B \rangle$  span_diff  $x'$  by blast
      also have  $\dots < e$ 
      by (metis  $\langle m > 0 \rangle$  less mult.commute pos_less_divide_eq)
      finally have  $|\text{representation } B\ (x' - x)\ b| \leq e$  by simp
      then show ?thesis
      by (simp add:  $\langle x \in \text{span } B \rangle \langle \text{independent } B \rangle$  representation_diff  $x'$ )
    qed
  qed
qed
qed
qed

```

4.1.2 Balls in Euclidean Space

```

lemma cball_subset_cball_iff:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  shows  $\text{cball } a\ r \subseteq \text{cball } a'\ r' \longleftrightarrow \text{dist } a\ a' + r \leq r' \vee r < 0$ 
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  assume ?lhs
  then show ?rhs
  proof (cases  $r < 0$ )
    case True
    then show ?rhs by simp
  next
    case False
    then have [simp]:  $r \geq 0$  by simp
    have  $\text{norm } (a - a') + r \leq r'$ 
    proof (cases  $a = a'$ )
      case True
      then show ?thesis
      using subsetD [where  $c = a + r *_R (\text{SOME } i. i \in \text{Basis}), \text{OF } \langle ?lhs \rangle$ ]
      subsetD [where  $c = a, \text{OF } \langle ?lhs \rangle$ ]
      by (force simp: SOME_Basis dist_norm)
    next

```

```

      case False
      have norm (a' - (a + (r / norm (a - a')) *R (a - a')))) = norm (a' - a -
(r / norm (a - a')) *R (a - a'))
        by (simp add: algebra_simps)
      also have ... = norm ((-1 - (r / norm (a - a')) *R (a - a'))
        by (simp add: algebra_simps)
      also from ⟨a ≠ a'⟩ have ... = |- norm (a - a') - r|
        by simp (simp add: field_simps)
      finally have [simp]: norm (a' - (a + (r / norm (a - a')) *R (a - a')))) =
|norm (a - a') + r|
        by linarith
      from ⟨a ≠ a'⟩ show ?thesis
        using subsetD [where c = a' + (1 + r / norm(a - a')) *R (a - a'), OF
⟨?lhs⟩]
        by (simp add: dist_norm scaleR_add_left)
      qed
    then show ?rhs
      by (simp add: dist_norm)
    qed
  qed metric

lemma cball_subset_ball_iff: cball a r ⊆ ball a' r' ⟷ dist a a' + r < r' ∨ r <
0
  (is ?lhs ⟷ ?rhs)
  for a :: 'a::euclidean_space
proof
  assume ?lhs
  then show ?rhs
  proof (cases r < 0)
    case True then
      show ?rhs by simp
    next
      case False
      then have [simp]: r ≥ 0 by simp
      have norm (a - a') + r < r'
      proof (cases a = a')
        case True
        then show ?thesis
          using subsetD [where c = a + r *R (SOME i. i ∈ Basis), OF ⟨?lhs⟩]
subsetD [where c = a, OF ⟨?lhs⟩]
          by (force simp: SOME_Basis dist_norm)
        next
          case False
          have False if norm (a - a') + r ≥ r'
          proof -
            from that have |r' - norm (a - a')| ≤ r
              by (simp split: abs_split)
            (metis ⟨0 ≤ r⟩ ⟨?lhs⟩ centre_in_cball dist_commute dist_norm less_asym
mem_ball subset_eq)
          qed
      qed
  qed

```

```

      then show ?thesis
        using subsetD [where c = a + (r' / norm(a - a') - 1) *R (a - a'), OF
  <?lhs>] <a ≠ a'>
        apply (simp add: dist_norm)
        apply (simp add: scaleR_left_diff_distrib)
        apply (simp add: field_simps)
        done
      qed
    then show ?thesis by force
  qed
  then show ?rhs by (simp add: dist_norm)
qed
next
  assume ?rhs
  then show ?lhs
    by metric
qed

lemma ball_subset_cball_iff: ball a r ⊆ cball a' r' ⟷ dist a a' + r ≤ r' ∨ r ≤
0
  (is ?lhs = ?rhs)
  for a :: 'a::euclidean_space
proof (cases r ≤ 0)
  case True
  then show ?thesis
    by metric
next
  case False
  show ?thesis
  proof
    assume ?lhs
    then have (cball a r ⊆ cball a' r')
      by (metis False closed_cball closure_ball closure_closed closure_mono not_less)
    with False show ?rhs
      by (fastforce iff: cball_subset_cball_iff)
  next
    assume ?rhs
    with False show ?lhs
      by metric
  qed
qed

lemma ball_subset_ball_iff:
  fixes a :: 'a :: euclidean_space
  shows ball a r ⊆ ball a' r' ⟷ dist a a' + r ≤ r' ∨ r ≤ 0
    (is ?lhs = ?rhs)
proof (cases r ≤ 0)
  case True then show ?thesis
    by metric

```

```

next
  case False show ?thesis
  proof
    assume ?lhs
    then have  $0 < r'$ 
      using False by metric
    then have  $(cball\ a\ r \subseteq cball\ a'\ r')$ 
      by (metis False<?lhs> closure_ball closure_mono not_less)
    then show ?rhs
      using False cball_subset_cball_iff by fastforce
  qed metric
qed

```

```

lemma ball_eq_ball_iff:
  fixes  $x :: 'a :: euclidean\_space$ 
  shows  $ball\ x\ d = ball\ y\ e \longleftrightarrow d \leq 0 \wedge e \leq 0 \vee x=y \wedge d=e$ 
    (is ?lhs = ?rhs)
  proof
    assume ?lhs
    then show ?rhs
      proof (cases  $d \leq 0 \vee e \leq 0$ )
        case True
          with <?lhs> show ?rhs
            by safe (simp_all only: ball_eq_empty [of y e, symmetric] ball_eq_empty
[of x d, symmetric])
        next
          case False
            with <?lhs> show ?rhs
              apply (auto simp: set_eq_subset ball_subset_ball_iff dist_norm norm_minus_commute
algebra_simps)
                apply (metis add_le_same_cancel1 le_add_same_cancel1 norm_ge_zero
norm_pths(2) order_trans)
                apply (metis add_increasing2 add_le_imp_le_right eq_iff norm_ge_zero)
                done
          qed
        next
          assume ?rhs then show ?lhs
            by (auto simp: set_eq_subset ball_subset_ball_iff)
          qed
    qed
  next
    assume ?rhs then show ?lhs
      by (auto simp: set_eq_subset ball_subset_ball_iff)
    qed
  qed

```

```

lemma cball_eq_cball_iff:
  fixes  $x :: 'a :: euclidean\_space$ 
  shows  $cball\ x\ d = cball\ y\ e \longleftrightarrow d < 0 \wedge e < 0 \vee x=y \wedge d=e$ 
    (is ?lhs = ?rhs)
  proof
    assume ?lhs
    then show ?rhs
      proof (cases  $d < 0 \vee e < 0$ )

```

```

    case True
      with ‹?lhs› show ?rhs
      by safe (simp_all only: cball_eq_empty [of y e, symmetric] cball_eq_empty
[of x d, symmetric])
    next
      case False
      with ‹?lhs› show ?rhs
      apply (auto simp: set_eq_subset cball_subset_cball_iff dist_norm norm_minus_commute
algebra_simps)
      apply (metis add_le_same_cancel1 le_add_same_cancel1 norm_ge_zero
norm_pths(2) order_trans)
      apply (metis add_increasing2 add_le_imp_le_right eq_iff norm_ge_zero)
      done
    qed
  next
    assume ?rhs then show ?lhs
      by (auto simp: set_eq_subset cball_subset_cball_iff)
    qed

lemma ball_eq_cball_iff:
  fixes x :: 'a :: euclidean_space
  shows ball x d = cball y e  $\longleftrightarrow$  d  $\leq$  0  $\wedge$  e < 0 (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    apply (auto simp: set_eq_subset ball_subset_cball_iff cball_subset_ball_iff
algebra_simps)
    apply (metis add_increasing2 add_le_cancel_right add_less_same_cancel1
dist_not_less_zero less_le_trans zero_le_dist)
    apply (metis add_less_same_cancel1 dist_not_less_zero less_le_trans not_le)
    using ‹?lhs› ball_eq_empty cball_eq_empty apply blast+
    done
  next
    assume ?rhs then show ?lhs by auto
  qed

lemma cball_eq_ball_iff:
  fixes x :: 'a :: euclidean_space
  shows cball x d = ball y e  $\longleftrightarrow$  d < 0  $\wedge$  e  $\leq$  0
  using ball_eq_cball_iff by blast

lemma finite_ball_avoid:
  fixes S :: 'a :: euclidean_space set
  assumes open S finite X p  $\in$  S
  shows  $\exists e > 0. \forall w \in \text{ball } p \ e. w \in S \wedge (w \neq p \longrightarrow w \notin X)$ 
proof -
  obtain e1 where 0 < e1 and e1_b: ball p e1  $\subseteq$  S
  using open_contains_ball_eq[OF ‹open S›] assms by auto
  obtain e2 where 0 < e2 and  $\forall x \in X. x \neq p \longrightarrow e2 \leq \text{dist } p \ x$ 

```

```

    using finite_set_avoid[OF ‹finite X›, of p] by auto
    hence  $\forall w \in \text{ball } p \text{ (min } e1 \ e2). w \in S \wedge (w \neq p \longrightarrow w \notin X)$  using e1_b by auto
    thus  $\exists e > 0. \forall w \in \text{ball } p \ e. w \in S \wedge (w \neq p \longrightarrow w \notin X)$  using ‹e2 > 0› ‹e1 > 0›
    apply (rule_tac x=min e1 e2 in exI)
    by auto
qed

```

```

lemma finite_cball_avoid:
  fixes S :: 'a :: euclidean_space set
  assumes open S finite X p ∈ S
  shows  $\exists e > 0. \forall w \in \text{cball } p \ e. w \in S \wedge (w \neq p \longrightarrow w \notin X)$ 
proof -
  obtain e1 where e1 > 0 and e1:  $\forall w \in \text{ball } p \ e1. w \in S \wedge (w \neq p \longrightarrow w \notin X)$ 
  using finite_ball_avoid[OF assms] by auto
  define e2 where e2  $\equiv e1 / 2$ 
  have e2 > 0 and e2 < e1 unfolding e2_def using ‹e1 > 0› by auto
  then have cball p e2  $\subseteq \text{ball } p \ e1$  by (subst cball_subset_ball_iff, auto)
  then show  $\exists e > 0. \forall w \in \text{cball } p \ e. w \in S \wedge (w \neq p \longrightarrow w \notin X)$  using ‹e2 > 0›
e1 by auto
qed

```

```

lemma dim_cball:
  assumes e > 0
  shows dim (cball (0 :: 'n::euclidean_space) e) = DIM('n)
proof -
  {
    fix x :: 'n::euclidean_space
    define y where y = (e / norm x) *R x
    then have y ∈ cball 0 e
    using assms by auto
    moreover have *: x = (norm x / e) *R y
    using y_def assms by simp
    moreover from * have x = (norm x / e) *R y
    by auto
    ultimately have x ∈ span (cball 0 e)
    using span_scale[of y cball 0 e norm x / e]
    span_superset[of cball 0 e]
    by (simp add: span_base)
  }
  then have span (cball 0 e) = (UNIV :: 'n::euclidean_space set)
  by auto
  then show ?thesis
  using dim_span[of cball (0 :: 'n::euclidean_space) e] by (auto)
qed

```

4.1.3 Boxes

abbreviation One :: 'a::euclidean_space **where**
 One $\equiv \sum \text{Basis}$


```

lemma One_non_0: assumes One = (0::'a::euclidean_space) shows False
proof -
  have dependent (Basis :: 'a set)
  apply (simp add: dependent_finite)
  apply (rule_tac x= $\lambda i. 1$  in exI)
  using SOME_Basis apply (auto simp: assms)
  done
  with independent_Basis show False by force
qed

```

```

corollary One_neq_0[iff]: One  $\neq$  0
by (metis One_non_0)

```

```

corollary Zero_neq_One[iff]: 0  $\neq$  One
by (metis One_non_0)

```

```

definition (in euclidean_space) eucl_less (infix <e 50) where
eucl_less a b  $\longleftrightarrow (\forall i \in \text{Basis}. a \cdot i < b \cdot i)$ 

```

```

definition box_eucl_less: box a b = {x. a <e x  $\wedge$  x <e b}
definition cbox a b = {x.  $\forall i \in \text{Basis}. a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i$ }

```

```

lemma box_def: box a b = {x.  $\forall i \in \text{Basis}. a \cdot i < x \cdot i \wedge x \cdot i < b \cdot i$ }
and in_box_eucl_less: x  $\in$  box a b  $\longleftrightarrow$  a <e x  $\wedge$  x <e b
and mem_box: x  $\in$  box a b  $\longleftrightarrow (\forall i \in \text{Basis}. a \cdot i < x \cdot i \wedge x \cdot i < b \cdot i)$ 
x  $\in$  cbox a b  $\longleftrightarrow (\forall i \in \text{Basis}. a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i)$ 
by (auto simp: box_eucl_less eucl_less_def cbox_def)

```

```

lemma cbox_Pair_eq: cbox (a, c) (b, d) = cbox a b  $\times$  cbox c d
by (force simp: cbox_def Basis_prod_def)

```

```

lemma cbox_Pair_iff [iff]: (x, y)  $\in$  cbox (a, c) (b, d)  $\longleftrightarrow$  x  $\in$  cbox a b  $\wedge$  y  $\in$ 
cbox c d
by (force simp: cbox_Pair_eq)

```

```

lemma cbox_Complex_eq: cbox (Complex a c) (Complex b d) = ( $\lambda(x,y).$  Complex
x y) ' (cbox a b  $\times$  cbox c d)
apply (auto simp: cbox_def Basis_complex_def)
apply (rule_tac x = (Re x, Im x) in image_eqI)
using complex_eq by auto

```

```

lemma cbox_Pair_eq_0: cbox (a, c) (b, d) = {}  $\longleftrightarrow$  cbox a b = {}  $\vee$  cbox c d
= {}
by (force simp: cbox_Pair_eq)

```

```

lemma swap_cbox_Pair [simp]: prod.swap ' cbox (c, a) (d, b) = cbox (a,c) (b,d)
by auto

```

lemma *mem_box_real[simp]*:

$(x::\text{real}) \in \text{box } a \ b \longleftrightarrow a < x \wedge x < b$
 $(x::\text{real}) \in \text{cbox } a \ b \longleftrightarrow a \leq x \wedge x \leq b$
by (*auto simp: mem_box*)

lemma *box_real[simp]*:

fixes $a \ b:: \text{real}$
shows $\text{box } a \ b = \{a <..< b\}$ $\text{cbox } a \ b = \{a .. b\}$
by *auto*

lemma *box_Int_box*:

fixes $a :: 'a::\text{euclidean\_space}$
shows $\text{box } a \ b \cap \text{box } c \ d =$
 $\text{box } (\sum i \in \text{Basis}. \max (a \cdot i) (c \cdot i) *_R i) (\sum i \in \text{Basis}. \min (b \cdot i) (d \cdot i) *_R i)$
unfolding *set_eq_iff* **and** *Int_iff* **and** *mem_box* **by** *auto*

lemma *rational_boxes*:

fixes $x :: 'a::\text{euclidean\_space}$
assumes $e > 0$
shows $\exists a \ b. (\forall i \in \text{Basis}. a \cdot i \in \mathbb{Q} \wedge b \cdot i \in \mathbb{Q}) \wedge x \in \text{box } a \ b \wedge \text{box } a \ b \subseteq \text{ball } x$
 e

proof –

define e' **where** $e' = e / (2 * \text{sqrt } (\text{real } (\text{DIM } ('a))))$
then have $e: e' > 0$
using *assms* **by** (*auto*)
have $\forall i. \exists y. y \in \mathbb{Q} \wedge y < x \cdot i \wedge x \cdot i - y < e'$ (**is** $\forall i. ?th \ i$)
proof
fix i
from *Rats_dense_in_real*[*of* $x \cdot i - e' x \cdot i$] e
show $?th \ i$ **by** *auto*

qed

from *choice*[*OF this*] **obtain** a **where**

$a: \forall xa. a \cdot xa \in \mathbb{Q} \wedge a \cdot xa < x \cdot xa \wedge x \cdot xa - a \cdot xa < e' ..$
have $\forall i. \exists y. y \in \mathbb{Q} \wedge x \cdot i < y \wedge y - x \cdot i < e'$ (**is** $\forall i. ?th \ i$)

proof

fix i
from *Rats_dense_in_real*[*of* $x \cdot i x \cdot i + e'$] e
show $?th \ i$ **by** *auto*

qed

from *choice*[*OF this*] **obtain** b **where**

$b: \forall xa. b \cdot xa \in \mathbb{Q} \wedge x \cdot xa < b \cdot xa \wedge b \cdot xa - x \cdot xa < e' ..$
let $?a = \sum i \in \text{Basis}. a \cdot i *_R i$ **and** $?b = \sum i \in \text{Basis}. b \cdot i *_R i$

show $?thesis$

proof (*rule exI*[*of* $?a$], *rule exI*[*of* $?b$], *safe*)

fix $y :: 'a$
assume $*$: $y \in \text{box } ?a \ ?b$
have $\text{dist } x \ y = \text{sqrt } (\sum i \in \text{Basis}. (\text{dist } (x \cdot i) (y \cdot i))^2)$
unfolding *L2_set_def[symmetric]* **by** (*rule euclidean_dist_l2*)
also have $\dots < \text{sqrt } (\sum (i::'a) \in \text{Basis}. e'^2 / \text{real } (\text{DIM } ('a)))$

```

proof (rule real_sqrt_less_mono, rule sum_strict_mono)
  fix i :: 'a
  assume i: i ∈ Basis
  have a i < y·i ∧ y·i < b i
    using * i by (auto simp: box_def)
  moreover have a i < x·i x·i - a i < e'
    using a by auto
  moreover have x·i < b i b i - x·i < e'
    using b by auto
  ultimately have |x·i - y·i| < 2 * e'
    by auto
  then have dist (x · i) (y · i) < e/sqrt (real (DIM('a)))
    unfolding e'_def by (auto simp: dist_real_def)
  then have (dist (x · i) (y · i))2 < (e/sqrt (real (DIM('a))))2
    by (rule power_strict_mono) auto
  then show (dist (x · i) (y · i))2 < e2 / real DIM('a)
    by (simp add: power_divide)
qed auto
also have ... = e
  using ⟨0 < e⟩ by simp
finally show y ∈ ball x e
  by (auto simp: ball_def)
qed (insert a b, auto simp: box_def)
qed

```

```

lemma open_UNION_box:
  fixes M :: 'a::euclidean_space set
  assumes open M
  defines a' ≡ λf :: 'a ⇒ real × real. (∑ (i::'a)∈Basis. fst (f i) *R i)
  defines b' ≡ λf :: 'a ⇒ real × real. (∑ (i::'a)∈Basis. snd (f i) *R i)
  defines I ≡ {f∈Basis →E ℚ × ℚ. box (a' f) (b' f) ⊆ M}
  shows M = (⋃ f∈I. box (a' f) (b' f))
proof -
  have x ∈ (⋃ f∈I. box (a' f) (b' f)) if x ∈ M for x
  proof -
    obtain e where e: e > 0 ball x e ⊆ M
      using openE[OF ⟨open M⟩ ⟨x ∈ M⟩] by auto
    moreover obtain a b where ab:
      x ∈ box a b
      ∀ i ∈ Basis. a · i ∈ ℚ
      ∀ i ∈ Basis. b · i ∈ ℚ
      box a b ⊆ ball x e
      using rational_boxes[OF e(1)] by metis
    ultimately show ?thesis
      by (intro UN_I[of λi∈Basis. (a · i, b · i)])
        (auto simp: euclidean_representation I_def a'_def b'_def)
  qed
  then show ?thesis by (auto simp: I_def)
qed

```

```

corollary open_countable_Union_open_box:
  fixes  $S :: 'a :: euclidean\_space$  set
  assumes open  $S$ 
  obtains  $\mathcal{D}$  where countable  $\mathcal{D}$   $\mathcal{D} \subseteq \text{Pow } S \wedge X. X \in \mathcal{D} \implies \exists a b. X = \text{box } a b$ 
 $\bigcup \mathcal{D} = S$ 
proof –
  let  $?a = \lambda f. (\sum (i::'a) \in \text{Basis}. \text{fst } (f i) *_R i)$ 
  let  $?b = \lambda f. (\sum (i::'a) \in \text{Basis}. \text{snd } (f i) *_R i)$ 
  let  $?I = \{f \in \text{Basis} \rightarrow_E \mathbb{Q} \times \mathbb{Q}. \text{box } (?a f) (?b f) \subseteq S\}$ 
  let  $?D = (\lambda f. \text{box } (?a f) (?b f)) \text{ ` } ?I$ 
  show ?thesis
proof
  have countable  $?I$ 
    by (simp add: countable_PiE countable_rat)
  then show countable  $?D$ 
    by blast
  show  $\bigcup ?D = S$ 
    using open_UNION_box [OF assms] by metis
qed auto
qed

lemma rational_cboxes:
  fixes  $x :: 'a :: euclidean\_space$ 
  assumes  $e > 0$ 
  shows  $\exists a b. (\forall i \in \text{Basis}. a \cdot i \in \mathbb{Q} \wedge b \cdot i \in \mathbb{Q}) \wedge x \in \text{cbox } a b \wedge \text{cbox } a b \subseteq \text{ball } x e$ 
proof –
  define  $e'$  where  $e' = e / (2 * \text{sqrt } (\text{real } (\text{DIM } ('a))))$ 
  then have  $e: e' > 0$ 
    using assms by auto
  have  $\forall i. \exists y. y \in \mathbb{Q} \wedge y < x \cdot i \wedge x \cdot i - y < e'$  (is  $\forall i. ?th i$ )
proof
  fix  $i$ 
  from Rats_dense_in_real[of  $x \cdot i - e' x \cdot i$ ]  $e$ 
  show ?th i by auto
qed
  from choice[OF this] obtain  $a$  where
     $a: \forall u. a u \in \mathbb{Q} \wedge a u < x \cdot u \wedge x \cdot u - a u < e' ..$ 
  have  $\forall i. \exists y. y \in \mathbb{Q} \wedge x \cdot i < y \wedge y - x \cdot i < e'$  (is  $\forall i. ?th i$ )
proof
  fix  $i$ 
  from Rats_dense_in_real[of  $x \cdot i x \cdot i + e'$ ]  $e$ 
  show ?th i by auto
qed
  from choice[OF this] obtain  $b$  where
     $b: \forall u. b u \in \mathbb{Q} \wedge x \cdot u < b u \wedge b u - x \cdot u < e' ..$ 
  let  $?a = \sum i \in \text{Basis}. a i *_R i$  and  $?b = \sum i \in \text{Basis}. b i *_R i$ 
  show ?thesis

```

```

proof (rule exI[of _ ?a], rule exI[of _ ?b], safe)
  fix y :: 'a
  assume *: y ∈ cbox ?a ?b
  have dist x y = sqrt (∑ i∈Basis. (dist (x · i) (y · i))2)
    unfolding L2_set_def[symmetric] by (rule euclidean_dist_l2)
  also have ... < sqrt (∑ (i::'a)∈Basis. e2 / real (DIM('a)))
proof (rule real_sqrt_less_mono, rule sum_strict_mono)
  fix i :: 'a
  assume i: i ∈ Basis
  have a i ≤ y·i ∧ y·i ≤ b i
    using * i by (auto simp: cbox_def)
  moreover have a i < x·i x·i - a i < e'
    using a by auto
  moreover have x·i < b i b i - x·i < e'
    using b by auto
  ultimately have |x·i - y·i| < 2 * e'
    by auto
  then have dist (x · i) (y · i) < e/sqrt (real (DIM('a)))
    unfolding e'_def by (auto simp: dist_real_def)
  then have (dist (x · i) (y · i))2 < (e/sqrt (real (DIM('a))))2
    by (rule power_strict_mono) auto
  then show (dist (x · i) (y · i))2 < e2 / real DIM('a)
    by (simp add: power_divide)
qed auto
  also have ... = e
    using ⟨0 < e⟩ by simp
  finally show y ∈ ball x e
    by (auto simp: ball_def)
next
  show x ∈ cbox (∑ i∈Basis. a i *R i) (∑ i∈Basis. b i *R i)
    using a b less_imp_le by (auto simp: cbox_def)
qed (use a b cbox_def in auto)
qed

```

lemma open_UNION_cbox:

```

fixes M :: 'a::euclidean_space set
assumes open M
defines a' ≡ λf. (∑ (i::'a)∈Basis. fst (f i) *R i)
defines b' ≡ λf. (∑ (i::'a)∈Basis. snd (f i) *R i)
defines I ≡ {f∈Basis →E Q × Q. cbox (a' f) (b' f) ⊆ M}
shows M = (⋃ f∈I. cbox (a' f) (b' f))
proof -
  have x ∈ (⋃ f∈I. cbox (a' f) (b' f)) if x ∈ M for x
  proof -
    obtain e where e: e > 0 ball x e ⊆ M
    using openE[OF ⟨open M⟩ ⟨x ∈ M⟩] by auto
    moreover obtain a b where ab: x ∈ cbox a b ∀ i ∈ Basis. a · i ∈ Q
      ∀ i ∈ Basis. b · i ∈ Q cbox a b ⊆ ball x e
    using rational_cboxes[OF e(1)] by metis

```

```

ultimately show ?thesis
  by (intro UN_I[of  $\lambda i \in \text{Basis}. (a \cdot i, b \cdot i)$ ])
     (auto simp: euclidean_representation I_def a'_def b'_def)
qed
then show ?thesis by (auto simp: I_def)
qed

corollary open_countable_Union_open_cbox:
  fixes  $S :: 'a :: \text{euclidean\_space}$  set
  assumes open S
  obtains  $\mathcal{D}$  where countable  $\mathcal{D}$   $\mathcal{D} \subseteq \text{Pow } S \wedge X. X \in \mathcal{D} \implies \exists a b. X = \text{cbox } a$ 
 $b \bigcup \mathcal{D} = S$ 
proof -
  let  $?a = \lambda f. (\sum (i :: 'a) \in \text{Basis}. \text{fst } (f i) *_R i)$ 
  let  $?b = \lambda f. (\sum (i :: 'a) \in \text{Basis}. \text{snd } (f i) *_R i)$ 
  let  $?I = \{f \in \text{Basis} \rightarrow_E \mathbb{Q} \times \mathbb{Q}. \text{cbox } (?a f) (?b f) \subseteq S\}$ 
  let  $?D = (\lambda f. \text{cbox } (?a f) (?b f)) \text{ ` } ?I$ 
  show ?thesis
  proof
    have countable ?I
      by (simp add: countable_PiE countable_rat)
    then show countable ?D
      by blast
    show  $\bigcup ?D = S$ 
      using open_UNION_cbox [OF assms] by metis
  qed auto
qed

lemma box_eq_empty:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  shows  $(\text{box } a b = \{\}) \longleftrightarrow (\exists i \in \text{Basis}. b \cdot i \leq a \cdot i)$  (is ?th1)
  and  $(\text{cbox } a b = \{\}) \longleftrightarrow (\exists i \in \text{Basis}. b \cdot i < a \cdot i)$  (is ?th2)
proof -
  {
    fix  $i x$ 
    assume  $i: i \in \text{Basis}$  and  $as: b \cdot i \leq a \cdot i$  and  $x: x \in \text{box } a b$ 
    then have  $a \cdot i < x \cdot i \wedge x \cdot i < b \cdot i$ 
      unfolding mem_box by (auto simp: box_def)
    then have  $a \cdot i < b \cdot i$  by auto
    then have False using as by auto
  }
  moreover
  {
    assume  $as: \forall i \in \text{Basis}. \neg (b \cdot i \leq a \cdot i)$ 
    let  $?x = (1/2) *_R (a + b)$ 
    {
      fix  $i :: 'a$ 
      assume  $i: i \in \text{Basis}$ 
      have  $a \cdot i < b \cdot i$ 

```

```

    using as[THEN bspec[where x=i]] i by auto
  then have  $a \cdot i < ((1/2) *_R (a+b)) \cdot i ((1/2) *_R (a+b)) \cdot i < b \cdot i$ 
    by (auto simp: inner_add_left)
}
then have  $\text{box } a \ b \neq \{\}$ 
  using mem_box(1)[of ?x a b] by auto
}
ultimately show ?th1 by blast

{
  fix i x
  assume i:  $i \in \text{Basis}$  and as: $b \cdot i < a \cdot i$  and x: $x \in \text{cbox } a \ b$ 
  then have  $a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i$ 
    unfolding mem_box by auto
  then have  $a \cdot i \leq b \cdot i$  by auto
  then have False using as by auto
}
moreover
{
  assume as: $\forall i \in \text{Basis}. \neg (b \cdot i < a \cdot i)$ 
  let ?x =  $(1/2) *_R (a + b)$ 
  {
    fix i :: 'a
    assume i: $i \in \text{Basis}$ 
    have  $a \cdot i \leq b \cdot i$ 
      using as[THEN bspec[where x=i]] i by auto
    then have  $a \cdot i \leq ((1/2) *_R (a+b)) \cdot i ((1/2) *_R (a+b)) \cdot i \leq b \cdot i$ 
      by (auto simp: inner_add_left)
  }
  then have  $\text{cbox } a \ b \neq \{\}$ 
    using mem_box(2)[of ?x a b] by auto
}
ultimately show ?th2 by blast
qed

```

```

lemma box_ne_empty:
  fixes a :: 'a::euclidean_space
  shows  $\text{cbox } a \ b \neq \{\} \longleftrightarrow (\forall i \in \text{Basis}. a \cdot i \leq b \cdot i)$ 
  and  $\text{box } a \ b \neq \{\} \longleftrightarrow (\forall i \in \text{Basis}. a \cdot i < b \cdot i)$ 
  unfolding box_eq_empty[of a b] by fastforce+

```

```

lemma
  fixes a :: 'a::euclidean_space
  shows cbox_sing [simp]:  $\text{cbox } a \ a = \{a\}$ 
    and box_sing [simp]:  $\text{box } a \ a = \{\}$ 
  unfolding set_eq_iff mem_box eq_iff [symmetric]
  by (auto intro!: euclidean_eqI[where 'a='a])
    (metis all_not_in_conv nonempty_Basis)

```

```

lemma subset_box_imp:
  fixes a :: 'a::euclidean_space
  shows ( $\forall i \in \text{Basis}. a \cdot i \leq c \cdot i \wedge d \cdot i \leq b \cdot i$ )  $\implies$   $\text{cbox } c \ d \subseteq \text{cbox } a \ b$ 
    and ( $\forall i \in \text{Basis}. a \cdot i < c \cdot i \wedge d \cdot i < b \cdot i$ )  $\implies$   $\text{cbox } c \ d \subseteq \text{box } a \ b$ 
    and ( $\forall i \in \text{Basis}. a \cdot i \leq c \cdot i \wedge d \cdot i \leq b \cdot i$ )  $\implies$   $\text{box } c \ d \subseteq \text{cbox } a \ b$ 
    and ( $\forall i \in \text{Basis}. a \cdot i < c \cdot i \wedge d \cdot i < b \cdot i$ )  $\implies$   $\text{box } c \ d \subseteq \text{box } a \ b$ 
  unfolding subset_eq [unfolded Ball_def] unfolding mem_box
  by (best intro: order_trans less_le_trans le_less_trans less_imp_le)+

```

```

lemma box_subset_cbox:
  fixes a :: 'a::euclidean_space
  shows  $\text{box } a \ b \subseteq \text{cbox } a \ b$ 
  unfolding subset_eq [unfolded Ball_def] mem_box
  by (fast intro: less_imp_le)

```

```

lemma subset_box:
  fixes a :: 'a::euclidean_space
  shows  $\text{cbox } c \ d \subseteq \text{cbox } a \ b \longleftrightarrow (\forall i \in \text{Basis}. c \cdot i \leq d \cdot i) \longrightarrow (\forall i \in \text{Basis}. a \cdot i \leq c \cdot i$ 
 $\wedge d \cdot i \leq b \cdot i)$  (is ?th1)
    and  $\text{cbox } c \ d \subseteq \text{box } a \ b \longleftrightarrow (\forall i \in \text{Basis}. c \cdot i \leq d \cdot i) \longrightarrow (\forall i \in \text{Basis}. a \cdot i < c \cdot i \wedge$ 
 $d \cdot i < b \cdot i)$  (is ?th2)
    and  $\text{box } c \ d \subseteq \text{cbox } a \ b \longleftrightarrow (\forall i \in \text{Basis}. c \cdot i < d \cdot i) \longrightarrow (\forall i \in \text{Basis}. a \cdot i \leq c \cdot i \wedge$ 
 $d \cdot i \leq b \cdot i)$  (is ?th3)
    and  $\text{box } c \ d \subseteq \text{box } a \ b \longleftrightarrow (\forall i \in \text{Basis}. c \cdot i < d \cdot i) \longrightarrow (\forall i \in \text{Basis}. a \cdot i \leq c \cdot i \wedge$ 
 $d \cdot i \leq b \cdot i)$  (is ?th4)
  proof -
    let ?lesscd =  $\forall i \in \text{Basis}. c \cdot i < d \cdot i$ 
    let ?lerhs =  $\forall i \in \text{Basis}. a \cdot i \leq c \cdot i \wedge d \cdot i \leq b \cdot i$ 
    show ?th1 ?th2
      by (fastforce simp: mem_box)+
    have acdb:  $a \cdot i \leq c \cdot i \wedge d \cdot i \leq b \cdot i$ 
      if  $i: i \in \text{Basis}$  and  $\text{box } c \ d \subseteq \text{cbox } a \ b$  and  $cd: \bigwedge i. i \in \text{Basis} \implies c \cdot i < d \cdot i$  for  $i$ 
    proof -
      have  $\text{box } c \ d \neq \{\}$ 
        using that
        unfolding box_eq_empty by force
      { let ?x =  $(\sum j \in \text{Basis}. (\text{if } j=i \text{ then } ((\min (a \cdot j) (d \cdot j)) + c \cdot j)/2 \text{ else } (c \cdot j + d \cdot j)/2))$ 
 $*_R j) :: 'a$ 
        assume *:  $a \cdot i > c \cdot i$ 
        then have  $c \cdot j < ?x \cdot j \wedge ?x \cdot j < d \cdot j$  if  $j \in \text{Basis}$  for  $j$ 
          using  $cd$  that by (fastforce simp add: i *)
        then have  $?x \in \text{box } c \ d$ 
          unfolding mem_box by auto
        moreover have  $?x \notin \text{cbox } a \ b$ 
          using  $i \ cd$  * by (force simp: mem_box)
        ultimately have False using box by auto
      }
    then have  $a \cdot i \leq c \cdot i$  by force

```



```

moreover
{ let ?x = ( $\sum_{j \in \text{Basis}} (\text{if } j=i \text{ then } ((\max (b \cdot j) (c \cdot j)) + d \cdot j) / 2 \text{ else } (c \cdot j + d \cdot j) / 2)$ )
*_R j)::'a
  assume *:  $b \cdot i < d \cdot i$ 
  then have  $d \cdot j > ?x \cdot j \wedge ?x \cdot j > c \cdot j$  if  $j \in \text{Basis}$  for  $j$ 
    using cd that by (fastforce simp add: i *)
  then have  $?x \in \text{box } c \ d$ 
    unfolding mem_box by auto
  moreover have  $?x \notin \text{cbox } a \ b$ 
    using i cd * by (force simp: mem_box)
  ultimately have False using box by auto
}
then have  $b \cdot i \geq d \cdot i$  by (rule ccontr) auto
ultimately show ?thesis by auto
qed
show ?th3
  using acdb by (fastforce simp add: mem_box)
have acdb':  $a \cdot i \leq c \cdot i \wedge d \cdot i \leq b \cdot i$ 
  if  $i \in \text{Basis}$   $\text{box } c \ d \subseteq \text{box } a \ b \wedge i \in \text{Basis} \implies c \cdot i < d \cdot i$  for  $i$ 
    using box_subset_cbox[of a b] that acdb by auto
show ?th4
  using acdb' by (fastforce simp add: mem_box)
qed

lemma eq_cbox:  $\text{cbox } a \ b = \text{cbox } c \ d \longleftrightarrow \text{cbox } a \ b = \{\} \wedge \text{cbox } c \ d = \{\} \vee a = c \wedge b = d$ 
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then have  $\text{cbox } a \ b \subseteq \text{cbox } c \ d \wedge \text{cbox } c \ d \subseteq \text{cbox } a \ b$ 
    by auto
  then show ?rhs
    by (force simp: subset_box box_eq_empty intro: antisym euclidean_eqI)
next
  assume ?rhs
  then show ?lhs
    by force
qed

lemma eq_cbox_box [simp]:  $\text{cbox } a \ b = \text{box } c \ d \longleftrightarrow \text{cbox } a \ b = \{\} \wedge \text{box } c \ d = \{\}$ 
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  assume L: ?lhs
  then have  $\text{cbox } a \ b \subseteq \text{box } c \ d \wedge \text{box } c \ d \subseteq \text{cbox } a \ b$ 
    by auto
  then show ?rhs
    apply (simp add: subset_box)
    using L box_ne_empty box_sing apply (fastforce simp add:)

```

done
qed *force*

lemma *eq_box_cbox [simp]:* $\text{box } a \ b = \text{cbox } c \ d \longleftrightarrow \text{box } a \ b = \{\} \wedge \text{cbox } c \ d = \{\}$
by (*metis eq_cbox_box*)

lemma *eq_box:* $\text{box } a \ b = \text{box } c \ d \longleftrightarrow \text{box } a \ b = \{\} \wedge \text{box } c \ d = \{\} \vee a = c \wedge b = d$
(is *?lhs* $\longleftrightarrow$ *?rhs*)

proof

assume *L: ?lhs*

then have $\text{box } a \ b \subseteq \text{box } c \ d \ \text{box } c \ d \subseteq \text{box } a \ b$

by *auto*

then show *?rhs*

apply (*simp add: subset_box*)

using *box_ne_empty(2) L*

apply *auto*

apply (*meson euclidean_eqI less_eq_real_def not_less*)+

done

qed *force*

lemma *subset_box_complex:*

$\text{cbox } a \ b \subseteq \text{cbox } c \ d \longleftrightarrow$

$(\text{Re } a \leq \text{Re } b \wedge \text{Im } a \leq \text{Im } b) \longrightarrow \text{Re } a \geq \text{Re } c \wedge \text{Im } a \geq \text{Im } c \wedge \text{Re } b \leq \text{Re } d \wedge \text{Im } b \leq \text{Im } d$

$\text{box } a \ b \subseteq \text{box } c \ d \longleftrightarrow$

$(\text{Re } a \leq \text{Re } b \wedge \text{Im } a \leq \text{Im } b) \longrightarrow \text{Re } a > \text{Re } c \wedge \text{Im } a > \text{Im } c \wedge \text{Re } b < \text{Re } d \wedge \text{Im } b < \text{Im } d$

$\text{box } a \ b \subseteq \text{cbox } c \ d \longleftrightarrow$

$(\text{Re } a < \text{Re } b \wedge \text{Im } a < \text{Im } b) \longrightarrow \text{Re } a \geq \text{Re } c \wedge \text{Im } a \geq \text{Im } c \wedge \text{Re } b \leq \text{Re } d \wedge \text{Im } b \leq \text{Im } d$

$\text{box } a \ b \subseteq \text{box } c \ d \longleftrightarrow$

$(\text{Re } a < \text{Re } b \wedge \text{Im } a < \text{Im } b) \longrightarrow \text{Re } a \geq \text{Re } c \wedge \text{Im } a \geq \text{Im } c \wedge \text{Re } b \leq \text{Re } d \wedge \text{Im } b \leq \text{Im } d$

by (*subst subset_box; force simp: Basis_complex_def*)+

lemma *in_cbox_complex_iff:*

$x \in \text{cbox } a \ b \longleftrightarrow \text{Re } x \in \{\text{Re } a.. \text{Re } b\} \wedge \text{Im } x \in \{\text{Im } a.. \text{Im } b\}$

by (*cases x; cases a; cases b*) (*auto simp: cbox_Complex_eq*)

lemma *box_Complex_eq:*

$\text{box } (\text{Complex } a \ c) (\text{Complex } b \ d) = (\lambda(x,y). \text{Complex } x \ y) \text{ ` } (\text{box } a \ b \times \text{box } c \ d)$

by (*auto simp: box_def Basis_complex_def image_iff complex_eq_iff*)

lemma *in_box_complex_iff:*

$x \in \text{box } a \ b \longleftrightarrow \text{Re } x \in \{\text{Re } a <.. < \text{Re } b\} \wedge \text{Im } x \in \{\text{Im } a <.. < \text{Im } b\}$

by (*cases x; cases a; cases b*) (*auto simp: box_Complex_eq*)

lemma *Int_interval*:

fixes $a :: 'a::euclidean\_space$

shows $cbox\ a\ b \cap cbox\ c\ d =$

$cbox\ (\sum_{i \in Basis}. \max\ (a \cdot i)\ (c \cdot i) *_{\mathbb{R}} i)\ (\sum_{i \in Basis}. \min\ (b \cdot i)\ (d \cdot i) *_{\mathbb{R}} i)$

unfolding *set_eq_iff* **and** *Int_iff* **and** *mem_box*

by *auto*

lemma *disjoint_interval*:

fixes $a :: 'a::euclidean\_space$

shows $cbox\ a\ b \cap cbox\ c\ d = \{\} \longleftrightarrow (\exists i \in Basis. (b \cdot i < a \cdot i \vee d \cdot i < c \cdot i \vee b \cdot i < c \cdot i \vee d \cdot i < a \cdot i))$ (**is** *?th1*)

and $cbox\ a\ b \cap box\ c\ d = \{\} \longleftrightarrow (\exists i \in Basis. (b \cdot i < a \cdot i \vee d \cdot i \leq c \cdot i \vee b \cdot i \leq c \cdot i \vee d \cdot i \leq a \cdot i))$ (**is** *?th2*)

and $box\ a\ b \cap cbox\ c\ d = \{\} \longleftrightarrow (\exists i \in Basis. (b \cdot i \leq a \cdot i \vee d \cdot i < c \cdot i \vee b \cdot i \leq c \cdot i \vee d \cdot i \leq a \cdot i))$ (**is** *?th3*)

and $box\ a\ b \cap box\ c\ d = \{\} \longleftrightarrow (\exists i \in Basis. (b \cdot i \leq a \cdot i \vee d \cdot i \leq c \cdot i \vee b \cdot i \leq c \cdot i \vee d \cdot i \leq a \cdot i))$ (**is** *?th4*)

proof –

let $?z = (\sum_{i \in Basis}. (((\max\ (a \cdot i)\ (c \cdot i)) + (\min\ (b \cdot i)\ (d \cdot i))) / 2) *_{\mathbb{R}} i) :: 'a$

have $** : \bigwedge P\ Q. (\bigwedge i :: 'a. i \in Basis \implies Q\ ?z\ i \implies P\ i) \implies$

$(\bigwedge i\ x :: 'a. i \in Basis \implies P\ i \implies Q\ x\ i) \implies (\forall x. \exists i \in Basis. Q\ x\ i) \longleftrightarrow (\exists i \in Basis. P\ i)$

by *blast*

note $* = set\_eq\_iff\ Int\_iff\ empty\_iff\ mem\_box\ ball\_conj\_distrib[symmetric]\ eq\_False\ ball\_simps(10)$

show *?th1* **unfolding** $*$ **by** (*intro* $**$) *auto*

show *?th2* **unfolding** $*$ **by** (*intro* $**$) *auto*

show *?th3* **unfolding** $*$ **by** (*intro* $**$) *auto*

show *?th4* **unfolding** $*$ **by** (*intro* $**$) *auto*

qed

lemma *UN_box_eq_UNIV*: $(\bigcup i :: nat. box\ (-\ (real\ i *_{\mathbb{R}}\ One))\ (real\ i *_{\mathbb{R}}\ One)) = UNIV$

proof –

have $|x \cdot b| < real\_of\_int\ (\lceil Max\ ((\lambda b. |x \cdot b|)\ 'Basis) \rceil + 1)$

if [*simp*]: $b \in Basis$ **for** $x\ b :: 'a$

proof –

have $|x \cdot b| \leq real\_of\_int\ \lceil |x \cdot b| \rceil$

by (*rule* *le_of_int_ceiling*)

also have $\dots \leq real\_of\_int\ \lceil Max\ ((\lambda b. |x \cdot b|)\ 'Basis) \rceil$

by (*auto* *intro!*: *ceiling_mono*)

also have $\dots < real\_of\_int\ (\lceil Max\ ((\lambda b. |x \cdot b|)\ 'Basis) \rceil + 1)$

by *simp*

finally show *?thesis* .

qed

then have $\exists n :: nat. \forall b \in Basis. |x \cdot b| < real\ n$ **for** $x :: 'a$

by (*metis* *order.strict_trans_reals_Archimedean2*)

moreover have $\bigwedge x\ b :: 'a. \bigwedge n :: nat. |x \cdot b| < real\ n \longleftrightarrow -\ real\ n < x \cdot b \wedge x \cdot b < real\ n$

```

    by auto
  ultimately show ?thesis
  by (auto simp: box_def inner_sum_left inner_Basis sum.If_cases)
qed

lemma image_affinity_cbox: fixes m::real
  fixes a b c :: 'a::euclidean_space
  shows ( $\lambda x. m *_{\mathbb{R}} x + c$ ) ' cbox a b =
    (if cbox a b = {} then {}
     else (if  $0 \leq m$  then cbox ( $m *_{\mathbb{R}} a + c$ ) ( $m *_{\mathbb{R}} b + c$ )
      else cbox ( $m *_{\mathbb{R}} b + c$ ) ( $m *_{\mathbb{R}} a + c$ ))))
proof (cases m = 0)
case True
{
  fix x
  assume  $\forall i \in \text{Basis}. x \cdot i \leq c \cdot i \ \forall i \in \text{Basis}. c \cdot i \leq x \cdot i$ 
  then have  $x = c$ 
  by (simp add: dual_order.antisym euclidean_eqI)
}
moreover have  $c \in \text{cbox } (m *_{\mathbb{R}} a + c) (m *_{\mathbb{R}} b + c)$ 
unfolding True by (auto)
ultimately show ?thesis using True by (auto simp: cbox_def)
next
case False
{
  fix y
  assume  $\forall i \in \text{Basis}. a \cdot i \leq y \cdot i \ \forall i \in \text{Basis}. y \cdot i \leq b \cdot i \ m > 0$ 
  then have  $\forall i \in \text{Basis}. (m *_{\mathbb{R}} a + c) \cdot i \leq (m *_{\mathbb{R}} y + c) \cdot i$  and  $\forall i \in \text{Basis}. (m *_{\mathbb{R}} y + c) \cdot i \leq (m *_{\mathbb{R}} b + c) \cdot i$ 
  by (auto simp: inner_distrib)
}
moreover
{
  fix y
  assume  $\forall i \in \text{Basis}. a \cdot i \leq y \cdot i \ \forall i \in \text{Basis}. y \cdot i \leq b \cdot i \ m < 0$ 
  then have  $\forall i \in \text{Basis}. (m *_{\mathbb{R}} b + c) \cdot i \leq (m *_{\mathbb{R}} y + c) \cdot i$  and  $\forall i \in \text{Basis}. (m *_{\mathbb{R}} y + c) \cdot i \leq (m *_{\mathbb{R}} a + c) \cdot i$ 
  by (auto simp: mult_left_mono_neg inner_distrib)
}
moreover
{
  fix y
  assume  $m > 0$  and  $\forall i \in \text{Basis}. (m *_{\mathbb{R}} a + c) \cdot i \leq y \cdot i$  and  $\forall i \in \text{Basis}. y \cdot i \leq (m *_{\mathbb{R}} b + c) \cdot i$ 
  then have  $y \in (\lambda x. m *_{\mathbb{R}} x + c)$  ' cbox a b
  unfolding image_iff Bex_def mem_box
  apply (intro exI[where  $x = (1 / m) *_{\mathbb{R}} (y - c)$ ])
  apply (auto simp: pos_le_divide_eq pos_divide_le_eq mult.commute inner_distrib inner_diff_left)

```

```

    done
  }
  moreover
  {
    fix y
    assume  $\forall i \in \text{Basis}. (m *_{\mathbb{R}} b + c) \cdot i \leq y \cdot i \ \forall i \in \text{Basis}. y \cdot i \leq (m *_{\mathbb{R}} a + c) \cdot i$ 
     $m < 0$ 
    then have  $y \in (\lambda x. m *_{\mathbb{R}} x + c)$  ‘cbox  $a \ b$ 
    unfolding image_iff Bex_def mem_box
    apply (intro exI[where  $x = (1 / m) *_{\mathbb{R}} (y - c)$ ])
    apply (auto simp: neg_le_divide_eq neg_divide_le_eq mult.commute inner_distrib inner_diff_left)
    done
  }
  ultimately show ?thesis using False by (auto simp: cbox_def)
qed

```

```

lemma image_smult_cbox:  $(\lambda x. m *_{\mathbb{R}} (x :: \text{euclidean\_space}))$  ‘cbox  $a \ b =$ 
  (if cbox  $a \ b = \{\}$  then  $\{\}$  else if  $0 \leq m$  then cbox  $(m *_{\mathbb{R}} a) (m *_{\mathbb{R}} b)$  else cbox
   $(m *_{\mathbb{R}} b) (m *_{\mathbb{R}} a)$ )
  using image_affinity_cbox[of  $m \ 0 \ a \ b$ ] by auto

```

```

lemma swap_continuous:
  assumes continuous_on  $(\text{cbox } (a,c) (b,d)) (\lambda(x,y). f \ x \ y)$ 
  shows continuous_on  $(\text{cbox } (c,a) (d,b)) (\lambda(x,y). f \ y \ x)$ 
proof -
  have  $(\lambda(x,y). f \ y \ x) = (\lambda(x,y). f \ x \ y) \circ \text{prod.swap}$ 
  by auto
  then show ?thesis
  apply (rule ssubst)
  apply (rule continuous_on_compose)
  apply (simp add: split_def)
  apply (rule continuous_intros | simp add: assms) +
  done
qed

```

4.1.4 General Intervals

```

definition is_interval  $(s :: ('a :: \text{euclidean\_space}) \text{ set}) \longleftrightarrow$ 
   $(\forall a \in s. \forall b \in s. \forall x. (\forall i \in \text{Basis}. ((a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i) \vee (b \cdot i \leq x \cdot i \wedge x \cdot i \leq a \cdot i)))$ 
   $\longrightarrow x \in s)$ 

```

```

lemma is_interval_1:
  is_interval  $(s :: \text{real set}) \longleftrightarrow (\forall a \in s. \forall b \in s. \forall x. a \leq x \wedge x \leq b \longrightarrow x \in s)$ 
  unfolding is_interval_def by auto

```

```

lemma is_interval_Int: is_interval  $X \implies \text{is\_interval } Y \implies \text{is\_interval } (X \cap Y)$ 
  unfolding is_interval_def

```

by *blast*

lemma *is_interval_cbox* [*simp*]: *is_interval* (*cbox* *a* (*b*::'*a*::euclidean_space')) (**is** *?th1*)
and *is_interval_box* [*simp*]: *is_interval* (*box* *a* *b*) (**is** *?th2*)
unfolding *is_interval_def* *mem_box* *Ball_def* *atLeastAtMost_iff*
by (*meson* *order_trans* *le_less_trans* *less_le_trans* *less_trans*)**+**

lemma *is_interval_empty* [*iff*]: *is_interval* {}
unfolding *is_interval_def* **by** *simp*

lemma *is_interval_univ* [*iff*]: *is_interval* *UNIV*
unfolding *is_interval_def* **by** *simp*

lemma *mem_is_intervalI*:
assumes *is_interval* *s*
and *a* $\in$ *s* *b* $\in$ *s*
and $\bigwedge i. i \in \text{Basis} \implies a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i \vee b \cdot i \leq x \cdot i \wedge x \cdot i \leq a \cdot i$
shows *x* $\in$ *s*
by (*rule* *assms*(1)[*simplified is_interval_def*, *rule_format*, *OF* *assms*(2,3,4)])

lemma *interval_subst*:
fixes *S*::'*a*::euclidean_space set
assumes *is_interval* *S*
and *x* $\in$ *S* *y* *j* $\in$ *S*
and *j* $\in$ *Basis*
shows $(\sum_{i \in \text{Basis}} (\text{if } i = j \text{ then } y \cdot i \cdot i \text{ else } x \cdot i) *_{\mathbb{R}} i) \in S$
by (*rule* *mem_is_intervalI*[*OF* *assms*(1,2)]) (*auto simp: assms*)

lemma *mem_box_componentwiseI*:
fixes *S*::'*a*::euclidean_space set
assumes *is_interval* *S*
assumes $\bigwedge i. i \in \text{Basis} \implies x \cdot i \in ((\lambda x. x \cdot i) ' S)$
shows *x* $\in$ *S*
proof –
from *assms* **have** $\forall i \in \text{Basis}. \exists s \in S. x \cdot i = s \cdot i$
by *auto*
with *finite_Basis* **obtain** *s* **and** *bs*::'*a* list
where *s*: $\bigwedge i. i \in \text{Basis} \implies x \cdot i = s \cdot i$ **and** $\bigwedge i. i \in \text{Basis} \implies s \cdot i \in S$
and *bs*: *set* *bs* = *Basis* *distinct* *bs*
by (*metis* *finite_distinct_list*)
from *nonempty_Basis* *s* **obtain** *j* **where** *j*: *j* $\in$ *Basis* *s* *j* $\in$ *S*
by *blast*

define *y* **where**

y = *rec_list* (*s* *j*) ($\lambda j \_ Y. (\sum_{i \in \text{Basis}} (\text{if } i = j \text{ then } s \cdot i \cdot i \text{ else } Y \cdot i) *_{\mathbb{R}} i)$)
have *x* = $(\sum_{i \in \text{Basis}} (\text{if } i \in \text{set } bs \text{ then } s \cdot i \cdot i \text{ else } s \cdot j \cdot i) *_{\mathbb{R}} i)$
using *bs* **by** (*auto simp: s*(1)[*symmetric*] *euclidean_representation*)
also have [*symmetric*]: *y* *bs* = ...

```

    using bs(2) bs(1)[THEN equalityD1]
  by (induct bs) (auto simp: y_def euclidean_representation intro!: euclidean_eqI[where
'a='a])
  also have y bs ∈ S
    using bs(1)[THEN equalityD1]
  apply (induct bs)
  apply (auto simp: y_def j)
  apply (rule interval_subst[OF assms(1)])
  apply (auto simp: s)
  done
  finally show ?thesis .
qed

```

```

lemma cbox01_nonempty [simp]: cbox 0 One ≠ {}
  by (simp add: box_ne_empty inner_Basis inner_sum_left sum_nonneg)

```

```

lemma box01_nonempty [simp]: box 0 One ≠ {}
  by (simp add: box_ne_empty inner_Basis inner_sum_left)

```

```

lemma empty_as_interval: {} = cbox One (0::'a::euclidean_space)
  using nonempty_Basis box01_nonempty box_eq_empty(1) box_ne_empty(1)
  by blast

```

```

lemma interval_subset_is_interval:
  assumes is_interval S
  shows cbox a b ⊆ S ⟷ cbox a b = {} ∨ a ∈ S ∧ b ∈ S (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs using box_ne_empty(1) mem_box(2) by fastforce
next
  assume ?rhs
  have cbox a b ⊆ S if a ∈ S b ∈ S
    using assms unfolding is_interval_def
  apply (clarsimp simp add: mem_box)
  using that by blast
  with ‹?rhs› show ?lhs
    by blast
qed

```

```

lemma is_real_interval_union:
  is_interval (X ∪ Y)
  if X: is_interval X and Y: is_interval Y and I: (X ≠ {} ⟹ Y ≠ {} ⟹ X
  ∩ Y ≠ {})
  for X Y::real set
proof -
  consider X ≠ {} Y ≠ {} | X = {} | Y = {} by blast
  then show ?thesis
  proof cases
    case 1

```

```

    then obtain  $r$  where  $r \in X \vee X \cap Y = \{\}$   $r \in Y \vee X \cap Y = \{\}$ 
    by blast
    then show ?thesis
    using I 1 X Y unfolding is_interval_1
    by (metis (full_types) Un_iff le_cases)
  qed (use that in auto)
qed

```

```

lemma is_interval_translationI:
  assumes is_interval X
  shows is_interval ((+) x ' X)
  unfolding is_interval_def
proof safe
  fix b d e
  assume  $b \in X$   $d \in X$ 
   $\forall i \in \text{Basis}. (x + b) \cdot i \leq e \cdot i \wedge e \cdot i \leq (x + d) \cdot i \vee$ 
     $(x + d) \cdot i \leq e \cdot i \wedge e \cdot i \leq (x + b) \cdot i$ 
  hence  $e - x \in X$ 
  by (intro mem_is_intervalI[OF assms  $\langle b \in X \rangle \langle d \in X \rangle$ , of  $e - x$ ])
    (auto simp: algebra_simps)
  thus  $e \in (+) x ' X$  by force
qed

```

```

lemma is_interval_uminusI:
  assumes is_interval X
  shows is_interval (uminus ' X)
  unfolding is_interval_def
proof safe
  fix b d e
  assume  $b \in X$   $d \in X$ 
   $\forall i \in \text{Basis}. (- b) \cdot i \leq e \cdot i \wedge e \cdot i \leq (- d) \cdot i \vee$ 
     $(- d) \cdot i \leq e \cdot i \wedge e \cdot i \leq (- b) \cdot i$ 
  hence  $- e \in X$ 
  by (intro mem_is_intervalI[OF assms  $\langle b \in X \rangle \langle d \in X \rangle$ , of  $- e$ ])
    (auto simp: algebra_simps)
  thus  $e \in \text{uminus ' } X$  by force
qed

```

```

lemma is_interval_uminus[simp]: is_interval (uminus ' x) = is_interval x
  using is_interval_uminusI[of x] is_interval_uminusI[of uminus ' x]
  by (auto simp: image_image)

```

```

lemma is_interval_neg_translationI:
  assumes is_interval X
  shows is_interval ((-) x ' X)
proof -
  have  $(-) x ' X = (+) x ' \text{uminus ' } X$ 
  by (force simp: algebra_simps)
  also have is_interval ...

```



```

  by (metis is_interval_uminusI is_interval_translationI assms)
  finally show ?thesis .
qed

```

```

lemma is_interval_translation[simp]:
  is_interval ((+) x ` X) = is_interval X
  using is_interval_neg_translationI[of (+) x ` X x]
  by (auto intro!: is_interval_translationI simp: image_image)

```

```

lemma is_interval_minus_translation[simp]:
  shows is_interval ((-) x ` X) = is_interval X
proof -
  have (-) x ` X = (+) x ` uminus ` X
    by (force simp: algebra_simps)
  also have is_interval ... = is_interval X
    by simp
  finally show ?thesis .
qed

```

```

lemma is_interval_minus_translation'[simp]:
  shows is_interval ((λx. x - c) ` X) = is_interval X
  using is_interval_translation[of -c X]
  by (metis image_cong uminus_add_conv_diff)

```

```

lemma is_interval_cball_1[intro, simp]: is_interval (cball a b) for a b::real
  by (simp add: cball_eq_atLeastAtMost is_interval_def)

```

```

lemma is_interval_ball_real: is_interval (ball a b) for a b::real
  by (simp add: ball_eq_greaterThanLessThan is_interval_def)

```

4.1.5 Bounded Projections

```

lemma bounded_inner_imp_bdd_above:
  assumes bounded s
  shows bdd_above ((λx. x · a) ` s)
by (simp add: assms bounded_imp_bdd_above bounded_linear_image bounded_linear_inner_left)

```

```

lemma bounded_inner_imp_bdd_below:
  assumes bounded s
  shows bdd_below ((λx. x · a) ` s)
by (simp add: assms bounded_imp_bdd_below bounded_linear_image bounded_linear_inner_left)

```

4.1.6 Structural rules for pointwise continuity

```

lemma continuous_infnorm[continuous_intros]:
  continuous F f ⟹ continuous F (λx. infnorm (f x))
  unfolding continuous_def by (rule tendsto_infnorm)

```

```

lemma continuous_inner[continuous_intros]:
  assumes continuous F f

```

and *continuous* $F\ g$
shows *continuous* $F\ (\lambda x. \text{inner } (f\ x) (g\ x))$
using *assms* **unfolding** *continuous_def* **by** (*rule tendsto_inner*)

4.1.7 Structural rules for setwise continuity

lemma *continuous_on_infnorm*[*continuous_intros*]:
 $\text{continuous\_on } s\ f \implies \text{continuous\_on } s\ (\lambda x. \text{infnorm } (f\ x))$
unfolding *continuous_on* **by** (*fast intro: tendsto_infnorm*)

lemma *continuous_on_inner*[*continuous_intros*]:
fixes $g :: 'a::\text{topological\_space} \Rightarrow 'b::\text{real\_inner}$
assumes *continuous_on* $s\ f$
and *continuous_on* $s\ g$
shows *continuous_on* $s\ (\lambda x. \text{inner } (f\ x) (g\ x))$
using *bounded_bilinear_inner* *assms*
by (*rule bounded_bilinear.continuous_on*)

4.1.8 Openness of halfspaces.

lemma *open_halfspace_lt*: *open* $\{x. \text{inner } a\ x < b\}$
by (*simp add: open_Collect_less continuous_on_inner*)

lemma *open_halfspace_gt*: *open* $\{x. \text{inner } a\ x > b\}$
by (*simp add: open_Collect_less continuous_on_inner*)

lemma *open_halfspace_component_lt*: *open* $\{x::'a::\text{euclidean\_space}. x \cdot i < a\}$
by (*simp add: open_Collect_less continuous_on_inner*)

lemma *open_halfspace_component_gt*: *open* $\{x::'a::\text{euclidean\_space}. x \cdot i > a\}$
by (*simp add: open_Collect_less continuous_on_inner*)

lemma *eucl_less_eq_halfspaces*:
fixes $a :: 'a::\text{euclidean\_space}$
shows $\{x. x <_e a\} = (\bigcap i \in \text{Basis}. \{x. x \cdot i < a \cdot i\})$
 $\{x. a <_e x\} = (\bigcap i \in \text{Basis}. \{x. a \cdot i < x \cdot i\})$
by (*auto simp: eucl_less_def*)

lemma *open_Collect_eucl_less*[*simp, intro*]:
fixes $a :: 'a::\text{euclidean\_space}$
shows *open* $\{x. x <_e a\}$ *open* $\{x. a <_e x\}$
by (*auto simp: eucl_less_eq_halfspaces open_halfspace_component_lt open_halfspace_component_gt*)

4.1.9 Closure and Interior of halfspaces and hyperplanes

lemma *continuous_at_inner*: *continuous* $(\text{at } x) (\text{inner } a)$
unfolding *continuous_at* **by** (*intro tendsto_intros*)

lemma *closed_halfspace_le*: *closed* $\{x. \text{inner } a\ x \leq b\}$
by (*simp add: closed_Collect_le continuous_on_inner*)

```

lemma closed_halfspace_ge: closed {x. inner a x ≥ b}
  by (simp add: closed_Collect_le continuous_on_inner)

lemma closed_hyperplane: closed {x. inner a x = b}
  by (simp add: closed_Collect_eq continuous_on_inner)

lemma closed_halfspace_component_le: closed {x::'a::euclidean_space. x·i ≤ a}
  by (simp add: closed_Collect_le continuous_on_inner)

lemma closed_halfspace_component_ge: closed {x::'a::euclidean_space. x·i ≥ a}
  by (simp add: closed_Collect_le continuous_on_inner)

lemma closed_interval_left:
  fixes b :: 'a::euclidean_space
  shows closed {x::'a. ∀ i∈Basis. x·i ≤ b·i}
  by (simp add: Collect_ball_eq closed_INT closed_Collect_le continuous_on_inner)

lemma closed_interval_right:
  fixes a :: 'a::euclidean_space
  shows closed {x::'a. ∀ i∈Basis. a·i ≤ x·i}
  by (simp add: Collect_ball_eq closed_INT closed_Collect_le continuous_on_inner)

lemma interior_halfspace_le [simp]:
  assumes a ≠ 0
  shows interior {x. a · x ≤ b} = {x. a · x < b}
proof -
  have *: a · x < b if x: x ∈ S and S: S ⊆ {x. a · x ≤ b} and open S for S x
  proof -
    obtain e where e>0 and e: cball x e ⊆ S
    using ⟨open S⟩ open_contains_cball x by blast
    then have x + (e / norm a) *R a ∈ cball x e
    by (simp add: dist_norm)
    then have x + (e / norm a) *R a ∈ S
    using e by blast
    then have x + (e / norm a) *R a ∈ {x. a · x ≤ b}
    using S by blast
    moreover have e * (a · a) / norm a > 0
    by (simp add: ⟨0 < e⟩ assms)
    ultimately show ?thesis
    by (simp add: algebra_simps)
  qed
  show ?thesis
  by (rule interior_unique) (auto simp: open_halfspace_lt *)
qed

lemma interior_halfspace_ge [simp]:
  a ≠ 0 ⇒ interior {x. a · x ≥ b} = {x. a · x > b}
using interior_halfspace_le [of -a -b] by simp

```

```

lemma closure_halfspace_lt [simp]:
  assumes  $a \neq 0$ 
  shows  $\text{closure } \{x. a \cdot x < b\} = \{x. a \cdot x \leq b\}$ 
proof -
  have [simp]:  $-\{x. a \cdot x < b\} = \{x. a \cdot x \geq b\}$ 
  by (force simp:)
  then show ?thesis
  using interior_halfspace_ge [of a b] assms
  by (force simp: closure_interior)
qed

```

```

lemma closure_halfspace_gt [simp]:
   $a \neq 0 \implies \text{closure } \{x. a \cdot x > b\} = \{x. a \cdot x \geq b\}$ 
using closure_halfspace_lt [of -a -b] by simp

```

```

lemma interior_hyperplane [simp]:
  assumes  $a \neq 0$ 
  shows  $\text{interior } \{x. a \cdot x = b\} = \{\}$ 
proof -
  have [simp]:  $\{x. a \cdot x = b\} = \{x. a \cdot x \leq b\} \cap \{x. a \cdot x \geq b\}$ 
  by (force simp:)
  then show ?thesis
  by (auto simp: assms)
qed

```

```

lemma frontier_halfspace_le:
  assumes  $a \neq 0 \vee b \neq 0$ 
  shows  $\text{frontier } \{x. a \cdot x \leq b\} = \{x. a \cdot x = b\}$ 
proof (cases  $a = 0$ )
  case True with assms show ?thesis by simp
next
  case False then show ?thesis
  by (force simp: frontier_def closed_halfspace_le)
qed

```

```

lemma frontier_halfspace_ge:
  assumes  $a \neq 0 \vee b \neq 0$ 
  shows  $\text{frontier } \{x. a \cdot x \geq b\} = \{x. a \cdot x = b\}$ 
proof (cases  $a = 0$ )
  case True with assms show ?thesis by simp
next
  case False then show ?thesis
  by (force simp: frontier_def closed_halfspace_ge)
qed

```

```

lemma frontier_halfspace_lt:
  assumes  $a \neq 0 \vee b \neq 0$ 
  shows  $\text{frontier } \{x. a \cdot x < b\} = \{x. a \cdot x = b\}$ 

```

```

proof (cases a = 0)
  case True with assms show ?thesis by simp
next
  case False then show ?thesis
    by (force simp: frontier_def interior_open open_halfspace_lt)
qed

```

```

lemma frontier_halfspace_gt:
  assumes  $a \neq 0 \vee b \neq 0$ 
  shows frontier  $\{x. a \cdot x > b\} = \{x. a \cdot x = b\}$ 
proof (cases a = 0)
  case True with assms show ?thesis by simp
next
  case False then show ?thesis
    by (force simp: frontier_def interior_open open_halfspace_gt)
qed

```

4.1.10 Some more convenient intermediate-value theorem formulations

```

lemma connected_ivt_hyperplane:
  assumes connected S and  $xy: x \in S \ y \in S$  and  $b: \text{inner } a \ x \leq b \ b \leq \text{inner } a \ y$ 
  shows  $\exists z \in S. \text{inner } a \ z = b$ 
proof (rule ccontr)
  assume as:  $\neg (\exists z \in S. \text{inner } a \ z = b)$ 
  let ?A =  $\{x. \text{inner } a \ x < b\}$ 
  let ?B =  $\{x. \text{inner } a \ x > b\}$ 
  have open ?A open ?B
    using open_halfspace_lt and open_halfspace_gt by auto
  moreover have  $?A \cap ?B = \{\}$  by auto
  moreover have  $S \subseteq ?A \cup ?B$  using as by auto
  ultimately show False
    using  $\langle \text{connected } S \rangle [\text{unfolded connected_def not_ex},$ 
       $\text{THEN spec[where } x=?A], \text{ THEN spec[where } x=?B]]$ 
    using xy b by auto
qed

```

```

lemma connected_ivt_component:
  fixes  $x::'a::\text{euclidean\_space}$ 
  shows  $\text{connected } S \implies x \in S \implies y \in S \implies x \cdot k \leq a \implies a \leq y \cdot k \implies (\exists z \in S. z \cdot k = a)$ 
  using connected_ivt_hyperplane[of S x y k::'a a]
  by (auto simp: inner_commute)

```

4.1.11 Limit Component Bounds

```

lemma Lim_component_le:
  fixes  $f :: 'a \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes  $(f \longrightarrow l)$  net

```

```

    and  $\neg$  (trivial_limit net)
    and eventually ( $\lambda x. f(x) \cdot i \leq b$ ) net
  shows  $l \cdot i \leq b$ 
  by (rule tendsto_le[OF assms(2) tendsto_const tendsto_inner[OF assms(1) tendsto_const] assms(3)])

```

```

lemma Lim_component_ge:
  fixes  $f :: 'a \Rightarrow 'b::euclidean\_space$ 
  assumes ( $f \longrightarrow l$ ) net
    and  $\neg$  (trivial_limit net)
    and eventually ( $\lambda x. b \leq (f\ x) \cdot i$ ) net
  shows  $b \leq l \cdot i$ 
  by (rule tendsto_le[OF assms(2) tendsto_inner[OF assms(1) tendsto_const] tendsto_const assms(3)])

```

```

lemma Lim_component_eq:
  fixes  $f :: 'a \Rightarrow 'b::euclidean\_space$ 
  assumes net: ( $f \longrightarrow l$ ) net  $\neg$  trivial_limit net
    and ev:eventually ( $\lambda x. f(x) \cdot i = b$ ) net
  shows  $l \cdot i = b$ 
  using ev[unfolded order_eq_iff eventually_conj_iff]
  using Lim_component_ge[OF net, of b i]
  using Lim_component_le[OF net, of i b]
  by auto

```

```

lemma open_box[intro]: open (box a b)
proof -
  have open ( $\bigcap i \in \text{Basis}. ((\cdot) \ i) - \{a \cdot i <..< b \cdot i\}$ )
    by (auto intro!: continuous_open_vimage continuous_inner continuous_ident continuous_const)
  also have ( $\bigcap i \in \text{Basis}. ((\cdot) \ i) - \{a \cdot i <..< b \cdot i\}$ ) = box a b
    by (auto simp: box_def inner_commute)
  finally show ?thesis .
qed

```

```

lemma closed_cbox[intro]:
  fixes  $a \ b :: 'a::euclidean\_space$ 
  shows closed (cbox a b)
proof -
  have closed ( $\bigcap i \in \text{Basis}. (\lambda x. x \cdot i) - \{a \cdot i .. b \cdot i\}$ )
    by (intro closed_INT ballI continuous_closed_vimage allI
      linear_continuous_at closed_real_atLeastAtMost finite_Basis bounded_linear_inner_left)
  also have ( $\bigcap i \in \text{Basis}. (\lambda x. x \cdot i) - \{a \cdot i .. b \cdot i\}$ ) = cbox a b
    by (auto simp: cbox_def)
  finally show closed (cbox a b) .
qed

```

```

lemma interior_cbox [simp]:
  fixes  $a \ b :: 'a::euclidean\_space$ 

```

```

shows interior (cbox a b) = box a b (is ?L = ?R)
proof(rule subset_antisym)
show ?R  $\subseteq$  ?L
  using box_subset_cbox open_box
  by (rule interior_maximal)
{
  fix x
  assume x  $\in$  interior (cbox a b)
  then obtain s where s: open s x  $\in$  s s  $\subseteq$  cbox a b ..
  then obtain e where e>0 and e: $\forall x'. \text{dist } x' x < e \longrightarrow x' \in \text{cbox } a \ b$ 
  unfolding open_dist and subset_eq by auto
  {
    fix i :: 'a
    assume i: i  $\in$  Basis
    have dist (x - (e / 2) *R i) x < e
      and dist (x + (e / 2) *R i) x < e
      unfolding dist_norm
      apply auto
      unfolding norm_minus_cancel
      using norm_Basis[OF i] <e>0>
      apply auto
      done
    then have a  $\cdot$  i  $\leq$  (x - (e / 2) *R i)  $\cdot$  i and (x + (e / 2) *R i)  $\cdot$  i  $\leq$  b  $\cdot$  i
      using e[THEN spec[where x=x - (e/2) *R i]]
      and e[THEN spec[where x=x + (e/2) *R i]]
      unfolding mem_box
      using i
      by blast+
    then have a  $\cdot$  i < x  $\cdot$  i and x  $\cdot$  i < b  $\cdot$  i
      using <e>0> i
      by (auto simp: inner_diff_left inner_Basis inner_add_left)
  }
  then have x  $\in$  box a b
    unfolding mem_box by auto
}
then show ?L  $\subseteq$  ?R ..
qed

```

```

lemma bounded_cbox [simp]:
  fixes a :: 'a::euclidean_space
  shows bounded (cbox a b)
proof -
  let ?b =  $\sum i \in \text{Basis}. |a \cdot i| + |b \cdot i|$ 
  {
    fix x :: 'a
    assume  $\bigwedge i. i \in \text{Basis} \implies a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i$ 
    then have ( $\sum i \in \text{Basis}. |x \cdot i|$ )  $\leq$  ?b
      by (force simp: intro!: sum_mono)
    then have norm x  $\leq$  ?b

```

```

    using norm_le_l1[of x] by auto
  }
  then show ?thesis
    unfolding cbox_def bounded_iff by force
qed

```

```

lemma bounded_box [simp]:
  fixes a :: 'a::euclidean_space
  shows bounded (box a b)
  using bounded_cbox[of a b] box_subset_cbox[of a b] bounded_subset[of cbox a b
box a b]
  by simp

```

```

lemma not_interval_UNIV [simp]:
  fixes a :: 'a::euclidean_space
  shows cbox a b  $\neq$  UNIV box a b  $\neq$  UNIV
  using bounded_box[of a b] bounded_cbox[of a b] by force+

```

```

lemma not_interval_UNIV2 [simp]:
  fixes a :: 'a::euclidean_space
  shows UNIV  $\neq$  cbox a b UNIV  $\neq$  box a b
  using bounded_box[of a b] bounded_cbox[of a b] by force+

```

```

lemma box_midpoint:
  fixes a :: 'a::euclidean_space
  assumes box a b  $\neq$  {}
  shows  $((1/2) *_R (a + b)) \in \text{box } a \ b$ 
proof -
  have  $a \cdot i < ((1 / 2) *_R (a + b)) \cdot i \wedge ((1 / 2) *_R (a + b)) \cdot i < b \cdot i$  if  $i \in$ 
  Basis for  $i$ 
  using assms that by (auto simp: inner_add_left box_ne_empty)
  then show ?thesis unfolding mem_box by auto
qed

```

```

lemma open_cbox_convex:
  fixes x :: 'a::euclidean_space
  assumes x:  $x \in \text{box } a \ b$ 
  and y:  $y \in \text{cbox } a \ b$ 
  and e:  $0 < e \leq 1$ 
  shows  $(e *_R x + (1 - e) *_R y) \in \text{box } a \ b$ 
proof -
  {
    fix i :: 'a
    assume i:  $i \in \text{Basis}$ 
    have  $a \cdot i = e * (a \cdot i) + (1 - e) * (a \cdot i)$ 
      unfolding left_diff_distrib by simp
    also have  $\dots < e * (x \cdot i) + (1 - e) * (y \cdot i)$ 
    proof (rule add_less_le_mono)
      show  $e * (a \cdot i) < e * (x \cdot i)$ 

```



```

    using ‹0 < e› i mem_box(1) x by auto
  show (1 - e) * (a • i) ≤ (1 - e) * (y • i)
    by (meson diff_ge_0_iff_ge ‹e ≤ 1› i mem_box(2) mult_left_mono y)
qed
finally have a • i < (e *R x + (1 - e) *R y) • i
  unfolding inner_simps by auto
moreover
{
  have b • i = e * (b • i) + (1 - e) * (b • i)
    unfolding left_diff_distrib by simp
  also have ... > e * (x • i) + (1 - e) * (y • i)
    proof (rule add_less_le_mono)
      show e * (x • i) < e * (b • i)
        using ‹0 < e› i mem_box(1) x by auto
      show (1 - e) * (y • i) ≤ (1 - e) * (b • i)
        by (meson diff_ge_0_iff_ge ‹e ≤ 1› i mem_box(2) mult_left_mono y)
    qed
  finally have (e *R x + (1 - e) *R y) • i < b • i
    unfolding inner_simps by auto
}
ultimately have a • i < (e *R x + (1 - e) *R y) • i ∧ (e *R x + (1 - e) *R
y) • i < b • i
  by auto
}
then show ?thesis
  unfolding mem_box by auto
qed

```

```

lemma closure_cbox [simp]: closure (cbox a b) = cbox a b
  by (simp add: closed_cbox)

```

```

lemma closure_box [simp]:
  fixes a :: 'a::euclidean_space
  assumes box a b ≠ {}
  shows closure (box a b) = cbox a b
proof -
  have ab: a < e b
    using assms by (simp add: eucl_less_def box_ne_empty)
  let ?c = (1 / 2) *R (a + b)
  {
    fix x
    assume as: x ∈ cbox a b
    define f where [abs_def]: f n = x + (inverse (real n + 1)) *R (?c - x) for n
    {
      fix n
      assume fn: f n < e b ⟶ a < e f n ⟶ f n = x and xc: x ≠ ?c
      have *: 0 < inverse (real n + 1) inverse (real n + 1) ≤ 1
        unfolding inverse_le_1_iff by auto
      have (inverse (real n + 1)) *R ((1 / 2) *R (a + b)) + (1 - inverse (real n

```

```

+ 1)) *R x =
  x + (inverse (real n + 1)) *R (((1 / 2) *R (a + b)) - x)
  by (auto simp: algebra_simps)
then have f n <e b and a <e f n
  using open_cbox_convex[OF box_midpoint[OF assms] as *]
  unfolding f_def by (auto simp: box_def eucl_less_def)
then have False
  using fn unfolding f_def using xc by auto
}
moreover
{
  assume ¬ (f ⟶ x) sequentially
  {
    fix e :: real
    assume e > 0
    then obtain N :: nat where N: inverse (real (N + 1)) < e
      using reals_Archimedean by auto
    have inverse (real n + 1) < e if N ≤ n for n
      by (auto intro!: that le_less_trans [OF _ N])
    then have ∃ N::nat. ∀ n≥N. inverse (real n + 1) < e by auto
  }
  then have ((λn. inverse (real n + 1)) ⟶ 0) sequentially
    unfolding lim_sequentially by (auto simp: dist_norm)
  then have (f ⟶ x) sequentially
    unfolding f_def
    using tendsto_add[OF tendsto_const, of λn::nat. (inverse (real n + 1)) *R
      ((1 / 2) *R (a + b) - x) 0 sequentially x]
    using tendsto_scaleR [OF _ tendsto_const, of λn::nat. inverse (real n +
      1) 0 sequentially ((1 / 2) *R (a + b) - x)]
    by auto
  }
  ultimately have x ∈ closure (box a b)
    using as box_midpoint[OF assms]
    unfolding closure_def islimpt_sequential
    by (cases x=?c) (auto simp: in_box_eucl_less)
}
then show ?thesis
  using closure_minimal[OF box_subset_cbox, of a b] by blast
qed

```

lemma *bounded_subset_box_symmetric*:

fixes $S :: ('a::euclidean\_space) \text{ set}$
assumes *bounded S*
obtains a **where** $S \subseteq \text{box } (-a) \ a$

proof –

obtain b **where** $b > 0$ **and** $b: \forall x \in S. \text{norm } x \leq b$
using *assms[unfolded bounded_pos]* **by** *auto*
define $a :: 'a$ **where** $a = (\sum i \in \text{Basis}. (b + 1) *_{\mathbb{R}} i)$
have $(-a) \cdot i < x \cdot i$ **and** $x \cdot i < a \cdot i$ **if** $x \in S$ **and** $i: i \in \text{Basis}$ **for** $x \ i$

```

    using  $b$  Basis_le_norm[OF  $i$ , of  $x$ ] that by (auto simp:  $a\_def$ )
  then have  $S \subseteq \text{box } (-a) a$ 
    by (auto simp: simp add:  $\text{box\_def}$ )
  then show ?thesis ..
qed

```

```

lemma bounded_subset_cbox_symmetric:
  fixes  $S :: ('a::euclidean\_space) \text{ set}$ 
  assumes  $\text{bounded } S$ 
  obtains  $a$  where  $S \subseteq \text{cbox } (-a) a$ 
proof -
  obtain  $a$  where  $S \subseteq \text{box } (-a) a$ 
    using bounded_subset_box_symmetric[OF  $\text{assms}$ ] by auto
  then show ?thesis
    by (meson  $\text{box\_subset\_cbox dual\_order.trans that}$ )
qed

```

```

lemma frontier_cbox:
  fixes  $a b :: 'a::euclidean\_space$ 
  shows  $\text{frontier } (\text{cbox } a b) = \text{cbox } a b - \text{box } a b$ 
  unfolding frontier_def unfolding interior_cbox and closure_closed[OF  $\text{closed\_cbox}$ ]
  ..

```

```

lemma frontier_box:
  fixes  $a b :: 'a::euclidean\_space$ 
  shows  $\text{frontier } (\text{box } a b) = (\text{if } \text{box } a b = \{\} \text{ then } \{\} \text{ else } \text{cbox } a b - \text{box } a b)$ 
proof (cases  $\text{box } a b = \{\}$ )
  case True
  then show ?thesis
    using frontier_empty by auto
next
  case False
  then show ?thesis
    unfolding frontier_def and closure_box[OF  $\text{False}$ ] and interior_open[OF  $\text{open\_box}$ ]
    by auto
qed

```

```

lemma Int_interval_mixed_eq_empty:
  fixes  $a :: 'a::euclidean\_space$ 
  assumes  $\text{box } c d \neq \{\}$ 
  shows  $\text{box } a b \cap \text{cbox } c d = \{\} \longleftrightarrow \text{box } a b \cap \text{box } c d = \{\}$ 
  unfolding closure_box[OF  $\text{assms}$ , symmetric]
  unfolding open_Int_closure_eq_empty[OF  $\text{open\_box}$ ] ..

```

4.1.12 Class Instances

```

lemma compact_lemma:
  fixes  $f :: \text{nat} \Rightarrow 'a::euclidean\_space$ 

```

```

assumes bounded (range f)
shows  $\forall d \subseteq \text{Basis}. \exists l :: 'a. \exists r.$ 
   $\text{strict\_mono } r \wedge (\forall e > 0. \text{eventually } (\lambda n. \forall i \in d. \text{dist } (f \ (r \ n)) \cdot i) \ (l \cdot i) < e)$ 
  sequentially)
by (rule compact_lemma_general[where unproj= $\lambda e. \sum i \in \text{Basis}. e \ i \ *_R \ i$ ])
  (auto intro!: assms bounded_linear_inner_left bounded_linear_image
    simp: euclidean_representation)

instance euclidean_space  $\subseteq$  heine_borel
proof
  fix f :: nat  $\Rightarrow$  'a
  assume f: bounded (range f)
  then obtain l::'a and r where r: strict_mono r
  and l:  $\forall e > 0. \text{eventually } (\lambda n. \forall i \in \text{Basis}. \text{dist } (f \ (r \ n)) \cdot i) \ (l \cdot i) < e)$  sequentially
  using compact_lemma [OF f] by blast
  {
    fix e::real
    assume e > 0
    hence e / real_of_nat DIM('a) > 0 by (simp)
    with l have eventually ( $\lambda n. \forall i \in \text{Basis}. \text{dist } (f \ (r \ n)) \cdot i) \ (l \cdot i) < e / (\text{real\_of\_nat } \text{DIM}('a))$ ) sequentially
    by simp
    moreover
    {
      fix n
      assume n:  $\forall i \in \text{Basis}. \text{dist } (f \ (r \ n)) \cdot i) \ (l \cdot i) < e / (\text{real\_of\_nat } \text{DIM}('a))$ 
      have  $\text{dist } (f \ (r \ n)) \ l \leq (\sum i \in \text{Basis}. \text{dist } (f \ (r \ n)) \cdot i) \ (l \cdot i)$ 
      apply (subst euclidean_dist_l2)
      using zero_le_dist
      apply (rule L2_set_le_sum)
      done
      also have  $\dots < (\sum i \in (\text{Basis}::'a \text{ set}). e / (\text{real\_of\_nat } \text{DIM}('a)))$ 
      apply (rule sum_strict_mono)
      using n
      apply auto
      done
      finally have  $\text{dist } (f \ (r \ n)) \ l < e$ 
      by auto
    }
    ultimately have eventually ( $\lambda n. \text{dist } (f \ (r \ n)) \ l < e$ ) sequentially
    by (rule eventually_mono)
  }
  then have *:  $((f \circ r) \longrightarrow l)$  sequentially
  unfolding o_def tendsto_iff by simp
  with r show  $\exists l \ r. \text{strict\_mono } r \wedge ((f \circ r) \longrightarrow l)$  sequentially
  by auto
qed

instance euclidean_space  $\subseteq$  banach ..

```

```

instance euclidean_space  $\subseteq$  second_countable_topology
proof
  define a where  $a\ f = (\sum i \in \text{Basis}. \text{fst}\ (f\ i) *_{\mathbb{R}} i)$  for  $f :: 'a \Rightarrow \text{real} \times \text{real}$ 
  then have  $a: \bigwedge f. (\sum i \in \text{Basis}. \text{fst}\ (f\ i) *_{\mathbb{R}} i) = a\ f$ 
    by simp
  define b where  $b\ f = (\sum i \in \text{Basis}. \text{snd}\ (f\ i) *_{\mathbb{R}} i)$  for  $f :: 'a \Rightarrow \text{real} \times \text{real}$ 
  then have  $b: \bigwedge f. (\sum i \in \text{Basis}. \text{snd}\ (f\ i) *_{\mathbb{R}} i) = b\ f$ 
    by simp
  define B where  $B = (\lambda f. \text{box}\ (a\ f)\ (b\ f))\ `(\text{Basis} \rightarrow_E (\mathbb{Q} \times \mathbb{Q}))$ 

  have Ball B open by (simp add: B_def open_box)
  moreover have  $(\forall A. \text{open}\ A \longrightarrow (\exists B' \subseteq B. \bigcup B' = A))$ 
  proof safe
    fix A :: 'a set
    assume open A
    show  $\exists B' \subseteq B. \bigcup B' = A$ 
      apply (rule exI[of _  $\{b \in B. b \subseteq A\}$ ])
      apply (subst (3) open_UNION_box[OF  $\langle \text{open}\ A \rangle$ ])
      apply (auto simp: a b B_def)
      done
  qed
  ultimately
  have topological_basis B
    unfolding topological_basis_def by blast
  moreover
  have countable B
    unfolding B_def
    by (intro countable_image countable_PiE finite_Basis countable_SIGMA countable_rat)
  ultimately show  $\exists B :: 'a \text{ set set}. \text{countable}\ B \wedge \text{open} = \text{generate\_topology}\ B$ 
    by (blast intro: topological_basis_imp_subbasis)
qed

instance euclidean_space  $\subseteq$  polish_space ..

```

4.1.13 Compact Boxes

```

lemma compact_cbox [simp]:
  fixes a :: 'a::euclidean_space
  shows compact (cbox a b)
  using bounded_closed_imp_seq_compact[of cbox a b] using bounded_cbox[of a b]
  by (auto simp: compact_eq_seq_compact_metric)

proposition is_interval_compact:
   $\text{is\_interval}\ S \wedge \text{compact}\ S \longleftrightarrow (\exists a\ b. S = \text{cbox}\ a\ b) \quad (\text{is ?lhs} = \text{?rhs})$ 
proof (cases  $S = \{\}$ )
  case True

```

```

with empty_as_interval show ?thesis by auto
next
case False
show ?thesis
proof
  assume L: ?lhs
  then have is_interval S compact S by auto
  define a where  $a \equiv \sum_{i \in \text{Basis}} (\text{INF } x \in S. x \cdot i) *_R i$ 
  define b where  $b \equiv \sum_{i \in \text{Basis}} (\text{SUP } x \in S. x \cdot i) *_R i$ 
  have 1:  $\bigwedge x i. \llbracket x \in S; i \in \text{Basis} \rrbracket \implies (\text{INF } x \in S. x \cdot i) \leq x \cdot i$ 
  by (simp add: cInf_lower bounded_inner_imp_bdd_below compact_imp_bounded
L)
  have 2:  $\bigwedge x i. \llbracket x \in S; i \in \text{Basis} \rrbracket \implies x \cdot i \leq (\text{SUP } x \in S. x \cdot i)$ 
  by (simp add: cSup_upper bounded_inner_imp_bdd_above compact_imp_bounded
L)
  have 3:  $x \in S$  if inf:  $\bigwedge i. i \in \text{Basis} \implies (\text{INF } x \in S. x \cdot i) \leq x \cdot i$ 
    and sup:  $\bigwedge i. i \in \text{Basis} \implies x \cdot i \leq (\text{SUP } x \in S. x \cdot i)$  for x
  proof (rule mem_box_componentwiseI [OF ‹is_interval S›])
    fix i::'a
    assume i: i ∈ Basis
    have cont: continuous_on S (λx. x · i)
    by (intro continuous_intros)
    obtain a where a ∈ S and a:  $\bigwedge y. y \in S \implies a \cdot i \leq y \cdot i$ 
    using continuous_attains_inf [OF ‹compact S› False cont] by blast
    obtain b where b ∈ S and b:  $\bigwedge y. y \in S \implies y \cdot i \leq b \cdot i$ 
    using continuous_attains_sup [OF ‹compact S› False cont] by blast
    have a · i ≤ (INF x ∈ S. x · i)
    by (simp add: False a cINF_greatest)
    also have ... ≤ x · i
    by (simp add: i inf)
    finally have ai: a · i ≤ x · i .
    have x · i ≤ (SUP x ∈ S. x · i)
    by (simp add: i sup)
    also have (SUP x ∈ S. x · i) ≤ b · i
    by (simp add: False b cSUP_least)
    finally have bi: x · i ≤ b · i .
    show x · i ∈ (λx. x · i) ' S
    apply (rule_tac x =  $\sum_{j \in \text{Basis}} ( \text{if } j = i \text{ then } x \cdot i \text{ else } a \cdot j ) *_R j$  in
image_eqI)
    apply (simp add: i)
    apply (rule mem_is_intervalI [OF ‹is_interval S› ‹a ∈ S› ‹b ∈ S›])
    using i ai bi apply force
  done
qed
have S = cbox a b
by (auto simp: a_def b_def mem_box intro: 1 2 3)
then show ?rhs
by blast
next

```

```

    assume R: ?rhs
  then show ?lhs
    using compact_cbox is_interval_cbox by blast
qed
qed

```

4.1.14 Componentwise limits and continuity

But is the premise really necessary? Need to generalise $\text{dist } ?x \ ?y = L2\_set$
 $(\lambda i. \text{dist } (?x \cdot i) \ (?y \cdot i)) \text{ Basis}$

lemma *Euclidean_dist_upper*: $i \in \text{Basis} \implies \text{dist } (x \cdot i) \ (y \cdot i) \leq \text{dist } x \ y$
 by (metis (no_types) member_le_L2_set euclidean_dist_l2 finite_Basis)

But is the premise $i \in \text{Basis}$ really necessary?

```

lemma open_preimage_inner:
  assumes open S i ∈ Basis
  shows open {x. x · i ∈ S}
proof (rule openI, simp)
  fix x
  assume x: x · i ∈ S
  with assms obtain e where 0 < e and e: ball (x · i) e ⊆ S
    by (auto simp: open_contains_ball_eq)
  have ∃ e>0. ball (y · i) e ⊆ S if dxy: dist x y < e / 2 for y
  proof (intro exI conjI)
    have dist (x · i) (y · i) < e / 2
      by (meson ⟨i ∈ Basis⟩ dual_order.trans Euclidean_dist_upper not_le that)
    then have dist (x · i) z < e if dist (y · i) z < e / 2 for z
      by (metis dist_commute dist_triangle_half_l that)
    then have ball (y · i) (e / 2) ⊆ ball (x · i) e
      using mem_ball by blast
    with e show ball (y · i) (e / 2) ⊆ S
      by (metis order_trans)
  qed (simp add: ⟨0 < e⟩)
  then show ∃ e>0. ball x e ⊆ {s. s · i ∈ S}
    by (metis (no_types, lifting) ⟨0 < e⟩ ⟨open S⟩ half_gt_zero_iff mem_Collect_eq
    mem_ball open_contains_ball_eq subsetI)
qed

```

proposition *tendsto_componentwise_iff*:
 fixes $f :: \_ \Rightarrow 'b::\text{euclidean\_space}$
 shows $(f \longrightarrow l) \ F \longleftrightarrow (\forall i \in \text{Basis}. ((\lambda x. (f \ x \cdot i)) \longrightarrow (l \cdot i)) \ F)$
 (is ?lhs = ?rhs)

```

proof
  assume ?lhs
  then show ?rhs
    unfolding tendsto_def
    apply clarify
    apply (drule_tac x={s. s · i ∈ S} in spec)

```

```

    apply (auto simp: open_preimage_inner)
  done
next
  assume R: ?rhs
  then have  $\bigwedge e. e > 0 \implies \forall i \in \text{Basis}. \forall_F x \text{ in } F. \text{dist } (f x \cdot i) (l \cdot i) < e$ 
    unfolding tendsto_iff by blast
  then have R':  $\bigwedge e. e > 0 \implies \forall_F x \text{ in } F. \forall i \in \text{Basis}. \text{dist } (f x \cdot i) (l \cdot i) < e$ 
    by (simp add: eventually_ball_finite_distrib [symmetric])
  show ?lhs
  unfolding tendsto_iff
  proof clarify
    fix e::real
    assume 0 < e
    have *:  $L2\_set (\lambda i. \text{dist } (f x \cdot i) (l \cdot i)) \text{Basis} < e$ 
      if  $\forall i \in \text{Basis}. \text{dist } (f x \cdot i) (l \cdot i) < e / \text{real DIM('b)}$  for x
    proof -
      have  $L2\_set (\lambda i. \text{dist } (f x \cdot i) (l \cdot i)) \text{Basis} \leq \text{sum } (\lambda i. \text{dist } (f x \cdot i) (l \cdot i)) \text{Basis}$ 
    Basis
      by (simp add: L2_set_le_sum)
    also have  $\dots < \text{DIM('b)} * (e / \text{real DIM('b)})$ 
      apply (rule sum_bounded_above_strict)
      using that by auto
    also have  $\dots = e$ 
      by (simp add: field_simps)
    finally show  $L2\_set (\lambda i. \text{dist } (f x \cdot i) (l \cdot i)) \text{Basis} < e$  .
  qed
  have  $\forall_F x \text{ in } F. \forall i \in \text{Basis}. \text{dist } (f x \cdot i) (l \cdot i) < e / \text{DIM('b)}$ 
    apply (rule R')
    using 0 < e by simp
  then show  $\forall_F x \text{ in } F. \text{dist } (f x) l < e$ 
    apply (rule eventually_mono)
    apply (subst euclidean_dist_l2)
    using * by blast
  qed
qed

```

corollary *continuous_componentwise:*

```

  continuous F f  $\longleftrightarrow (\forall i \in \text{Basis}. \text{continuous } F (\lambda x. (f x \cdot i)))$ 
by (simp add: continuous_def tendsto_componentwise_iff [symmetric])

```

corollary *continuous_on_componentwise:*

```

  fixes S :: 'a :: t2_space set
  shows continuous_on S f  $\longleftrightarrow (\forall i \in \text{Basis}. \text{continuous\_on } S (\lambda x. (f x \cdot i)))$ 
  apply (simp add: continuous_on_eq_continuous_within)
  using continuous_componentwise by blast

```

lemma *linear_componentwise_iff:*

```

  (linear f')  $\longleftrightarrow (\forall i \in \text{Basis}. \text{linear } (\lambda x. f' x \cdot i))$ 

```



```

apply (auto simp: linear_iff inner_left_distrib)
apply (metis inner_left_distrib euclidean_eq_iff)
by (metis euclidean_eqI inner_scaleR_left)

lemma bounded_linear_componentwise_iff:
  (bounded_linear f')  $\longleftrightarrow$  ( $\forall i \in \text{Basis}. \text{bounded\_linear } (\lambda x. f' x \cdot i)$ )
  (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
    by (simp add: bounded_linear_inner_left_comp)
next
  assume ?rhs
  then have ( $\forall i \in \text{Basis}. \exists K. \forall x. |f' x \cdot i| \leq \text{norm } x * K$ ) linear f'
    by (auto simp: bounded_linear_def bounded_linear_axioms_def linear_componentwise_iff
  [symmetric] ball_conj_distrib)
  then obtain F where F:  $\bigwedge i x. i \in \text{Basis} \implies |f' x \cdot i| \leq \text{norm } x * F i$ 
    by metis
  have  $\text{norm } (f' x) \leq \text{norm } x * \text{sum } F \text{ Basis}$  for x
    proof -
      have  $\text{norm } (f' x) \leq (\sum i \in \text{Basis}. |f' x \cdot i|)$ 
        by (rule norm_le_l1)
      also have  $\dots \leq (\sum i \in \text{Basis}. \text{norm } x * F i)$ 
        by (metis F sum_mono)
      also have  $\dots = \text{norm } x * \text{sum } F \text{ Basis}$ 
        by (simp add: sum_distrib_left)
      finally show ?thesis .
    qed
  then show ?lhs
    by (force simp: bounded_linear_def bounded_linear_axioms_def linear_f')
qed

```

4.1.15 Continuous Extension

definition clamp :: 'a::euclidean_space $\Rightarrow$ 'a $\Rightarrow$ 'a $\Rightarrow$ 'a **where**
 clamp a b x = (if ($\forall i \in \text{Basis}. a \cdot i \leq b \cdot i$)
 then ($\sum i \in \text{Basis}. (\text{if } x \cdot i < a \cdot i \text{ then } a \cdot i \text{ else if } x \cdot i \leq b \cdot i \text{ then } x \cdot i \text{ else } b \cdot i) * i$)
 else a)

```

lemma clamp_in_interval[simp]:
  assumes  $\bigwedge i. i \in \text{Basis} \implies a \cdot i \leq b \cdot i$ 
  shows clamp a b x  $\in$  cbox a b
  unfolding clamp_def
  using box_ne_empty(1)[of a b] assms by (auto simp: cbox_def)

```

```

lemma clamp_cancel_cbox[simp]:
  fixes x a b :: 'a::euclidean_space
  assumes x: x  $\in$  cbox a b
  shows clamp a b x = x
  using assms

```

by (auto simp: clamp_def mem_box intro!: euclidean_eqI[where 'a='a])

lemma clamp_empty_interval:

assumes $i \in \text{Basis } a \cdot i > b \cdot i$

shows $\text{clamp } a \ b = (\lambda \_. a)$

using assms

by (force simp: clamp_def[abs_def] split: if_splits intro!: ext)

lemma dist_clamps_le_dist_args:

fixes $x :: 'a :: \text{euclidean\_space}$

shows $\text{dist } (\text{clamp } a \ b \ y) (\text{clamp } a \ b \ x) \leq \text{dist } y \ x$

proof cases

assume le: $(\forall i \in \text{Basis}. a \cdot i \leq b \cdot i)$

then have $(\sum i \in \text{Basis}. (\text{dist } (\text{clamp } a \ b \ y \cdot i) (\text{clamp } a \ b \ x \cdot i))^2) \leq$
 $(\sum i \in \text{Basis}. (\text{dist } (y \cdot i) (x \cdot i))^2)$

by (auto intro!: sum_mono simp: clamp_def dist_real_def abs_le_square_iff[symmetric])

then show ?thesis

by (auto intro: real_sqrt_le_mono

simp: euclidean_dist_l2[where y=x] euclidean_dist_l2[where y=clamp a b

x] L2_set_def)

qed (auto simp: clamp_def)

lemma clamp_continuous_at:

fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{metric\_space}$

and $x :: 'a$

assumes $f\_cont: \text{continuous\_on } (cbox \ a \ b) \ f$

shows $\text{continuous } (at \ x) (\lambda x. f (\text{clamp } a \ b \ x))$

proof cases

assume le: $(\forall i \in \text{Basis}. a \cdot i \leq b \cdot i)$

show ?thesis

unfolding continuous_at_eps_delta

proof safe

fix $x :: 'a$

fix $e :: \text{real}$

assume $e > 0$

moreover have $\text{clamp } a \ b \ x \in cbox \ a \ b$

by (simp add: le)

moreover note $f\_cont[\text{simplified continuous\_on\_iff}]$

ultimately

obtain d where $d: 0 < d$

$\bigwedge x'. x' \in cbox \ a \ b \implies \text{dist } x' (\text{clamp } a \ b \ x) < d \implies \text{dist } (f \ x') (f (\text{clamp } a \ b \ x)) < e$

by force

show $\exists d > 0. \forall x'. \text{dist } x' \ x < d \longrightarrow$

$\text{dist } (f (\text{clamp } a \ b \ x')) (f (\text{clamp } a \ b \ x)) < e$

using le

by (auto intro!: d clamp_in_interval dist_clamps_le_dist_args[THEN le_less_trans])

qed

qed (auto simp: clamp_empty_interval)

```

lemma clamp_continuous_on:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::metric\_space$ 
  assumes  $f\_cont: continuous\_on (cbox\ a\ b)\ f$ 
  shows  $continuous\_on\ S\ (\lambda x. f\ (clamp\ a\ b\ x))$ 
  using assms
  by (auto intro: continuous_at_imp_continuous_on clamp_continuous_at)

lemma clamp_bounded:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::metric\_space$ 
  assumes  $bounded: bounded\ (f\ ' (cbox\ a\ b))$ 
  shows  $bounded\ (range\ (\lambda x. f\ (clamp\ a\ b\ x)))$ 
proof cases
  assume  $le: (\forall i \in Basis. a \cdot i \leq b \cdot i)$ 
  from  $bounded$  obtain  $c$  where  $f\_bound: \forall x \in f\ ' cbox\ a\ b. dist\ undefined\ x \leq c$ 
  by (auto simp: bounded_any_center[where  $a=undefined$ ])
  then show ?thesis
  by (auto intro!: exI[where  $x=c$ ] clamp_in_interval[OF  $le$ [rule_format]]
      simp: bounded_any_center[where  $a=undefined$ ])
qed (auto simp: clamp_empty_interval image_def)

```

```

definition ext_cont ::  $('a::euclidean\_space \Rightarrow 'b::metric\_space) \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b$ 
  where  $ext\_cont\ f\ a\ b = (\lambda x. f\ (clamp\ a\ b\ x))$ 

```

```

lemma ext_cont_cancel_cbox[simp]:
  fixes  $x\ a\ b :: 'a::euclidean\_space$ 
  assumes  $x: x \in cbox\ a\ b$ 
  shows  $ext\_cont\ f\ a\ b\ x = f\ x$ 
  using assms
  unfolding ext_cont_def
  by (auto simp: clamp_def mem_box intro!: euclidean_eqI[where  $'a='a$ ] arg_cong[where  $f=f$ ])

```

```

lemma continuous_on_ext_cont[continuous_intros]:
   $continuous\_on\ (cbox\ a\ b)\ f \Longrightarrow continuous\_on\ S\ (ext\_cont\ f\ a\ b)$ 
  by (auto intro!: clamp_continuous_on simp: ext_cont_def)

```

4.1.16 Separability

```

lemma univ_second_countable_sequence:
  obtains  $B :: nat \Rightarrow 'a::euclidean\_space\ set$ 
  where  $inj\ B \wedge n. open(B\ n) \wedge S. open\ S \Longrightarrow \exists k. S = \bigcup \{B\ n \mid n. n \in k\}$ 
proof –
  obtain  $\mathcal{B} :: 'a\ set\ set$ 
  where countable  $\mathcal{B}$ 
  and  $opn: \bigwedge C. C \in \mathcal{B} \Longrightarrow open\ C$ 
  and  $Un: \bigwedge S. open\ S \Longrightarrow \exists U. U \subseteq \mathcal{B} \wedge S = \bigcup U$ 

```

```

    using univ_second_countable by blast
  have *: infinite (range (λn. ball (0::'a) (inverse(Suc n))))
    apply (rule Infinite_Set.range_inj_infinite)
    apply (simp add: inj_on_def ball_eq_ball_iff)
  done
  have infinite B
  proof
    assume finite B
    then have finite (Union ' (Pow B))
      by simp
    then have finite (range (λn. ball (0::'a) (inverse(Suc n))))
      apply (rule rev_finite_subset)
      by (metis (no_types, lifting) PowI image_eqI image_subset_iff Un [OF
open_ball])
    with * show False by simp
  qed
  obtain f :: nat ⇒ 'a set where B = range f inj f
    by (blast intro: countable_as_injective_image [OF ⟨countable B⟩ ⟨infinite B⟩])
  have *: ∃ k. S = ⋃ {f n | n. n ∈ k} if open S for S
    using Un [OF that]
    apply clarify
    apply (rule_tac x=f- 'U in exI)
    using ⟨inj f⟩ ⟨B = range f⟩ apply force
  done
  show ?thesis
    apply (rule that [OF ⟨inj f⟩ _ *])
    apply (auto simp: ⟨B = range f⟩ opn)
  done
qed

proposition separable:
  fixes S :: 'a::{metric_space, second_countable_topology} set
  obtains T where countable T T ⊆ S S ⊆ closure T
  proof -
    obtain B :: 'a set set
      where countable B
        and {} ∉ B
        and ope: ⋀ C. C ∈ B ⇒ openin(top_of_set S) C
        and if_ope: ⋀ T. openin(top_of_set S) T ⇒ ∃ U. U ⊆ B ∧ T = ⋃ U
      by (meson subset_second_countable)
    then obtain f where f: ⋀ C. C ∈ B ⇒ f C ∈ C
      by (metis equalsOI)
    show ?thesis
    proof
      show countable (f ' B)
        by (simp add: ⟨countable B⟩)
      show f ' B ⊆ S
        using ope f openin_imp_subset by blast
      show S ⊆ closure (f ' B)

```

```

proof (clarsimp simp: closure_approachable)
  fix  $x$  and  $e::\text{real}$ 
  assume  $x \in S$   $0 < e$ 
  have  $\text{openin } (\text{top\_of\_set } S) (S \cap \text{ball } x \ e)$ 
    by (simp add: openin_Int_open)
  with  $\text{if\_ope}$  obtain  $\mathcal{U}$  where  $\mathcal{U}: \mathcal{U} \subseteq \mathcal{B} \ S \cap \text{ball } x \ e = \bigcup \mathcal{U}$ 
    by meson
  show  $\exists C \in \mathcal{B}. \text{dist } (f \ C) \ x < e$ 
  proof (cases  $\mathcal{U} = \{\}$ )
    case True
    then show ?thesis
      using  $\langle 0 < e \rangle \ \mathcal{U} \ \langle x \in S \rangle$  by auto
  next
    case False
    then obtain  $C$  where  $C \in \mathcal{U}$  by blast
    show ?thesis
      proof
        show  $\text{dist } (f \ C) \ x < e$ 
        by (metis Int_iff Union_iff  $\mathcal{U} \ \langle C \in \mathcal{U} \rangle \text{dist\_commute } f \text{mem\_ball}$ 
subsetCE)
        show  $C \in \mathcal{B}$ 
        using  $\langle \mathcal{U} \subseteq \mathcal{B} \rangle \ \langle C \in \mathcal{U} \rangle$  by blast
      qed
    qed
  qed
qed
qed

```

4.1.17 Diameter

```

lemma diameter_cball [simp]:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  shows  $\text{diameter}(\text{cball } a \ r) = (\text{if } r < 0 \text{ then } 0 \text{ else } 2*r)$ 
proof -
  have  $\text{diameter}(\text{cball } a \ r) = 2*r$  if  $r \geq 0$ 
  proof (rule order_antisym)
    show  $\text{diameter}(\text{cball } a \ r) \leq 2*r$ 
    proof (rule diameter_le)
      fix  $x \ y$  assume  $x \in \text{cball } a \ r \ y \in \text{cball } a \ r$ 
      then have  $\text{norm } (x - a) \leq r \ \text{norm } (a - y) \leq r$ 
        by (auto simp: dist_norm norm_minus_commute)
      then have  $\text{norm } (x - y) \leq r + r$ 
        using norm_diff_triangle_le by blast
      then show  $\text{norm } (x - y) \leq 2*r$  by simp
    qed (simp add: that)
    have  $2*r = \text{dist } (a + r *_{\mathbb{R}} (\text{SOME } i. i \in \text{Basis})) (a - r *_{\mathbb{R}} (\text{SOME } i. i \in \text{Basis}))$ 
  apply (simp add: dist_norm)
    by (metis abs_of_nonneg mult.right_neutral norm_numerical norm_scaleR)

```

```

norm_some_Basis real_norm_def scaleR_2 that)
  also have ...  $\leq$  diameter (cball a r)
  apply (rule diameter_bounded_bound)
  using that by (auto simp: dist_norm)
  finally show  $2*r \leq$  diameter (cball a r) .
qed
then show ?thesis by simp
qed

```

```

lemma diameter_ball [simp]:
  fixes a :: 'a::euclidean_space
  shows diameter(ball a r) = (if r < 0 then 0 else 2*r)
proof -
  have diameter(ball a r) = 2*r if r > 0
  by (metis bounded_ball diameter_closure closure_ball diameter_cball less_eq_real_def
linorder_not_less that)
  then show ?thesis
  by (simp add: diameter_def)
qed

```

```

lemma diameter_closed_interval [simp]: diameter {a..b} = (if b < a then 0 else
b-a)
proof -
  have {a .. b} = cball ((a+b)/2) ((b-a)/2)
  by (auto simp: dist_norm abs_if field_split_simps split: if_split_asm)
  then show ?thesis
  by simp
qed

```

```

lemma diameter_open_interval [simp]: diameter {a<..b} = (if b < a then 0 else
b-a)
proof -
  have {a <..b} = ball ((a+b)/2) ((b-a)/2)
  by (auto simp: dist_norm abs_if field_split_simps split: if_split_asm)
  then show ?thesis
  by simp
qed

```

```

lemma diameter_cbox:
  fixes a b :: 'a::euclidean_space
  shows  $(\forall i \in \text{Basis}. a \cdot i \leq b \cdot i) \implies$  diameter (cbox a b) = dist a b
  by (force simp: diameter_def intro!: cSup_eq_maximum L2_set_mono
simp: euclidean_dist_l2[where 'a='a] cbox_def dist_norm)

```

4.1.18 Relating linear images to open/closed/interior/closure/connected

```

proposition open_surjective_linear_image:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::euclidean_space

```

```

    assumes open A linear f surj f
    shows open(f ' A)
  unfolding open_dist
  proof clarify
    fix x
    assume x ∈ A
    have bounded (inv f ' Basis)
      by (simp add: finite_imp_bounded)
    with bounded_pos obtain B where B > 0 and B:  $\bigwedge x. x \in \text{inv } f ' \text{Basis} \implies$ 
    norm x ≤ B
    by metis
    obtain e where e > 0 and e:  $\bigwedge z. \text{dist } z \ x < e \implies z \in A$ 
    by (metis open_dist ⟨x ∈ A⟩ ⟨open A⟩)
    define δ where δ ≡ e / B / DIM('b)
    show  $\exists e > 0. \forall y. \text{dist } y (f x) < e \longrightarrow y \in f ' A$ 
    proof (intro exI conjI)
      show δ > 0
        using ⟨e > 0⟩ ⟨B > 0⟩ by (simp add: δ_def field_split_simps)
      have y ∈ f ' A if dist y (f x) * (B * real DIM('b)) < e for y
      proof -
        define u where u ≡ y - f x
        show ?thesis
        proof (rule image_eqI)
          show y = f (x + (∑ i ∈ Basis. (u • i) *R inv f i))
          apply (simp add: linear_add linear_sum linear.scaleR ⟨linear f⟩ surj_f_inv_f
            ⟨surj f⟩)
          apply (simp add: euclidean_representation u_def)
          done
          have dist (x + (∑ i ∈ Basis. (u • i) *R inv f i)) x ≤ (∑ i ∈ Basis. norm ((u
            • i) *R inv f i))
          by (simp add: dist_norm sum_norm_le)
          also have ... = (∑ i ∈ Basis. |u • i| * norm (inv f i))
          by simp
          also have ... ≤ (∑ i ∈ Basis. |u • i|) * B
          by (simp add: B sum_distrib_right sum_mono mult_left_mono)
          also have ... ≤ DIM('b) * dist y (f x) * B
          apply (rule mult_right_mono [OF sum_bounded_above])
          using ⟨0 < B⟩ by (auto simp: Basis_le_norm dist_norm u_def)
          also have ... < e
          by (metis mult.commute mult.left_commute that)
          finally show x + (∑ i ∈ Basis. (u • i) *R inv f i) ∈ A
          by (rule e)
        qed
      qed
    qed
    then show  $\forall y. \text{dist } y (f x) < \delta \longrightarrow y \in f ' A$ 
    using ⟨e > 0⟩ ⟨B > 0⟩
    by (auto simp: δ_def field_split_simps)
  qed
  qed

```

```

corollary open_bijjective_linear_image_eq:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes linear  $f$  bij  $f$ 
  shows  $\text{open}(f \text{ ` } A) \longleftrightarrow \text{open } A$ 
proof
  assume  $\text{open}(f \text{ ` } A)$ 
  then have  $\text{open}(f - \text{ ` } (f \text{ ` } A))$ 
    using assms by (force simp: linear_continuous_at linear_conv_bounded_linear
continuous_open_vimage)
  then show  $\text{open } A$ 
    by (simp add: assms bij_is_inj inj_vimage_image_eq)
next
  assume  $\text{open } A$ 
  then show  $\text{open}(f \text{ ` } A)$ 
    by (simp add: assms bij_is_surj open_surjective_linear_image)
qed

```

```

corollary interior_bijjective_linear_image:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes linear  $f$  bij  $f$ 
  shows  $\text{interior}(f \text{ ` } S) = f \text{ ` } \text{interior } S$  (is ?lhs = ?rhs)
proof safe
  fix  $x$ 
  assume  $x: x \in ?lhs$ 
  then obtain  $T$  where  $\text{open } T$  and  $x \in T$  and  $T \subseteq f \text{ ` } S$ 
    by (metis interiorE)
  then show  $x \in ?rhs$ 
    by (metis (no_types, opaque_lifting) assms subsetD interior_maximal open_bijjective_linear_image_c
subset_image_iff)
next
  fix  $x$ 
  assume  $x: x \in \text{interior } S$ 
  then show  $f x \in \text{interior}(f \text{ ` } S)$ 
    by (meson assms imageI image_mono interiorI interior_subset open_bijjective_linear_image_eq
open_interior)
qed

```

```

lemma interior_injective_linear_image:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'a::\text{euclidean\_space}$ 
  assumes linear  $f$  inj  $f$ 
  shows  $\text{interior}(f \text{ ` } S) = f \text{ ` } (\text{interior } S)$ 
by (simp add: linear_injective_imp_surjective assms bijI interior_bijjective_linear_image)

```

```

lemma interior_surjective_linear_image:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'a::\text{euclidean\_space}$ 
  assumes linear  $f$  surj  $f$ 
  shows  $\text{interior}(f \text{ ` } S) = f \text{ ` } (\text{interior } S)$ 
by (simp add: assms interior_injective_linear_image linear_surjective_imp_injective)

```


lemma *interior_negations*:

fixes $S :: 'a::\text{euclidean\_space}$ *set*

shows $\text{interior}(\text{uminus } S) = \text{image } \text{uminus } (\text{interior } S)$

by (*simp add: bij_uminus interior_bijective_linear_image linear_uminus*)

lemma *connected_linear_image*:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{real\_normed\_vector}$

assumes *linear f and connected s*

shows *connected (f ` s)*

using *connected_continuous_image assms linear_continuous_on linear_conv_bounded_linear*

by *blast*

4.1.19 "Isometry" (up to constant bounds) of Injective Linear Map

proposition *injective_imp_isometric*:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$

assumes *s: closed s subspace s*

and $f: \text{bounded\_linear } f \ \forall x \in s. f \ x = 0 \longrightarrow x = 0$

shows $\exists e > 0. \forall x \in s. \text{norm } (f \ x) \geq e * \text{norm } x$

proof (*cases s $\subseteq \{0::'a\}$*)

case *True*

have $\text{norm } x \leq \text{norm } (f \ x)$ **if** $x \in s$ **for** x

proof $-$

from *True* **that** **have** $x = 0$ **by** *auto*

then show *?thesis* **by** *simp*

qed

then show *?thesis*

by (*auto intro!: exI[where x=1]*)

next

case *False*

interpret $f: \text{bounded\_linear } f$ **by** *fact*

from *False* **obtain** a **where** $a: a \neq 0 \ a \in s$

by *auto*

from *False* **have** $s \neq \{\}$

by *auto*

let $?S = \{f \ x \mid x. x \in s \wedge \text{norm } x = \text{norm } a\}$

let $?S' = \{x::'a. x \in s \wedge \text{norm } x = \text{norm } a\}$

let $?S'' = \{x::'a. \text{norm } x = \text{norm } a\}$

have $?S'' = \text{frontier } (\text{cball } 0 \ (\text{norm } a))$

by (*simp add: sphere_def dist_norm*)

then have *compact ?S''* **by** (*metis compact_cball compact_frontier*)

moreover have $?S' = s \cap ?S''$ **by** *auto*

ultimately have *compact ?S'*

using *closed_Int_compact[of s ?S'']* **using** $s(1)$ **by** *auto*

moreover have $*:f ` ?S' = ?S$ **by** *auto*

ultimately have *compact ?S*

```

    using compact_continuous_image[OF linear_continuous_on[OF f(1)], of ?S]
  by auto
  then have closed ?S
    using compact_imp_closed by auto
  moreover from a have ?S ≠ {} by auto
  ultimately obtain b' where b' ∈ ?S ∀ y ∈ ?S. norm b' ≤ norm y
    using distance_attains_inf[of ?S 0] unfolding dist_0_norm by auto
  then obtain b where b ∈ s
    and ba: norm b = norm a
    and b: ∀ x ∈ {x ∈ s. norm x = norm a}. norm (f b) ≤ norm (f x)
    unfolding *[symmetric] unfolding image_iff by auto

  let ?e = norm (f b) / norm b
  have norm b > 0
    using ba and a and norm_ge_zero by auto
  moreover have norm (f b) > 0
    using f(2)[THEN bspec[where x=b], OF ⟨b ∈ s⟩]
    using ⟨norm b > 0⟩ by simp
  ultimately have 0 < norm (f b) / norm b by simp
  moreover
  have norm (f b) / norm b * norm x ≤ norm (f x) if x ∈ s for x
  proof (cases x = 0)
    case True
      then show norm (f b) / norm b * norm x ≤ norm (f x)
        by auto
    next
      case False
        with ⟨a ≠ 0⟩ have *: 0 < norm a / norm x
          unfolding zero_less_norm_iff[symmetric] by simp
        have ∀ x ∈ s. c *R x ∈ s for c
          using s[unfolded subspace_def] by simp
        with ⟨x ∈ s⟩ ⟨x ≠ 0⟩ have (norm a / norm x) *R x ∈ {x ∈ s. norm x = norm a}
          by simp
        with ⟨x ≠ 0⟩ ⟨a ≠ 0⟩ show norm (f b) / norm b * norm x ≤ norm (f x)
          using b[THEN bspec[where x=(norm a / norm x) *R x]]
          unfolding f.scaleR and ba
          by (auto simp: mult.commute pos_le_divide_eq pos_divide_le_eq)
      qed
    ultimately show ?thesis by auto
  qed

```

```

proposition closed_injective_image_subspace:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes subspace s bounded_linear f ∀ x ∈ s. f x = 0 ⟶ x = 0 closed s
  shows closed(f ` s)
proof –
  obtain e where e > 0 and e: ∀ x ∈ s. e * norm x ≤ norm (f x)
    using injective_imp_isometric[OF assms(4,1,2,3)] by auto

```

```

show ?thesis
  using complete_isometric_image[OF ‹ $e > 0$ › assms(1,2) e] and assms(4)
  unfolding complete_eq_closed[symmetric] by auto
qed

```

```

lemma closure_bounded_linear_image_subset:
  assumes f: bounded_linear f
  shows f ` closure S  $\subseteq$  closure (f ` S)
  using linear_continuous_on [OF f] closed_closure closure_subset
  by (rule image_closure_subset)

```

```

lemma closure_linear_image_subset:
  fixes f :: 'm::euclidean_space  $\Rightarrow$  'n::real_normed_vector
  assumes linear f
  shows f ` (closure S)  $\subseteq$  closure (f ` S)
  using assms unfolding linear_conv_bounded_linear
  by (rule closure_bounded_linear_image_subset)

```

```

lemma closed_injective_linear_image:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes S: closed S and f: linear f inj f
  shows closed (f ` S)
proof -
  obtain g where g: linear g g  $\circ$  f = id
  using linear_injective_left_inverse [OF f] by blast
  then have cfg: continuous_on (range f) g
  using linear_continuous_on linear_conv_bounded_linear by blast
  have [simp]: g ` f ` S = S
  using g by (simp add: image_comp)
  have cgf: closed (g ` f ` S)
  by (simp add: ‹g  $\circ$  f = id› S image_comp)
  have [simp]: (range f  $\cap$  g - ` S) = f ` S
  using g unfolding o_def id_def image_def by auto metis+
  show ?thesis
proof (rule closedin_closed_trans [of range f])
  show closedin (top_of_set (range f)) (f ` S)
  using continuous_closedin_preimage [OF cfg cgf] by simp
  show closed (range f)
  apply (rule closed_injective_image_subspace)
  using f apply (auto simp: linear_linear linear_injective_0)
  done
qed
qed

```

```

lemma closed_injective_linear_image_eq:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes f: linear f inj f
  shows (closed (image f s)  $\longleftrightarrow$  closed s)

```

by (metis closed_injective_linear_image_closure_eq closure_linear_image_subset
closure_subset_eq f(1) f(2) inj_image_subset_iff)

lemma closure_injective_linear_image:

fixes f :: 'a::euclidean_space $\Rightarrow$ 'b::euclidean_space
shows $\llbracket \text{linear } f; \text{inj } f \rrbracket \Longrightarrow f \text{ ` } (\text{closure } S) = \text{closure } (f \text{ ` } S)$
apply (rule subset_antisym)
apply (simp add: closure_linear_image_subset)
by (simp add: closure_minimal closed_injective_linear_image_closure_subset
image_mono)

lemma closure_bounded_linear_image:

fixes f :: 'a::euclidean_space $\Rightarrow$ 'b::euclidean_space
shows $\llbracket \text{linear } f; \text{bounded } S \rrbracket \Longrightarrow f \text{ ` } (\text{closure } S) = \text{closure } (f \text{ ` } S)$
apply (rule subset_antisym, simp add: closure_linear_image_subset)
apply (rule closure_minimal, simp add: closure_subset image_mono)
by (meson bounded_closure closed_closure compact_continuous_image_com-
pact_eq_bounded_closed linear_continuous_on linear_conv_bounded_linear)

lemma closure_scaleR:

fixes S :: 'a::real_normed_vector set
shows $((*_R) \text{ ` } c) \text{ ` } (\text{closure } S) = \text{closure } (((*_R) \text{ ` } c) \text{ ` } S)$
proof
show $((*_R) \text{ ` } c) \text{ ` } (\text{closure } S) \subseteq \text{closure } (((*_R) \text{ ` } c) \text{ ` } S)$
using bounded_linear_scaleR_right
by (rule closure_bounded_linear_image_subset)
show $\text{closure } (((*_R) \text{ ` } c) \text{ ` } S) \subseteq ((*_R) \text{ ` } c) \text{ ` } (\text{closure } S)$
by (intro closure_minimal image_mono closure_subset closed_scaling closed_closure)
qed

4.1.20 Some properties of a canonical subspace

lemma closed_substandard: closed $\{x::'a::euclidean\_space. \forall i \in \text{Basis}. P \ i \longrightarrow x \cdot i = 0\}$

(is closed ?A)

proof –

let ?D = $\{i \in \text{Basis}. P \ i\}$

have closed $(\bigcap i \in ?D. \{x::'a. x \cdot i = 0\})$

by (simp add: closed_INT closed_Collect_eq continuous_on_inner)

also have $(\bigcap i \in ?D. \{x::'a. x \cdot i = 0\}) = ?A$

by auto

finally show closed ?A .

qed

lemma closed_subspace:

fixes s :: 'a::euclidean_space set

assumes subspace s

shows closed s

proof –

```

have  $\dim s \leq \text{card } (\text{Basis} :: 'a \text{ set})$ 
  using  $\dim\_subset\_UNIV$  by auto
with  $\text{ex\_card}[OF \text{ this}]$  obtain  $d :: 'a \text{ set}$  where  $t: \text{card } d = \dim s$  and  $d: d \subseteq$ 
Basis
  by auto
let  $?t = \{x :: 'a. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\}$ 
have  $\exists f. \text{linear } f \wedge f' \{x :: 'a. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\} = s \wedge$ 
   $\text{inj\_on } f \{x :: 'a. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\}$ 
  using  $\dim\_substandard[of d] \ t \ d \ \text{assms}$ 
  by (intro  $\text{subspace\_isomorphism}[OF \text{ subspace\_substandard}[of \lambda i. i \notin d]]$ ) (auto
simp:  $\text{inner\_Basis}$ )
then obtain  $f$  where  $f$ :
   $\text{linear } f$ 
   $f' \{x. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\} = s$ 
   $\text{inj\_on } f \{x. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\}$ 
  by blast
interpret  $f$ :  $\text{bounded\_linear } f$ 
  using  $f$  by (simp add:  $\text{linear\_conv\_bounded\_linear}$ )
have  $x \in ?t \implies f x = 0 \implies x = 0$  for  $x$ 
  using  $f.\text{zero } d \ f(3)[THEN \text{inj\_on } D, \text{ of } x \ 0]$  by auto
moreover have  $\text{closed } ?t$  by (rule  $\text{closed\_substandard}$ )
moreover have  $\text{subspace } ?t$  by (rule  $\text{subspace\_substandard}$ )
ultimately show  $?thesis$ 
  using  $\text{closed\_injective\_image\_subspace}[of ?t f]$ 
  unfolding  $f(2)$  using  $f(1)$  unfolding  $\text{linear\_conv\_bounded\_linear}$  by auto
qed

lemma  $\text{complete\_subspace}$ :  $\text{subspace } s \implies \text{complete } s$ 
  for  $s :: 'a :: \text{euclidean\_space} \text{ set}$ 
  using  $\text{complete\_eq\_closed } \text{closed\_subspace}$  by auto

lemma  $\text{closed\_span } [iff]$ :  $\text{closed } (\text{span } s)$ 
  for  $s :: 'a :: \text{euclidean\_space} \text{ set}$ 
  by (simp add:  $\text{closed\_subspace}$ )

lemma  $\dim\_closure$  [simp]:  $\dim (\text{closure } s) = \dim s$  (is  $?dc = ?d$ )
  for  $s :: 'a :: \text{euclidean\_space} \text{ set}$ 
proof -
  have  $?dc \leq ?d$ 
    using  $\text{closure\_minimal}[OF \text{ span\_superset}, \text{ of } s]$ 
    using  $\text{closed\_subspace}[OF \text{ subspace\_span}, \text{ of } s]$ 
    using  $\dim\_subset[of \text{ closure } s \text{ span } s]$ 
    by simp
  then show  $?thesis$ 
    using  $\dim\_subset[OF \text{ closure\_subset}, \text{ of } s]$ 
    by simp
qed

```

4.1.21 Set Distance

lemma *setdist_compact_closed*:

fixes $A :: 'a::heine\_borel\ set$

assumes $A: compact\ A$ **and** $B: closed\ B$

and $A \neq \{\}$ $B \neq \{\}$

shows $\exists x \in A. \exists y \in B. dist\ x\ y = setdist\ A\ B$

proof –

obtain x **where** $x \in A$ $setdist\ A\ B = infdist\ x\ B$

by (*metis* $A\ assms(3)$ *setdist_attains_inf setdist_sym*)

moreover

obtain y **where** $y \in B$ $infdist\ x\ B = dist\ x\ y$

using $B \neq \{\}$ *infdist_attains_inf* **by** *blast*

ultimately show *?thesis*

using $\langle x \in A \rangle \langle y \in B \rangle$ **by** *auto*

qed

lemma *setdist_closed_compact*:

fixes $S :: 'a::heine\_borel\ set$

assumes $S: closed\ S$ **and** $T: compact\ T$

and $S \neq \{\}$ $T \neq \{\}$

shows $\exists x \in S. \exists y \in T. dist\ x\ y = setdist\ S\ T$

using *setdist_compact_closed* [*OF* $T\ S \neq \{\}$ $\neq \{\}$]

by (*metis* *dist_commute setdist_sym*)

lemma *setdist_eq_0_compact_closed*:

assumes $S: compact\ S$ **and** $T: closed\ T$

shows $setdist\ S\ T = 0 \longleftrightarrow S = \{\} \vee T = \{\} \vee S \cap T \neq \{\}$

proof (*cases* $S = \{\} \vee T = \{\}$)

case *True*

then show *?thesis*

by *force*

next

case *False*

then show *?thesis*

by (*metis* $S\ T\ disjoint\_iff\_not\_equal\ in\_closed\_iff\_infdist\_zero\ setdist\_attains\_inf\ setdist\_eq\_0I\ setdist\_sym$)

qed

corollary *setdist_gt_0_compact_closed*:

assumes $S: compact\ S$ **and** $T: closed\ T$

shows $setdist\ S\ T > 0 \longleftrightarrow (S \neq \{\} \wedge T \neq \{\} \wedge S \cap T = \{\})$

using *setdist_pos_le* [*of* $S\ T$] *setdist_eq_0_compact_closed* [*OF* *assms*] **by** *linarith*

lemma *setdist_eq_0_closed_compact*:

assumes $S: closed\ S$ **and** $T: compact\ T$

shows $setdist\ S\ T = 0 \longleftrightarrow S = \{\} \vee T = \{\} \vee S \cap T \neq \{\}$

using *setdist_eq_0_compact_closed* [*OF* $T\ S$]

by (*metis* *Int_commute setdist_sym*)

```

lemma setdist_eq_0_bounded:
  fixes  $S :: 'a::heine\_borel\ set$ 
  assumes  $\text{bounded } S \vee \text{bounded } T$ 
  shows  $\text{setdist } S\ T = 0 \longleftrightarrow S = \{\} \vee T = \{\} \vee \text{closure } S \cap \text{closure } T \neq \{\}$ 
proof (cases  $S = \{\} \vee T = \{\}$ )
  case False
  then show ?thesis
    using setdist_eq_0_compact_closed [of closure  $S$  closure  $T$ ]
      setdist_eq_0_closed_compact [of closure  $S$  closure  $T$ ] assms
    by (force simp: bounded_closure compact_eq_bounded_closed)
qed force

```

```

lemma setdist_eq_0_sing_1:
   $\text{setdist } \{x\}\ S = 0 \longleftrightarrow S = \{\} \vee x \in \text{closure } S$ 
by (metis in_closure_iff_infdist_zero infdist_def infdist_eq_setdist)

```

```

lemma setdist_eq_0_sing_2:
   $\text{setdist } S\ \{x\} = 0 \longleftrightarrow S = \{\} \vee x \in \text{closure } S$ 
by (metis setdist_eq_0_sing_1 setdist_sym)

```

```

lemma setdist_neq_0_sing_1:
   $\llbracket \text{setdist } \{x\}\ S = a; a \neq 0 \rrbracket \implies S \neq \{\} \wedge x \notin \text{closure } S$ 
by (metis setdist_closure_2 setdist_empty2 setdist_eq_0I singletonI)

```

```

lemma setdist_neq_0_sing_2:
   $\llbracket \text{setdist } S\ \{x\} = a; a \neq 0 \rrbracket \implies S \neq \{\} \wedge x \notin \text{closure } S$ 
by (simp add: setdist_neq_0_sing_1 setdist_sym)

```

```

lemma setdist_sing_in_set:
   $x \in S \implies \text{setdist } \{x\}\ S = 0$ 
by (simp add: setdist_eq_0I)

```

```

lemma setdist_eq_0_closed:
   $\text{closed } S \implies (\text{setdist } \{x\}\ S = 0 \longleftrightarrow S = \{\} \vee x \in S)$ 
by (simp add: setdist_eq_0_sing_1)

```

```

lemma setdist_eq_0_closedin:
  shows  $\llbracket \text{closedin } (\text{top\_of\_set } U)\ S; x \in U \rrbracket$ 
     $\implies (\text{setdist } \{x\}\ S = 0 \longleftrightarrow S = \{\} \vee x \in S)$ 
by (auto simp: closedin_limpt setdist_eq_0_sing_1 closure_def)

```

```

lemma setdist_gt_0_closedin:
  shows  $\llbracket \text{closedin } (\text{top\_of\_set } U)\ S; x \in U; S \neq \{\}; x \notin S \rrbracket$ 
     $\implies \text{setdist } \{x\}\ S > 0$ 
  using less_eq_real_def setdist_eq_0_closedin by fastforce

```

```

no_notation
  eucl_less (infix <e 50)

```

end

4.2 Convex Sets and Functions on (Normed) Euclidean Spaces

```
theory Convex_Euclidean_Space
imports
  Convex
  Topology_Euclidean_Space
begin
```

4.2.1 Topological Properties of Convex Sets and Functions

```
lemma aff_dim_cball:
  fixes a :: 'n::euclidean_space
  assumes e > 0
  shows aff_dim (cball a e) = int (DIM('n))
proof -
  have ( $\lambda x. a + x$ ) ' (cball 0 e)  $\subseteq$  cball a e
    unfolding cball_def dist_norm by auto
  then have aff_dim (cball (0 :: 'n::euclidean_space) e)  $\leq$  aff_dim (cball a e)
    using aff_dim_translation_eq[of a cball 0 e]
      aff_dim_subset[of (+) a ' cball 0 e cball a e]
    by auto
  moreover have aff_dim (cball (0 :: 'n::euclidean_space) e) = int (DIM('n))
    using hull_inc[of (0 :: 'n::euclidean_space) cball 0 e]
      centre_in_cball[of (0 :: 'n::euclidean_space)] assms
    by (simp add: dim_cball[of e] aff_dim_zero[of cball 0 e])
  ultimately show ?thesis
    using aff_dim_le_DIM[of cball a e] by auto
qed
```

```
lemma aff_dim_open:
  fixes S :: 'n::euclidean_space set
  assumes open S
    and S  $\neq$  {}
  shows aff_dim S = int (DIM('n))
proof -
  obtain x where x  $\in$  S
    using assms by auto
  then obtain e where e: e > 0 cball x e  $\subseteq$  S
    using open_contains_cball[of S] assms by auto
  then have aff_dim (cball x e)  $\leq$  aff_dim S
    using aff_dim_subset by auto
  with e show ?thesis
    using aff_dim_cball[of e x] aff_dim_le_DIM[of S] by auto
qed
```



```

lemma low_dim_interior:
  fixes  $S :: 'n::\text{euclidean\_space}$  set
  assumes  $\neg \text{aff\_dim } S = \text{int } (\text{DIM } 'n)$ 
  shows  $\text{interior } S = \{\}$ 
proof -
  have  $\text{aff\_dim } (\text{interior } S) \leq \text{aff\_dim } S$ 
    using interior_subset aff_dim_subset [of interior S S] by auto
  then show ?thesis
    using aff_dim_open [of interior S] aff_dim_le_DIM [of S] assms by auto
qed

corollary empty_interior_lowdim:
  fixes  $S :: 'n::\text{euclidean\_space}$  set
  shows  $\text{dim } S < \text{DIM } 'n \implies \text{interior } S = \{\}$ 
by (metis low_dim_interior affine_hull_UNIV dim_affine_hull less_not_refl dim_UNIV)

corollary aff_dim_nonempty_interior:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  shows  $\text{interior } S \neq \{\} \implies \text{aff\_dim } S = \text{DIM } 'a$ 
by (metis low_dim_interior)

```

4.2.2 Relative interior of a set

definition *rel_interior* $S =$
 $\{x. \exists T. \text{openin } (\text{top\_of\_set } (\text{affine hull } S)) \ T \wedge x \in T \wedge T \subseteq S\}$

```

lemma rel_interior_mono:
   $\llbracket S \subseteq T; \text{affine hull } S = \text{affine hull } T \rrbracket$ 
   $\implies (\text{rel\_interior } S) \subseteq (\text{rel\_interior } T)$ 
by (auto simp: rel_interior_def)

```

```

lemma rel_interior_maximal:
   $\llbracket T \subseteq S; \text{openin } (\text{top\_of\_set } (\text{affine hull } S)) \ T \rrbracket \implies T \subseteq (\text{rel\_interior } S)$ 
by (auto simp: rel_interior_def)

```

```

lemma rel_interior:  $\text{rel\_interior } S = \{x \in S. \exists T. \text{open } T \wedge x \in T \wedge T \cap \text{affine hull } S \subseteq S\}$ 
  (is ?lhs = ?rhs)

```

```

proof
  show ?lhs  $\subseteq$  ?rhs
    by (force simp add: rel_interior_def openin_open)
  { fix  $x \ T$ 
    assume  $*$ :  $x \in S \text{ open } T \ x \in T \ T \cap \text{affine hull } S \subseteq S$ 
    then have  $**$ :  $x \in T \cap \text{affine hull } S$ 
      using hull_inc by auto
    with  $*$  have  $\exists T_b. (\exists T_a. \text{open } T_a \wedge T_b = \text{affine hull } S \cap T_a) \wedge x \in T_b \wedge T_b$ 
     $\subseteq S$ 
    by (rule_tac  $x = T \cap (\text{affine hull } S)$  in exI) auto
  }

```

```

    }
    then show ?rhs  $\subseteq$  ?lhs
    by (force simp add: rel_interior_def openin_open)
qed

```

```

lemma mem_rel_interior:  $x \in \text{rel\_interior } S \longleftrightarrow (\exists T. \text{open } T \wedge x \in T \cap S \wedge T \cap \text{affine\_hull } S \subseteq S)$ 
  by (auto simp: rel_interior)

```

```

lemma mem_rel_interior_ball:
   $x \in \text{rel\_interior } S \longleftrightarrow x \in S \wedge (\exists e. e > 0 \wedge \text{ball } x \ e \cap \text{affine\_hull } S \subseteq S)$ 
  (is ?lhs = ?rhs)
proof
  assume ?rhs then show ?lhs
    by (simp add: rel_interior) (meson Elementary_Metric_Spaces.open_ball centre_in_ball)
qed (force simp: rel_interior open_contains_ball)

```

```

lemma rel_interior_ball:
   $\text{rel\_interior } S = \{x \in S. \exists e. e > 0 \wedge \text{ball } x \ e \cap \text{affine\_hull } S \subseteq S\}$ 
  using mem_rel_interior_ball [of _ S] by auto

```

```

lemma mem_rel_interior_cball:
   $x \in \text{rel\_interior } S \longleftrightarrow x \in S \wedge (\exists e. e > 0 \wedge \text{cball } x \ e \cap \text{affine\_hull } S \subseteq S)$ 
  (is ?lhs = ?rhs)
proof
  assume ?rhs then obtain e where  $x \in S \ e > 0 \ \text{cball } x \ e \cap \text{affine\_hull } S \subseteq S$ 
    by (auto simp: rel_interior)
  then have  $\text{ball } x \ e \cap \text{affine\_hull } S \subseteq S$ 
    by auto
  then show ?lhs
    using  $\langle 0 < e \rangle \langle x \in S \rangle \text{rel\_interior\_ball}$  by auto
qed (force simp: rel_interior open_contains_cball)

```

```

lemma rel_interior_cball:
   $\text{rel\_interior } S = \{x \in S. \exists e. e > 0 \wedge \text{cball } x \ e \cap \text{affine\_hull } S \subseteq S\}$ 
  using mem_rel_interior_cball [of _ S] by auto

```

```

lemma rel_interior_empty [simp]:  $\text{rel\_interior } \{\} = \{\}$ 
  by (auto simp: rel_interior_def)

```

```

lemma affine_hull_sing [simp]:  $\text{affine\_hull } \{a :: 'n::\text{euclidean\_space}\} = \{a\}$ 
  by (metis affine_hull_eq affine_sing)

```

```

lemma rel_interior_sing [simp]:
  fixes  $a :: 'n::\text{euclidean\_space}$  shows  $\text{rel\_interior } \{a\} = \{a\}$ 
proof -
  have  $\exists x::\text{real}. 0 < x$ 
    using zero_less_one by blast

```

```

then show ?thesis
  by (auto simp: rel_interior_ball)
qed

```

```

lemma subset_rel_interior:
  fixes  $S\ T :: 'n::\text{euclidean\_space}\ \text{set}$ 
  assumes  $S \subseteq T$ 
  and  $\text{affine hull } S = \text{affine hull } T$ 
  shows  $\text{rel\_interior } S \subseteq \text{rel\_interior } T$ 
  using assms by (auto simp: rel_interior_def)

```

```

lemma rel_interior_subset:  $\text{rel\_interior } S \subseteq S$ 
  by (auto simp: rel_interior_def)

```

```

lemma rel_interior_subset_closure:  $\text{rel\_interior } S \subseteq \text{closure } S$ 
  using rel_interior_subset by (auto simp: closure_def)

```

```

lemma interior_subset_rel_interior:  $\text{interior } S \subseteq \text{rel\_interior } S$ 
  by (auto simp: rel_interior interior_def)

```

```

lemma interior_rel_interior:
  fixes  $S :: 'n::\text{euclidean\_space}\ \text{set}$ 
  assumes  $\text{aff\_dim } S = \text{int}(\text{DIM}('n))$ 
  shows  $\text{rel\_interior } S = \text{interior } S$ 
proof –
  have  $\text{affine hull } S = \text{UNIV}$ 
  using assms  $\text{affine\_hull\_UNIV[of } S\text{]}$  by auto
  then show ?thesis
    unfolding rel_interior interior_def by auto
qed

```

```

lemma rel_interior_interior:
  fixes  $S :: 'n::\text{euclidean\_space}\ \text{set}$ 
  assumes  $\text{affine hull } S = \text{UNIV}$ 
  shows  $\text{rel\_interior } S = \text{interior } S$ 
  using assms unfolding rel_interior interior_def by auto

```

```

lemma rel_interior_open:
  fixes  $S :: 'n::\text{euclidean\_space}\ \text{set}$ 
  assumes open  $S$ 
  shows  $\text{rel\_interior } S = S$ 
  by (metis assms interior_eq interior_subset_rel_interior rel_interior_subset
    set_eq_subset)

```

```

lemma interior_rel_interior_gen:
  fixes  $S :: 'n::\text{euclidean\_space}\ \text{set}$ 
  shows  $\text{interior } S = (\text{if } \text{aff\_dim } S = \text{int}(\text{DIM}('n)) \text{ then } \text{rel\_interior } S \text{ else } \{\})$ 
  by (metis interior_rel_interior low_dim_interior)

```

```

lemma rel_interior_nonempty_interior:
  fixes S :: 'n::euclidean_space set
  shows interior S  $\neq$  {}  $\implies$  rel_interior S = interior S
by (metis interior_rel_interior_gen)

lemma affine_hull_nonempty_interior:
  fixes S :: 'n::euclidean_space set
  shows interior S  $\neq$  {}  $\implies$  affine hull S = UNIV
by (metis affine_hull_UNIV interior_rel_interior_gen)

lemma rel_interior_affine_hull [simp]:
  fixes S :: 'n::euclidean_space set
  shows rel_interior (affine hull S) = affine hull S
proof -
  have *: rel_interior (affine hull S)  $\subseteq$  affine hull S
  using rel_interior_subset by auto
  {
    fix x
    assume x: x  $\in$  affine hull S
    define e :: real where e = 1
    then have e > 0 ball x e  $\cap$  affine hull (affine hull S)  $\subseteq$  affine hull S
    using hull_hull[of _ S] by auto
    then have x  $\in$  rel_interior (affine hull S)
    using x rel_interior_ball[of affine hull S] by auto
  }
  then show ?thesis using * by auto
qed

lemma rel_interior_UNIV [simp]: rel_interior (UNIV :: ('n::euclidean_space)
set) = UNIV
by (metis open_UNIV rel_interior_open)

lemma rel_interior_convex_shrink:
  fixes S :: 'a::euclidean_space set
  assumes convex S
  and c  $\in$  rel_interior S
  and x  $\in$  S
  and 0 < e
  and e  $\leq$  1
  shows x - e *R (x - c)  $\in$  rel_interior S
proof -
  obtain d where d > 0 and d: ball c d  $\cap$  affine hull S  $\subseteq$  S
  using assms(2) unfolding mem_rel_interior_ball by auto
  {
    fix y
    assume as: dist (x - e *R (x - c)) y < e * d y  $\in$  affine hull S
    have *: y = (1 - (1 - e)) *R ((1 / e) *R y - ((1 - e) / e) *R x) + (1 - e)
    *R x
    using <e > 0 by (auto simp: scaleR_left_diff_distrib scaleR_right_diff_distrib)
  }

```

```

have x ∈ affine hull S
  using assms hull_subset[of S] by auto
moreover have 1 / e + - ((1 - e) / e) = 1
  using ‹e > 0› left_diff_distrib[of 1 (1-e) 1/e] by auto
ultimately have **: (1 / e) *R y - ((1 - e) / e) *R x ∈ affine hull S
  using as affine_affine_hull[of S] mem_affine[of affine hull S y x (1 / e) -((1
- e) / e)]
  by (simp add: algebra_simps)
have c - ((1 / e) *R y - ((1 - e) / e) *R x) = (1 / e) *R (e *R c - y + (1
- e) *R x)
  using ‹e > 0›
  by (auto simp: euclidean_eq_iff[where 'a='a] field_simps inner_simps)
then have dist c ((1 / e) *R y - ((1 - e) / e) *R x) = |1/e| * norm (e *R c
- y + (1 - e) *R x)
  unfolding dist_norm norm_scaleR[symmetric] by auto
also have ... = |1/e| * norm (x - e *R (x - c) - y)
  by (auto intro!: arg_cong[where f=norm] simp add: algebra_simps)
also have ... < d
  using as[unfolded dist_norm] and ‹e > 0›
  by (auto simp: pos_divide_less_eq[OF ‹e > 0›] mult.commute)
finally have (1 / e) *R y - ((1 - e) / e) *R x ∈ S
  using ** d by auto
then have y ∈ S
  using * convexD [OF ‹convex S›] assms(3-5)
  by (metis diff_add_cancel diff_ge_0_iff_ge le_add_same_cancel1 less_eq_real_def)
}
then have ball (x - e *R (x - c)) (e*d) ∩ affine hull S ⊆ S
  by auto
moreover have e * d > 0
  using ‹e > 0› ‹d > 0› by simp
moreover have c: c ∈ S
  using assms rel_interior_subset by auto
moreover from c have x - e *R (x - c) ∈ S
  using convexD_alt[of S x c e] assms
  by (metis diff_add_eq diff_diff_eq2 less_eq_real_def scaleR_diff_left scaleR_one
scale_right_diff_distrib)
ultimately show ?thesis
  using mem_rel_interior_ball[of x - e *R (x - c) S] ‹e > 0› by auto
qed

```

lemma interior_real_atLeast [simp]:

fixes a :: real

shows interior {a..} = {a<..}

proof -

{

fix y

have ball y (y - a) ⊆ {a..}

by (auto simp: dist_norm)

moreover assume a < y

```

    ultimately have  $y \in \text{interior } \{a.. \}$ 
    by (force simp add: mem_interior)
  }
  moreover
  {
    fix  $y$ 
    assume  $y \in \text{interior } \{a.. \}$ 
    then obtain  $e$  where  $e: e > 0$   $\text{cball } y \ e \subseteq \{a.. \}$ 
    using mem_interior_cball[of  $y \ \{a.. \}$ ] by auto
    moreover from  $e$  have  $y - e \in \text{cball } y \ e$ 
    by (auto simp: cball_def dist_norm)
    ultimately have  $a \leq y - e$  by blast
    then have  $a < y$  using  $e$  by auto
  }
  ultimately show ?thesis by auto
qed

lemma continuous_ge_on_Ioo:
  assumes continuous_on  $\{c..d\}$   $g \wedge x. x \in \{c < .. < d\} \implies g \ x \geq a \ c < d \ x \in \{c..d\}$ 
  shows  $g \ (x::\text{real}) \geq (a::\text{real})$ 
proof -
  from assms(3) have  $\{c..d\} = \text{closure } \{c < .. < d\}$  by (rule closure_greaterThanLessThan[symmetric])
  also from assms(2) have  $\{c < .. < d\} \subseteq (g - ' \{a.. \} \cap \{c..d\})$  by auto
  hence  $\text{closure } \{c < .. < d\} \subseteq \text{closure } (g - ' \{a.. \} \cap \{c..d\})$  by (rule closure_mono)
  also from assms(1) have  $\text{closed } (g - ' \{a.. \} \cap \{c..d\})$ 
  by (auto simp: continuous_on_closed_vimage)
  hence  $\text{closure } (g - ' \{a.. \} \cap \{c..d\}) = g - ' \{a.. \} \cap \{c..d\}$  by simp
  finally show ?thesis using  $\langle x \in \{c..d\} \rangle$  by auto
qed

lemma interior_real_atMost [simp]:
  fixes  $a :: \text{real}$ 
  shows  $\text{interior } \{..a\} = \{..<a\}$ 
proof -
  {
    fix  $y$ 
    have  $\text{ball } y \ (a - y) \subseteq \{..a\}$ 
    by (auto simp: dist_norm)
    moreover assume  $a > y$ 
    ultimately have  $y \in \text{interior } \{..a\}$ 
    by (force simp add: mem_interior)
  }
  moreover
  {
    fix  $y$ 
    assume  $y \in \text{interior } \{..a\}$ 
    then obtain  $e$  where  $e: e > 0$   $\text{cball } y \ e \subseteq \{..a\}$ 
    using mem_interior_cball[of  $y \ \{..a\}$ ] by auto
  }

```

```

    moreover from  $e$  have  $y + e \in \text{cball } y \ e$ 
    by (auto simp: cball_def dist_norm)
    ultimately have  $a \geq y + e$  by auto
    then have  $a > y$  using  $e$  by auto
  }
  ultimately show ?thesis by auto
qed

```

```

lemma interior_atLeastAtMost_real [simp]: interior  $\{a..b\} = \{a <..<b :: \text{real}\}$ 
proof-
  have  $\{a..b\} = \{a.. \} \cap \{..b\}$  by auto
  also have interior  $\dots = \{a <.. \} \cap \{..<b\}$ 
    by (simp)
  also have  $\dots = \{a <..<b\}$  by auto
  finally show ?thesis .
qed

```

```

lemma interior_atLeastLessThan [simp]:
  fixes  $a::\text{real}$  shows interior  $\{a..<b\} = \{a <..<b\}$ 
  by (metis atLeastLessThan_def greaterThanLessThan_def interior_atLeastAtMost_real
    interior_Int interior_interior interior_real_atLeast)

```

```

lemma interior_lessThanAtMost [simp]:
  fixes  $a::\text{real}$  shows interior  $\{a <..b\} = \{a <..<b\}$ 
  by (metis atLeastAtMost_def greaterThanAtMost_def interior_atLeastAtMost_real
    interior_Int
    interior_interior interior_real_atLeast)

```

```

lemma interior_greaterThanLessThan_real [simp]: interior  $\{a <..<b\} = \{a <..<b :: \text{real}\}$ 
  by (metis interior_atLeastAtMost_real interior_interior)

```

```

lemma frontier_real_atMost [simp]:
  fixes  $a :: \text{real}$ 
  shows frontier  $\{..a\} = \{a\}$ 
  unfolding frontier_def by auto

```

```

lemma frontier_real_atLeast [simp]: frontier  $\{a.. \} = \{a::\text{real}\}$ 
  by (auto simp: frontier_def)

```

```

lemma frontier_real_greaterThan [simp]: frontier  $\{a <.. \} = \{a::\text{real}\}$ 
  by (auto simp: interior_open frontier_def)

```

```

lemma frontier_real_lessThan [simp]: frontier  $\{..<a\} = \{a::\text{real}\}$ 
  by (auto simp: interior_open frontier_def)

```

```

lemma rel_interior_real_box [simp]:
  fixes  $a \ b :: \text{real}$ 
  assumes  $a < b$ 

```

```

  shows  $rel\_interior \{a .. b\} = \{a <..< b\}$ 
proof -
  have  $box\ a\ b \neq \{\}$ 
    using assms
    unfolding set_eq_iff
    by (auto intro!: exI[of  $\_ (a + b) / 2$ ] simp: box_def)
  then show ?thesis
    using interior_rel_interior_gen[of cbox a b, symmetric]
    by (simp split: if_split_asm del: box_real add: box_real[symmetric])
qed

```

```

lemma rel_interior_real_semiline [simp]:
  fixes  $a :: real$ 
  shows  $rel\_interior \{a.. \} = \{a<.. \}$ 
proof -
  have  $\ast: \{a<.. \} \neq \{\}$ 
    unfolding set_eq_iff by (auto intro!: exI[of  $\_ a + 1$ ])
  then show ?thesis using interior_real_atLeast interior_rel_interior_gen[of
 $\{a.. \}$ ]
    by (auto split: if_split_asm)
qed

```

Relative open sets

definition $rel\_open\ S \longleftrightarrow rel\_interior\ S = S$

lemma *rel_open*: $rel\_open\ S \longleftrightarrow openin\ (top\_of\_set\ (affine\ hull\ S))\ S$ (*is ?lhs = ?rhs*)

```

proof
  assume ?lhs
  then show ?rhs
    unfolding rel_open_def rel_interior_def
    using openin_subopen[of top_of_set (affine hull S) S] by auto
qed (auto simp: rel_open_def rel_interior_def)

```

lemma *openin_rel_interior*: $openin\ (top\_of\_set\ (affine\ hull\ S))\ (rel\_interior\ S)$
 using *openin_subopen* by (*fastforce simp add: rel_interior_def*)

lemma *openin_set_rel_interior*:
 $openin\ (top\_of\_set\ S)\ (rel\_interior\ S)$
 by (*rule openin_subset_trans [OF openin_rel_interior rel_interior_subset hull_subset]*)

lemma *affine_rel_open*:
 fixes $S :: 'n::euclidean\_space\ set$
 assumes *affine S*
 shows $rel\_open\ S$
 unfolding *rel_open_def*
 using *assms rel_interior_affine_hull*[of *S*] *affine_hull_eq*[of *S*]
 by *metis*


```

lemma affine_closed:
  fixes S :: 'n::euclidean_space set
  assumes affine S
  shows closed S
proof -
  {
    assume S ≠ {}
    then obtain L where L: subspace L affine_parallel S L
      using assms affine_parallel_subspace[of S] by auto
    then obtain a where a: S = ((+) a ' L)
      using affine_parallel_def[of L S] affine_parallel_commut by auto
    from L have closed L using closed_subspace by auto
    then have closed S
      using closed_translation a by auto
  }
  then show ?thesis by auto
qed

lemma closure_affine_hull:
  fixes S :: 'n::euclidean_space set
  shows closure S ⊆ affine hull S
  by (intro closure_minimal hull_subset affine_closed affine_affine_hull)

lemma closed_affine_hull [iff]:
  fixes S :: 'n::euclidean_space set
  shows closed (affine hull S)
  by (metis affine_affine_hull affine_closed)

lemma closure_same_affine_hull [simp]:
  fixes S :: 'n::euclidean_space set
  shows affine hull (closure S) = affine hull S
proof -
  have affine hull (closure S) ⊆ affine hull S
    using hull_mono[of closure S affine hull S affine]
      closure_affine_hull[of S] hull_hull[of affine S]
    by auto
  moreover have affine hull (closure S) ⊇ affine hull S
    using hull_mono[of S closure S affine] closure_subset by auto
  ultimately show ?thesis by auto
qed

lemma closure_aff_dim [simp]:
  fixes S :: 'n::euclidean_space set
  shows aff_dim (closure S) = aff_dim S
proof -
  have aff_dim S ≤ aff_dim (closure S)
    using aff_dim_subset closure_subset by auto
  moreover have aff_dim (closure S) ≤ aff_dim (affine hull S)

```

```

    using aff_dim_subset_closure_affine_hull by blast
    moreover have aff_dim (affine hull S) = aff_dim S
    using aff_dim_affine_hull by auto
    ultimately show ?thesis by auto
qed

lemma rel_interior_closure_convex_shrink:
  fixes S :: ::euclidean_space set
  assumes convex S
    and c ∈ rel_interior S
    and x ∈ closure S
    and e > 0
    and e ≤ 1
  shows x - e *R (x - c) ∈ rel_interior S
proof -
  obtain d where d > 0 and d: ball c d ∩ affine hull S ⊆ S
    using assms(2) unfolding mem_rel_interior_ball by auto
  have ∃ y ∈ S. norm (y - x) * (1 - e) < e * d
  proof (cases x ∈ S)
    case True
    then show ?thesis using ⟨e > 0⟩ ⟨d > 0⟩ by force
  next
    case False
    then have x: x islimpt S
    using assms(3)[unfolded closure_def] by auto
    show ?thesis
    proof (cases e = 1)
      case True
      obtain y where y ∈ S y ≠ x dist y x < 1
        using x[unfolded islimpt_approachable, THEN spec[where x=1]] by auto
      then show ?thesis
        unfolding True using ⟨d > 0⟩ by (force simp add: )
    next
      case False
      then have 0 < e * d / (1 - e) and *: 1 - e > 0
        using ⟨e ≤ 1⟩ ⟨e > 0⟩ ⟨d > 0⟩ by auto
      then obtain y where y ∈ S y ≠ x dist y x < e * d / (1 - e)
        using x[unfolded islimpt_approachable, THEN spec[where x=e*d / (1 -
e)]] by auto
      then show ?thesis
        unfolding dist_norm using pos_less_divide_eq[OF *] by force
    qed
  qed
  then obtain y where y ∈ S and y: norm (y - x) * (1 - e) < e * d
    by auto
  define z where z = c + ((1 - e) / e) *R (x - y)
  have *: x - e *R (x - c) = y - e *R (y - z)
  unfolding z_def using ⟨e > 0⟩
  by (auto simp: scaleR_right_diff_distrib scaleR_right_distrib scaleR_left_diff_distrib)

```

```

have zball:  $z \in \text{ball } c \ d$ 
  using mem_ball z_def dist_norm[of c]
  using y and assms(4,5)
  by (simp add: norm_minus_commute) (simp add: field_simps)
have  $x \in \text{affine hull } S$ 
  using closure_affine_hull assms by auto
moreover have  $y \in \text{affine hull } S$ 
  using  $\langle y \in S \rangle$  hull_subset[of S] by auto
moreover have  $c \in \text{affine hull } S$ 
  using assms rel_interior_subset hull_subset[of S] by auto
ultimately have  $z \in \text{affine hull } S$ 
  using z_def affine_affine_hull[of S]
  mem_affine_3_minus [of affine hull S c x y (1 - e) / e]
  assms
  by simp
then have  $z \in S$  using d zball by auto
obtain d1 where  $d1 > 0$  and  $d1: \text{ball } z \ d1 \leq \text{ball } c \ d$ 
  using zball open_ball[of c d] openE[of ball c d z] by auto
then have  $\text{ball } z \ d1 \cap \text{affine hull } S \subseteq \text{ball } c \ d \cap \text{affine hull } S$ 
  by auto
then have  $\text{ball } z \ d1 \cap \text{affine hull } S \subseteq S$ 
  using d by auto
then have  $z \in \text{rel\_interior } S$ 
  using mem_rel_interior_ball using  $\langle d1 > 0 \rangle \langle z \in S \rangle$  by auto
then have  $y - e *_{\mathbb{R}} (y - z) \in \text{rel\_interior } S$ 
  using rel_interior_convex_shrink[of S z y e] assms  $\langle y \in S \rangle$  by auto
then show ?thesis using * by auto
qed

```

```

lemma rel_interior_eq:
   $\text{rel\_interior } s = s \longleftrightarrow \text{openin}(\text{top\_of\_set } (\text{affine hull } s)) \ s$ 
  using rel_open rel_open_def by blast

```

```

lemma rel_interior_openin:
   $\text{openin}(\text{top\_of\_set } (\text{affine hull } s)) \ s \implies \text{rel\_interior } s = s$ 
  by (simp add: rel_interior_eq)

```

```

lemma rel_interior_affine:
  fixes  $S :: 'n::\text{euclidean\_space}$  set
  shows  $\text{affine } S \implies \text{rel\_interior } S = S$ 
  using affine_rel_open rel_open_def by auto

```

```

lemma rel_interior_eq_closure:
  fixes  $S :: 'n::\text{euclidean\_space}$  set
  shows  $\text{rel\_interior } S = \text{closure } S \longleftrightarrow \text{affine } S$ 
proof (cases  $S = \{\}$ )
  case True
  then show ?thesis
  by auto

```

```

next
  case False show ?thesis
  proof
    assume eq: rel_interior S = closure S
    have openin (top_of_set (affine hull S)) S
      by (metis eq closure_subset openin_rel_interior rel_interior_subset subset_antisym)
    moreover have closedin (top_of_set (affine hull S)) S
      by (metis closed_subset closure_subset_eq eq hull_subset rel_interior_subset)
    ultimately have S = {} ∨ S = affine hull S
      using convex_connected connected_clopen convex_affine_hull by metis
    with False have affine hull S = S
      by auto
    then show affine S
      by (metis affine_hull_eq)
  next
    assume affine S
    then show rel_interior S = closure S
      by (simp add: rel_interior_affine affine_closed)
  qed
qed

```

Relative interior preserves under linear transformations

```

lemma rel_interior_translation_aux:
  fixes a :: 'n::euclidean_space
  shows ((λx. a + x) ' rel_interior S) ⊆ rel_interior ((λx. a + x) ' S)
proof -
  {
    fix x
    assume x: x ∈ rel_interior S
    then obtain T where open T x ∈ T ∩ S T ∩ affine hull S ⊆ S
      using mem_rel_interior[of x S] by auto
    then have open ((λx. a + x) ' T)
      and a + x ∈ ((λx. a + x) ' T) ∩ ((λx. a + x) ' S)
      and ((λx. a + x) ' T) ∩ affine hull ((λx. a + x) ' S) ⊆ (λx. a + x) ' S
      using affine_hull_translation[of a S] open_translation[of T a] x by auto
    then have a + x ∈ rel_interior ((λx. a + x) ' S)
      using mem_rel_interior[of a+x ((λx. a + x) ' S)] by auto
  }
  then show ?thesis by auto
qed

```

```

lemma rel_interior_translation:
  fixes a :: 'n::euclidean_space
  shows rel_interior ((λx. a + x) ' S) = (λx. a + x) ' rel_interior S
proof -
  have (λx. (-a) + x) ' rel_interior ((λx. a + x) ' S) ⊆ rel_interior S
    using rel_interior_translation_aux[of -a (λx. a + x) ' S]

```

```

      translation_assoc[of  $-a$   $a$ ]
    by auto
  then have  $((\lambda x. a + x) ' rel\_interior\ S) \supseteq rel\_interior\ ((\lambda x. a + x) ' S)$ 
    using translation_inverse_subset[of  $a$   $rel\_interior\ ((+) a ' S) rel\_interior\ S$ ]
    by auto
  then show ?thesis
    using rel_interior_translation_aux[of  $a$   $S$ ] by auto
qed

```

```

lemma affine_hull_linear_image:
  assumes bounded_linear f
  shows  $f ' (affine\ hull\ s) = affine\ hull\ f ' s$ 
proof -
  interpret f: bounded_linear f by fact
  have affine  $\{x. f\ x \in affine\ hull\ f ' s\}$ 
    unfolding affine_def
    by (auto simp: f.scaleR f.add affine_affine_hull[unfolded affine_def, rule_format])
  moreover have affine  $\{x. x \in f ' (affine\ hull\ s)\}$ 
    using affine_affine_hull[unfolded affine_def, of s]
    unfolding affine_def by (auto simp: f.scaleR [symmetric] f.add [symmetric])
  ultimately show ?thesis
    by (auto simp: hull_inc elim!: hull_induct)
qed

```

```

lemma rel_interior_injective_on_span_linear_image:
  fixes  $f :: 'm::euclidean\_space \Rightarrow 'n::euclidean\_space$ 
  and  $S :: 'm::euclidean\_space\ set$ 
  assumes bounded_linear f
  and inj_on f (span S)
  shows  $rel\_interior\ (f ' S) = f ' (rel\_interior\ S)$ 
proof -
  {
    fix z
    assume  $z: z \in rel\_interior\ (f ' S)$ 
    then have  $z \in f ' S$ 
      using rel_interior_subset[of  $f ' S$ ] by auto
    then obtain x where  $x: x \in S\ f\ x = z$  by auto
    obtain e2 where  $e2: e2 > 0\ cball\ z\ e2 \cap affine\ hull\ (f ' S) \subseteq (f ' S)$ 
      using z rel_interior_cball[of  $f ' S$ ] by auto
    obtain K where  $K: K > 0 \wedge x. norm\ (f\ x) \leq norm\ x * K$ 
      using assms Real_Vector_Spaces.bounded_linear.pos_bounded[of f] by auto
    define e1 where  $e1 = 1 / K$ 
    then have  $e1: e1 > 0 \wedge x. e1 * norm\ (f\ x) \leq norm\ x$ 
      using K pos_le_divide_eq[of e1] by auto
    define e where  $e = e1 * e2$ 
    then have  $e > 0$  using e1 e2 by auto
  }

```

```

fix y
assume y: y ∈ cball x e ∩ affine hull S
then have h1: f y ∈ affine hull (f ` S)
  using affine_hull_linear_image[of f S] assms by auto
from y have norm (x-y) ≤ e1 * e2
  using cball_def[of x e] dist_norm[of x y] e_def by auto
moreover have f x - f y = f (x - y)
  using assms linear_diff[of f x y] linear_conv_bounded_linear[of f] by auto
moreover have e1 * norm (f (x-y)) ≤ norm (x - y)
  using e1 by auto
ultimately have e1 * norm ((f x)-(f y)) ≤ e1 * e2
  by auto
then have f y ∈ cball z e2
  using cball_def[of f x e2] dist_norm[of f x f y] e1 x by auto
then have f y ∈ f ` S
  using y e2 h1 by auto
then have y ∈ S
  using assms y hull_subset[of S] affine_hull_subset_span
    inj_on_image_mem_iff [OF ⟨inj_on f (span S)⟩]
  by (metis Int_iff span_superset subsetCE)
}
then have z ∈ f ` (rel_interior S)
  using mem_rel_interior_cball[of x S] ⟨e > 0⟩ x by auto
}
moreover
{
  fix x
  assume x: x ∈ rel_interior S
  then obtain e2 where e2: e2 > 0 cball x e2 ∩ affine hull S ⊆ S
    using rel_interior_cball[of S] by auto
  have x ∈ S using x rel_interior_subset by auto
  then have *: f x ∈ f ` S by auto
  have ∀ x ∈ span S. f x = 0 ⟶ x = 0
    using assms subspace_span linear_conv_bounded_linear[of f]
      linear_injective_on_subspace_0[of f span S]
    by auto
  then obtain e1 where e1: e1 > 0 ∀ x ∈ span S. e1 * norm x ≤ norm (f x)
    using assms injective_imp_isometric[of span S f]
      subspace_span[of S] closed_subspace[of span S]
    by auto
  define e where e = e1 * e2
  hence e > 0 using e1 e2 by auto
  {
    fix y
    assume y: y ∈ cball (f x) e ∩ affine hull (f ` S)
    then have y ∈ f ` (affine hull S)
      using affine_hull_linear_image[of f S] assms by auto
    then obtain xy where xy: xy ∈ affine hull S f xy = y by auto
    with y have norm (f x - f xy) ≤ e1 * e2

```

```

    using cball_def[of f x e] dist_norm[of f x y] e_def by auto
  moreover have  $f\ x - f\ xy = f\ (x - xy)$ 
    using assms linear_diff[of f x xy] linear_conv_bounded_linear[of f] by auto
  moreover have *:  $x - xy \in \text{span } S$ 
    using subspace_diff[of span S x xy] subspace_span  $\langle x \in S \rangle xy$ 
    affine_hull_subset_span[of S] span_superset
    by auto
  moreover from * have  $e1 * \text{norm } (x - xy) \leq \text{norm } (f\ (x - xy))$ 
    using e1 by auto
  ultimately have  $e1 * \text{norm } (x - xy) \leq e1 * e2$ 
    by auto
  then have  $xy \in \text{cball } x\ e2$ 
    using cball_def[of x e2] dist_norm[of x xy] e1 by auto
  then have  $y \in f\ 'S$ 
    using xy e2 by auto
}
then have  $f\ x \in \text{rel\_interior } (f\ 'S)$ 
  using mem_rel_interior_cball[of (f x) (f 'S)] *  $\langle e > 0 \rangle$  by auto
}
ultimately show ?thesis by auto
qed

```

```

lemma rel_interior_injective_linear_image:
  fixes f :: 'm::euclidean_space  $\Rightarrow$  'n::euclidean_space
  assumes bounded_linear f
    and inj f
  shows  $\text{rel\_interior } (f\ 'S) = f\ '(\text{rel\_interior } S)$ 
  using assms rel_interior_injective_on_span_linear_image[of f S]
    subset_inj_on[of f UNIV span S]
  by auto

```

4.2.3 Openness and compactness are preserved by convex hull operation

```

lemma open_convex_hull[intro]:
  fixes S :: 'a::real_normed_vector set
  assumes open S
  shows open (convex_hull S)
proof (clarsimp simp: open_contains_cball convex_hull_explicit)
  fix T and u :: 'a $\Rightarrow$ real
  assume obt:  $\text{finite } T\ T \subseteq S\ \forall x \in T. 0 \leq u\ x\ \text{sum } u\ T = 1$ 

  from assms[unfolded open_contains_cball] obtain b
    where b:  $\bigwedge x. x \in S \implies 0 < b\ x \wedge \text{cball } x\ (b\ x) \subseteq S$  by metis
  have b 'T  $\neq \{\}$ 
    using obt by auto
  define i where  $i = b\ 'T$ 
  let ? $\Phi = \lambda y. \exists F. \text{finite } F \wedge F \subseteq S \wedge (\exists u. (\forall x \in F. 0 \leq u\ x) \wedge \text{sum } u\ F = 1 \wedge$ 
     $(\sum_{v \in F} u\ v *_R v) = y)$ 

```

```

let ?a =  $\sum_{v \in T} u \ v \ *_R \ v$ 
show  $\exists e > 0. \text{ cball } ?a \ e \subseteq \{y. ?\Phi \ y\}$ 
proof (intro exI subsetI conjI)
  show  $0 < \text{Min } i$ 
    unfolding i_def and Min_gr_iff[OF finite_imageI[OF obt(1)]]  $\langle b \ ' \ T \neq \{\} \rangle$ 
    using b  $\langle T \subseteq S \rangle$  by auto
next
fix y
assume  $y \in \text{cball } ?a \ (\text{Min } i)$ 
then have  $y: \text{norm } (?a - y) \leq \text{Min } i$ 
  unfolding dist_norm[symmetric] by auto
{ fix x
  assume  $x \in T$ 
  then have  $\text{Min } i \leq b \ x$ 
    by (simp add: i_def obt(1))
  then have  $x + (y - ?a) \in \text{cball } x \ (b \ x)$ 
    using y unfolding mem_cball dist_norm by auto
  moreover have  $x \in S$ 
    using  $\langle x \in T \rangle \langle T \subseteq S \rangle$  by auto
  ultimately have  $x + (y - ?a) \in S$ 
    using y b by blast
}
moreover
have *: inj_on  $(\lambda v. v + (y - ?a)) \ T$ 
  unfolding inj_on_def by auto
have  $(\sum_{v \in (\lambda v. v + (y - ?a)) \ ' \ T} u \ (v - (y - ?a)) \ *_R \ v) = y$ 
  unfolding sum.reindex[OF *] o_def using obt(4)
by (simp add: sum.distrib sum_subtractf scaleR_left.sum[symmetric] scaleR_right_distrib)
ultimately show  $y \in \{y. ?\Phi \ y\}$ 
proof (intro CollectI exI conjI)
  show finite  $((\lambda v. v + (y - ?a)) \ ' \ T)$ 
    by (simp add: obt(1))
  show  $\text{sum } (\lambda v. u \ (v - (y - ?a))) \ ((\lambda v. v + (y - ?a)) \ ' \ T) = 1$ 
    unfolding sum.reindex[OF *] o_def using obt(4) by auto
qed (use obt(1, 3) in auto)
qed
qed

lemma compact_convex_combinations:
  fixes  $S \ T :: 'a::\text{real\_normed\_vector\_set}$ 
  assumes compact S compact T
  shows compact  $\{ (1 - u) \ *_R \ x + u \ *_R \ y \mid x \ y \ u. \ 0 \leq u \wedge u \leq 1 \wedge x \in S \wedge y \in T \}$ 
proof -
  let ?X =  $\{0..1\} \times S \times T$ 
  let ?h =  $(\lambda z. (1 - \text{fst } z) \ *_R \ \text{fst } (\text{snd } z) + \text{fst } z \ *_R \ \text{snd } (\text{snd } z))$ 
  have *:  $\{ (1 - u) \ *_R \ x + u \ *_R \ y \mid x \ y \ u. \ 0 \leq u \wedge u \leq 1 \wedge x \in S \wedge y \in T \} =$ 
    ?h  $\ ' \ ?X$ 
    by force

```



```

have continuous_on ?X ( $\lambda z. (1 - \text{fst } z) *_R \text{fst } (\text{snd } z) + \text{fst } z *_R \text{snd } (\text{snd } z)$ )
  unfolding continuous_on by (rule ballI) (intro tendsto_intros)
with assms show ?thesis
  by (simp add: * compact_Times compact_continuous_image)
qed

```

```

lemma finite_imp_compact_convex_hull:
  fixes S :: 'a::real_normed_vector set
  assumes finite S
  shows compact (convex hull S)
proof (cases S = {})
  case True
  then show ?thesis by simp
next
  case False
  with assms show ?thesis
  proof (induct rule: finite_ne_induct)
    case (singleton x)
    show ?case by simp
  next
    case (insert x A)
    let ?f =  $\lambda(u, y::'a). u *_R x + (1 - u) *_R y$ 
    let ?T =  $\{0..1::\text{real}\} \times (\text{convex hull } A)$ 
    have continuous_on ?T ?f
      unfolding split_def continuous_on by (intro ballI tendsto_intros)
    moreover have compact ?T
      by (intro compact_Times compact_Icc insert)
    ultimately have compact (?f ' ?T)
      by (rule compact_continuous_image)
    also have ?f ' ?T = convex hull (insert x A)
      unfolding convex_hull_insert [OF 'A  $\neq \{\}$ ]
      apply safe
      apply (rule_tac x=a in exI, simp)
      apply (rule_tac x=1 - a in exI, simp, fast)
      apply (rule_tac x=(u, b) in image_eqI, simp_all)
      done
    finally show compact (convex hull (insert x A)) .
  qed
qed

```

```

lemma compact_convex_hull:
  fixes S :: 'a::euclidean_space set
  assumes compact S
  shows compact (convex hull S)
proof (cases S = {})
  case True
  then show ?thesis using compact_empty by simp
next
  case False

```

```

then obtain  $w$  where  $w \in S$  by auto
show ?thesis
  unfolding caratheodory[of  $S$ ]
proof (induct (DIM('a) + 1))
  case 0
  have *:  $\{x. \exists sa. \text{finite } sa \wedge sa \subseteq S \wedge \text{card } sa \leq 0 \wedge x \in \text{convex hull } sa\} = \{\}$ 
    using compact_empty by auto
  from 0 show ?case unfolding * by simp
next
  case (Suc  $n$ )
  show ?case
  proof (cases  $n = 0$ )
    case True
    have  $\{x. \exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq \text{Suc } n \wedge x \in \text{convex hull } T\} = S$ 
      unfolding set_eq_iff and mem_Collect_eq
    proof (rule, rule)
      fix  $x$ 
      assume  $\exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq \text{Suc } n \wedge x \in \text{convex hull } T$ 
      then obtain  $T$  where  $T: \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq \text{Suc } n \wedge x \in \text{convex hull } T$ 
      T
      by auto
      show  $x \in S$ 
      proof (cases  $\text{card } T = 0$ )
        case True
        then show ?thesis
          using  $T(4)$  unfolding card_0_eq[OF  $T(1)$ ] by simp
      next
        case False
        then have  $\text{card } T = \text{Suc } 0$  using  $T(3)$   $\langle n=0 \rangle$  by auto
        then obtain  $a$  where  $T = \{a\}$  unfolding card_Suc_eq by auto
        then show ?thesis using  $T(2,4)$  by simp
      qed
    next
      fix  $x$  assume  $x \in S$ 
      then show  $\exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq \text{Suc } n \wedge x \in \text{convex hull } T$ 
        by (rule_tac  $x=\{x\}$  in exI) (use convex_hull_singleton in auto)
      qed
    then show ?thesis using assms by simp
  next
    case False
    have  $\{x. \exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq \text{Suc } n \wedge x \in \text{convex hull } T\} =$ 
       $\{(1 - u) *_{\mathbb{R}} x + u *_{\mathbb{R}} y \mid x y u. 0 \leq u \wedge u \leq 1 \wedge x \in S \wedge y \in \{x. \exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq n$ 
       $\wedge x \in \text{convex hull } T\}\}$ 
      unfolding set_eq_iff and mem_Collect_eq
    proof (rule, rule)
      fix  $x$ 
      assume  $\exists u v c. x = (1 - c) *_{\mathbb{R}} u + c *_{\mathbb{R}} v \wedge 0 \leq c \wedge c \leq 1 \wedge u \in S \wedge (\exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq n \wedge v \in$ 

```

```

convex hull T)
  then obtain u v c T where obt:  $x = (1 - c) *_R u + c *_R v$ 
     $0 \leq c \wedge c \leq 1$   $u \in S$   $\text{finite } T$   $T \subseteq S$   $\text{card } T \leq n$   $v \in \text{convex hull } T$ 
    by auto
  moreover have  $(1 - c) *_R u + c *_R v \in \text{convex hull insert } u \ T$ 
  by (meson convexD_alt convex_convex_hull hull_inc hull_mono in_mono
insertCI obt(2) obt(7) subset_insertI)
  ultimately show  $\exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq \text{Suc } n \wedge x \in \text{convex hull } T$ 
    by (rule_tac x=insert u T in exI) (auto simp: card_insert_if)
next
  fix x
  assume  $\exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq \text{Suc } n \wedge x \in \text{convex hull } T$ 
  then obtain T where T:  $\text{finite } T$   $T \subseteq S$   $\text{card } T \leq \text{Suc } n$   $x \in \text{convex hull } T$ 
T
    by auto
  show  $\exists u v c. x = (1 - c) *_R u + c *_R v \wedge$ 
     $0 \leq c \wedge c \leq 1 \wedge u \in S \wedge (\exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq n \wedge v \in \text{convex hull } T)$ 
  proof (cases  $\text{card } T = \text{Suc } n$ )
  case False
    then have  $\text{card } T \leq n$  using T(3) by auto
    then show ?thesis
      using  $\langle w \in S \rangle$  and T
      by (rule_tac x=w in exI, rule_tac x=x in exI, rule_tac x=1 in exI)
  auto
next
  case True
    then obtain a u where au:  $T = \text{insert } a \ u$   $a \notin u$ 
      by (metis card_le_Suc_iff order_refl)
    show ?thesis
    proof (cases  $u = \{\}$ )
    case True
      then have  $x = a$  using T(4)[unfolded au] by auto
      show ?thesis unfolding  $\langle x = a \rangle$ 
        using  $T \ \langle n \neq 0 \rangle$  unfolding au
        by (rule_tac x=a in exI, rule_tac x=a in exI, rule_tac x=1 in exI)
    force
  next
  case False
    obtain ux vx b where obt:  $ux \geq 0$   $vx \geq 0$   $ux + vx = 1$ 
       $b \in \text{convex hull } u$   $x = ux *_R a + vx *_R b$ 
      using T(4)[unfolded au convex_hull_insert[OF False]]
      by auto
    have *:  $1 - vx = ux$  using obt(3) by auto
    show ?thesis
      using obt T(1-3) card_insert_disjoint[OF _ au(2)] unfolding au *
      by (rule_tac x=a in exI, rule_tac x=b in exI, rule_tac x=vx in exI)
  force

```

```

      qed
    qed
  qed
  then show ?thesis
    using compact_convex_combinations[OF assms Suc] by simp
  qed
qed
qed

```

4.2.4 Extremal points of a simplex are some vertices

```

lemma dist_increases_online:
  fixes a b d :: 'a::real_inner
  assumes d ≠ 0
  shows dist a (b + d) > dist a b ∨ dist a (b - d) > dist a b
proof (cases inner a d - inner b d > 0)
  case True
  then have 0 < inner d d + (inner a d * 2 - inner b d * 2)
    using assms
    by (intro add_pos_pos) auto
  then show ?thesis
    unfolding dist_norm and norm_eq_sqrt_inner and real_sqrt_less_iff
    by (simp add: algebra_simps inner_commute)
  next
  case False
  then have 0 < inner d d + (inner b d * 2 - inner a d * 2)
    using assms
    by (intro add_pos_nonneg) auto
  then show ?thesis
    unfolding dist_norm and norm_eq_sqrt_inner and real_sqrt_less_iff
    by (simp add: algebra_simps inner_commute)
qed

```

```

lemma norm_increases_online:
  fixes d :: 'a::real_inner
  shows d ≠ 0 ⇒ norm (a + d) > norm a ∨ norm (a - d) > norm a
  using dist_increases_online[of d a 0] unfolding dist_norm by auto

```

```

lemma simplex_furthest_lt:
  fixes S :: 'a::real_inner set
  assumes finite S
  shows ∀ x ∈ convex hull S. x ∉ S ⇒ (∃ y ∈ convex hull S. norm (x - a) <
norm(y - a))
  using assms
proof induct
  fix x S
  assume as: finite S x ∉ S ∀ x ∈ convex hull S. x ∉ S ⇒ (∃ y ∈ convex hull S. norm
(x - a) < norm (y - a))
  show ∀ xa ∈ convex hull insert x S. xa ∉ insert x S ⇒

```

```

  ( $\exists y \in \text{convex hull insert } x \ S. \text{norm } (x a - a) < \text{norm } (y - a)$ )
proof (intro impI ballI, cases  $S = \{\}$ )
  case False
  fix y
  assume y:  $y \in \text{convex hull insert } x \ S \ y \notin \text{insert } x \ S$ 
  obtain u v b where obt:  $u \geq 0 \ v \geq 0 \ u + v = 1 \ b \in \text{convex hull } S \ y = u *_R x +$ 
 $v *_R b$ 
  using y(1)[unfolded convex_hull_insert[OF False]] by auto
  show  $\exists z \in \text{convex hull insert } x \ S. \text{norm } (y - a) < \text{norm } (z - a)$ 
proof (cases  $y \in \text{convex hull } S$ )
  case True
  then obtain z where  $z \in \text{convex hull } S \ \text{norm } (y - a) < \text{norm } (z - a)$ 
  using as(3)[THEN bspec[where  $x=y$ ]] and y(2) by auto
  then show ?thesis
  by (meson hull_mono subsetD subset_insertI)
next
case False
show ?thesis
proof (cases  $u = 0 \vee v = 0$ )
case True
with False show ?thesis
using obt y by auto
next
case False
then obtain w where  $w > 0 \ w < u \ w < v$ 
using field_lbound_gt_zero[of u v] and obt(1,2) by auto
have  $x \neq b$ 
proof
assume  $x = b$ 
then have  $y = b$  unfolding obt(5)
using obt(3) by (auto simp: scaleR_left_distrib[symmetric])
then show False using obt(4) and False
using  $\langle x = b \rangle \ y(2)$  by blast
qed
then have *:  $w *_R (x - b) \neq 0$  using w(1) by auto
show ?thesis
using dist_increases_online[OF *, of a y]
proof (elim disjE)
assume  $\text{dist } a \ y < \text{dist } a \ (y + w *_R (x - b))$ 
then have  $\text{norm } (y - a) < \text{norm } ((u + w) *_R x + (v - w) *_R b - a)$ 
unfolding dist_commute[of a]
unfolding dist_norm obt(5)
by (simp add: algebra_simps)
moreover have  $(u + w) *_R x + (v - w) *_R b \in \text{convex hull insert } x \ S$ 
unfolding convex_hull_insert[OF  $\langle S \neq \{\} \rangle$ ]
proof (intro CollectI conjI exI)
show  $u + w \geq 0 \ v - w \geq 0$ 
using obt(1) w by auto
qed (use obt in auto)

```

```

      ultimately show ?thesis by auto
    next
      assume  $\text{dist } a \ y < \text{dist } a \ (y - w *_R (x - b))$ 
      then have  $\text{norm } (y - a) < \text{norm } ((u - w) *_R x + (v + w) *_R b - a)$ 
        unfolding  $\text{dist\_commute}[of \ a]$ 
        unfolding  $\text{dist\_norm } \text{obt}(5)$ 
        by ( $\text{simp add: algebra\_simps}$ )
      moreover have  $(u - w) *_R x + (v + w) *_R b \in \text{convex hull insert } x \ S$ 
        unfolding  $\text{convex\_hull\_insert}[OF \ \langle S \neq \{\} \rangle]$ 
      proof (intro CollectI conjI exI)
        show  $u - w \geq 0 \ \vee \ v + w \geq 0$ 
          using  $\text{obt}(1) \ w$  by auto
        qed (use obt in auto)
      ultimately show ?thesis by auto
    qed
  qed
qed
qed auto
qed (auto simp: assms)

lemma simplex_furthest_le:
  fixes  $S :: 'a::\text{real\_inner\_set}$ 
  assumes  $\text{finite } S$ 
  and  $S \neq \{\}$ 
  shows  $\exists y \in S. \forall x \in \text{convex hull } S. \text{norm } (x - a) \leq \text{norm } (y - a)$ 
proof -
  have  $\text{convex hull } S \neq \{\}$ 
    using  $\text{hull\_subset}[of \ S \ \text{convex}]$  and  $\text{assms}(2)$  by auto
  then obtain  $x$  where  $x \in \text{convex hull } S \ \wedge \ y \in \text{convex hull } S. \text{norm } (y - a) \leq$ 
 $\text{norm } (x - a)$ 
    using  $\text{distance\_attains\_sup}[OF \ \text{finite\_imp\_compact\_convex\_hull}[OF \ \langle \text{finite } S \rangle], \text{ of } a]$ 
    unfolding  $\text{dist\_commute}[of \ a]$ 
    unfolding  $\text{dist\_norm}$ 
    by auto
  show ?thesis
  proof (cases  $x \in S$ )
    case False
      then obtain  $y$  where  $y \in \text{convex hull } S \ \text{norm } (x - a) < \text{norm } (y - a)$ 
        using  $\text{simplex\_furthest\_lt}[OF \ \text{assms}(1), \text{ THEN } \text{bspec}[\text{where } x=x]]$  and  $x(1)$ 
        by auto
      then show ?thesis
        using  $x(2)[\text{THEN } \text{bspec}[\text{where } x=y]]$  by auto
    next
      case True
        with  $x$  show ?thesis by auto
  qed
qed

```

```

lemma simplex_furthest_le_exists:
  fixes S :: ('a::real_inner) set
  shows finite S  $\implies \forall x \in (\text{convex hull } S). \exists y \in S. \text{norm } (x - a) \leq \text{norm } (y - a)$ 
  using simplex_furthest_le[of S] by (cases S = {}) auto

lemma simplex_extremal_le:
  fixes S :: 'a::real_inner set
  assumes finite S
    and S  $\neq \{\}$ 
  shows  $\exists u \in S. \exists v \in S. \forall x \in \text{convex hull } S. \forall y \in \text{convex hull } S. \text{norm } (x - y) \leq$ 
 $\text{norm } (u - v)$ 
  proof -
    have convex_hull_S  $\neq \{\}$ 
      using hull_subset[of S convex] and assms(2) by auto
    then obtain u v where obt:  $u \in \text{convex hull } S \wedge v \in \text{convex hull } S$ 
       $\wedge \forall x \in \text{convex hull } S. \forall y \in \text{convex hull } S. \text{norm } (x - y) \leq \text{norm } (u - v)$ 
      using compact_sup_maxdistance[OF finite_imp_compact_convex_hull[OF
        assms(1)]]
      by (auto simp: dist_norm)
    then show ?thesis
      proof (cases  $u \notin S \vee v \notin S$ , elim disjE)
        assume u  $\notin S$ 
        then obtain y where  $y \in \text{convex hull } S \wedge \text{norm } (u - v) < \text{norm } (y - v)$ 
          using simplex_furthest_lt[OF assms(1), THEN bspec[where x=u]] and
            obt(1)
          by auto
        then show ?thesis
          using obt(3)[THEN bspec[where x=y], THEN bspec[where x=v]] and obt(2)
          by auto
      next
        assume v  $\notin S$ 
        then obtain y where  $y \in \text{convex hull } S \wedge \text{norm } (v - u) < \text{norm } (y - u)$ 
          using simplex_furthest_lt[OF assms(1), THEN bspec[where x=v]] and
            obt(2)
          by auto
        then show ?thesis
          using obt(3)[THEN bspec[where x=u], THEN bspec[where x=y]] and obt(1)
          by (auto simp: norm_minus_commute)
      qed auto
    qed auto
  qed

lemma simplex_extremal_le_exists:
  fixes S :: 'a::real_inner set
  shows finite S  $\implies x \in \text{convex hull } S \implies y \in \text{convex hull } S \implies$ 
 $\exists u \in S. \exists v \in S. \text{norm } (x - y) \leq \text{norm } (u - v)$ 
  using convex_hull_empty_simplex_extremal_le[of S]
  by (cases S = {}) auto

```

4.2.5 Closest point of a convex set is unique, with a continuous projection

definition *closest_point* :: 'a::{real_inner,heine_borel} set $\Rightarrow$ 'a $\Rightarrow$ 'a
 where *closest_point* *S* *a* = (SOME *x*. *x* $\in$ *S* $\wedge$ ($\forall y \in S$. *dist* *a* *x* $\leq$ *dist* *a* *y*))

lemma *closest_point_exists*:
 assumes *closed* *S*
 and *S* $\neq$ {}
 shows *closest_point_in_set*: *closest_point* *S* *a* $\in$ *S*
 and $\forall y \in S$. *dist* *a* (*closest_point* *S* *a*) $\leq$ *dist* *a* *y*
 unfolding *closest_point_def*
 by (rule_tac someI2_ex, auto intro: distance_attains_inf[OF *assms*(1,2), of *a*])+

lemma *closest_point_le*: *closed* *S* $\implies x \in S \implies \text{dist } a \text{ (closest\_point } S \text{ } a) \leq \text{dist } a \text{ } x$
 using *closest_point_exists*[of *S*] by auto

lemma *closest_point_self*:
 assumes *x* $\in$ *S*
 shows *closest_point* *S* *x* = *x*
 unfolding *closest_point_def*
 by (rule some1_equality, rule ex1I[of _ *x*]) (use *assms* in auto)

lemma *closest_point_refl*: *closed* *S* $\implies S \neq \{\}$ $\implies \text{closest\_point } S \text{ } x = x \longleftrightarrow x \in S$
 using *closest_point_in_set*[of *S* *x*] *closest_point_self*[of *x* *S*]
 by auto

lemma *closer_points_lemma*:
 assumes *inner* *y* *z* > 0
 shows $\exists u > 0. \forall v > 0. v \leq u \longrightarrow \text{norm}(v *_{\mathbb{R}} z - y) < \text{norm } y$
proof –
 have *z*: *inner* *z* *z* > 0
 unfolding *inner_gt_zero_iff* using *assms* by auto
 have *norm* (*v* $*_{\mathbb{R}}$ *z* – *y*) $<$ *norm* *y*
 if $0 < v$ and $v \leq \text{inner } y \text{ } z / \text{inner } z \text{ } z$ for *v*
 unfolding *norm_lt* using *z* *assms* that
 by (simp add: field_simps *inner_diff* *inner_commute* *mult_strict_left_mono*[OF _ $\langle 0 < v \rangle$])
 then show ?thesis
 using *assms* *z*
 by (rule_tac *x* = *inner* *y* *z* / *inner* *z* *z* in exI) auto
 qed

lemma *closer_point_lemma*:
 assumes *inner* (*y* – *x*) (*z* – *x*) > 0
 shows $\exists u > 0. u \leq 1 \wedge \text{dist } (x + u *_{\mathbb{R}} (z - x)) \text{ } y < \text{dist } x \text{ } y$
proof –


```

obtain  $u$  where  $u > 0$ 
  and  $u$ :  $\bigwedge v. \llbracket 0 < v; v \leq u \rrbracket \implies \text{norm } (v *_R (z - x) - (y - x)) < \text{norm } (y - x)$ 
  using closer_points_lemma[OF assms] by auto
show ?thesis
  using  $u$ [of min  $u$  1] and  $\langle u > 0 \rangle$ 
  by (metis diff_diff_add dist_commute dist_norm_less_eq_real_def not_less_u
zero_less_one)
qed

```

```

lemma any_closest_point_dot:
  assumes convex  $S$  closed  $S$   $x \in S$   $y \in S$   $\forall z \in S. \text{dist } a \ x \leq \text{dist } a \ z$ 
  shows  $\text{inner } (a - x) (y - x) \leq 0$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then obtain  $u$  where  $u > 0$   $u \leq 1$   $\text{dist } (x + u *_R (y - x)) \ a < \text{dist } x \ a$ 
    using closer_point_lemma[of  $a \ x \ y$ ] by auto
  let  $?z = (1 - u) *_R x + u *_R y$ 
  have  $?z \in S$ 
    using convexD_alt[OF assms(1,3,4), of  $u$ ] using  $u$  by auto
  then show False
    using assms(5)[THEN bspec[where  $x = ?z$ ]] and  $u(3)$ 
    by (auto simp: dist_commute algebra_simps)
qed

```

```

lemma any_closest_point_unique:
  fixes  $x :: 'a :: \text{real\_inner}$ 
  assumes convex  $S$  closed  $S$   $x \in S$   $y \in S$ 
     $\forall z \in S. \text{dist } a \ x \leq \text{dist } a \ z \ \forall z \in S. \text{dist } a \ y \leq \text{dist } a \ z$ 
  shows  $x = y$ 
  using any_closest_point_dot[OF assms(1-4,5)] and any_closest_point_dot[OF
assms(1-2,4,3,6)]
  unfolding norm_pths(1) and norm_le_square
  by (auto simp: algebra_simps)

```

```

lemma closest_point_unique:
  assumes convex  $S$  closed  $S$   $x \in S$   $\forall z \in S. \text{dist } a \ x \leq \text{dist } a \ z$ 
  shows  $x = \text{closest\_point } S \ a$ 
  using any_closest_point_unique[OF assms(1-3) _ assms(4), of closest_point
S  $a$ ]
  using closest_point_exists[OF assms(2)] and assms(3) by auto

```

```

lemma closest_point_dot:
  assumes convex  $S$  closed  $S$   $x \in S$ 
  shows  $\text{inner } (a - \text{closest\_point } S \ a) (x - \text{closest\_point } S \ a) \leq 0$ 
  using any_closest_point_dot[OF assms(1,2) _ assms(3)]
  by (metis assms(2) assms(3) closest_point_in_set closest_point_le_empty_iff)

```

```

lemma closest_point_lt:
  assumes convex  $S$  closed  $S$   $x \in S$   $x \neq \text{closest\_point } S \ a$ 

```

shows $\text{dist } a (\text{closest\_point } S \ a) < \text{dist } a \ x$
using $\text{closest\_point\_unique}[\text{where } a=a] \ \text{closest\_point\_le}[\text{where } a=a] \ \text{assms}$
by *fastforce*

lemma *setdist_closest_point*:

$\llbracket \text{closed } S; S \neq \{\} \rrbracket \implies \text{setdist } \{a\} \ S = \text{dist } a (\text{closest\_point } S \ a)$
by (*metis* $\text{closest\_point\_exists}(2) \ \text{closest\_point\_in\_set\_emptyE} \ \text{insert\_iff} \ \text{setdist\_unique}$)

lemma *closest_point_lipschitz*:

assumes *convex* S

and $\text{closed } S \ S \neq \{\}$

shows $\text{dist } (\text{closest\_point } S \ x) (\text{closest\_point } S \ y) \leq \text{dist } x \ y$

proof –

have $\text{inner } (x - \text{closest\_point } S \ x) (\text{closest\_point } S \ y - \text{closest\_point } S \ x) \leq 0$

and $\text{inner } (y - \text{closest\_point } S \ y) (\text{closest\_point } S \ x - \text{closest\_point } S \ y) \leq 0$

by (*simp_all* *add*: *assms* $\text{closest\_point\_dot} \ \text{closest\_point\_in\_set}$)

then show *?thesis* **unfolding** *dist_norm* **and** *norm_le*

using $\text{inner\_ge\_zero}[\text{of } (x - \text{closest\_point } S \ x) - (y - \text{closest\_point } S \ y)]$

by (*simp* *add*: $\text{inner\_add} \ \text{inner\_diff} \ \text{inner\_commute}$)

qed

lemma *continuous_at_closest_point*:

assumes *convex* S

and $\text{closed } S$

and $S \neq \{\}$

shows *continuous* $(\text{at } x) (\text{closest\_point } S)$

unfolding *continuous_at_eps_delta*

using $\text{le\_less\_trans}[\text{OF } \text{closest\_point\_lipschitz}[\text{OF } \text{assms}]]$ **by** *auto*

lemma *continuous_on_closest_point*:

assumes *convex* S

and $\text{closed } S$

and $S \neq \{\}$

shows *continuous_on* $t (\text{closest\_point } S)$

by (*metis* *continuous_at_imp_continuous_on* *continuous_at_closest_point* [OF *assms*])

proposition *closest_point_in_rel_interior*:

assumes $\text{closed } S \ S \neq \{\}$ **and** $x: x \in \text{affine hull } S$

shows $\text{closest\_point } S \ x \in \text{rel\_interior } S \longleftrightarrow x \in \text{rel\_interior } S$

proof (*cases* $x \in S$)

case *True*

then show *?thesis*

by (*simp* *add*: $\text{closest\_point\_self}$)

next

case *False*

then have *False* **if** *asm*: $\text{closest\_point } S \ x \in \text{rel\_interior } S$

proof –

```

    obtain e where e > 0 and clox: closest_point S x ∈ S
      and e: cball (closest_point S x) e ∩ affine hull S ⊆ S
    using asm mem_rel_interior_cball by blast
    then have clo_notx: closest_point S x ≠ x
      using ⟨x ∉ S⟩ by auto
    define y where y ≡ closest_point S x -
      (min 1 (e / norm(closest_point S x - x))) *R (closest_point S
x - x)
    have x - y = (1 - min 1 (e / norm (closest_point S x - x))) *R (x -
closest_point S x)
    by (simp add: y_def algebra_simps)
    then have norm (x - y) = abs ((1 - min 1 (e / norm (closest_point S x -
x)))) * norm(x - closest_point S x)
    by simp
    also have ... < norm(x - closest_point S x)
    using clo_notx ⟨e > 0⟩
    by (auto simp: mult_less_cancel_right2 field_split_simps)
    finally have no_less: norm (x - y) < norm (x - closest_point S x) .
    have y ∈ affine hull S
    unfolding y_def
    by (meson affine_affine_hull clox hull_subset mem_affine_3_minus2 subsetD
x)
    moreover have dist (closest_point S x) y ≤ e
    using ⟨e > 0⟩ by (auto simp: y_def min_mult_distrib_right)
    ultimately have y ∈ S
    using subsetD [OF e] by simp
    then have dist x (closest_point S x) ≤ dist x y
    by (simp add: closest_point_le ⟨closed S⟩)
    with no_less show False
    by (simp add: dist_norm)
  qed
  moreover have x ∉ rel_interior S
    using rel_interior_subset False by blast
  ultimately show ?thesis by blast
qed

```

Various point-to-set separating/supporting hyperplane theorems

```

lemma supporting_hyperplane_closed_point:
  fixes z :: 'a::{real_inner,heine_borel}
  assumes convex S
    and closed S
    and S ≠ {}
    and z ∉ S
  shows ∃ a b. ∃ y ∈ S. inner a z < b ∧ inner a y = b ∧ (∀ x ∈ S. inner a x ≥ b)
proof -
  obtain y where y ∈ S and y: ∀ x ∈ S. dist z y ≤ dist z x
    by (metis distance_attains_inf [OF assms(2-3)])
  show ?thesis

```

```

proof (intro exI bexI conjI ballI)
  show  $(y - z) \cdot z < (y - z) \cdot y$ 
    by (metis  $\langle y \in S \rangle$  assms(4) diff_gt_0_iff_gt inner_commute inner_diff_left
inner_gt_zero_iff_right_minus_eq)
  show  $(y - z) \cdot y \leq (y - z) \cdot x$  if  $x \in S$  for  $x$ 
  proof (rule ccontr)
    have *:  $\bigwedge u. 0 \leq u \wedge u \leq 1 \longrightarrow \text{dist } z \ y \leq \text{dist } z \ ((1 - u) *_R y + u *_R x)$ 
      using assms(1)[unfolded convex_alt] and  $y$  and  $\langle x \in S \rangle$  and  $\langle y \in S \rangle$  by auto
    assume  $\neg (y - z) \cdot y \leq (y - z) \cdot x$ 
    then obtain  $v$  where  $v > 0 \ v \leq 1 \ \text{dist } (y + v *_R (x - y)) \ z < \text{dist } y \ z$ 
      using closer_point_lemma[of  $z \ y \ x$ ] by (auto simp: inner_diff)
    then show False
      using *[of  $v$ ] by (auto simp: dist_commute algebra_simps)
  qed
qed (use  $\langle y \in S \rangle$  in auto)
qed

lemma separating_hyperplane_closed_point:
  fixes  $z :: 'a :: \{\text{real\_inner}, \text{heine\_borel}\}$ 
  assumes convex  $S$ 
    and closed  $S$ 
    and  $z \notin S$ 
  shows  $\exists a \ b. \text{inner } a \ z < b \wedge (\forall x \in S. \text{inner } a \ x > b)$ 
proof (cases  $S = \{\}$ )
  case True
    then show ?thesis
      by (simp add: gt_ex)
  next
  case False
    obtain  $y$  where  $y \in S$  and  $y: \bigwedge x. x \in S \implies \text{dist } z \ y \leq \text{dist } z \ x$ 
      by (metis distance_attains_inf[OF assms(2)] False)
    show ?thesis
      proof (intro exI conjI ballI)
        show  $(y - z) \cdot z < \text{inner } (y - z) \ z + (\text{norm } (y - z))^2 / 2$ 
          using  $\langle y \in S \rangle \ \langle z \notin S \rangle$  by auto
        next
          fix  $x$ 
          assume  $x \in S$ 
          have False if *:  $0 < \text{inner } (z - y) \ (x - y)$ 
          proof -
            obtain  $u$  where  $u > 0 \ u \leq 1 \ \text{dist } (y + u *_R (x - y)) \ z < \text{dist } y \ z$ 
              using * closer_point_lemma by blast
            then show False using  $y$ [of  $y + u *_R (x - y)$ ] convexD_alt [OF  $\langle \text{convex } S \rangle$ ]
              using  $\langle x \in S \rangle \ \langle y \in S \rangle$  by (auto simp: dist_commute algebra_simps)
          qed
          moreover have  $0 < (\text{norm } (y - z))^2$ 
            using  $\langle y \in S \rangle \ \langle z \notin S \rangle$  by auto
          then have  $0 < \text{inner } (y - z) \ (y - z)$ 
            unfolding power2_norm_eq_inner by simp

```

```

ultimately show  $(y - z) \cdot z + (\text{norm } (y - z))^2 / 2 < (y - z) \cdot x$ 
  by (force simp: field_simps power2_norm_eq_inner inner_commute inner_diff)
qed
qed

```

```

lemma separating_hyperplane_closed_0:
  assumes convex  $(S :: ('a :: euclidean\_space) \text{ set})$ 
    and closed  $S$ 
    and  $0 \notin S$ 
  shows  $\exists a \ b. a \neq 0 \wedge 0 < b \wedge (\forall x \in S. \text{inner } a \ x > b)$ 
proof (cases  $S = \{\}$ )
  case True
  have  $(\text{SOME } i. i \in \text{Basis}) \neq (0 :: 'a)$ 
    by (metis Basis_zero SOME_Basis)
  then show ?thesis
    using True zero_less_one by blast
  next
  case False
  then show ?thesis
    using False using separating_hyperplane_closed_point[OF assms]
    by (metis all_not_in_conv inner_zero_left inner_zero_right less_eq_real_def not_le)
qed

```

Now set-to-set for closed/compact sets

```

lemma separating_hyperplane_closed_compact:
  fixes  $S :: ('a :: euclidean\_space) \text{ set}$ 
  assumes convex  $S$ 
    and closed  $S$ 
    and convex  $T$ 
    and compact  $T$ 
    and  $T \neq \{\}$ 
    and  $S \cap T = \{\}$ 
  shows  $\exists a \ b. (\forall x \in S. \text{inner } a \ x < b) \wedge (\forall x \in T. \text{inner } a \ x > b)$ 
proof (cases  $S = \{\}$ )
  case True
  obtain  $b$  where  $b: b > 0 \ \forall x \in T. \text{norm } x \leq b$ 
    using compact_imp_bounded[OF assms(4)] unfolding bounded_pos by auto
  obtain  $z :: 'a$  where  $z: \text{norm } z = b + 1$ 
    using vector_choose_size[of  $b + 1$ ] and  $b(1)$  by auto
  then have  $z \notin T$  using  $b(2)$ [THEN bspec[where  $x=z$ ]] by auto
  then obtain  $a \ b$  where  $ab: \text{inner } a \ z < b \ \forall x \in T. b < \text{inner } a \ x$ 
    using separating_hyperplane_closed_point[OF assms(3) compact_imp_closed[OF
  assms(4)], of  $z$ ]
    by auto
  then show ?thesis
    using True by auto

```

```

next
  case False
  then obtain y where  $y \in S$  by auto
  obtain a b where  $0 < b$  and  $\S: \bigwedge x. x \in (\bigcup x \in S. \bigcup y \in T. \{x - y\}) \implies b <$ 
inner a x
  using separating_hyperplane_closed_point[OF convex_differences[OF assms(1,3)],
of 0]
  using closed_compact_differences assms by fastforce
  have ab:  $b + \text{inner } a \ y < \text{inner } a \ x$  if  $x \in S \ y \in T$  for  $x \ y$ 
  using  $\S$  [of  $x - y$ ] that by (auto simp add: inner_diff_right less_diff_eq)
  define k where  $k = (\text{SUP } x \in T. a \cdot x)$ 
  have  $k + b / 2 < a \cdot x$  if  $x \in S$  for  $x$ 
  proof -
    have  $k \leq \text{inner } a \ x - b$ 
    unfolding k_def
    using  $\langle T \neq \{\} \rangle$  ab that by (fastforce intro: cSUP_least)
    then show ?thesis
    using  $\langle 0 < b \rangle$  by auto
  qed
  moreover
  have  $-(k + b / 2) < -a \cdot x$  if  $x \in T$  for  $x$ 
  proof -
    have  $\text{inner } a \ x - b / 2 < k$ 
    unfolding k_def
    proof (subst less_cSUP_iff)
      show  $T \neq \{\}$  by fact
      show bdd_above  $((\cdot) \ a \ ' T)$ 
      using ab[rule_format, of y]  $\langle y \in S \rangle$ 
      by (intro bdd_aboveI2[where  $M = \text{inner } a \ y - b$ ]) (auto simp: field_simps
intro: less_imp_le)
      show  $\exists y \in T. a \cdot x - b / 2 < a \cdot y$ 
      using  $\langle 0 < b \rangle$  that by force
    qed
  qed
  then show ?thesis
  by auto
  qed
  ultimately show ?thesis
  by (metis inner_minus_left neg_less_iff_less)
qed

```

```

lemma separating_hyperplane_compact_closed:
  fixes S :: 'a::euclidean_space set
  assumes convex S
    and compact S
    and  $S \neq \{\}$ 
    and convex T
    and closed T
    and  $S \cap T = \{\}$ 
  shows  $\exists a \ b. (\forall x \in S. \text{inner } a \ x < b) \wedge (\forall x \in T. \text{inner } a \ x > b)$ 

```

```

proof -
  obtain a b where  $(\forall x \in T. \text{inner } a \ x < b) \wedge (\forall x \in S. b < \text{inner } a \ x)$ 
  by (metis disjoint_iff_not_equal separating_hyperplane_closed_compact assms)
  then show ?thesis
  by (metis inner_minus_left neg_less_iff_less)
qed

```

General case without assuming closure and getting non-strict separation

```

lemma separating_hyperplane_set_0:
  assumes convex S  $(0 :: 'a :: euclidean\_space) \notin S$ 
  shows  $\exists a. a \neq 0 \wedge (\forall x \in S. 0 \leq \text{inner } a \ x)$ 
proof -
  let ?k =  $\lambda c. \{x :: 'a. 0 \leq \text{inner } c \ x\}$ 
  have *:  $\text{frontier } (\text{cball } 0 \ 1) \cap \bigcap f \neq \{\}$  if as:  $f \subseteq ?k \wedge S \text{ finite } f$  for f
  proof -
    obtain c where  $c: f = ?k \wedge c \subseteq S \text{ finite } c$ 
    using finite_subset_image[OF as(2,1)] by auto
    then obtain a b where  $ab: a \neq 0 \ 0 < b \ \forall x \in \text{convex hull } c. b < \text{inner } a \ x$ 
    using separating_hyperplane_closed_0[OF convex_convex_hull, of c]
    using finite_imp_compact_convex_hull[OF c(3), THEN compact_imp_closed]
  and assms(2)
    using subset_hull[of convex, OF assms(1), symmetric, of c]
    by force
  have  $\text{norm } (a /_{\mathbb{R}} \text{norm } a) = 1$ 
  by (simp add: ab(1))
  moreover have  $(\forall y \in c. 0 \leq y \cdot (a /_{\mathbb{R}} \text{norm } a))$ 
  using hull_subset[of c convex] ab by (force simp: inner_commute)
  ultimately have  $\exists x. \text{norm } x = 1 \wedge (\forall y \in c. 0 \leq \text{inner } y \ x)$ 
  by blast
  then show  $\text{frontier } (\text{cball } 0 \ 1) \cap \bigcap f \neq \{\}$ 
  unfolding c(1) frontier_cball_sphere_def dist_norm by auto
qed
have  $\text{frontier } (\text{cball } 0 \ 1) \cap (\bigcap (?k \wedge S)) \neq \{\}$ 
by (rule compact_imp_fip) (use * closed_halfspace_ge in auto)
then obtain x where  $\text{norm } x = 1 \ \forall y \in S. x \in ?k \ y$ 
unfolding frontier_cball_dist_norm_sphere_def by auto
then show ?thesis
by (metis inner_commute mem_Collect_eq norm_eq_zero zero_neq_one)
qed

```

```

lemma separating_hyperplane_sets:
  fixes S T ::  $'a :: euclidean\_space \text{ set}$ 
  assumes convex S
  and convex T
  and  $S \neq \{\}$ 
  and  $T \neq \{\}$ 
  and  $S \cap T = \{\}$ 

```

```

shows  $\exists a b. a \neq 0 \wedge (\forall x \in S. \text{inner } a \ x \leq b) \wedge (\forall x \in T. \text{inner } a \ x \geq b)$ 
proof -
  from separating_hyperplane_set_0[OF convex_differences[OF assms(2,1)]]
  obtain a where a  $\neq 0 \ \forall x \in \{x - y \mid x \in T \wedge y \in S\}. 0 \leq \text{inner } a \ x$ 
    using assms(3-5) by force
  then have *:  $\bigwedge x y. x \in T \implies y \in S \implies \text{inner } a \ y \leq \text{inner } a \ x$ 
    by (force simp: inner_diff)
  then have bdd: bdd_above ((( $\cdot$ ) a) 'S)
    using  $\langle T \neq \{\} \rangle$  by (auto intro: bdd_aboveI2[OF *])
  show ?thesis
    using  $\langle a \neq 0 \rangle$ 
    by (intro exI[of _ a] exI[of _ SUP  $x \in S. a \cdot x$ ])
      (auto intro!: cSUP_upper bdd cSUP_least  $\langle a \neq 0 \rangle \langle S \neq \{\} \rangle$  *)
qed

```

4.2.6 More convexity generalities

```

lemma convex_closure [intro,simp]:
  fixes S :: 'a::real_normed_vector set
  assumes convex S
  shows convex (closure S)
  apply (rule convexI)
  unfolding closure_sequential
  apply (elim exE)
  subgoal for x y u v f g
    by (rule_tac x= $\lambda n. u *_R f \ n + v *_R g \ n$  in exI) (force intro: tendsto_intros
dest: convexD [OF assms])
  done

```

```

lemma convex_interior [intro,simp]:
  fixes S :: 'a::real_normed_vector set
  assumes convex S
  shows convex (interior S)
  unfolding convex_alt Ball_def mem_interior
proof clarify
  fix x y u
  assume u:  $0 \leq u \leq 1$ 
  fix e d
  assume ed: ball x e  $\subseteq$  S ball y d  $\subseteq$  S  $0 < d \ 0 < e$ 
  show  $\exists e > 0. \text{ball } ((1 - u) *_R x + u *_R y) \ e \subseteq S$ 
  proof (intro exI conjI subsetI)
    fix z
    assume z:  $z \in \text{ball } ((1 - u) *_R x + u *_R y) \ (\min d \ e)$ 
    have  $(1 - u) *_R (z - u *_R (y - x)) + u *_R (z + (1 - u) *_R (y - x)) \in S$ 
    proof (rule_tac assms[unfolded convex_alt, rule_format])
      show  $z - u *_R (y - x) \in S \ z + (1 - u) *_R (y - x) \in S$ 
      using ed z u by (auto simp add: algebra_simps dist_norm)
    qed (use u in auto)
    then show  $z \in S$ 
  qed

```



```

    using u by (auto simp: algebra_simps)
  qed(use u ed in auto)
qed

```

```

lemma convex_hull_eq_empty[simp]: convex hull  $S = \{\}$   $\longleftrightarrow S = \{\}$ 
  using hull_subset[of  $S$  convex] convex_hull_empty by auto

```

4.2.7 Convex set as intersection of halfspaces

```

lemma convex_halfspace_intersection:
  fixes  $S :: ('a::euclidean\_space) set$ 
  assumes closed  $S$  convex  $S$ 
  shows  $S = \bigcap \{h. S \subseteq h \wedge (\exists a b. h = \{x. \text{inner } a \ x \leq b\})\}$ 
proof -
  { fix  $z$ 
    assume  $\forall T. S \subseteq T \wedge (\exists a b. T = \{x. \text{inner } a \ x \leq b\}) \longrightarrow z \in T \ z \notin S$ 
    then have  $\S: \bigwedge a b. S \subseteq \{x. \text{inner } a \ x \leq b\} \Longrightarrow z \in \{x. \text{inner } a \ x \leq b\}$ 
      by blast
    obtain  $a b$  where  $\text{inner } a \ z < b$  ( $\forall x \in S. \text{inner } a \ x > b$ )
      using  $\langle z \notin S \rangle$  assms separating_hyperplane_closed_point by blast
    then have False
      using  $\S$  [of  $-a -b$ ] by fastforce
  }
  then show ?thesis
    by force
qed

```

4.2.8 Convexity of general and special intervals

```

lemma is_interval_convex:
  fixes  $S :: 'a::euclidean\_space set$ 
  assumes is_interval  $S$ 
  shows convex  $S$ 
proof (rule convexI)
  fix  $x y$  and  $u v :: real$ 
  assume  $x \in S \ y \in S$  and  $uv: 0 \leq u \ 0 \leq v \ u + v = 1$ 
  then have  $*: u = 1 - v \ 1 - v \geq 0$  and  $** : v = 1 - u \ 1 - u \geq 0$ 
    by auto
  {
    fix  $a b$ 
    assume  $\neg b \leq u * a + v * b$ 
    then have  $u * a < (1 - v) * b$ 
      unfolding not_le using  $\langle 0 \leq v \rangle$  by (auto simp: field_simps)
    then have  $a < b$ 
      using  $*(1)$  less_eq_real_def uv(1) by auto
    then have  $a \leq u * a + v * b$ 
      unfolding * using  $\langle 0 \leq v \rangle$ 
      by (auto simp: field_simps intro!: mult_right_mono)
  }
  moreover

```

```

{
  fix a b
  assume  $\neg u * a + v * b \leq a$ 
  then have  $v * b > (1 - u) * a$ 
    unfolding not_le using  $\langle 0 \leq v \rangle$  by (auto simp: field_simps)
  then have  $a < b$ 
    unfolding * using  $\langle 0 \leq v \rangle$ 
    by (rule_tac mult_left_less_imp_less) (auto simp: field_simps)
  then have  $u * a + v * b \leq b$ 
    unfolding **
    using  $**(2)$   $\langle 0 \leq u \rangle$  by (auto simp: algebra_simps mult_right_mono)
}
ultimately show  $u *_R x + v *_R y \in S$ 
  using DIM_positive[where 'a='a]
  by (intro mem_is_intervalI [OF assms  $\langle x \in S \rangle \langle y \in S \rangle$ ]) (auto simp: inner_simps)
qed

```

```

lemma is_interval_connected:
  fixes  $S :: 'a::euclidean\_space$  set
  shows  $is\_interval\ S \implies connected\ S$ 
  using is_interval_convex convex_connected by auto

```

```

lemma convex_box [simp]:  $convex\ (cbox\ a\ b)$   $convex\ (box\ a\ (b::'a::euclidean\_space))$ 
  by (auto simp add: is_interval_convex)

```

A non-singleton connected set is perfect (i.e. has no isolated points).

```

lemma connected_imp_perfect:
  fixes  $a :: 'a::metric\_space$ 
  assumes  $connected\ S$   $a \in S$  and  $S: \bigwedge x. S \neq \{x\}$ 
  shows  $a\ islimpt\ S$ 
proof -
  have  $False$  if  $a \in T$  open  $T \bigwedge y. \llbracket y \in S; y \in T \rrbracket \implies y = a$  for  $T$ 
  proof -
    obtain  $e$  where  $e > 0$  and  $e: cball\ a\ e \subseteq T$ 
    using  $\langle open\ T \rangle \langle a \in T \rangle$  by (auto simp: open_contains_cball)
    have  $openin\ (top\_of\_set\ S)\ \{a\}$ 
    unfolding openin_open using that  $\langle a \in S \rangle$  by blast
    moreover have  $closedin\ (top\_of\_set\ S)\ \{a\}$ 
    by (simp add: assms)
    ultimately show  $False$ 
    using  $\langle connected\ S \rangle connected\_clopen\ S$  by blast
  qed
  then show ?thesis
    unfolding islimpt_def by blast
qed

```

```

lemma islimpt_Ioc [simp]:
  fixes  $a :: real$ 

```

```

assumes  $a < b$ 
shows  $x \text{ islimpt } \{a <..b\} \longleftrightarrow x \in \{a..b\}$  (is ?lhs = ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
  by (metis assms closed_atLeastAtMost closed_limpt closure_greaterThanAtMost
closure_subset islimpt_subset)
next
  assume ?rhs
  then have  $x \in \text{closure } \{a <..<b\}$ 
  using assms closure_greaterThanLessThan by blast
  then show ?lhs
  by (metis (no_types) Diff_empty Diff_insert0 interior_lessThanAtMost interior_limit_point interior_subset islimpt_in_closure islimpt_subset)
qed

```

```

lemma islimpt_Ico [simp]:
  fixes  $a :: \text{real}$ 
  assumes  $a < b$  shows  $x \text{ islimpt } \{a..<b\} \longleftrightarrow x \in \{a..b\}$ 
  by (metis assms closure_atLeastLessThan closure_greaterThanAtMost islimpt_Ioc
limpt_of_closure)

```

```

lemma islimpt_Icc [simp]:
  fixes  $a :: \text{real}$ 
  assumes  $a < b$  shows  $x \text{ islimpt } \{a..b\} \longleftrightarrow x \in \{a..b\}$ 
  by (metis assms closure_atLeastLessThan islimpt_Ico limpt_of_closure)

```

```

lemma connected_imp_perfect_aff_dim:
   $\llbracket \text{connected } S; \text{aff\_dim } S \neq 0; a \in S \rrbracket \implies a \text{ islimpt } S$ 
  using aff_dimSing connected_imp_perfect by blast

```

4.2.9 On real, is_interval, convex and connected are all equivalent

```

lemma mem_is_interval_1_I:
  fixes  $a \ b \ c :: \text{real}$ 
  assumes is_interval  $S$ 
  assumes  $a \in S \ c \in S$ 
  assumes  $a \leq b \leq c$ 
  shows  $b \in S$ 
  using assms is_interval_1 by blast

```

```

lemma is_interval_connected_1:
  fixes  $S :: \text{real set}$ 
  shows is_interval  $S \longleftrightarrow \text{connected } S$ 
  by (meson connected_iff_interval is_interval_1)

```

```

lemma is_interval_convex_1:
  fixes  $S :: \text{real set}$ 
  shows is_interval  $S \longleftrightarrow \text{convex } S$ 

```

by (metis is_interval_convex convex_connected is_interval_connected_1)

lemma *connected_compact_interval_1*:
 $connected\ S \wedge compact\ S \longleftrightarrow (\exists a\ b. S = \{a..b::real\})$
 by (auto simp: is_interval_connected_1 [symmetric] is_interval_compact)

lemma *connected_convex_1*:
 fixes $S :: real\ set$
 shows $connected\ S \longleftrightarrow convex\ S$
 by (metis is_interval_convex convex_connected is_interval_connected_1)

lemma *connected_convex_1_gen*:
 fixes $S :: 'a :: euclidean\_space\ set$
 assumes $DIM('a) = 1$
 shows $connected\ S \longleftrightarrow convex\ S$
proof –
 obtain $f :: 'a \Rightarrow real$ **where** $linf$: *linear f* **and** $inj\ f$
 using *subspace_isomorphism[OF subspace_UNIV subspace_UNIV,*
 where $'a = 'a$ **and** $'b = real$
 unfolding *Euclidean_Space.dim_UNIV*
 by (auto simp: assms)
then have $f - ' (f\ ' S) = S$
 by (simp add: inj_vimage_image_eq)
then show ?thesis
 by (metis connected_convex_1 convex_linear_vimage linf convex_connected
 connected_linear_image)
qed

lemma [*simp*]:
 fixes $r\ s :: real$
 shows $is\_interval\ io: is\_interval\ \{..<r\}$
 and $is\_interval\ oi: is\_interval\ \{r<.. \}$
 and $is_interval\ oo: is_interval\ \{r<..
 and $is\_interval\ oc: is\_interval\ \{r<..s\}$
 and $is_interval\ co: is_interval\ \{r..
 by (simp_all add: is_interval_convex_1)$$

4.2.10 Another intermediate value theorem formulation

lemma *ivt_increasing_component_on_1*:
 fixes $f :: real \Rightarrow 'a :: euclidean\_space$
 assumes $a \leq b$
 and *continuous_on* $\{a..b\}\ f$
 and $(f\ a) \cdot k \leq y \leq (f\ b) \cdot k$
 shows $\exists x \in \{a..b\}. (f\ x) \cdot k = y$
proof –
 have $f\ a \in f\ ' cbox\ a\ b$ $f\ b \in f\ ' cbox\ a\ b$
 using $\langle a \leq b \rangle$ **by** auto
then show ?thesis

```

    using connected_ivt_component[of f 'cbox a b f a f b k y]
    by (simp add: connected_continuous_image assms)
qed

```

```

lemma ivt_increasing_component_1:
  fixes f :: real  $\Rightarrow$  'a::euclidean_space
  shows  $a \leq b \implies \forall x \in \{a..b\}. \text{continuous } (at x) f \implies$ 
     $f a \cdot k \leq y \implies y \leq f b \cdot k \implies \exists x \in \{a..b\}. (f x) \cdot k = y$ 
  by (rule ivt_increasing_component_on_1) (auto simp: continuous_at_imp_continuous_on)

```

```

lemma ivt_decreasing_component_on_1:
  fixes f :: real  $\Rightarrow$  'a::euclidean_space
  assumes  $a \leq b$ 
    and continuous_on {a..b} f
    and  $(f b) \cdot k \leq y$ 
    and  $y \leq (f a) \cdot k$ 
  shows  $\exists x \in \{a..b\}. (f x) \cdot k = y$ 
  using ivt_increasing_component_on_1[of a b  $\lambda x. - f x k - y$ ] neg_equal_iff_equal
  using assms continuous_on_minus by force

```

```

lemma ivt_decreasing_component_1:
  fixes f :: real  $\Rightarrow$  'a::euclidean_space
  shows  $a \leq b \implies \forall x \in \{a..b\}. \text{continuous } (at x) f \implies$ 
     $f b \cdot k \leq y \implies y \leq f a \cdot k \implies \exists x \in \{a..b\}. (f x) \cdot k = y$ 
  by (rule ivt_decreasing_component_on_1) (auto simp: continuous_at_imp_continuous_on)

```

4.2.11 A bound within an interval

```

lemma convex_hull_eq_real_cbox:
  fixes x y :: real assumes  $x \leq y$ 
  shows  $\text{convex hull } \{x, y\} = \text{cbox } x y$ 
proof (rule hull_unique)
  show  $\{x, y\} \subseteq \text{cbox } x y$  using  $\langle x \leq y \rangle$  by auto
  show  $\text{convex } (\text{cbox } x y)$ 
    by (rule convex_box)
next
  fix S assume  $\{x, y\} \subseteq S$  and  $\text{convex } S$ 
  then show  $\text{cbox } x y \subseteq S$ 
    unfolding is_interval_convex_1 [symmetric] is_interval_def Basis_real_def
    by - (clarify, simp (no_asm_use), fast)
qed

```

```

lemma unit_interval_convex_hull:
   $\text{cbox } (0::'a::euclidean_space) \text{ One} = \text{convex hull } \{x. \forall i \in \text{Basis}. (x \cdot i = 0) \vee (x \cdot i = 1)\}$ 
  (is ?int = convex hull ?points)
proof -
  have  $\text{One}[simp]: \bigwedge i. i \in \text{Basis} \implies \text{One} \cdot i = 1$ 
    by (simp add: inner_sum_left sum.If_cases inner_Basis)

```

```

have ?int = {x.  $\forall i \in \text{Basis}. x \cdot i \in \text{cbox } 0 \ 1$ }
  by (auto simp: cbox_def)
also have ... = ( $\sum i \in \text{Basis}. (\lambda x. x *_R i) \text{ ` cbox } 0 \ 1$ )
  by (simp only: box_eq_set_sum_Basis)
also have ... = ( $\sum i \in \text{Basis}. (\lambda x. x *_R i) \text{ ` } (\text{convex hull } \{0, 1\})$ )
  by (simp only: convex_hull_eq_real_cbox_zero_le_one)
also have ... = ( $\sum i \in \text{Basis}. \text{convex hull } ((\lambda x. x *_R i) \text{ ` } \{0, 1\})$ )
  by (simp add: convex_hull_linear_image)
also have ... = convex hull ( $\sum i \in \text{Basis}. (\lambda x. x *_R i) \text{ ` } \{0, 1\}$ )
  by (simp only: convex_hull_set_sum)
also have ... = convex hull {x.  $\forall i \in \text{Basis}. x \cdot i \in \{0, 1\}$ }
  by (simp only: box_eq_set_sum_Basis)
also have convex hull {x.  $\forall i \in \text{Basis}. x \cdot i \in \{0, 1\}$ } = convex hull ?points
  by simp
finally show ?thesis .
qed

```

And this is a finite set of vertices.

```

lemma unit_cube_convex_hull:
  obtains S :: 'a::euclidean_space set
  where finite S and cbox 0 ( $\sum \text{Basis}$ ) = convex hull S
proof
  show finite {x::'a.  $\forall i \in \text{Basis}. x \cdot i = 0 \vee x \cdot i = 1$ }
  proof (rule finite_subset, clarify)
    show finite (( $\lambda S. \sum i \in \text{Basis}. (\text{if } i \in S \text{ then } 1 \text{ else } 0) *_R i$ ) ` Pow Basis)
      using finite_Basis by blast
    fix x :: 'a
    assume x:  $\forall i \in \text{Basis}. x \cdot i = 0 \vee x \cdot i = 1$ 
    show x  $\in (\lambda S. \sum i \in \text{Basis}. (\text{if } i \in S \text{ then } 1 \text{ else } 0) *_R i) \text{ ` Pow Basis}$ 
      apply (rule image_eqI[where x={i. i  $\in$  Basis  $\wedge$  x·i = 1}])
      using x
      by (subst euclidean_eq_iff, auto)
  qed
  show cbox 0 One = convex hull {x.  $\forall i \in \text{Basis}. x \cdot i = 0 \vee x \cdot i = 1$ }
    using unit_interval_convex_hull by blast
qed

```

Hence any cube (could do any nonempty interval).

```

lemma cube_convex_hull:
  assumes d > 0
  obtains S :: 'a::euclidean_space set where
    finite S and cbox (x - ( $\sum i \in \text{Basis}. d *_R i$ )) (x + ( $\sum i \in \text{Basis}. d *_R i$ )) = convex hull S
proof -
  let ?d = ( $\sum i \in \text{Basis}. d *_R i$ )::'a
  have *: cbox (x - ?d) (x + ?d) = ( $\lambda y. x - ?d + (2 *_R y)$ ) ` cbox 0 ( $\sum \text{Basis}$ )
  proof (intro set_eqI iffI)
    fix y
    assume y  $\in$  cbox (x - ?d) (x + ?d)

```

```

then have  $\text{inverse } (2 * d) *_{\mathbb{R}} (y - (x - ?d)) \in \text{cbox } 0 (\sum \text{Basis})$ 
using assms by (simp add: mem_box inner_simps) (simp add: field_simps)
with  $\langle 0 < d \rangle$  show  $y \in (\lambda y. x - \text{sum } ((*_R) d) \text{Basis} + (2 * d) *_{\mathbb{R}} y) \text{ ' cbox } 0 \text{ One}$ 
by (auto intro: image_eqI [where  $x = \text{inverse } (2 * d) *_{\mathbb{R}} (y - (x - ?d))$ ])
next
fix  $y$ 
assume  $y \in (\lambda y. x - ?d + (2 * d) *_{\mathbb{R}} y) \text{ ' cbox } 0 \text{ One}$ 
then obtain  $z$  where  $z: z \in \text{cbox } 0 \text{ One } y = x - ?d + (2*d) *_{\mathbb{R}} z$ 
by auto
then show  $y \in \text{cbox } (x - ?d) (x + ?d)$ 
using  $z$  assms by (auto simp: mem_box inner_simps)
qed
obtain  $S$  where  $\text{finite } S \text{ cbox } 0 (\sum \text{Basis}::'a) = \text{convex hull } S$ 
using unit_cube_convex_hull by auto
then show ?thesis
by (rule_tac that [of  $(\lambda y. x - ?d + (2 * d) *_{\mathbb{R}} y) \text{ ' } S$ ] (auto simp: convex_hull_affinity *))
qed

```

4.2.12 Representation of any interval as a finite convex hull

lemma *image_stretch_interval*:

```

 $(\lambda x. \sum_{k \in \text{Basis}. (m\ k * (x \cdot k)) *_{\mathbb{R}} k) \text{ ' cbox } a (b::'a::\text{euclidean\_space}) =$ 
 $(\text{if } (\text{cbox } a\ b) = \{\} \text{ then } \{\} \text{ else}$ 
 $\text{cbox } (\sum_{k \in \text{Basis}. (\min (m\ k * (a \cdot k)) (m\ k * (b \cdot k))) *_{\mathbb{R}} k::'a)$ 
 $(\sum_{k \in \text{Basis}. (\max (m\ k * (a \cdot k)) (m\ k * (b \cdot k))) *_{\mathbb{R}} k))$ 
proof cases
assume  $*$ :  $\text{cbox } a\ b \neq \{\}$ 
show ?thesis
unfolding box_ne_empty if_not_P [OF  $*$ ]
apply (simp add: cbox_def image_Collect set_eq_iff euclidean_eq_iff [where
 $'a = 'a$ ] ball_conj_distrib [symmetric])
apply (subst choice_Basis_iff [symmetric])
proof (intro allI ball_cong refl)
fix  $x\ i :: 'a$  assume  $i \in \text{Basis}$ 
with  $*$  have  $a\_le\_b: a \cdot i \leq b \cdot i$ 
unfolding box_ne_empty by auto
show  $(\exists xa. x \cdot i = m\ i * xa \wedge a \cdot i \leq xa \wedge xa \leq b \cdot i) \longleftrightarrow$ 
 $\min (m\ i * (a \cdot i)) (m\ i * (b \cdot i)) \leq x \cdot i \wedge x \cdot i \leq \max (m\ i * (a \cdot i)) (m$ 
 $i * (b \cdot i))$ 
proof (cases  $m\ i = 0$ )
case True
with  $a\_le\_b$  show ?thesis by auto
next
case False
then have  $*$ :  $\bigwedge a\ b. a = m\ i * b \longleftrightarrow b = a / m\ i$ 
by (auto simp: field_simps)
from False have

```

```

      min (m i * (a · i)) (m i * (b · i)) = (if 0 < m i then m i * (a · i) else m
i * (b · i))
      max (m i * (a · i)) (m i * (b · i)) = (if 0 < m i then m i * (b · i) else m
i * (a · i))
    using a_le_b by (auto simp: min_def max_def mult_le_cancel_left)
    with False show ?thesis using a_le_b *
    by (simp add: le_divide_eq divide_le_eq) (simp add: ac_simps)
  qed
qed
qed simp

```

```

lemma interval_image_stretch_interval:
  ∃ u v. (λx. ∑ k∈Basis. (m k * (x·k))*R k) ‘ cbox a (b::'a::euclidean_space) =
cbox u (v::'a::euclidean_space)
  unfolding image_stretch_interval by auto

```

```

lemma cbox_translation: cbox (c + a) (c + b) = image (λx. c + x) (cbox a b)
  using image_affinity_cbox [of 1 c a b]
  using box_ne_empty [of a+c b+c] box_ne_empty [of a b]
  by (auto simp: inner_left_distrib add.commute)

```

```

lemma cbox_image_unit_interval:
  fixes a :: 'a::euclidean_space
  assumes cbox a b ≠ {}
  shows cbox a b =
    (+) a ‘ (λx. ∑ k∈Basis. ((b · k - a · k) * (x · k)) *R k) ‘ cbox 0 One
  using assms
  apply (simp add: box_ne_empty image_stretch_interval cbox_translation [symmetric])
  apply (simp add: min_def max_def algebra_simps sum_subtractf euclidean_representation)
  done

```

```

lemma closed_interval_as_convex_hull:
  fixes a :: 'a::euclidean_space
  obtains S where finite S cbox a b = convex hull S
  proof (cases cbox a b = {})
    case True with convex_hull_empty that show ?thesis
      by blast
    next
      case False
      obtain S::'a set where finite S and eq: cbox 0 One = convex hull S
        by (blast intro: unit_cube_convex_hull)
      let ?S = (+) a ‘ (λx. ∑ k∈Basis. ((b · k - a · k) * (x · k)) *R k) ‘ S
      show thesis
      proof
        show finite ?S
          by (simp add: ⟨finite S⟩)
        have lin: linear (λx. ∑ k∈Basis. ((b · k - a · k) * (x · k)) *R k)
          by (rule linear_compose_sum) (auto simp: algebra_simps linearI)
        show cbox a b = convex hull ?S

```



```

    using convex_hull_linear_image [OF lin]
    by (simp add: convex_hull_translation eq cbox_image_unit_interval [OF
False])
qed
qed

```

4.2.13 Bounded convex function on open set is continuous

```

lemma convex_on_bounded_continuous:
  fixes S :: ('a::real_normed_vector) set
  assumes open S
    and convex_on S f
    and  $\forall x \in S. |f x| \leq b$ 
  shows continuous_on S f
proof -
  have  $\exists d > 0. \forall x'. \text{norm } (x' - x) < d \longrightarrow |f x' - f x| < e$  if  $x \in S$   $e > 0$  for  $x$ 
  and  $e :: \text{real}$ 
  proof -
    define B where  $B = |b| + 1$ 
    then have B:  $0 < B \wedge x. x \in S \implies |f x| \leq B$ 
    using assms(3) by auto
    obtain k where  $k > 0$  and  $k$ :  $\text{cball } x \ k \subseteq S$ 
    using  $\langle x \in S \rangle$  assms(1) open_contains_cball_eq by blast
    show  $\exists d > 0. \forall x'. \text{norm } (x' - x) < d \longrightarrow |f x' - f x| < e$ 
    proof (intro exI conjI allI impI)
      fix y
      assume as:  $\text{norm } (y - x) < \min (k / 2) (e / (2 * B) * k)$ 
      show  $|f y - f x| < e$ 
      proof (cases  $y = x$ )
      case False
      define t where  $t = k / \text{norm } (y - x)$ 
      have  $2 < t \wedge 0 < t$ 
      unfolding t_def using as False and  $\langle k > 0 \rangle$ 
      by (auto simp: field_simps)
      have  $y \in S$ 
      apply (rule k[THEN subsetD])
      unfolding mem_cball dist_norm
      apply (rule order_trans[of _  $2 * \text{norm } (x - y)$ ])
      using as
      by (auto simp: field_simps norm_minus_commute)
      {
        define w where  $w = x + t *_R (y - x)$ 
        have  $w \in S$ 
        using  $\langle k > 0 \rangle$  by (auto simp: dist_norm t_def w_def k[THEN subsetD])
        have  $(1 / t) *_R x + - x + ((t - 1) / t) *_R x = (1 / t - 1 + (t - 1) /$ 
 $t) *_R x$ 
        by (auto simp: algebra_simps)
        also have  $\dots = 0$ 
        using  $\langle t > 0 \rangle$  by (auto simp: field_simps)
      }
    end
  end

```

```

finally have  $w: (1 / t) *_{\mathcal{R}} w + ((t - 1) / t) *_{\mathcal{R}} x = y$ 
  unfolding  $w\_def$  using  $False$  and  $\langle t > 0 \rangle$ 
  by (auto simp: algebra_simps)
have  $2: 2 * B < e * t$ 
  unfolding  $t\_def$  using  $\langle 0 < e \rangle \langle 0 < k \rangle \langle B > 0 \rangle$  and  $as$  and  $False$ 
  by (auto simp: field_simps)
have  $f y - f x \leq (f w - f x) / t$ 
  using  $assms(2)[unfolded\ convex\_on\_def, rule\_format, of\ w\ x\ 1/t\ (t -$ 
 $1)/t, unfolded\ w]$ 
  using  $\langle 0 < t \rangle \langle 2 < t \rangle$  and  $\langle x \in S \rangle \langle w \in S \rangle$ 
  by (auto simp: field_simps)
also have  $\dots < e$ 
  using  $B(2)[OF\ \langle w \in S \rangle]$  and  $B(2)[OF\ \langle x \in S \rangle]$   $2\ \langle t > 0 \rangle$  by (auto simp:
 $field\_simps$ )
  finally have  $th1: f y - f x < e$  .
}
moreover
{
  define  $w$  where  $w = x - t *_{\mathcal{R}} (y - x)$ 
  have  $w \in S$ 
  using  $\langle k > 0 \rangle$  by (auto simp: dist_norm t_def w_def k[THEN subsetD])
  have  $(1 / (1 + t)) *_{\mathcal{R}} x + (t / (1 + t)) *_{\mathcal{R}} x = (1 / (1 + t) + t / (1 +$ 
 $t)) *_{\mathcal{R}} x$ 
  by (auto simp: algebra_simps)
  also have  $\dots = x$ 
  using  $\langle t > 0 \rangle$  by (auto simp: field_simps)
  finally have  $w: (1 / (1 + t)) *_{\mathcal{R}} w + (t / (1 + t)) *_{\mathcal{R}} y = x$ 
  unfolding  $w\_def$  using  $False$  and  $\langle t > 0 \rangle$ 
  by (auto simp: algebra_simps)
  have  $2 * B < e * t$ 
  unfolding  $t\_def$ 
  using  $\langle 0 < e \rangle \langle 0 < k \rangle \langle B > 0 \rangle$  and  $as$  and  $False$ 
  by (auto simp: field_simps)
  then have  $*: (f w - f y) / t < e$ 
  using  $B(2)[OF\ \langle w \in S \rangle]$  and  $B(2)[OF\ \langle y \in S \rangle]$ 
  using  $\langle t > 0 \rangle$ 
  by (auto simp: field_simps)
  have  $f x \leq 1 / (1 + t) * f w + (t / (1 + t)) * f y$ 
  using  $assms(2)[unfolded\ convex\_on\_def, rule\_format, of\ w\ y\ 1/(1+t)\ t$ 
 $/\ (1+t), unfolded\ w]$ 
  using  $\langle 0 < t \rangle \langle 2 < t \rangle$  and  $\langle y \in S \rangle \langle w \in S \rangle$ 
  by (auto simp: field_simps)
  also have  $\dots = (f w + t * f y) / (1 + t)$ 
  using  $\langle t > 0 \rangle$  by (simp add: add_divide_distrib)
  also have  $\dots < e + f y$ 
  using  $\langle t > 0 \rangle * \langle e > 0 \rangle$  by (auto simp: field_simps)
  finally have  $f x - f y < e$  by auto
}
ultimately show  $?thesis$  by auto

```

```

      qed (use ‹0 < e› in auto)
    qed (use ‹0 < e› ‹0 < k› ‹0 < B› in ‹auto simp: field_simps›)
  qed
  then show ?thesis
    by (metis continuous_on_iff dist_norm real_norm_def)
qed

```

4.2.14 Upper bound on a ball implies upper and lower bounds

```

lemma convex_bounds_lemma:
  fixes x :: 'a::real_normed_vector
  assumes convex_on (cball x e) f
  and  $\forall y \in \text{cball } x \ e. f \ y \leq b$  and  $y: y \in \text{cball } x \ e$ 
  shows  $|f \ y| \leq b + 2 * |f \ x|$ 
proof (cases  $0 \leq e$ )
case True
  define z where  $z = 2 *_R x - y$ 
  have *:  $x - (2 *_R x - y) = y - x$ 
  by (simp add: scaleR_2)
  have z:  $z \in \text{cball } x \ e$ 
  using y unfolding z_def mem_cball dist_norm * by (auto simp: norm_minus_commute)
  have  $(1 / 2) *_R y + (1 / 2) *_R z = x$ 
  unfolding z_def by (auto simp: algebra_simps)
  then show  $|f \ y| \leq b + 2 * |f \ x|$ 
  using assms(1)[unfolded convex_on_def, rule_format, OF y z, of  $1/2 \ 1/2$ ]
  using assms(2)[rule_format, OF y] assms(2)[rule_format, OF z]
  by (auto simp: field_simps)
next
case False
  have  $\text{dist } x \ y < 0$ 
  using False y unfolding mem_cball not_le by (auto simp del: dist_not_less_zero)
  then show  $|f \ y| \leq b + 2 * |f \ x|$ 
  using zero_le_dist[of x y] by auto
qed

```

Hence a convex function on an open set is continuous

```

lemma real_of_nat_ge_one_iff:  $1 \leq \text{real } (n::\text{nat}) \longleftrightarrow 1 \leq n$ 
  by auto

```

```

lemma convex_on_continuous:
  assumes open (s::('a::euclidean_space) set) convex_on s f
  shows continuous_on s f
  unfolding continuous_on_eq_continuous_at[OF assms(1)]
proof
  note dimge1 = DIM_positive[where 'a='a]
  fix x
  assume  $x \in s$ 
  then obtain e where  $e: \text{cball } x \ e \subseteq s$  and  $e > 0$ 
  using assms(1) unfolding open_contains_cball by auto

```

```

define  $d$  where  $d = e / \text{real } \text{DIM}('a)$ 
have  $0 < d$ 
  unfolding  $d\_def$  using  $\langle e > 0 \rangle \text{ dimge1}$  by auto
let  $?d = (\sum_{i \in \text{Basis}. d *_{\mathbb{R}} i) :: 'a$ 
obtain  $c$ 
  where  $c$ : finite  $c$ 
  and  $c1$ :  $\text{convex hull } c \subseteq \text{cball } x \ e$ 
  and  $c2$ :  $\text{cball } x \ d \subseteq \text{convex hull } c$ 
proof
  define  $c$  where  $c = (\sum_{i \in \text{Basis}. (\lambda a. a *_{\mathbb{R}} i) \cdot \{x \cdot i - d, x \cdot i + d\})$ 
  show finite  $c$ 
    unfolding  $c\_def$  by (simp add: finite_set_sum)
  have  $\bigwedge i. i \in \text{Basis} \implies \text{convex hull } \{x \cdot i - d, x \cdot i + d\} = \text{cbox } (x \cdot i - d)$ 
    ( $x \cdot i + d$ )
    using  $\langle 0 < d \rangle \text{ convex\_hull\_eq\_real\_cbox}$  by auto
  then have  $1$ :  $\text{convex hull } c = \{a. \forall i \in \text{Basis}. a \cdot i \in \text{cbox } (x \cdot i - d) (x \cdot i + d)\}$ 
    unfolding  $\text{box\_eq\_set\_sum\_Basis } c\_def \text{ convex\_hull\_set\_sum}$ 
    apply (subst convex\_hull\_linear\_image [symmetric])
    by (force simp add: linear\_iff scaleR\_add\_left) +
  then have  $2$ :  $\text{convex hull } c = \{a. \forall i \in \text{Basis}. a \cdot i \in \text{cball } (x \cdot i) \ d\}$ 
    by (simp add: dist\_norm abs\_le\_iff algebra\_simps)
  show  $\text{cball } x \ d \subseteq \text{convex hull } c$ 
    unfolding  $2$ 
    by (clarsimp simp: dist\_norm) (metis inner\_commute inner\_diff\_right
norm\_bound\_Basis\_le)
  have  $e'$ :  $e = (\sum_{(i::'a) \in \text{Basis}. d)$ 
    by (simp add: d\_def)
  show  $\text{convex hull } c \subseteq \text{cball } x \ e$ 
    unfolding  $2$ 
proof clarsimp
  show  $\text{dist } x \ y \leq e$  if  $\forall i \in \text{Basis}. \text{dist } (x \cdot i) (y \cdot i) \leq d$  for  $y$ 
proof –
  have  $\bigwedge i. i \in \text{Basis} \implies 0 \leq \text{dist } (x \cdot i) (y \cdot i)$ 
    by simp
  have  $(\sum_{i \in \text{Basis}. \text{dist } (x \cdot i) (y \cdot i)) \leq e$ 
    using  $e'$  sum\_mono that by fastforce
  then show ?thesis
    by (metis (mono\_tags) euclidean\_dist\_l2 order\_trans [OF L2\_set\_le\_sum]
zero\_le\_dist)
  qed
qed
qed
define  $k$  where  $k = \text{Max } (f \cdot c)$ 
have  $\text{convex\_on } (\text{convex hull } c) \ f$ 
  using assms( $2$ )  $c1$  convex\_on\_subset  $e(1)$  by blast
then have  $k$ :  $\forall y \in \text{convex hull } c. f \ y \leq k$ 
  using  $c$  convex\_on\_convex\_hull\_bound  $k\_def$  by fastforce
have  $e \leq e * \text{real } \text{DIM}('a)$ 

```

```

    using e(2) real_of_nat_ge_one_iff by auto
  then have  $d \leq e$ 
    by (simp add: d_def field_split_simps)
  then have dsube:  $\text{cball } x \ d \subseteq \text{cball } x \ e$ 
    by (rule subset_cball)
  have conv: convex_on (cball x d) f
    using ⟨convex_on (convex hull c) f⟩ c2 convex_on_subset by blast
  then have  $\bigwedge y. y \in \text{cball } x \ d \implies |f \ y| \leq k + 2 * |f \ x|$ 
    by (rule convex_bounds_lemma) (use c2 k in blast)
  then have continuous_on (ball x d) f
    by (meson Elementary_Metric_Spaces.open_ball_ball_subset_cball conv convex_on_bounded_continuous
        convex_on_subset mem_ball_imp_mem_cball)
  then show continuous (at x) f
    unfolding continuous_on_eq_continuous_at[OF open_ball]
    using ⟨ $d > 0$ ⟩ by auto
qed

end

```

4.3 Operator Norm

```

theory Operator_Norm
imports Complex_Main
begin

```

This formulation yields zero if $'a$ is the trivial vector space.

definition

```

onorm :: ('a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector)  $\Rightarrow$  real where
onorm f = (SUP x. norm (f x) / norm x)

```

proposition onorm_bound:

```

assumes  $0 \leq b$  and  $\bigwedge x. \text{norm } (f \ x) \leq b * \text{norm } x$ 
shows  $\text{onorm } f \leq b$ 
unfolding onorm_def
proof (rule cSUP_least)
  fix x
  show  $\text{norm } (f \ x) / \text{norm } x \leq b$ 
    using assms by (cases  $x = 0$ ) (simp_all add: pos_divide_le_eq)
qed simp

```

In non-trivial vector spaces, the first assumption is redundant.

lemma onorm_le:

```

fixes f :: 'a::{real_normed_vector, perfect_space}  $\Rightarrow$  'b::real_normed_vector
assumes  $\bigwedge x. \text{norm } (f \ x) \leq b * \text{norm } x$ 
shows  $\text{onorm } f \leq b$ 
proof (rule onorm_bound [OF __ assms])
  have  $\{0::'a\} \neq \text{UNIV}$  by (metis not_open_singleton open_UNIV)

```

```

then obtain a :: 'a where a ≠ 0 by fast
have 0 ≤ b * norm a
  by (rule order_trans [OF norm_ge_zero assms])
with ⟨a ≠ 0⟩ show 0 ≤ b
  by (simp add: zero_le_mult_iff)
qed

```

```

lemma le_onorm:
  assumes bounded_linear f
  shows norm (f x) / norm x ≤ onorm f
proof -
  interpret f: bounded_linear f by fact
  obtain b where 0 ≤ b and ∀x. norm (f x) ≤ norm x * b
    using f.nonneg_bounded by auto
  then have ∀x. norm (f x) / norm x ≤ b
    by (clarify, case_tac x = 0,
        simp_all add: f.zero_pos_divide_le_eq mult.commute)
  then have bdd_above (range (λx. norm (f x) / norm x))
    unfolding bdd_above_def by fast
  with UNIV_I show ?thesis
    unfolding onorm_def by (rule cSUP_upper)
qed

```

```

lemma onorm:
  assumes bounded_linear f
  shows norm (f x) ≤ onorm f * norm x
proof -
  interpret f: bounded_linear f by fact
  show ?thesis
  proof (cases)
    assume x = 0
    then show ?thesis by (simp add: f.zero)
  next
    assume x ≠ 0
    have norm (f x) / norm x ≤ onorm f
      by (rule le_onorm [OF assms])
    then show norm (f x) ≤ onorm f * norm x
      by (simp add: pos_divide_le_eq ⟨x ≠ 0⟩)
  qed
qed

```

```

lemma onorm_pos_le:
  assumes f: bounded_linear f
  shows 0 ≤ onorm f
  using le_onorm [OF f, where x=0] by simp

```

```

lemma onorm_zero: onorm (λx. 0) = 0
proof (rule order_antisym)
  show onorm (λx. 0) ≤ 0

```

```

    by (simp add: onorm_bound)
  show  $0 \leq \text{onorm } (\lambda x. 0)$ 
    using bounded_linear_zero by (rule onorm_pos_le)
qed

```

```

lemma onorm_eq_0:
  assumes f: bounded_linear f
  shows  $\text{onorm } f = 0 \iff (\forall x. f\ x = 0)$ 
  using onorm [OF f] by (auto simp: fun_eq_iff [symmetric] onorm_zero)

```

```

lemma onorm_pos_lt:
  assumes f: bounded_linear f
  shows  $0 < \text{onorm } f \iff \neg (\forall x. f\ x = 0)$ 
  by (simp add: less_le onorm_pos_le [OF f] onorm_eq_0 [OF f])

```

```

lemma onorm_id_le:  $\text{onorm } (\lambda x. x) \leq 1$ 
  by (rule onorm_bound) simp_all

```

```

lemma onorm_id:  $\text{onorm } (\lambda x. x :: 'a :: \{\text{real\_normed\_vector}, \text{perfect\_space}\}) = 1$ 
proof (rule antisym[OF onorm_id_le])
  have  $\{0 :: 'a\} \neq \text{UNIV}$  by (metis not_open_singleton open_UNIV)
  then obtain  $x :: 'a$  where  $x \neq 0$  by fast
  hence  $1 \leq \text{norm } x / \text{norm } x$ 
    by simp
  also have  $\dots \leq \text{onorm } (\lambda x :: 'a. x)$ 
    by (rule le_onorm) (rule bounded_linear_ident)
  finally show  $1 \leq \text{onorm } (\lambda x :: 'a. x)$  .
qed

```

```

lemma onorm_compose:
  assumes f: bounded_linear f
  assumes g: bounded_linear g
  shows  $\text{onorm } (f \circ g) \leq \text{onorm } f * \text{onorm } g$ 
proof (rule onorm_bound)
  show  $0 \leq \text{onorm } f * \text{onorm } g$ 
    by (intro mult_nonneg_nonneg onorm_pos_le f g)
next
  fix x
  have  $\text{norm } (f\ (g\ x)) \leq \text{onorm } f * \text{norm } (g\ x)$ 
    by (rule onorm [OF f])
  also have  $\text{onorm } f * \text{norm } (g\ x) \leq \text{onorm } f * (\text{onorm } g * \text{norm } x)$ 
    by (rule mult_left_mono [OF onorm [OF g] onorm_pos_le [OF f]])
  finally show  $\text{norm } ((f \circ g)\ x) \leq \text{onorm } f * \text{onorm } g * \text{norm } x$ 
    by (simp add: mult.assoc)
qed

```

```

lemma onorm_scaleR_lemma:
  assumes f: bounded_linear f
  shows  $\text{onorm } (\lambda x. r *_{\mathbb{R}} f\ x) \leq |r| * \text{onorm } f$ 

```

```

proof (rule onorm_bound)
  show  $0 \leq |r| * \text{onorm } f$ 
    by (intro mult_nonneg_nonneg onorm_pos_le abs_ge_zero f)
next
  fix x
  have  $|r| * \text{norm } (f x) \leq |r| * (\text{onorm } f * \text{norm } x)$ 
    by (intro mult_left_mono onorm_abs_ge_zero f)
  then show  $\text{norm } (r *_R f x) \leq |r| * \text{onorm } f * \text{norm } x$ 
    by (simp only: norm_scaleR mult.assoc)
qed

```

```

lemma onorm_scaleR:
  assumes f: bounded_linear f
  shows  $\text{onorm } (\lambda x. r *_R f x) = |r| * \text{onorm } f$ 
proof (cases r = 0)
  assume  $r \neq 0$ 
  show ?thesis
  proof (rule order_antisym)
    show  $\text{onorm } (\lambda x. r *_R f x) \leq |r| * \text{onorm } f$ 
      using f by (rule onorm_scaleR_lemma)
    next
    have bounded_linear  $(\lambda x. r *_R f x)$ 
      using bounded_linear_scaleR_right f by (rule bounded_linear_compose)
    then have  $\text{onorm } (\lambda x. \text{inverse } r *_R r *_R f x) \leq |\text{inverse } r| * \text{onorm } (\lambda x. r *_R f x)$ 
      by (rule onorm_scaleR_lemma)
    with  $\langle r \neq 0 \rangle$  show  $|r| * \text{onorm } f \leq \text{onorm } (\lambda x. r *_R f x)$ 
      by (simp add: inverse_eq_divide pos_le_divide_eq mult.commute)
  qed
qed (simp add: onorm_zero)

```

```

lemma onorm_scaleR_left_lemma:
  assumes r: bounded_linear r
  shows  $\text{onorm } (\lambda x. r x *_R f) \leq \text{onorm } r * \text{norm } f$ 
proof (rule onorm_bound)
  fix x
  have  $\text{norm } (r x *_R f) = \text{norm } (r x) * \text{norm } f$ 
    by simp
  also have  $\dots \leq \text{onorm } r * \text{norm } x * \text{norm } f$ 
    by (intro mult_right_mono onorm_r norm_ge_zero)
  finally show  $\text{norm } (r x *_R f) \leq \text{onorm } r * \text{norm } f * \text{norm } x$ 
    by (simp add: ac_simps)
qed (intro mult_nonneg_nonneg norm_ge_zero onorm_pos_le r)

```

```

lemma onorm_scaleR_left:
  assumes f: bounded_linear r
  shows  $\text{onorm } (\lambda x. r x *_R f) = \text{onorm } r * \text{norm } f$ 
proof (cases f = 0)
  assume  $f \neq 0$ 

```



```

show ?thesis
proof (rule order_antisym)
  show onorm ( $\lambda x. r\ x *_{\mathbb{R}} f$ )  $\leq$  onorm  $r * \text{norm } f$ 
    using  $f$  by (rule onorm_scaleR_left_lemma)
next
have bl1: bounded_linear ( $\lambda x. r\ x *_{\mathbb{R}} f$ )
  by (metis bounded_linear_scaleR_const  $f$ )
have bounded_linear ( $\lambda x. r\ x * \text{norm } f$ )
  by (metis bounded_linear_mult_const  $f$ )
from onorm_scaleR_left_lemma[OF this, of inverse ( $\text{norm } f$ )]
have onorm  $r \leq$  onorm ( $\lambda x. r\ x * \text{norm } f$ ) * inverse ( $\text{norm } f$ )
  using  $\langle f \neq 0 \rangle$ 
  by (simp add: inverse_eq_divide)
also have onorm ( $\lambda x. r\ x * \text{norm } f$ )  $\leq$  onorm ( $\lambda x. r\ x *_{\mathbb{R}} f$ )
  by (rule onorm_bound)
  (auto simp: abs_mult bl1 onorm_pos_le intro!: order_trans[OF _ onorm])
finally show onorm  $r * \text{norm } f \leq$  onorm ( $\lambda x. r\ x *_{\mathbb{R}} f$ )
  using  $\langle f \neq 0 \rangle$ 
  by (simp add: inverse_eq_divide pos_le_divide_eq mult.commute)
qed
qed (simp add: onorm_zero)

lemma onorm_neg:
  shows onorm ( $\lambda x. - f\ x$ ) = onorm  $f$ 
  unfolding onorm_def by simp

lemma onorm_triangle:
  assumes  $f$ : bounded_linear  $f$ 
  assumes  $g$ : bounded_linear  $g$ 
  shows onorm ( $\lambda x. f\ x + g\ x$ )  $\leq$  onorm  $f$  + onorm  $g$ 
proof (rule onorm_bound)
  show  $0 \leq$  onorm  $f$  + onorm  $g$ 
    by (intro add_nonneg_nonneg onorm_pos_le  $f\ g$ )
next
fix  $x$ 
have norm ( $f\ x + g\ x$ )  $\leq$  norm ( $f\ x$ ) + norm ( $g\ x$ )
  by (rule norm_triangle_ineq)
also have norm ( $f\ x$ ) + norm ( $g\ x$ )  $\leq$  onorm  $f * \text{norm } x$  + onorm  $g * \text{norm } x$ 
  by (intro add_mono onorm  $f\ g$ )
finally show norm ( $f\ x + g\ x$ )  $\leq$  (onorm  $f$  + onorm  $g$ ) * norm  $x$ 
  by (simp only: distrib_right)
qed

lemma onorm_triangle_le:
  assumes bounded_linear  $f$ 
  assumes bounded_linear  $g$ 
  assumes onorm  $f$  + onorm  $g \leq e$ 
  shows onorm ( $\lambda x. f\ x + g\ x$ )  $\leq e$ 
  using assms by (rule onorm_triangle [THEN order_trans])

```

```

lemma onorm_triangle_lt:
  assumes bounded_linear f
  assumes bounded_linear g
  assumes onorm f + onorm g < e
  shows onorm ( $\lambda x. f\ x + g\ x$ ) < e
  using assms by (rule onorm_triangle [THEN order_le_less_trans])

lemma onorm_sum:
  assumes finite S
  assumes  $\bigwedge s. s \in S \implies \text{bounded\_linear } (f\ s)$ 
  shows onorm ( $\lambda x. \text{sum } (\lambda s. f\ s\ x)\ S$ )  $\leq \text{sum } (\lambda s. \text{onorm } (f\ s))\ S$ 
  using assms
  by (induction) (auto simp: onorm_zero intro!: onorm_triangle_le bounded_linear_sum)

lemmas onorm_sum_le = onorm_sum [THEN order_trans]

end

```

4.4 Line Segment

```

theory Line_Segment
imports
  Convex
  Topology_Euclidean_Space
begin

```

4.4.1 Topological Properties of Convex Sets, Metric Spaces and Functions

```

lemma convex_supp_sum:
  assumes convex S and 1: supp_sum u I = 1
  and  $\bigwedge i. i \in I \implies 0 \leq u\ i \wedge (u\ i = 0 \vee f\ i \in S)$ 
  shows supp_sum ( $\lambda i. u\ i *_R f\ i$ ) I  $\in S$ 
proof -
  have fin: finite {i  $\in$  I. u i  $\neq$  0}
  using 1 sum.infinite by (force simp: supp_sum_def support_on_def)
  then have supp_sum ( $\lambda i. u\ i *_R f\ i$ ) I = sum ( $\lambda i. u\ i *_R f\ i$ ) {i  $\in$  I. u i  $\neq$  0}
  by (force intro: sum.mono_neutral_left simp: supp_sum_def support_on_def)
  also have ...  $\in S$ 
  using 1 assms by (force simp: supp_sum_def support_on_def intro: convex_sum [OF fin  $\langle$ convex S $\rangle$ ])
  finally show ?thesis .
qed

lemma sphere_eq_empty [simp]:
  fixes a :: 'a::{'real_normed_vector', perfect_space'}
  shows sphere a r = {}  $\longleftrightarrow r < 0$ 
by (auto simp: sphere_def dist_norm) (metis dist_norm le_less_linear vector_choose_dist)

```

```

lemma cone_closure:
  fixes  $S :: 'a::real\_normed\_vector\ set$ 
  assumes cone  $S$ 
  shows cone (closure  $S$ )
proof (cases  $S = \{\}$ )
  case True
  then show ?thesis by auto
next
  case False
  then have  $0 \in S \wedge (\forall c. c > 0 \longrightarrow (*_R) c \cdot S = S)$ 
    using cone_iff[of  $S$ ] assms by auto
  then have  $0 \in \text{closure } S \wedge (\forall c. c > 0 \longrightarrow (*_R) c \cdot \text{closure } S = \text{closure } S)$ 
    using closure_subset by (auto simp: closure_scaleR)
  then show ?thesis
    using False cone_iff[of closure  $S$ ] by auto
qed

```

```

corollary component_complement_connected:
  fixes  $S :: 'a::real\_normed\_vector\ set$ 
  assumes connected  $S$   $C \in \text{components } (-S)$ 
  shows connected( $-C$ )
  using component_diff_connected [of  $S$  UNIV] assms
  by (auto simp: Compl_eq_Diff_UNIV)

```

```

proposition clopen:
  fixes  $S :: 'a :: real\_normed\_vector\ set$ 
  shows closed  $S \wedge \text{open } S \longleftrightarrow S = \{\} \vee S = \text{UNIV}$ 
  by (force intro!: connected_UNIV [unfolded connected_clopen, rule_format])

```

```

corollary compact_open:
  fixes  $S :: 'a :: euclidean\_space\ set$ 
  shows compact  $S \wedge \text{open } S \longleftrightarrow S = \{\}$ 
  by (auto simp: compact_eq_bounded_closed clopen)

```

```

corollary finite_imp_not_open:
  fixes  $S :: 'a::\{real\_normed\_vector, perfect\_space\}\ set$ 
  shows  $\llbracket \text{finite } S; \text{open } S \rrbracket \implies S = \{\}$ 
  using clopen [of  $S$ ] finite_imp_closed not_bounded_UNIV by blast

```

```

corollary empty_interior_finite:
  fixes  $S :: 'a::\{real\_normed\_vector, perfect\_space\}\ set$ 
  shows finite  $S \implies \text{interior } S = \{\}$ 
  by (metis interior_subset finite_subset open_interior [of  $S$ ] finite_imp_not_open)

```

Balls, being convex, are connected.

```

lemma convex_local_global_minimum:
  fixes  $s :: 'a::real\_normed\_vector\ set$ 

```

```

assumes  $e > 0$ 
and convex_on  $s$   $f$ 
and ball  $x$   $e \subseteq s$ 
and  $\forall y \in \text{ball } x \ e. f\ x \leq f\ y$ 
shows  $\forall y \in s. f\ x \leq f\ y$ 
proof (rule ccontr)
have  $x \in s$  using assms(1,3) by auto
assume  $\neg ?thesis$ 
then obtain  $y$  where  $y \in s$  and  $y: f\ x > f\ y$  by auto
then have  $xy: 0 < \text{dist } x\ y$  by auto
then obtain  $u$  where  $0 < u \leq 1$  and  $u: u < e / \text{dist } x\ y$ 
using field_lbound_gt_zero[of  $1\ e / \text{dist } x\ y$ ]  $xy\ \langle e > 0 \rangle$  by auto
then have  $f\ ((1-u) *_{\mathbb{R}} x + u *_{\mathbb{R}} y) \leq (1-u) * f\ x + u * f\ y$ 
using  $\langle x \in s \rangle\ \langle y \in s \rangle$ 
using assms(2)[unfolded convex_on_def,
  THEN bspec[where  $x=x$ ], THEN bspec[where  $x=y$ ], THEN spec[where
 $x=1-u$ ]]
by auto
moreover
have  $*$ :  $x - ((1-u) *_{\mathbb{R}} x + u *_{\mathbb{R}} y) = u *_{\mathbb{R}} (x - y)$ 
by (simp add: algebra_simps)
have  $(1-u) *_{\mathbb{R}} x + u *_{\mathbb{R}} y \in \text{ball } x\ e$ 
unfolding mem_ball_dist_norm
unfolding  $*$  and norm_scaleR and abs_of_pos[OF  $\langle 0 < u \rangle$ ]
unfolding dist_norm[symmetric]
using  $u$ 
unfolding pos_less_divide_eq[OF xy]
by auto
then have  $f\ x \leq f\ ((1-u) *_{\mathbb{R}} x + u *_{\mathbb{R}} y)$ 
using assms(4) by auto
ultimately show False
using mult_strict_left_mono[OF y  $\langle u > 0 \rangle$ ]
unfolding left_diff_distrib
by auto
qed

```

```

lemma convex_ball [iff]:
fixes  $x :: 'a::\text{real\_normed\_vector}$ 
shows convex (ball  $x\ e$ )
proof (auto simp: convex_def)
fix  $y\ z$ 
assume  $yz: \text{dist } x\ y < e\ \text{dist } x\ z < e$ 
fix  $u\ v :: \text{real}$ 
assume  $uv: 0 \leq u\ 0 \leq v\ u + v = 1$ 
have  $\text{dist } x\ (u *_{\mathbb{R}} y + v *_{\mathbb{R}} z) \leq u * \text{dist } x\ y + v * \text{dist } x\ z$ 
using  $uv\ yz$ 
using convex_on_dist [of ball  $x\ e\ x$ , unfolded convex_on_def,
  THEN bspec[where  $x=y$ ], THEN bspec[where  $x=z$ ]]
by auto

```

```

    then show  $\text{dist } x (u *_R y + v *_R z) < e$ 
      using  $\text{convex\_bound\_lt}[OF \text{ } yz \text{ } uv]$  by auto
  qed

```

```

lemma  $\text{convex\_cball}$  [iff]:
  fixes  $x :: 'a::\text{real\_normed\_vector}$ 
  shows  $\text{convex } (\text{cball } x \text{ } e)$ 
proof -
  {
    fix  $y \text{ } z$ 
    assume  $yz: \text{dist } x \text{ } y \leq e \text{ } \text{dist } x \text{ } z \leq e$ 
    fix  $u \text{ } v :: \text{real}$ 
    assume  $uv: 0 \leq u \text{ } 0 \leq v \text{ } u + v = 1$ 
    have  $\text{dist } x (u *_R y + v *_R z) \leq u * \text{dist } x \text{ } y + v * \text{dist } x \text{ } z$ 
      using  $uv \text{ } yz$ 
      using  $\text{convex\_on\_dist}$  [of  $\text{cball } x \text{ } e \text{ } x$ ,  $\text{unfolded convex\_on\_def}$ ,
        THEN  $\text{bspec}[\text{where } x=y]$ , THEN  $\text{bspec}[\text{where } x=z]$ ]
      by auto
    then have  $\text{dist } x (u *_R y + v *_R z) \leq e$ 
      using  $\text{convex\_bound\_le}[OF \text{ } yz \text{ } uv]$  by auto
  }
  then show ?thesis by (auto simp:  $\text{convex\_def Ball\_def}$ )
qed

```

```

lemma  $\text{connected\_ball}$  [iff]:
  fixes  $x :: 'a::\text{real\_normed\_vector}$ 
  shows  $\text{connected } (\text{ball } x \text{ } e)$ 
  using  $\text{convex\_connected convex\_ball}$  by auto

```

```

lemma  $\text{connected\_cball}$  [iff]:
  fixes  $x :: 'a::\text{real\_normed\_vector}$ 
  shows  $\text{connected } (\text{cball } x \text{ } e)$ 
  using  $\text{convex\_connected convex\_cball}$  by auto

```

```

lemma  $\text{bounded\_convex\_hull}$ :
  fixes  $s :: 'a::\text{real\_normed\_vector set}$ 
  assumes  $\text{bounded } s$ 
  shows  $\text{bounded } (\text{convex hull } s)$ 
proof -
  from  $\text{assms}$  obtain  $B$  where  $B: \forall x \in s. \text{norm } x \leq B$ 
  unfolding  $\text{bounded\_iff}$  by auto
  show ?thesis
    by (simp add:  $\text{bounded\_subset}[OF \text{ } \text{bounded\_cball}, \text{ of } \_ \text{ } 0 \text{ } B]$   $B \text{ subsetI subset\_hull}$ )
qed

```

```

lemma  $\text{finite\_imp\_bounded\_convex\_hull}$ :
  fixes  $s :: 'a::\text{real\_normed\_vector set}$ 
  shows  $\text{finite } s \implies \text{bounded } (\text{convex hull } s)$ 

```

```

using bounded_convex_hull finite_imp_bounded
by auto

```

4.4.2 Midpoint

```

definition midpoint :: 'a::real_vector  $\Rightarrow$  'a  $\Rightarrow$  'a
  where midpoint a b = (inverse (2::real)) *R (a + b)

```

```

lemma midpoint_idem [simp]: midpoint x x = x
  unfolding midpoint_def by simp

```

```

lemma midpoint_sym: midpoint a b = midpoint b a
  unfolding midpoint_def by (auto simp add: scaleR_right_distrib)

```

```

lemma midpoint_eq_iff: midpoint a b = c  $\longleftrightarrow$  a + b = c + c
proof -
  have midpoint a b = c  $\longleftrightarrow$  scaleR 2 (midpoint a b) = scaleR 2 c
  by simp
  then show ?thesis
    unfolding midpoint_def scaleR_2 [symmetric] by simp
qed

```

```

lemma
  fixes a::real
  assumes a  $\leq$  b shows ge_midpoint_1: a  $\leq$  midpoint a b
    and le_midpoint_1: midpoint a b  $\leq$  b
  by (simp_all add: midpoint_def assms)

```

```

lemma dist_midpoint:
  fixes a b :: 'a::real_normed_vector shows
    dist a (midpoint a b) = (dist a b) / 2 (is ?t1)
    dist b (midpoint a b) = (dist a b) / 2 (is ?t2)
    dist (midpoint a b) a = (dist a b) / 2 (is ?t3)
    dist (midpoint a b) b = (dist a b) / 2 (is ?t4)
proof -
  have *:  $\bigwedge x y::'a. 2 *_{\mathbf{R}} x = - y \implies \text{norm } x = (\text{norm } y) / 2$ 
    unfolding equation_minus_iff by auto
  have **:  $\bigwedge x y::'a. 2 *_{\mathbf{R}} x = y \implies \text{norm } x = (\text{norm } y) / 2$ 
    by auto
  note scaleR_right_distrib [simp]
  show ?t1
    unfolding midpoint_def dist_norm
    apply (rule **)
    apply (simp add: scaleR_right_diff_distrib)
    apply (simp add: scaleR_2)
    done
  show ?t2
    unfolding midpoint_def dist_norm
    apply (rule *)

```

```

    apply (simp add: scaleR_right_diff_distrib)
    apply (simp add: scaleR_2)
    done
  show ?t3
    unfolding midpoint_def dist_norm
    apply (rule *)
    apply (simp add: scaleR_right_diff_distrib)
    apply (simp add: scaleR_2)
    done
  show ?t4
    unfolding midpoint_def dist_norm
    apply (rule **)
    apply (simp add: scaleR_right_diff_distrib)
    apply (simp add: scaleR_2)
    done
qed

lemma midpoint_eq_endpoint [simp]:
  midpoint a b = a  $\longleftrightarrow$  a = b
  midpoint a b = b  $\longleftrightarrow$  a = b
  unfolding midpoint_eq_iff by auto

lemma midpoint_plus_self [simp]: midpoint a b + midpoint a b = a + b
  using midpoint_eq_iff by metis

lemma midpoint_linear_image:
  linear f  $\implies$  midpoint(f a)(f b) = f(midpoint a b)
by (simp add: linear_iff midpoint_def)



### 4.4.3 Open and closed segments

definition closed_segment :: 'a::real_vector  $\Rightarrow$  'a  $\Rightarrow$  'a set
  where closed_segment a b = {(1 - u) *R a + u *R b | u::real. 0  $\leq$  u  $\wedge$  u  $\leq$  1}

definition open_segment :: 'a::real_vector  $\Rightarrow$  'a  $\Rightarrow$  'a set where
  open_segment a b  $\equiv$  closed_segment a b - {a,b}

lemmas segment = open_segment_def closed_segment_def

lemma in_segment:
  x  $\in$  closed_segment a b  $\longleftrightarrow$  ( $\exists$  u. 0  $\leq$  u  $\wedge$  u  $\leq$  1  $\wedge$  x = (1 - u) *R a + u *R b)
  x  $\in$  open_segment a b  $\longleftrightarrow$  a  $\neq$  b  $\wedge$  ( $\exists$  u. 0 < u  $\wedge$  u < 1  $\wedge$  x = (1 - u) *R a + u *R b)
  using less_eq_real_def by (auto simp: segment algebra_simps)

lemma closed_segment_linear_image:
  closed_segment (f a) (f b) = f ` (closed_segment a b) if linear f
proof -

```

```

interpret linear f by fact
show ?thesis
  by (force simp add: in_segment add scale)
qed

```

```

lemma open_segment_linear_image:
   $\llbracket \text{linear } f; \text{inj } f \rrbracket \implies \text{open\_segment } (f\ a)\ (f\ b) = f\ ' \ (\text{open\_segment } a\ b)$ 
  by (force simp: open_segment_def closed_segment_linear_image inj_on_def)

```

```

lemma closed_segment_translation:
   $\text{closed\_segment } (c + a)\ (c + b) = \text{image } (\lambda x. c + x)\ (\text{closed\_segment } a\ b)$ 
  apply safe
  apply (rule_tac x=x-c in image_eqI)
  apply (auto simp: in_segment algebra_simps)
  done

```

```

lemma open_segment_translation:
   $\text{open\_segment } (c + a)\ (c + b) = \text{image } (\lambda x. c + x)\ (\text{open\_segment } a\ b)$ 
  by (simp add: open_segment_def closed_segment_translation translation_diff)

```

```

lemma closed_segment_of_real:
   $\text{closed\_segment } (\text{of\_real } x)\ (\text{of\_real } y) = \text{of\_real } ' \ \text{closed\_segment } x\ y$ 
  apply (auto simp: image_iff in_segment scaleR_conv_of_real)
  apply (rule_tac x=(1-u)*x + u*y in bexI)
  apply (auto simp: in_segment)
  done

```

```

lemma open_segment_of_real:
   $\text{open\_segment } (\text{of\_real } x)\ (\text{of\_real } y) = \text{of\_real } ' \ \text{open\_segment } x\ y$ 
  apply (auto simp: image_iff in_segment scaleR_conv_of_real)
  apply (rule_tac x=(1-u)*x + u*y in bexI)
  apply (auto simp: in_segment)
  done

```

```

lemma closed_segment_Reals:
   $\llbracket x \in \text{Reals}; y \in \text{Reals} \rrbracket \implies \text{closed\_segment } x\ y = \text{of\_real } ' \ \text{closed\_segment } (\text{Re } x)\ (\text{Re } y)$ 
  by (metis closed_segment_of_real of_real_Re)

```

```

lemma open_segment_Reals:
   $\llbracket x \in \text{Reals}; y \in \text{Reals} \rrbracket \implies \text{open\_segment } x\ y = \text{of\_real } ' \ \text{open\_segment } (\text{Re } x)\ (\text{Re } y)$ 
  by (metis open_segment_of_real of_real_Re)

```

```

lemma open_segment_PairD:
   $(x, x') \in \text{open\_segment } (a, a')\ (b, b')$ 
   $\implies (x \in \text{open\_segment } a\ b \vee a = b) \wedge (x' \in \text{open\_segment } a'\ b' \vee a' = b')$ 
  by (auto simp: in_segment)

```


lemma *closed_segment_PairD*:

$(x, x') \in \text{closed\_segment } (a, a') (b, b') \implies x \in \text{closed\_segment } a \ b \wedge x' \in \text{closed\_segment } a' \ b'$
by (auto simp: closed_segment_def)

lemma *closed_segment_translation_eq* [simp]:

$d + x \in \text{closed\_segment } (d + a) (d + b) \longleftrightarrow x \in \text{closed\_segment } a \ b$
proof –
have *: $\bigwedge d \ x \ a \ b. x \in \text{closed\_segment } a \ b \implies d + x \in \text{closed\_segment } (d + a) (d + b)$
apply (simp add: closed_segment_def)
apply (erule ex_forward)
apply (simp add: algebra_simps)
done
show ?thesis
using * [where $d = -d$] *
by (fastforce simp add:)
qed

lemma *open_segment_translation_eq* [simp]:

$d + x \in \text{open\_segment } (d + a) (d + b) \longleftrightarrow x \in \text{open\_segment } a \ b$
by (simp add: open_segment_def)

lemma *of_real_closed_segment* [simp]:

$\text{of\_real } x \in \text{closed\_segment } (\text{of\_real } a) (\text{of\_real } b) \longleftrightarrow x \in \text{closed\_segment } a \ b$
apply (auto simp: in_segment scaleR_conv_of_real elim!: ex_forward)
using of_real_eq_iff **by** fastforce

lemma *of_real_open_segment* [simp]:

$\text{of\_real } x \in \text{open\_segment } (\text{of\_real } a) (\text{of\_real } b) \longleftrightarrow x \in \text{open\_segment } a \ b$
apply (auto simp: in_segment scaleR_conv_of_real elim!: ex_forward del: exE)
using of_real_eq_iff **by** fastforce

lemma *convex_contains_segment*:

$\text{convex } S \longleftrightarrow (\forall a \in S. \forall b \in S. \text{closed\_segment } a \ b \subseteq S)$
unfolding convex_alt closed_segment_def **by** auto

lemma *closed_segment_in_Reals*:

$\llbracket x \in \text{closed\_segment } a \ b; a \in \text{Reals}; b \in \text{Reals} \rrbracket \implies x \in \text{Reals}$
by (meson subsetD convex_Reals convex_contains_segment)

lemma *open_segment_in_Reals*:

$\llbracket x \in \text{open\_segment } a \ b; a \in \text{Reals}; b \in \text{Reals} \rrbracket \implies x \in \text{Reals}$
by (metis Diff_iff closed_segment_in_Reals open_segment_def)

lemma *closed_segment_subset*: $\llbracket x \in S; y \in S; \text{convex } S \rrbracket \implies \text{closed\_segment } x \ y \subseteq S$

by (simp add: convex_contains_segment)

lemma *closed_segment_subset_convex_hull*:
 $\llbracket x \in \text{convex\_hull } S; y \in \text{convex\_hull } S \rrbracket \implies \text{closed\_segment } x \ y \subseteq \text{convex\_hull } S$
using *convex_contains_segment* **by** *blast*

lemma *segment_convex_hull*:
 $\text{closed\_segment } a \ b = \text{convex\_hull } \{a, b\}$
proof –
have *: $\bigwedge x. \{x\} \neq \{\}$ **by** *auto*
show ?thesis
unfolding *segment_convex_hull_insert* [OF *] *convex_hull_singleton*
by (*safe*; *rule_tac* $x=1$ – *u* **in** *exI*; *force*)
qed

lemma *open_closed_segment*: $u \in \text{open\_segment } w \ z \implies u \in \text{closed\_segment } w \ z$
by (*auto simp add: closed_segment_def open_segment_def*)

lemma *segment_open_subset_closed*:
 $\text{open\_segment } a \ b \subseteq \text{closed\_segment } a \ b$
by (*auto simp: closed_segment_def open_segment_def*)

lemma *bounded_closed_segment*:
fixes $a :: 'a :: \text{real\_normed\_vector}$ **shows** *bounded* ($\text{closed\_segment } a \ b$)
by (*rule* *boundedI* [**where** $B = \max (\text{norm } a) (\text{norm } b)$])
(*auto simp: closed_segment_def max_def convex_bound_le intro!: norm_triangle_le*)

lemma *bounded_open_segment*:
fixes $a :: 'a :: \text{real\_normed\_vector}$ **shows** *bounded* ($\text{open\_segment } a \ b$)
by (*rule* *bounded_subset* [OF *bounded_closed_segment segment_open_subset_closed*])

lemmas *bounded_segment* = *bounded_closed_segment open_closed_segment*

lemma *ends_in_segment* [*iff*]: $a \in \text{closed\_segment } a \ b \ b \in \text{closed\_segment } a \ b$
unfolding *segment_convex_hull*
by (*auto intro!: hull_subset* [*unfolded subset_eq, rule_format*])

lemma *eventually_closed_segment*:
fixes $x0 :: 'a :: \text{real\_normed\_vector}$
assumes *open* $X0 \ x0 \in X0$
shows $\forall_F x \text{ in } \text{at } x0 \text{ within } U. \text{closed\_segment } x0 \ x \subseteq X0$
proof –
from *openE* [OF *assms*]
obtain e **where** $0 < e \text{ ball } x0 \ e \subseteq X0$.
then have $\forall_F x \text{ in } \text{at } x0 \text{ within } U. x \in \text{ball } x0 \ e$
by (*auto simp: dist_commute eventually_at*)
then show ?thesis
proof *eventually_elim*
case (*elim x*)

```

    have  $x0 \in \text{ball } x0 \ e$  using  $\langle e > 0 \rangle$  by simp
    from convex_ball[unfolded convex_contains_segment, rule_format, OF this
elim]
    have closed_segment  $x0 \ x \subseteq \text{ball } x0 \ e$  .
    also note  $\langle \dots \subseteq X0 \rangle$ 
    finally show ?case .
qed
qed

```

```

lemma closed_segment_commute: closed_segment a b = closed_segment b a
proof -
  have  $\{a, b\} = \{b, a\}$  by auto
  thus ?thesis
    by (simp add: segment_convex_hull)
qed

```

```

lemma segment_bound1:
  assumes  $x \in \text{closed\_segment } a \ b$ 
  shows  $\text{norm } (x - a) \leq \text{norm } (b - a)$ 
proof -
  obtain  $u$  where  $x = (1 - u) *_R a + u *_R b$   $0 \leq u \leq 1$ 
  using assms by (auto simp add: closed_segment_def)
  then show  $\text{norm } (x - a) \leq \text{norm } (b - a)$ 
    apply clarify
    apply (auto simp: algebra_simps)
    apply (simp add: scaleR_diff_right [symmetric] mult_left_le_one_le)
    done
qed

```

```

lemma segment_bound:
  assumes  $x \in \text{closed\_segment } a \ b$ 
  shows  $\text{norm } (x - a) \leq \text{norm } (b - a)$   $\text{norm } (x - b) \leq \text{norm } (b - a)$ 
by (metis assms closed_segment_commute dist_commute dist_norm segment_bound1)+

```

```

lemma open_segment_commute: open_segment a b = open_segment b a
proof -
  have  $\{a, b\} = \{b, a\}$  by auto
  thus ?thesis
    by (simp add: closed_segment_commute open_segment_def)
qed

```

```

lemma closed_segment_idem [simp]: closed_segment a a = {a}
unfolding segment by (auto simp add: algebra_simps)

```

```

lemma open_segment_idem [simp]: open_segment a a = {}
by (simp add: open_segment_def)

```

```

lemma closed_segment_eq_open: closed_segment a b = open_segment a b  $\cup \{a, b\}$ 
using open_segment_def by auto

```

lemma *convex_contains_open_segment*:

convex $s \longleftrightarrow (\forall a \in s. \forall b \in s. \text{open\_segment } a \ b \subseteq s)$

by (*simp add: convex_contains_segment closed_segment_eq_open*)

lemma *closed_segment_eq_real_ivl1*:

fixes $a \ b :: \text{real}$

assumes $a \leq b$

shows $\text{closed\_segment } a \ b = \{a \dots b\}$

proof *safe*

fix x

assume $x \in \text{closed\_segment } a \ b$

then obtain u **where** $0 \leq u \leq 1$ **and** $x\_def: x = (1 - u) * a + u * b$

by (*auto simp: closed_segment_def*)

have $u * a \leq u * b$ $(1 - u) * a \leq (1 - u) * b$

by (*auto intro!: mult_left_mono u assms*)

then show $x \in \{a \dots b\}$

unfolding x_def **by** (*auto simp: algebra_simps*)

next

show $\bigwedge x. x \in \{a \dots b\} \implies x \in \text{closed\_segment } a \ b$

by (*force simp: closed_segment_def divide_simps algebra_simps*

intro: exI[where $x = (x - a) / (b - a)$ for x])

qed

lemma *closed_segment_eq_real_ivl*:

fixes $a \ b :: \text{real}$

shows $\text{closed\_segment } a \ b = (\text{if } a \leq b \text{ then } \{a \dots b\} \text{ else } \{b \dots a\})$

using *closed_segment_eq_real_ivl1* [*of* $a \ b$] *closed_segment_eq_real_ivl1* [*of* $b \ a$]

by (*auto simp: closed_segment_commute*)

lemma *open_segment_eq_real_ivl*:

fixes $a \ b :: \text{real}$

shows $\text{open\_segment } a \ b = (\text{if } a \leq b \text{ then } \{a < \dots < b\} \text{ else } \{b < \dots < a\})$

by (*auto simp: closed_segment_eq_real_ivl open_segment_def split: if_split_asm*)

lemma *closed_segment_real_eq*:

fixes $u :: \text{real}$ **shows** $\text{closed\_segment } u \ v = (\lambda x. (v - u) * x + u) \text{ ` } \{0..1\}$

by (*simp add: add.commute* [*of* u] *image_affinity_atLeastAtMost* [*where* $c = u$]

closed_segment_eq_real_ivl)

lemma *closed_segment_same_Re*:

assumes $\text{Re } a = \text{Re } b$

shows $\text{closed\_segment } a \ b = \{z. \text{Re } z = \text{Re } a \wedge \text{Im } z \in \text{closed\_segment } (\text{Im } a) \ (\text{Im } b)\}$

proof *safe*

fix z **assume** $z \in \text{closed\_segment } a \ b$

then obtain u **where** $u \in \{0..1\}$ $z = a + \text{of\_real } u * (b - a)$

by (*auto simp: closed_segment_def scaleR_conv_of_real algebra_simps*)

from *assms* **show** $\text{Re } z = \text{Re } a$ **by** (*auto simp: u*)

```

    from u(1) show  $Im\ z \in closed\_segment\ (Im\ a)\ (Im\ b)$ 
    by (force simp: u closed_segment_def algebra_simps)
next
  fix z assume [simp]:  $Re\ z = Re\ a$  and  $Im\ z \in closed\_segment\ (Im\ a)\ (Im\ b)$ 
  then obtain u where  $u: u \in \{0..1\}$   $Im\ z = Im\ a + of\_real\ u * (Im\ b - Im\ a)$ 
    by (auto simp: closed_segment_def scaleR_conv_of_real algebra_simps)
  from u(1) show  $z \in closed\_segment\ a\ b$  using assms
  by (force simp: u closed_segment_def algebra_simps scaleR_conv_of_real complex_eq_iff)
qed

```

```

lemma closed_segment_same_Im:
  assumes  $Im\ a = Im\ b$ 
  shows  $closed\_segment\ a\ b = \{z. Im\ z = Im\ a \wedge Re\ z \in closed\_segment\ (Re\ a)\ (Re\ b)\}$ 
proof safe
  fix z assume  $z \in closed\_segment\ a\ b$ 
  then obtain u where  $u: u \in \{0..1\}$   $z = a + of\_real\ u * (b - a)$ 
    by (auto simp: closed_segment_def scaleR_conv_of_real algebra_simps)
  from assms show  $Im\ z = Im\ a$  by (auto simp: u)
  from u(1) show  $Re\ z \in closed\_segment\ (Re\ a)\ (Re\ b)$ 
    by (force simp: u closed_segment_def algebra_simps)
next
  fix z assume [simp]:  $Im\ z = Im\ a$  and  $Re\ z \in closed\_segment\ (Re\ a)\ (Re\ b)$ 
  then obtain u where  $u: u \in \{0..1\}$   $Re\ z = Re\ a + of\_real\ u * (Re\ b - Re\ a)$ 
    by (auto simp: closed_segment_def scaleR_conv_of_real algebra_simps)
  from u(1) show  $z \in closed\_segment\ a\ b$  using assms
  by (force simp: u closed_segment_def algebra_simps scaleR_conv_of_real complex_eq_iff)
qed

```

```

lemma dist_in_closed_segment:
  fixes  $a :: 'a :: euclidean\_space$ 
  assumes  $x \in closed\_segment\ a\ b$ 
  shows  $dist\ x\ a \leq dist\ a\ b \wedge dist\ x\ b \leq dist\ a\ b$ 
proof (intro conjI)
  obtain u where  $0 \leq u \leq 1$  and  $x: x = (1 - u) *_R\ a + u *_R\ b$ 
    using assms by (force simp: in_segment algebra_simps)
  have  $dist\ x\ a = u * dist\ a\ b$ 
    apply (simp add: dist_norm algebra_simps x)
  by (metis <0 ≤ u> abs_of_nonneg norm_minus_commute norm_scaleR real_vector.scale_right_diff_distrib)
  also have  $\dots \leq dist\ a\ b$ 
    by (simp add: mult_left_le_one_le u)
  finally show  $dist\ x\ a \leq dist\ a\ b$  .
  have  $dist\ x\ b = norm\ ((1-u) *_R\ a - (1-u) *_R\ b)$ 
    by (simp add: dist_norm algebra_simps x)
  also have  $\dots = (1-u) * dist\ a\ b$ 
  proof -
    have  $norm\ ((1 - 1 * u) *_R\ (a - b)) = (1 - 1 * u) * norm\ (a - b)$ 

```

```

    using ⟨u ≤ 1⟩ by force
  then show ?thesis
    by (simp add: dist_norm real_vector.scale_right_diff_distrib)
qed
also have ... ≤ dist a b
  by (simp add: mult_left_le_one_le u)
finally show dist x b ≤ dist a b .
qed

lemma dist_in_open_segment:
  fixes a :: 'a :: euclidean_space
  assumes x ∈ open_segment a b
  shows dist x a < dist a b ∧ dist x b < dist a b
proof (intro conjI)
  obtain u where u: 0 < u u < 1 and x: x = (1 - u) *R a + u *R b
  using assms by (force simp: in_segment algebra_simps)
  have dist x a = u * dist a b
  apply (simp add: dist_norm algebra_simps x)
  by (metis abs_of_nonneg less_eq_real_def norm_minus_commute norm_scaleR
    real_vector.scale_right_diff_distrib ⟨0 < u⟩)
  also have *: ... < dist a b
  using assms mult_less_cancel_right2 u(2) by fastforce
  finally show dist x a < dist a b .
  have ab_ne0: dist a b ≠ 0
  using * by fastforce
  have dist x b = norm ((1-u) *R a - (1-u) *R b)
  by (simp add: dist_norm algebra_simps x)
  also have ... = (1-u) * dist a b
proof -
  have norm ((1 - 1 * u) *R (a - b)) = (1 - 1 * u) * norm (a - b)
  using ⟨u < 1⟩ by force
  then show ?thesis
    by (simp add: dist_norm real_vector.scale_right_diff_distrib)
qed
also have ... < dist a b
  using ab_ne0 ⟨0 < u⟩ by simp
finally show dist x b < dist a b .
qed

```

```

lemma dist_decreases_open_segment_0:
  fixes x :: 'a :: euclidean_space
  assumes x ∈ open_segment 0 b
  shows dist c x < dist c 0 ∨ dist c x < dist c b
proof (rule ccontr, clarsimp simp: not_less)
  obtain u where u: 0 ≠ b 0 < u u < 1 and x: x = u *R b
  using assms by (auto simp: in_segment)
  have xb: x • b < b • b
  using u x by auto
  assume norm c ≤ dist c x

```

```

then have  $c \cdot c \leq (c - x) \cdot (c - x)$ 
  by (simp add: dist_norm norm_le)
moreover have  $0 < x \cdot b$ 
  using  $u \ x$  by auto
ultimately have less:  $c \cdot b < x \cdot b$ 
  by (simp add: x algebra_simps inner_commute u)
assume  $\text{dist } c \ b \leq \text{dist } c \ x$ 
then have  $(c - b) \cdot (c - b) \leq (c - x) \cdot (c - x)$ 
  by (simp add: dist_norm norm_le)
then have  $(b \cdot b) * (1 - u * u) \leq 2 * (b \cdot c) * (1 - u)$ 
  by (simp add: x algebra_simps inner_commute)
then have  $(1 + u) * (b \cdot b) * (1 - u) \leq 2 * (b \cdot c) * (1 - u)$ 
  by (simp add: algebra_simps)
then have  $(1 + u) * (b \cdot b) \leq 2 * (b \cdot c)$ 
  using  $\langle u < 1 \rangle$  by auto
with  $xb$  have  $c \cdot b \geq x \cdot b$ 
  by (auto simp: x algebra_simps inner_commute)
with less show False by auto
qed

proposition dist_decreases_open_segment:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  assumes  $x \in \text{open\_segment } a \ b$ 
  shows  $\text{dist } c \ x < \text{dist } c \ a \vee \text{dist } c \ x < \text{dist } c \ b$ 
proof -
  have  $*$ :  $x - a \in \text{open\_segment } 0 \ (b - a)$  using  $\text{assms}$ 
  by (metis diff_self open_segment_translation_eq uminus_add_conv_diff)
  show ?thesis
  using dist_decreases_open_segment_0 [OF  $*$ , of  $c - a$ ]  $\text{assms}$ 
  by (simp add: dist_norm)
qed

corollary open_segment_furthest_le:
  fixes  $a \ b \ x \ y :: 'a :: \text{euclidean\_space}$ 
  assumes  $x \in \text{open\_segment } a \ b$ 
  shows  $\text{norm } (y - x) < \text{norm } (y - a) \vee \text{norm } (y - x) < \text{norm } (y - b)$ 
  by (metis  $\text{assms}$  dist_decreases_open_segment dist_norm)

corollary dist_decreases_closed_segment:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  assumes  $x \in \text{closed\_segment } a \ b$ 
  shows  $\text{dist } c \ x \leq \text{dist } c \ a \vee \text{dist } c \ x \leq \text{dist } c \ b$ 
apply (cases  $x \in \text{open\_segment } a \ b$ )
  using dist_decreases_open_segment less_eq_real_def apply blast
by (metis DiffI  $\text{assms}$  empty_iff insertE open_segment_def order_refl)

corollary segment_furthest_le:
  fixes  $a \ b \ x \ y :: 'a :: \text{euclidean\_space}$ 
  assumes  $x \in \text{closed\_segment } a \ b$ 

```

shows $\text{norm } (y - x) \leq \text{norm } (y - a) \vee \text{norm } (y - x) \leq \text{norm } (y - b)$
by (metis assms dist_decreases_closed_segment dist_norm)

lemma convex_intermediate_ball:

fixes $a :: 'a :: \text{euclidean\_space}$

shows $\llbracket \text{ball } a \ r \subseteq T; T \subseteq \text{cball } a \ r \rrbracket \implies \text{convex } T$

apply (simp add: convex_contains_open_segment, clarify)

by (metis (no_types, opaque_lifting) less_le_trans mem_ball mem_cball subsetCE
dist_decreases_open_segment)

lemma csegment_midpoint_subset: $\text{closed\_segment } (\text{midpoint } a \ b) \ b \subseteq \text{closed\_segment } a \ b$

apply (clarsimp simp: midpoint_def in_segment)

apply (rule_tac $x = (1 + u) / 2$ in exI)

apply (auto simp: algebra_simps add_divide_distrib diff_divide_distrib)

by (metis field_sum_of_halves scaleR_left.add)

lemma notin_segment_midpoint:

fixes $a :: 'a :: \text{euclidean\_space}$

shows $a \neq b \implies a \notin \text{closed\_segment } (\text{midpoint } a \ b) \ b$

by (auto simp: dist_midpoint dest!: dist_in_closed_segment)

More lemmas, especially for working with the underlying formula

lemma segment_eq_compose:

fixes $a :: 'a :: \text{real\_vector}$

shows $(\lambda u. (1 - u) *_{\mathbb{R}} a + u *_{\mathbb{R}} b) = (\lambda x. a + x) \circ (\lambda u. u *_{\mathbb{R}} (b - a))$

by (simp add: o_def algebra_simps)

lemma segment_degen_1:

fixes $a :: 'a :: \text{real\_vector}$

shows $(1 - u) *_{\mathbb{R}} a + u *_{\mathbb{R}} b = b \longleftrightarrow a = b \vee u = 1$

proof –

{ **assume** $(1 - u) *_{\mathbb{R}} a + u *_{\mathbb{R}} b = b$

then have $(1 - u) *_{\mathbb{R}} a = (1 - u) *_{\mathbb{R}} b$

by (simp add: algebra_simps)

then have $a = b \vee u = 1$

by simp

} **then show** ?thesis

by (auto simp: algebra_simps)

qed

lemma segment_degen_0:

fixes $a :: 'a :: \text{real\_vector}$

shows $(1 - u) *_{\mathbb{R}} a + u *_{\mathbb{R}} b = a \longleftrightarrow a = b \vee u = 0$

using segment_degen_1 [of 1-u b a]

by (auto simp: algebra_simps)

lemma add_scaleR_degen:


```

fixes  $a\ b :: 'a :: \text{real\_vector}$ 
assumes  $(u *_R b + v *_R a) = (u *_R a + v *_R b)$   $u \neq v$ 
shows  $a=b$ 
by (metis (no_types, opaque_lifting) add commute add_diff_eq diff_add_cancel
real_vector.scale_cancel_left real_vector.scale_left_diff_distrib assms)

```

```

lemma closed_segment_image_interval:
   $\text{closed\_segment } a\ b = (\lambda u. (1 - u) *_R a + u *_R b) \text{ ` } \{0..1\}$ 
by (auto simp: set_eq_iff image_iff closed_segment_def)

```

```

lemma open_segment_image_interval:
   $\text{open\_segment } a\ b = (\text{if } a=b \text{ then } \{\} \text{ else } (\lambda u. (1 - u) *_R a + u *_R b) \text{ ` } \{0 <..<1\})$ 
by (auto simp: open_segment_def closed_segment_def segment_degen_0 segment_degen_1)

```

```

lemmas segment_image_interval = closed_segment_image_interval open_segment_image_interval

```

```

lemma closed_segment_neq_empty [simp]:  $\text{closed\_segment } a\ b \neq \{\}$ 
by auto

```

```

lemma open_segment_eq_empty [simp]:  $\text{open\_segment } a\ b = \{\} \longleftrightarrow a = b$ 
proof -
  { assume  $a1: \text{open\_segment } a\ b = \{\}$ 
    have  $\{\} \neq \{0 :: \text{real} <..<1\}$ 
    by simp
    then have  $a = b$ 
    using  $a1$  open_segment_image_interval by fastforce
  } then show ?thesis by auto
qed

```

```

lemma open_segment_eq_empty' [simp]:  $\{\} = \text{open\_segment } a\ b \longleftrightarrow a = b$ 
using open_segment_eq_empty by blast

```

```

lemmas segment_eq_empty = closed_segment_neq_empty open_segment_eq_empty

```

```

lemma inj_segment:
  fixes  $a :: 'a :: \text{real\_vector}$ 
  assumes  $a \neq b$ 
  shows inj_on  $(\lambda u. (1 - u) *_R a + u *_R b)$   $I$ 
proof
  fix  $x\ y$ 
  assume  $(1 - x) *_R a + x *_R b = (1 - y) *_R a + y *_R b$ 
  then have  $x *_R (b - a) = y *_R (b - a)$ 
  by (simp add: algebra_simps)
  with assms show  $x = y$ 
  by (simp add: real_vector.scale_right_imp_eq)
qed

```

```

lemma finite_closed_segment [simp]: finite(closed_segment a b)  $\longleftrightarrow$  a = b
  apply auto
  apply (rule ccontr)
  apply (simp add: segment_image_interval)
  using infinite_Icc [OF zero_less_one] finite_imageD [OF inj_segment] apply
blast
done

```

```

lemma finite_open_segment [simp]: finite(open_segment a b)  $\longleftrightarrow$  a = b
  by (auto simp: open_segment_def)

```

```

lemmas finite_segment = finite_closed_segment finite_open_segment

```

```

lemma closed_segment_eq_sing: closed_segment a b = {c}  $\longleftrightarrow$  a = c  $\wedge$  b = c
  by auto

```

```

lemma open_segment_eq_sing: open_segment a b  $\neq$  {c}
  by (metis finite_insert finite_open_segment insert_not_empty open_segment_image_interval)

```

```

lemmas segment_eq_sing = closed_segment_eq_sing open_segment_eq_sing

```

```

lemma open_segment_bound1:
  assumes x  $\in$  open_segment a b
  shows norm (x - a) < norm (b - a)
proof -
  obtain u where x = (1 - u) *R a + u *R b 0 < u u < 1 a  $\neq$  b
  using assms by (auto simp add: open_segment_image_interval split: if_split_asm)
  then show norm (x - a) < norm (b - a)
    apply clarify
    apply (auto simp: algebra_simps)
    apply (simp add: scaleR_diff_right [symmetric])
  done
qed

```

```

lemma compact_segment [simp]:
  fixes a :: 'a::real_normed_vector
  shows compact (closed_segment a b)
  by (auto simp: segment_image_interval intro!: compact_continuous_image continuous_intros)

```

```

lemma closed_segment [simp]:
  fixes a :: 'a::real_normed_vector
  shows closed (closed_segment a b)
  by (simp add: compact_imp_closed)

```

```

lemma closure_closed_segment [simp]:
  fixes a :: 'a::real_normed_vector
  shows closure(closed_segment a b) = closed_segment a b
  by simp

```

```

lemma open_segment_bound:
  assumes  $x \in \text{open\_segment } a \ b$ 
  shows  $\text{norm } (x - a) < \text{norm } (b - a)$   $\text{norm } (x - b) < \text{norm } (b - a)$ 
apply (simp add: assms open_segment_bound1)
by (metis assms norm_minus_commute open_segment_bound1 open_segment_commute)

```

```

lemma closure_open_segment [simp]:
   $\text{closure } (\text{open\_segment } a \ b) = (\text{if } a = b \text{ then } \{\} \text{ else } \text{closed\_segment } a \ b)$ 
  for  $a :: 'a::\text{euclidean\_space}$ 
proof (cases  $a = b$ )
  case True
  then show ?thesis
  by simp
next
  case False
  have  $\text{closure } ((\lambda u. u *_R (b - a)) \ ' \{0 <..< 1\}) = (\lambda u. u *_R (b - a)) \ ' \text{closure } \{0 <..< 1\}$ 
  apply (rule closure_injective_linear_image [symmetric])
  apply (use False in ‹auto intro!: injI›)
  done
  then have closure
     $((\lambda u. (1 - u) *_R a + u *_R b) \ ' \{0 <..< 1\}) =$ 
     $(\lambda x. (1 - x) *_R a + x *_R b) \ ' \text{closure } \{0 <..< 1\}$ 
  using closure_translation [of  $a$   $((\lambda x. x *_R b - x *_R a) \ ' \{0 <..< 1\})$ ]
  by (simp add: segment_eq_compose field_simps scaleR_diff_left scaleR_diff_right image_image)
  then show ?thesis
  by (simp add: segment_image_interval closure_greaterThanLessThan [symmetric]
    del: closure_greaterThanLessThan)
qed

```

```

lemma closed_open_segment_iff [simp]:
  fixes  $a :: 'a::\text{euclidean\_space}$  shows  $\text{closed}(\text{open\_segment } a \ b) \longleftrightarrow a = b$ 
  by (metis open_segment_def DiffE closure_eq closure_open_segment_ends_in_segment(1)
    insert_iff segment_image_interval(2))

```

```

lemma compact_open_segment_iff [simp]:
  fixes  $a :: 'a::\text{euclidean\_space}$  shows  $\text{compact}(\text{open\_segment } a \ b) \longleftrightarrow a = b$ 
  by (simp add: bounded_open_segment compact_eq_bounded_closed)

```

```

lemma convex_closed_segment [iff]:  $\text{convex } (\text{closed\_segment } a \ b)$ 
  unfolding segment_convex_hull by(rule convex_convex_hull)

```

```

lemma convex_open_segment [iff]:  $\text{convex } (\text{open\_segment } a \ b)$ 
proof –
  have  $\text{convex } ((\lambda u. u *_R (b - a)) \ ' \{0 <..< 1\})$ 
  by (rule convex_linear_image) auto
  then have  $\text{convex } (+) \ a \ ' (\lambda u. u *_R (b - a)) \ ' \{0 <..< 1\}$ 

```

```

    by (rule convex_translation)
  then show ?thesis
    by (simp add: image_image open_segment_image_interval segment_eq_compose
field_simps scaleR_diff_left scaleR_diff_right)
qed

```

lemmas *convex_segment* = *convex_closed_segment* *convex_open_segment*

lemma *subset_closed_segment*:

$$\text{closed\_segment } a \ b \subseteq \text{closed\_segment } c \ d \longleftrightarrow$$

$$a \in \text{closed\_segment } c \ d \wedge b \in \text{closed\_segment } c \ d$$

by *auto* (*meson contra_subsetD convex_closed_segment convex_contains_segment*)

lemma *subset_co_segment*:

$$\text{closed\_segment } a \ b \subseteq \text{open\_segment } c \ d \longleftrightarrow$$

$$a \in \text{open\_segment } c \ d \wedge b \in \text{open\_segment } c \ d$$

using *closed_segment_subset* **by** *blast*

lemma *subset_open_segment*:

fixes *a* :: 'a::euclidean_space

shows *open_segment* *a* *b* $\subseteq$ *open_segment* *c* *d* $\longleftrightarrow$

$$a = b \vee a \in \text{closed\_segment } c \ d \wedge b \in \text{closed\_segment } c \ d$$

(**is** ?lhs = ?rhs)

proof (*cases* *a* = *b*)

case *True* **then show** ?thesis **by** *simp*

next

case *False* **show** ?thesis

proof

assume *rhs*: ?rhs

with $\langle a \neq b \rangle$ **have** $c \neq d$

using *closed_segment_idem_singleton_iff* **by** *auto*

have $\exists uc. (1 - u) *_R ((1 - ua) *_R c + ua *_R d) + u *_R ((1 - ub) *_R c +$
 $ub *_R d) =$

$$(1 - uc) *_R c + uc *_R d \wedge 0 < uc \wedge uc < 1$$

if *neg*: $(1 - ua) *_R c + ua *_R d \neq (1 - ub) *_R c + ub *_R d$ $c \neq d$

and $a = (1 - ua) *_R c + ua *_R d$ $b = (1 - ub) *_R c + ub *_R d$

and *u*: $0 < u$ $u < 1$ **and** *uab*: $0 \leq ua$ $ua \leq 1$ $0 \leq ub$ $ub \leq 1$

for *u* *ua* *ub*

proof —

have $ua \neq ub$

using *neg* **by** *auto*

moreover **have** $(u - 1) * ua \leq 0$ **using** *u uab*

by (*simp* *add: mult_nonpos_nonneg*)

ultimately **have** *lt*: $(u - 1) * ua < u * ub$ **using** *u uab*

by (*metis antisym_conv diff_ge_0_iff_ge le_less_trans mult_eq_0_iff*
mult_le_0_iff not_less)

have $p * ua + q * ub < p + q$ **if** *p*: $0 < p$ **and** *q*: $0 < q$ **for** *p* *q*

proof —

have $\neg p \leq 0 \neg q \leq 0$

```

      using p q not_less by blast+
    then show ?thesis
  by (metis ‹ua ≠ ub› add_less_cancel_left add_less_cancel_right add_mono_thms_linordered_field(5)
      less_eq_real_def mult_cancel_left1 mult_less_cancel_left2 uab(2)
uab(4))
  qed
  then have (1 - u) * ua + u * ub < 1 using u ‹ua ≠ ub›
    by (metis diff_add_cancel diff_gt_0_iff_gt)
  with lt show ?thesis
    by (rule_tac x=ua + u*(ub-ua) in exI) (simp add: algebra_simps)
  qed
  with rhs ‹a ≠ b› ‹c ≠ d› show ?lhs
    unfolding open_segment_image_interval closed_segment_def
    by (fastforce simp add:)
next
  assume lhs: ?lhs
  with ‹a ≠ b› have c ≠ d
    by (meson finite_open_segment rev_finite_subset)
  have closure (open_segment a b) ⊆ closure (open_segment c d)
    using lhs closure_mono by blast
  then have closed_segment a b ⊆ closed_segment c d
    by (simp add: ‹a ≠ b› ‹c ≠ d›)
  then show ?rhs
    by (force simp: ‹a ≠ b›)
  qed
qed

lemma subset_oc_segment:
  fixes a :: 'a::euclidean_space
  shows open_segment a b ⊆ closed_segment c d ⟷
    a = b ∨ a ∈ closed_segment c d ∧ b ∈ closed_segment c d
apply (simp add: subset_open_segment [symmetric])
apply (rule iffI)
  apply (metis closure_closed_segment closure_mono closure_open_segment sub-
    set_closed_segment subset_open_segment)
  apply (meson dual_order.trans segment_open_subset_closed)
done

lemmas subset_segment = subset_closed_segment subset_co_segment subset_oc_segment
subset_open_segment

lemma dist_half_times2:
  fixes a :: 'a :: real_normed_vector
  shows dist ((1 / 2) *R (a + b)) x * 2 = dist (a+b) (2 *R x)
proof -
  have norm ((1 / 2) *R (a + b) - x) * 2 = norm (2 *R ((1 / 2) *R (a + b) -
x))
    by simp
  also have ... = norm ((a + b) - 2 *R x)

```

```

    by (simp add: real_vector.scale_right_diff_distrib)
  finally show ?thesis
    by (simp only: dist_norm)
qed

lemma closed_segment_as_ball:
  closed_segment a b = affine_hull {a,b} ∩ cball(inverse 2 *R (a + b))(norm(b
- a) / 2)
proof (cases b = a)
  case True then show ?thesis by (auto simp: hull_inc)
next
  case False
  then have *: ((∃ u v. x = u *R a + v *R b ∧ u + v = 1) ∧
    dist ((1 / 2) *R (a + b)) x * 2 ≤ norm (b - a)) =
    (∃ u. x = (1 - u) *R a + u *R b ∧ 0 ≤ u ∧ u ≤ 1) for x
  proof -
    have ((∃ u v. x = u *R a + v *R b ∧ u + v = 1) ∧
      dist ((1 / 2) *R (a + b)) x * 2 ≤ norm (b - a)) =
      ((∃ u. x = (1 - u) *R a + u *R b) ∧
        dist ((1 / 2) *R (a + b)) x * 2 ≤ norm (b - a))
    unfolding eq_diff_eq [symmetric] by simp
    also have ... = (∃ u. x = (1 - u) *R a + u *R b ∧
      norm ((a+b) - (2 *R x)) ≤ norm (b - a))
    by (simp add: dist_half_times2) (simp add: dist_norm)
    also have ... = (∃ u. x = (1 - u) *R a + u *R b ∧
      norm ((a+b) - (2 *R ((1 - u) *R a + u *R b))) ≤ norm (b - a))
    by auto
    also have ... = (∃ u. x = (1 - u) *R a + u *R b ∧
      norm ((1 - u * 2) *R (b - a)) ≤ norm (b - a))
    by (simp add: algebra_simps scaleR_2)
    also have ... = (∃ u. x = (1 - u) *R a + u *R b ∧
      |1 - u * 2| * norm (b - a) ≤ norm (b - a))
    by simp
    also have ... = (∃ u. x = (1 - u) *R a + u *R b ∧ |1 - u * 2| ≤ 1)
    by (simp add: mult_le_cancel_right2 False)
    also have ... = (∃ u. x = (1 - u) *R a + u *R b ∧ 0 ≤ u ∧ u ≤ 1)
    by auto
    finally show ?thesis .
  qed
show ?thesis
  by (simp add: affine_hull_2 Set.set_eq_iff closed_segment_def *)
qed

lemma open_segment_as_ball:
  open_segment a b =
    affine_hull {a,b} ∩ ball(inverse 2 *R (a + b))(norm(b - a) / 2)
proof (cases b = a)
  case True then show ?thesis by (auto simp: hull_inc)
next

```

```

case False
then have *: (( $\exists u v. x = u *_R a + v *_R b \wedge u + v = 1$ )  $\wedge$ 
 $\text{dist } ((1 / 2) *_R (a + b)) x * 2 < \text{norm } (b - a)$ ) =
( $\exists u. x = (1 - u) *_R a + u *_R b \wedge 0 < u \wedge u < 1$ ) for x
proof -
  have (( $\exists u v. x = u *_R a + v *_R b \wedge u + v = 1$ )  $\wedge$ 
 $\text{dist } ((1 / 2) *_R (a + b)) x * 2 < \text{norm } (b - a)$ ) =
    (( $\exists u. x = (1 - u) *_R a + u *_R b$ )  $\wedge$ 
 $\text{dist } ((1 / 2) *_R (a + b)) x * 2 < \text{norm } (b - a)$ )
  unfolding eq_diff_eq [symmetric] by simp
  also have ... = ( $\exists u. x = (1 - u) *_R a + u *_R b \wedge$ 
 $\text{norm } ((a+b) - (2 *_R x)) < \text{norm } (b - a)$ )
  by (simp add: dist_half_times2) (simp add: dist_norm)
  also have ... = ( $\exists u. x = (1 - u) *_R a + u *_R b \wedge$ 
 $\text{norm } ((a+b) - (2 *_R ((1 - u) *_R a + u *_R b))) < \text{norm } (b - a)$ )
  by auto
  also have ... = ( $\exists u. x = (1 - u) *_R a + u *_R b \wedge$ 
 $\text{norm } ((1 - u * 2) *_R (b - a)) < \text{norm } (b - a)$ )
  by (simp add: algebra_simps scaleR_2)
  also have ... = ( $\exists u. x = (1 - u) *_R a + u *_R b \wedge$ 
 $|1 - u * 2| * \text{norm } (b - a) < \text{norm } (b - a)$ )
  by simp
  also have ... = ( $\exists u. x = (1 - u) *_R a + u *_R b \wedge |1 - u * 2| < 1$ )
  by (simp add: mult_le_cancel_right2 False)
  also have ... = ( $\exists u. x = (1 - u) *_R a + u *_R b \wedge 0 < u \wedge u < 1$ )
  by auto
  finally show ?thesis .
qed
show ?thesis
using False by (force simp: affine_hull_2 Set.set_eq_iff open_segment_image_interval
*)
qed

lemmas segment_as_ball = closed_segment_as_ball open_segment_as_ball

lemma connected_segment [iff]:
  fixes x :: 'a :: real_normed_vector
  shows connected (closed_segment x y)
  by (simp add: convex_connected)

lemma is_interval_closed_segment_1 [intro, simp]: is_interval (closed_segment
a b) for a b::real
  unfolding closed_segment_eq_real_ivl
  by (auto simp: is_interval_def)

lemma IVT'_closed_segment_real:
  fixes f :: real  $\Rightarrow$  real
  assumes y  $\in$  closed_segment (f a) (f b)
  assumes continuous_on (closed_segment a b) f

```

shows $\exists x \in \text{closed\_segment } a \ b. f x = y$
using $\text{IVT}[of f a y b]$
 $\text{IVT}[of -f a -y b]$
 $\text{IVT}[of f b y a]$
 $\text{IVT}[of -f b -y a]$ *assms*
by (*cases* $a \leq b$; *cases* $f b \geq f a$) (*auto simp: closed_segment_eq_real_ivl continuous_on_minus*)

4.4.4 Betweenness

definition $\text{between} = (\lambda(a,b) x. x \in \text{closed\_segment } a \ b)$

lemma *betweenI*:

assumes $0 \leq u \ u \leq 1 \ x = (1 - u) *_R a + u *_R b$
shows $\text{between } (a, b) \ x$

using *assms* **unfolding** *between_def closed_segment_def* **by** *auto*

lemma *betweenE*:

assumes $\text{between } (a, b) \ x$
obtains u **where** $0 \leq u \ u \leq 1 \ x = (1 - u) *_R a + u *_R b$

using *assms* **unfolding** *between_def closed_segment_def* **by** *auto*

lemma *between_implies_scaled_diff*:

assumes $\text{between } (S, T) \ X \ \text{between } (S, T) \ Y \ S \neq Y$
obtains c **where** $(X - Y) = c *_R (S - Y)$

proof –

from $\langle \text{between } (S, T) \ X \rangle$ **obtain** u_X **where** $X = u_X *_R S + (1 - u_X) *_R T$

by (*metis add.commute betweenE eq_diff_eq*)

from $\langle \text{between } (S, T) \ Y \rangle$ **obtain** u_Y **where** $Y = u_Y *_R S + (1 - u_Y) *_R T$

by (*metis add.commute betweenE eq_diff_eq*)

have $X - Y = (u_X - u_Y) *_R (S - T)$

proof –

from $X \ Y$ **have** $X - Y = u_X *_R S - u_Y *_R S + ((1 - u_X) *_R T - (1 - u_Y) *_R T)$ **by** *simp*

also have $\dots = (u_X - u_Y) *_R S - (u_X - u_Y) *_R T$ **by** (*simp add: scaleR_left.diff*)

finally show *?thesis* **by** (*simp add: real_vector.scale_right_diff_distrib*)

qed

moreover from Y **have** $S - Y = (1 - u_Y) *_R (S - T)$

by (*simp add: real_vector.scale_left_diff_distrib real_vector.scale_right_diff_distrib*)

moreover note $\langle S \neq Y \rangle$

ultimately have $(X - Y) = ((u_X - u_Y) / (1 - u_Y)) *_R (S - Y)$ **by** *auto*

from this that show thesis **by** *blast*

qed

lemma *between_mem_segment*: $\text{between } (a,b) \ x \longleftrightarrow x \in \text{closed\_segment } a \ b$

unfolding *between_def* **by** *auto*


```

lemma between: between (a, b) (x::'a::euclidean_space)  $\longleftrightarrow$  dist a b = (dist a x)
+ (dist x b)
proof (cases a = b)
  case True
    then show ?thesis
      by (auto simp add: between_def dist_commute)
  next
    case False
      then have Fal: norm (a - b)  $\neq$  0 and Fal2: norm (a - b) > 0
        by auto
      have *:  $\bigwedge u. a - ((1 - u) *_R a + u *_R b) = u *_R (a - b)$ 
        by (auto simp add: algebra_simps)
      have norm (a - x) *R (x - b) = norm (x - b) *R (a - x) if x = (1 - u) *R
a + u *R b 0 ≤ u u ≤ 1 for u
      proof -
        have *: a - x = u *R (a - b) x - b = (1 - u) *R (a - b)
          unfolding that(1) by (auto simp add: algebra_simps)
        show norm (a - x) *R (x - b) = norm (x - b) *R (a - x)
          unfolding norm_minus_commute[of x a] * using ⟨0 ≤ u⟩ ⟨u ≤ 1⟩
          by simp
      qed
      moreover have  $\exists u. x = (1 - u) *_R a + u *_R b \wedge 0 \leq u \wedge u \leq 1$  if dist a b
= dist a x + dist x b
      proof -
        let ?β = norm (a - x) / norm (a - b)
        show  $\exists u. x = (1 - u) *_R a + u *_R b \wedge 0 \leq u \wedge u \leq 1$ 
        proof (intro exI conjI)
          show ?β ≤ 1
            using Fal2 unfolding that[unfolded dist_norm] norm_ge_zero by auto
          show x = (1 - ?β) *R a + (?β) *R b
          proof (subst euclidean_eq_iff; intro ballI)
            fix i :: 'a'
            assume i: i ∈ Basis
            have ((1 - ?β) *R a + (?β) *R b) • i
              = ((norm (a - b) - norm (a - x)) * (a • i) + norm (a - x) * (b •
i)) / norm (a - b)
            using Fal by (auto simp add: field_simps inner_simps)
            also have ... = x • i
            apply (rule divide_eq_imp[OF Fal])
            unfolding that[unfolded dist_norm]
            using that[unfolded dist_triangle_eq] i
            apply (subst (asm) euclidean_eq_iff)
            apply (auto simp add: field_simps inner_simps)
            done
          finally show x • i = ((1 - ?β) *R a + (?β) *R b) • i
            by auto
        qed
      qed (use Fal2 in auto)

```

```

qed
ultimately show ?thesis
  by (force simp add: between_def closed_segment_def dist_triangle_eq)
qed

```

```

lemma between_midpoint:
  fixes a :: 'a::euclidean_space
  shows between (a,b) (midpoint a b) (is ?t1)
    and between (b,a) (midpoint a b) (is ?t2)
proof -
  have *:  $\bigwedge x y z. x = (1/2::real) *_R z \implies y = (1/2) *_R z \implies \text{norm } z = \text{norm } x + \text{norm } y$ 
  by auto
  show ?t1 ?t2
    unfolding between_midpoint_def dist_norm
    by (auto simp add: field_simps inner_simps euclidean_eq_iff [where 'a='a]
intro!: *)
qed

```

```

lemma between_mem_convex_hull:
  between (a,b) x  $\longleftrightarrow$   $x \in \text{convex hull } \{a,b\}$ 
  unfolding between_mem_segment segment_convex_hull ..

```

```

lemma between_triv_iff [simp]: between (a,a) b  $\longleftrightarrow$   $a=b$ 
  by (auto simp: between_def)

```

```

lemma between_triv1 [simp]: between (a,b) a
  by (auto simp: between_def)

```

```

lemma between_triv2 [simp]: between (a,b) b
  by (auto simp: between_def)

```

```

lemma between_commute:
  between (a,b) = between (b,a)
by (auto simp: between_def closed_segment_commute)

```

```

lemma between_antisym:
  fixes a :: 'a :: euclidean_space
  shows  $\llbracket \text{between } (b,c) a; \text{between } (a,c) b \rrbracket \implies a = b$ 
by (auto simp: between_dist_commute)

```

```

lemma between_trans:
  fixes a :: 'a :: euclidean_space
  shows  $\llbracket \text{between } (b,c) a; \text{between } (a,c) d \rrbracket \implies \text{between } (b,c) d$ 
  using dist_triangle2 [of b c d] dist_triangle3 [of b d a]
  by (auto simp: between_dist_commute)

```

```

lemma between_norm:
  fixes a :: 'a :: euclidean_space

```

shows $\text{between } (a,b) \ x \longleftrightarrow \text{norm}(x - a) *_{\mathbb{R}} (b - x) = \text{norm}(b - x) *_{\mathbb{R}} (x - a)$

by (auto simp: between dist_triangle_eq norm_minus_commute algebra_simps)

lemma *between_swap*:

fixes $A \ B \ X \ Y :: 'a::\text{euclidean\_space}$

assumes $\text{between } (A, B) \ X$

assumes $\text{between } (A, B) \ Y$

shows $\text{between } (X, B) \ Y \longleftrightarrow \text{between } (A, Y) \ X$

using *assms* **by** (auto simp add: between)

lemma *between_translation* [simp]: $\text{between } (a + y, a + z) \ (a + x) \longleftrightarrow \text{between } (y, z) \ x$

by (auto simp: between_def)

lemma *between_trans_2*:

fixes $a :: 'a :: \text{euclidean\_space}$

shows $\llbracket \text{between } (b,c) \ a; \text{between } (a,b) \ d \rrbracket \implies \text{between } (c,d) \ a$

by (metis between_commute between_swap between_trans)

lemma *between_scaleR_lift* [simp]:

fixes $v :: 'a::\text{euclidean\_space}$

shows $\text{between } (a *_{\mathbb{R}} v, b *_{\mathbb{R}} v) \ (c *_{\mathbb{R}} v) \longleftrightarrow v = 0 \vee \text{between } (a, b) \ c$

by (simp add: between dist_norm scaleR_left_diff_distrib [symmetric] distrib_right [symmetric])

lemma *between_1*:

fixes $x::\text{real}$

shows $\text{between } (a,b) \ x \longleftrightarrow (a \leq x \wedge x \leq b) \vee (b \leq x \wedge x \leq a)$

by (auto simp: between_mem_segment closed_segment_eq_real_ivl)

end

4.5 Limits on the Extended Real Number Line

theory *Extended_Real_Limits*

imports

Topology_Euclidean_Space

HOL-Library.Extended_Real

HOL-Library.Extended_Nonnegative_Real

HOL-Library.Indicator_Function

begin

lemma *compact_UNIV*:

$\text{compact } (\text{UNIV} :: 'a::\{\text{complete\_linorder}, \text{linorder\_topology}, \text{second\_countable\_topology}\} \text{ set})$

using *compact_complete_linorder*

by (auto simp: seq_compact_eq_compact [symmetric] seq_compact_def)

```

lemma compact_eq_closed:
  fixes S :: 'a::{complete_linorder,linorder_topology,second_countable_topology}
  set
  shows compact S  $\longleftrightarrow$  closed S
  using closed_Int_compact[of S, OF _ compact_UNIV] compact_imp_closed
  by auto

```

```

lemma closed_contains_Sup_cl:
  fixes S :: 'a::{complete_linorder,linorder_topology,second_countable_topology}
  set
  assumes closed S
  and S  $\neq$  {}
  shows Sup S  $\in$  S
proof -
  from compact_eq_closed[of S] compact_attains_sup[of S] assms
  obtain s where S: s  $\in$  S  $\forall t \in S. t \leq s$ 
  by auto
  then have Sup S = s
  by (auto intro!: Sup_eqI)
  with S show ?thesis
  by simp
qed

```

```

lemma closed_contains_Inf_cl:
  fixes S :: 'a::{complete_linorder,linorder_topology,second_countable_topology}
  set
  assumes closed S
  and S  $\neq$  {}
  shows Inf S  $\in$  S
proof -
  from compact_eq_closed[of S] compact_attains_inf[of S] assms
  obtain s where S: s  $\in$  S  $\forall t \in S. s \leq t$ 
  by auto
  then have Inf S = s
  by (auto intro!: Inf_eqI)
  with S show ?thesis
  by simp
qed

```

```

instance enat :: second_countable_topology
proof
  show  $\exists B::\text{enat set set}. \text{countable } B \wedge \text{open} = \text{generate\_topology } B$ 
  proof (intro exI conjI)
    show countable (range lessThan  $\cup$  range greaterThan::enat set set)
    by auto
  qed (simp add: open_enat_def)
qed

```

```

instance ereal :: second_countable_topology

```

```

proof (standard, intro exI conjI)
  let ?B = ( $\bigcup r \in \mathbb{Q}. \{\dots < r\}, \{r < \dots\}\} :: \text{ereal set set})
  show countable ?B
    by (auto intro: countable_rat)
  show open = generate_topology ?B
  proof (intro ext iffI)
    fix S :: ereal set
    assume open S
    then show generate_topology ?B S
      unfolding open_generated_order
    proof induct
      case (Basis b)
      then obtain e where b =  $\{\dots < e\} \vee b = \{e < \dots\}$ 
      by auto
      moreover have  $\{\dots < e\} = \bigcup \{\{\dots < x\} \mid x \in \mathbb{Q} \wedge x < e\} \{e < \dots\} = \bigcup \{\{x < \dots\} \mid x.$ 
 $x \in \mathbb{Q} \wedge e < x\}$ 
      by (auto dest: ereal_dense3
        simp del: ex_simps
        simp add: ex_simps[symmetric] conj_commute Rats_def image_iff)
      ultimately show ?case
      by (auto intro: generate_topology.intros)
    qed (auto intro: generate_topology.intros)
  next
    fix S
    assume generate_topology ?B S
    then show open S
      by induct auto
  qed
qed$ 
```

This is a copy from *ereal :: second_countable_topology*. Maybe find a common super class of topological spaces where the rational numbers are densely embedded ?

```

instance ennreal :: second_countable_topology
proof (standard, intro exI conjI)
  let ?B = ( $\bigcup r \in \mathbb{Q}. \{\{\dots < r\}, \{r < \dots\}\} :: \text{ennreal set set})
  show countable ?B
    by (auto intro: countable_rat)
  show open = generate_topology ?B
  proof (intro ext iffI)
    fix S :: ennreal set
    assume open S
    then show generate_topology ?B S
      unfolding open_generated_order
    proof induct
      case (Basis b)
      then obtain e where b =  $\{\dots < e\} \vee b = \{e < \dots\}$ 
      by auto
      moreover have  $\{\dots < e\} = \bigcup \{\{\dots < x\} \mid x \in \mathbb{Q} \wedge x < e\} \{e < \dots\} = \bigcup \{\{x < \dots\} \mid x.$$ 
```

```

 $x \in \mathbb{Q} \wedge e < x\}$ 
  by (auto dest: ennreal_rat_dense
      simp del: ex_simps
      simp add: ex_simps[symmetric] conj_commute Rats_def image_iff)
  ultimately show ?case
  by (auto intro: generate_topology.intros)
qed (auto intro: generate_topology.intros)
next
fix S
assume generate_topology ?B S
then show open S
  by induct auto
qed
qed

```

lemma *ereal_open_closed_aux*:

```

fixes S :: ereal set
assumes open S
and closed S
and S:  $(-\infty) \notin S$ 
shows  $S = \{\}$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then have *:  $\text{Inf } S \in S$ 

  by (metis assms(2) closed_contains_Inf_cl)
{
  assume  $\text{Inf } S = -\infty$ 
  then have False
    using * assms(3) by auto
}
moreover
{
  assume  $\text{Inf } S = \infty$ 
  then have  $S = \{\infty\}$ 
    by (metis Inf_eq_PInfty  $\langle S \neq \{\} \rangle$ )
  then have False
    by (metis assms(1) not_open_singleton)
}
moreover
{
  assume fin:  $|\text{Inf } S| \neq \infty$ 
  from ereal_open_cont_interval[OF assms(1) * fin]
  obtain e where  $e > 0 \ \{\text{Inf } S - e < \dots < \text{Inf } S + e\} \subseteq S$  .
  then obtain b where  $\text{Inf } S - e < b < \text{Inf } S$ 
    using fin ereal_between[of Inf S e] dense[of Inf S - e]
    by auto
  then have  $b \in \{\text{Inf } S - e < \dots < \text{Inf } S + e\}$ 
    using e fin ereal_between[of Inf S e]

```

```

    by auto
  then have  $b \in S$ 
    using  $e$  by auto
  then have False
    using  $b$  by (metis complete_lattice_class.Inf_lower leD)
}
ultimately show False
  by auto
qed

```

```

lemma ereal_open_closed:
  fixes  $S :: \text{ereal set}$ 
  shows  $\text{open } S \wedge \text{closed } S \longleftrightarrow S = \{\} \vee S = \text{UNIV}$ 
proof -
  {
    assume lhs:  $\text{open } S \wedge \text{closed } S$ 
    {
      assume  $-\infty \notin S$ 
      then have  $S = \{\}$ 
        using lhs ereal_open_closed_aux by auto
    }
    moreover
    {
      assume  $-\infty \in S$ 
      then have  $-S = \{\}$ 
        using lhs ereal_open_closed_aux[of  $-S$ ] by auto
    }
    ultimately have  $S = \{\} \vee S = \text{UNIV}$ 
      by auto
  }
  then show ?thesis
    by auto
qed

```

```

lemma ereal_open_atLeast:
  fixes  $x :: \text{ereal}$ 
  shows  $\text{open } \{x..\} \longleftrightarrow x = -\infty$ 
proof
  assume  $x = -\infty$ 
  then have  $\{x..\} = \text{UNIV}$ 
    by auto
  then show  $\text{open } \{x..\}$ 
    by auto
next
  assume  $\text{open } \{x..\}$ 
  then have  $\text{open } \{x..\} \wedge \text{closed } \{x..\}$ 
    by auto
  then have  $\{x..\} = \text{UNIV}$ 
    unfolding ereal_open_closed by auto

```

```

    then show  $x = -\infty$ 
    by (simp add: bot_ereal_def atLeast_eq_UNIV_iff)
qed

```

```

lemma mono_closed_real:
  fixes  $S :: \text{real set}$ 
  assumes mono:  $\forall y z. y \in S \wedge y \leq z \longrightarrow z \in S$ 
  and closed  $S$ 
  shows  $S = \{\}$   $\vee S = \text{UNIV}$   $\vee (\exists a. S = \{a.. \})$ 
proof -
  {
    assume  $S \neq \{\}$ 
    { assume ex:  $\exists B. \forall x \in S. B \leq x$ 
      then have *:  $\forall x \in S. \text{Inf } S \leq x$ 
        using cInf_lower[of _ S] ex by (metis bdd_below_def)
      then have  $\text{Inf } S \in S$ 
        apply (subst closed_contains_Inf)
        using ex  $\langle S \neq \{\} \rangle \langle \text{closed } S \rangle$ 
        apply auto
        done
      then have  $\forall x. \text{Inf } S \leq x \longleftrightarrow x \in S$ 
        using mono[rule_format, of Inf S] *
        by auto
      then have  $S = \{\text{Inf } S .. \}$ 
        by auto
      then have  $\exists a. S = \{a .. \}$ 
        by auto
    }
    moreover
    {
      assume  $\neg (\exists B. \forall x \in S. B \leq x)$ 
      then have nex:  $\forall B. \exists x \in S. x < B$ 
        by (simp add: not_le)
      {
        fix  $y$ 
        obtain  $x$  where  $x \in S$  and  $x < y$ 
          using nex by auto
        then have  $y \in S$ 
          using mono[rule_format, of x y] by auto
      }
      then have  $S = \text{UNIV}$ 
        by auto
    }
    ultimately have  $S = \text{UNIV} \vee (\exists a. S = \{a .. \})$ 
      by blast
  }
  then show ?thesis
    by blast
qed

```



```

lemma mono_closed_ereal:
  fixes S :: real set
  assumes mono:  $\forall y z. y \in S \wedge y \leq z \longrightarrow z \in S$ 
    and closed S
  shows  $\exists a. S = \{x. a \leq \text{ereal } x\}$ 
proof -
  {
    assume S = {}
    then have ?thesis
      apply (rule_tac x=PInfty in exI)
      apply auto
      done
  }
  moreover
  {
    assume S = UNIV
    then have ?thesis
      apply (rule_tac x=- $\infty$  in exI)
      apply auto
      done
  }
  moreover
  {
    assume  $\exists a. S = \{a ..\}$ 
    then obtain a where  $S = \{a ..\}$ 
      by auto
    then have ?thesis
      apply (rule_tac x=ereal a in exI)
      apply auto
      done
  }
  ultimately show ?thesis
    using mono_closed_real[of S] assms by auto
qed

```

```

lemma Liminf_within:
  fixes f :: 'a::metric_space  $\Rightarrow$  'b::complete_lattice
  shows  $\text{Liminf (at } x \text{ within } S) f = (\text{SUP } e \in \{0 < ..\}. \text{INF } y \in (S \cap \text{ball } x \text{ } e - \{x\}).$ 
     $f y)$ 
  unfolding Liminf_def eventually_at
proof (rule SUP_eq, simp_all add: Ball_def Bex_def, safe)
  fix P d
  assume  $0 < d$  and  $\forall y. y \in S \longrightarrow y \neq x \wedge \text{dist } y \text{ } x < d \longrightarrow P y$ 
  then have  $S \cap \text{ball } x \text{ } d - \{x\} \subseteq \{x. P x\}$ 
    by (auto simp: dist_commute)
  then show  $\exists r > 0. \text{Inf } (f ` (\text{Collect } P)) \leq \text{Inf } (f ` (S \cap \text{ball } x \text{ } r - \{x\}))$ 
    by (intro exI[of _ d] INF_mono conjI <0 < d>) auto
next

```

```

fix d :: real
assume 0 < d
then show  $\exists P. (\exists d>0. \forall xa. xa \in S \longrightarrow xa \neq x \wedge \text{dist } xa \ x < d \longrightarrow P \ x) \wedge$ 
   $\text{Inf } (f \text{ ` } (S \cap \text{ball } x \ d - \{x\})) \leq \text{Inf } (f \text{ ` } (\text{Collect } P))$ 
  by (intro exI[of _  $\lambda y. y \in S \cap \text{ball } x \ d - \{x\}$ ])
    (auto intro!: INF_mono exI[of _ d] simp: dist_commute)
qed

```

```

lemma Limsup_within:
  fixes f :: 'a::metric_space  $\Rightarrow$  'b::complete_lattice
  shows  $\text{Limsup } (\text{at } x \text{ within } S) \ f = (\text{INF } e \in \{0 < ..\}. \text{SUP } y \in (S \cap \text{ball } x \ e - \{x\}).$ 
     $f \ y)$ 
  unfolding Limsup_def eventually_at
proof (rule INF_eq, simp_all add: Ball_def Bex_def, safe)
  fix P d
  assume 0 < d and  $\forall y. y \in S \longrightarrow y \neq x \wedge \text{dist } y \ x < d \longrightarrow P \ y$ 
  then have  $S \cap \text{ball } x \ d - \{x\} \subseteq \{x. P \ x\}$ 
    by (auto simp: dist_commute)
  then show  $\exists r>0. \text{Sup } (f \text{ ` } (S \cap \text{ball } x \ r - \{x\})) \leq \text{Sup } (f \text{ ` } (\text{Collect } P))$ 
    by (intro exI[of _ d] SUP_mono conjI  $\langle 0 < d \rangle$  auto)
next
  fix d :: real
  assume 0 < d
  then show  $\exists P. (\exists d>0. \forall xa. xa \in S \longrightarrow xa \neq x \wedge \text{dist } xa \ x < d \longrightarrow P \ xa) \wedge$ 
     $\text{Sup } (f \text{ ` } (\text{Collect } P)) \leq \text{Sup } (f \text{ ` } (S \cap \text{ball } x \ d - \{x\}))$ 
    by (intro exI[of _  $\lambda y. y \in S \cap \text{ball } x \ d - \{x\}$ ])
      (auto intro!: SUP_mono exI[of _ d] simp: dist_commute)
qed

```

```

lemma Liminf_at:
  fixes f :: 'a::metric_space  $\Rightarrow$  'b::complete_lattice
  shows  $\text{Liminf } (\text{at } x) \ f = (\text{SUP } e \in \{0 < ..\}. \text{INF } y \in (\text{ball } x \ e - \{x\}). f \ y)$ 
  using Liminf_within[of x UNIV f] by simp

```

```

lemma Limsup_at:
  fixes f :: 'a::metric_space  $\Rightarrow$  'b::complete_lattice
  shows  $\text{Limsup } (\text{at } x) \ f = (\text{INF } e \in \{0 < ..\}. \text{SUP } y \in (\text{ball } x \ e - \{x\}). f \ y)$ 
  using Limsup_within[of x UNIV f] by simp

```

```

lemma min_Liminf_at:
  fixes f :: 'a::metric_space  $\Rightarrow$  'b::complete_linorder
  shows  $\min (f \ x) (\text{Liminf } (\text{at } x) \ f) = (\text{SUP } e \in \{0 < ..\}. \text{INF } y \in \text{ball } x \ e. f \ y)$ 
  apply (simp add: inf_min [symmetric] Liminf_at)
  apply (subst inf_commute)
  apply (subst SUP_inf)
  apply auto
  apply (metis (no_types, lifting) INF_insert centre_in_ball greaterThan_iff im-
    age_cong inf_commute insert_Diff)
  done

```

4.5.1 Extended-Real.thy

```

lemma sum_constant_ereal:
  fixes a::ereal
  shows  $(\sum_{i \in I. a} = a * \text{card } I$ 
apply (cases finite I, induct set: finite, simp_all)
apply (cases a, auto, metis (no_types, opaque_lifting) add.commute mult.commute
semiring_normalization_rules(3))
done

```

```

lemma real_lim_then_eventually_real:
  assumes  $(u \longrightarrow \text{ereal } l) F$ 
  shows eventually  $(\lambda n. u\ n = \text{ereal}(\text{real\_of\_ereal}(u\ n))) F$ 
proof –
  have  $\text{ereal } l \in \{-\infty < .. < (\infty :: \text{ereal})\}$  by simp
  moreover have open  $\{-\infty < .. < (\infty :: \text{ereal})\}$  by simp
  ultimately have eventually  $(\lambda n. u\ n \in \{-\infty < .. < (\infty :: \text{ereal})\}) F$  using assms
tendsto_def by blast
  moreover have  $\bigwedge x. x \in \{-\infty < .. < (\infty :: \text{ereal})\} \implies x = \text{ereal}(\text{real\_of\_ereal } x)$ 
using ereal_real by auto
  ultimately show ?thesis by (metis (mono_tags, lifting) eventually_mono)
qed

```

```

lemma ereal_Inf_cmult:
  assumes  $c > (0 :: \text{real})$ 
  shows  $\text{Inf } \{\text{ereal } c * x \mid x. P\ x\} = \text{ereal } c * \text{Inf } \{x. P\ x\}$ 
proof –
  have  $(\lambda x :: \text{ereal}. c * x) (\text{Inf } \{x :: \text{ereal}. P\ x\}) = \text{Inf } ((\lambda x :: \text{ereal}. c * x) \{x :: \text{ereal}.$ 
P x\})
  apply (rule mono_bij_Inf)
  apply (simp add: assms_ereal_mult_left_mono less_imp_le mono_def)
  apply (rule bij_betw_byWitness[of _  $\lambda x. (x :: \text{ereal}) / c$ , auto simp add: assms
ereal_mult_divide])
  using assms_ereal_divide_eq apply auto
  done
  then show ?thesis by (simp only: setcompr_eq_image[symmetric])
qed

```

Continuity of addition

The next few lemmas remove an unnecessary assumption in *tendsto_add_ereal*, culminating in *tendsto_add_ereal_general* which essentially says that the addition is continuous on *ereal* times *ereal*, except at $(-\infty, \infty)$ and $(\infty, -\infty)$. It is much more convenient in many situations, see for instance the proof of *tendsto_sum_ereal* below.

```

lemma tendsto_add_ereal_PInf:
  fixes y :: ereal
  assumes y:  $y \neq -\infty$ 
  assumes f:  $(f \longrightarrow \infty) F$  and g:  $(g \longrightarrow y) F$ 

```

```

    shows  $((\lambda x. f\ x + g\ x) \longrightarrow \infty)\ F$ 
  proof -
    have  $\exists C. \text{eventually } (\lambda x. g\ x > \text{ereal } C)\ F$ 
  proof (cases y)
    case (real r)
      have  $y > y-1$  using y real by (simp add: ereal_between(1))
      then have eventually  $(\lambda x. g\ x > y - 1)\ F$  using g y order_tendsto_iff by
    auto
      moreover have  $y-1 = \text{ereal}(\text{real\_of\_ereal}(y-1))$ 
      by (metis real_ereal_eq_1(1) ereal_minus(1) real_of_ereal.simps(1))
      ultimately have eventually  $(\lambda x. g\ x > \text{ereal}(\text{real\_of\_ereal}(y - 1)))\ F$  by
    simp
      then show ?thesis by auto
    next
      case (PInf)
      have eventually  $(\lambda x. g\ x > \text{ereal } 0)\ F$  using g PInf by (simp add: tend-
    sto_PInfty)
      then show ?thesis by auto
    qed (simp add: y)
    then obtain  $C::\text{real}$  where ge: eventually  $(\lambda x. g\ x > \text{ereal } C)\ F$  by auto

    {
      fix  $M::\text{real}$ 
      have eventually  $(\lambda x. f\ x > \text{ereal}(M - C))\ F$  using f by (simp add: tend-
    sto_PInfty)
      then have eventually  $(\lambda x. (f\ x > \text{ereal}(M-C)) \wedge (g\ x > \text{ereal } C))\ F$ 
      by (auto simp add: ge eventually_conj_iff)
      moreover have  $\bigwedge x. ((f\ x > \text{ereal}(M-C)) \wedge (g\ x > \text{ereal } C)) \implies (f\ x + g\ x$ 
    >  $\text{ereal } M)$ 
      using ereal_add_strict_mono2 by fastforce
      ultimately have eventually  $(\lambda x. f\ x + g\ x > \text{ereal } M)\ F$  using eventu-
    ally_mono by force
    }
    then show ?thesis by (simp add: tendsto_PInfty)
  qed

```

One would like to deduce the next lemma from the previous one, but the fact that $-(x + y)$ is in general different from $(-x) + (-y)$ in *ereal* creates difficulties, so it is more efficient to copy the previous proof.

```

lemma tendsto_add_ereal_MInf:
  fixes y :: ereal
  assumes y:  $y \neq \infty$ 
  assumes f:  $(f \longrightarrow -\infty)\ F$  and g:  $(g \longrightarrow y)\ F$ 
  shows  $((\lambda x. f\ x + g\ x) \longrightarrow -\infty)\ F$ 
proof -
  have  $\exists C. \text{eventually } (\lambda x. g\ x < \text{ereal } C)\ F$ 
proof (cases y)
  case (real r)
    have  $y < y+1$  using y real by (simp add: ereal_between(1))

```

```

    then have eventually ( $\lambda x. g\ x < y + 1$ )  $F$  using  $g\ y\ order\_tendsto\_iff$  by
    force
    moreover have  $y+1 = ereal(real\_of\_ereal\ (y+1))$  by (simp add: real)
    ultimately have eventually ( $\lambda x. g\ x < ereal(real\_of\_ereal(y + 1))$ )  $F$  by
    simp
    then show ?thesis by auto
  next
    case (MInf)
    have eventually ( $\lambda x. g\ x < ereal\ 0$ )  $F$  using  $g\ MInf$  by (simp add: tend-
    sto_MInfty)
    then show ?thesis by auto
  qed (simp add: y)
  then obtain  $C::real$  where  $ge: eventually\ (\lambda x. g\ x < ereal\ C)\ F$  by auto

  {
    fix  $M::real$ 
    have eventually ( $\lambda x. f\ x < ereal(M - C)$ )  $F$  using  $f$  by (simp add: tend-
    sto_MInfty)
    then have eventually ( $\lambda x. (f\ x < ereal\ (M - C)) \wedge (g\ x < ereal\ C)$ )  $F$ 
    by (auto simp add: ge eventually_conj_iff)
    moreover have  $\bigwedge x. ((f\ x < ereal\ (M - C)) \wedge (g\ x < ereal\ C)) \implies (f\ x + g\ x$ 
     $< ereal\ M)$ 
    using  $ereal\_add\_strict\_mono2$  by fastforce
    ultimately have eventually ( $\lambda x. f\ x + g\ x < ereal\ M$ )  $F$  using eventu-
    ally_mono by force
  }
  then show ?thesis by (simp add: tendsto_MInfty)
qed

lemma tendsto_add_ereal_general1:
  fixes  $x\ y :: ereal$ 
  assumes  $y: |y| \neq \infty$ 
  assumes  $f: (f \longrightarrow x)\ F$  and  $g: (g \longrightarrow y)\ F$ 
  shows  $((\lambda x. f\ x + g\ x) \longrightarrow x + y)\ F$ 
proof (cases  $x$ )
  case (real  $r$ )
  have  $a: |x| \neq \infty$  by (simp add: real)
  show ?thesis by (rule tendsto_add_ereal[OF  $a, OF\ y, OF\ f, OF\ g$ ])
next
  case PInf
  then show ?thesis using tendsto_add_ereal_PInf assms by force
next
  case MInf
  then show ?thesis using tendsto_add_ereal_MInf assms
  by (metis  $abs\_ereal.simps(3)\ ereal\_MInfty\_eq\_plus$ )
qed

lemma tendsto_add_ereal_general2:
  fixes  $x\ y :: ereal$ 

```

```

assumes  $x: |x| \neq \infty$ 
and  $f: (f \longrightarrow x) F$  and  $g: (g \longrightarrow y) F$ 
shows  $((\lambda x. f\ x + g\ x) \longrightarrow x + y) F$ 
proof -
  have  $((\lambda x. g\ x + f\ x) \longrightarrow x + y) F$ 
  using tendsto_add_ereal_general1 [OF x, OF g, OF f] add.commute [of y, of
x] by simp
  moreover have  $\bigwedge x. g\ x + f\ x = f\ x + g\ x$  using add.commute by auto
  ultimately show ?thesis by simp
qed

```

The next lemma says that the addition is continuous on *ereal*, except at the pairs $(-\infty, \infty)$ and $(\infty, -\infty)$.

```

lemma tendsto_add_ereal_general [tendsto_intros]:
  fixes  $x\ y :: \text{ereal}$ 
  assumes  $\neg((x=\infty \wedge y=-\infty) \vee (x=-\infty \wedge y=\infty))$ 
  and  $f: (f \longrightarrow x) F$  and  $g: (g \longrightarrow y) F$ 
  shows  $((\lambda x. f\ x + g\ x) \longrightarrow x + y) F$ 
proof (cases x)
  case (real r)
  show ?thesis
    apply (rule tendsto_add_ereal_general2) using real assms by auto
next
  case (PInf)
  then have  $y \neq -\infty$  using assms by simp
  then show ?thesis using tendsto_add_ereal_PInf PInf assms by auto
next
  case (MInf)
  then have  $y \neq \infty$  using assms by simp
  then show ?thesis using tendsto_add_ereal_MInf MInf f g by (metis ereal_MInfty_eq_plus)
qed

```

Continuity of multiplication

In the same way as for addition, we prove that the multiplication is continuous on *ereal* times *ereal*, except at $(\infty, 0)$ and $(-\infty, 0)$ and $(0, \infty)$ and $(0, -\infty)$, starting with specific situations.

```

lemma tendsto_mult_real_ereal:
  assumes  $(u \longrightarrow \text{ereal } l) F$   $(v \longrightarrow \text{ereal } m) F$ 
  shows  $((\lambda n. u\ n * v\ n) \longrightarrow \text{ereal } l * \text{ereal } m) F$ 
proof -
  have ureal: eventually  $(\lambda n. u\ n = \text{ereal}(\text{real\_of\_ereal}(u\ n))) F$  by (rule real_lim_then_eventually_real
assms(1))
  then have  $((\lambda n. \text{ereal}(\text{real\_of\_ereal}(u\ n))) \longrightarrow \text{ereal } l) F$  using assms by
auto
  then have limu:  $((\lambda n. \text{real\_of\_ereal}(u\ n)) \longrightarrow l) F$  by auto
  have vreal: eventually  $(\lambda n. v\ n = \text{ereal}(\text{real\_of\_ereal}(v\ n))) F$  by (rule real_lim_then_eventually_real
assms(2))

```

```

then have (( $\lambda n$ .  $\text{ereal}(\text{real\_of\_ereal}(v\ n))$ )  $\longrightarrow$   $\text{ereal } m$ )  $F$  using  $\text{assms}$  by
auto
then have  $\text{limv}$ : (( $\lambda n$ .  $\text{real\_of\_ereal}(v\ n)$ )  $\longrightarrow$   $m$ )  $F$  by  $\text{auto}$ 

{
  fix  $n$  assume  $u\ n = \text{ereal}(\text{real\_of\_ereal}(u\ n))$   $v\ n = \text{ereal}(\text{real\_of\_ereal}(v\ n))$ 
  then have  $\text{ereal}(\text{real\_of\_ereal}(u\ n) * \text{real\_of\_ereal}(v\ n)) = u\ n * v\ n$  by
  ( $\text{metis times\_ereal.simps}(1)$ )
}
then have *:  $\text{eventually } (\lambda n$ .  $\text{ereal}(\text{real\_of\_ereal}(u\ n) * \text{real\_of\_ereal}(v\ n)) =$ 
 $u\ n * v\ n)$   $F$ 
using  $\text{eventually\_elim2}[OF\ \text{ureal vreal}]$  by  $\text{auto}$ 

have (( $\lambda n$ .  $\text{real\_of\_ereal}(u\ n) * \text{real\_of\_ereal}(v\ n)$ )  $\longrightarrow$   $l * m$ )  $F$  using
 $\text{tendsto\_mult}[OF\ \text{limu limv}]$  by  $\text{auto}$ 
then have (( $\lambda n$ .  $\text{ereal}(\text{real\_of\_ereal}(u\ n)) * \text{real\_of\_ereal}(v\ n)$ )  $\longrightarrow$   $\text{ereal}(l * m)$ )  $F$  by  $\text{auto}$ 
then show ?thesis using *  $\text{filterlim\_cong}$  by  $\text{fastforce}$ 
qed

```

lemma $\text{tendsto\_mult\_ereal\_PInf}$:

```

fixes  $f\ g::\_ \Rightarrow \text{ereal}$ 
assumes ( $f \longrightarrow l$ )  $F$   $l > 0$  ( $g \longrightarrow \infty$ )  $F$ 
shows (( $\lambda x$ .  $f\ x * g\ x$ )  $\longrightarrow \infty$ )  $F$ 
proof –
  obtain  $a::\text{real}$  where  $0 < \text{ereal } a < l$  using  $\text{assms}(2)$  using  $\text{ereal\_dense2}$  by
   $\text{blast}$ 
  have *:  $\text{eventually } (\lambda x$ .  $f\ x > a$ )  $F$  using  $\langle a < l \rangle$   $\text{assms}(1)$  by ( $\text{simp add:}$ 
 $\text{order\_tendsto\_iff}$ )
  {
    fix  $K::\text{real}$ 
    define  $M$  where  $M = \max K\ 1$ 
    then have  $M > 0$  by  $\text{simp}$ 
    then have  $\text{ereal}(M/a) > 0$  using  $\langle \text{ereal } a > 0 \rangle$  by  $\text{simp}$ 
    then have  $\bigwedge x. ((f\ x > a) \wedge (g\ x > M/a)) \implies (f\ x * g\ x > \text{ereal } a * \text{ereal}(M/a))$ 
      using  $\text{ereal\_mult\_mono\_strict'}$  [where  $?c = M/a$ ,  $OF\ \langle 0 < \text{ereal } a \rangle$ ] by
       $\text{auto}$ 
    moreover have  $\text{ereal } a * \text{ereal}(M/a) = M$  using  $\langle \text{ereal } a > 0 \rangle$  by  $\text{simp}$ 
    ultimately have  $\bigwedge x. ((f\ x > a) \wedge (g\ x > M/a)) \implies (f\ x * g\ x > M)$  by  $\text{simp}$ 
    moreover have  $M \geq K$  unfolding  $M\_def$  by  $\text{simp}$ 
    ultimately have  $\text{Imp}: \bigwedge x. ((f\ x > a) \wedge (g\ x > M/a)) \implies (f\ x * g\ x > K)$ 
      using  $\text{ereal\_less\_eq}(3)$   $\text{le\_less\_trans}$  by  $\text{blast}$ 

    have  $\text{eventually } (\lambda x$ .  $g\ x > M/a$ )  $F$  using  $\text{assms}(3)$  by ( $\text{simp add: tendsto\_PInfy}$ )
    then have  $\text{eventually } (\lambda x$ .  $(f\ x > a) \wedge (g\ x > M/a)$ )  $F$ 
      using * by ( $\text{auto simp add: eventually\_conj\_iff}$ )
    then have  $\text{eventually } (\lambda x$ .  $f\ x * g\ x > K$ )  $F$  using  $\text{eventually\_mono Imp}$  by
     $\text{force}$ 
  }

```

```

    }
    then show ?thesis by (auto simp add: tendsto_PInf ty)
qed

lemma tendsto_mult_ereal_pos:
  fixes f g ::  $\_ \Rightarrow \text{ereal}$ 
  assumes (f  $\longrightarrow$  l) F (g  $\longrightarrow$  m) F l > 0 m > 0
  shows (( $\lambda x. f\ x * g\ x$ )  $\longrightarrow$  l * m) F
proof (cases)
  assume *: l =  $\infty \vee m = \infty$ 
  then show ?thesis
  proof (cases)
    assume m =  $\infty$ 
    then show ?thesis using tendsto_mult_ereal_PInf assms by auto
  next
    assume  $\neg(m = \infty)$ 
    then have l =  $\infty$  using * by simp
    then have (( $\lambda x. g\ x * f\ x$ )  $\longrightarrow$  l * m) F using tendsto_mult_ereal_PInf
    assms by auto
    moreover have  $\bigwedge x. g\ x * f\ x = f\ x * g\ x$  using mult.commute by auto
    ultimately show ?thesis by simp
  qed
next
  assume  $\neg(l = \infty \vee m = \infty)$ 
  then have l <  $\infty$  m <  $\infty$  by auto
  then obtain lr mr where l =  $\text{ereal}\ lr$  m =  $\text{ereal}\ mr$ 
    using <l>0> <m>0> by (metis  $\text{ereal\_cases}$   $\text{ereal\_less}(6)$  not_less_iff_gr_or_eq)
  then show ?thesis using tendsto_mult_real_ereal assms by auto
qed

```

We reduce the general situation to the positive case by multiplying by suitable signs. Unfortunately, as ereal is not a ring, all the neat sign lemmas are not available there. We give the bare minimum we need.

```

lemma  $\text{ereal\_sgn\_abs}$ :
  fixes l ::  $\text{ereal}$ 
  shows  $\text{sgn}(l) * l = \text{abs}(l)$ 
apply (cases l) by (auto simp add:  $\text{sgn\_if}$   $\text{ereal\_less\_uminus\_reorder}$ )

```

```

lemma  $\text{sgn\_squared\_ereal}$ :
  assumes l  $\neq$  (0 ::  $\text{ereal}$ )
  shows  $\text{sgn}(l) * \text{sgn}(l) = 1$ 
apply (cases l) using assms by (auto simp add:  $\text{one\_ereal\_def}$   $\text{sgn\_if}$ )

```

```

lemma tendsto_mult_ereal [tendsto_intros]:
  fixes f g ::  $\_ \Rightarrow \text{ereal}$ 
  assumes (f  $\longrightarrow$  l) F (g  $\longrightarrow$  m) F  $\neg((l=0 \wedge \text{abs}(m) = \infty) \vee (m=0 \wedge \text{abs}(l) = \infty))$ 
  shows (( $\lambda x. f\ x * g\ x$ )  $\longrightarrow$  l * m) F
proof (cases)

```



```

  assume  $l=0 \vee m=0$ 
  then have  $\text{abs}(l) \neq \infty \text{ abs}(m) \neq \infty$  using assms(3) by auto
  then obtain lr mr where  $l = \text{ereal } lr \text{ } m = \text{ereal } mr$  by auto
  then show ?thesis using tendsto_mult_real_ereal assms by auto
next
  have sgn_finite:  $\bigwedge a::\text{ereal}. \text{abs}(\text{sgn } a) \neq \infty$ 
    by (metis MInfty_neq_ereal(2) PInfty_neq_ereal(2) abs_eq_infinity_cases
        ereal_times(1) ereal_times(3) ereal_uminus_eq_reorder sgn_ereal.elims)
  then have sgn_finite2:  $\bigwedge a b::\text{ereal}. \text{abs}(\text{sgn } a * \text{sgn } b) \neq \infty$ 
    by (metis abs_eq_infinity_cases abs_ereal.simps(2) abs_ereal.simps(3) ereal_mult_eq_MInfty
        ereal_mult_eq_PInfty)
  assume  $\neg(l=0 \vee m=0)$ 
  then have  $l \neq 0 \text{ } m \neq 0$  by auto
  then have  $\text{abs}(l) > 0 \text{ abs}(m) > 0$ 
    by (metis abs_ereal_ge0 abs_ereal_less0 abs_ereal_pos ereal_uminus_uminus
        ereal_uminus_zero less_le not_less)
  then have  $\text{sgn}(l) * l > 0 \text{ sgn}(m) * m > 0$  using ereal_sgn_abs by auto
  moreover have  $((\lambda x. \text{sgn}(l) * f x) \longrightarrow (\text{sgn}(l) * l)) F$ 
    by (rule tendsto_cmult_ereal, auto simp add: sgn_finite assms(1))
  moreover have  $((\lambda x. \text{sgn}(m) * g x) \longrightarrow (\text{sgn}(m) * m)) F$ 
    by (rule tendsto_cmult_ereal, auto simp add: sgn_finite assms(2))
  ultimately have *:  $((\lambda x. (\text{sgn}(l) * f x) * (\text{sgn}(m) * g x)) \longrightarrow (\text{sgn}(l) * l) * (\text{sgn}(m) * m)) F$ 
    using tendsto_mult_ereal_pos by force
  have  $((\lambda x. (\text{sgn}(l) * \text{sgn}(m)) * ((\text{sgn}(l) * f x) * (\text{sgn}(m) * g x))) \longrightarrow (\text{sgn}(l) * \text{sgn}(m)) * ((\text{sgn}(l) * l) * (\text{sgn}(m) * m))) F$ 
    by (rule tendsto_cmult_ereal, auto simp add: sgn_finite2 *)
  moreover have  $\bigwedge x. (\text{sgn}(l) * \text{sgn}(m)) * ((\text{sgn}(l) * f x) * (\text{sgn}(m) * g x)) = f x * g x$ 
    by (metis mult.left_neutral sgn_squared_ereal[OF l ≠ 0] sgn_squared_ereal[OF m ≠ 0]
        mult.assoc mult.commute)
  moreover have  $(\text{sgn}(l) * \text{sgn}(m)) * ((\text{sgn}(l) * l) * (\text{sgn}(m) * m)) = l * m$ 
    by (metis mult.left_neutral sgn_squared_ereal[OF l ≠ 0] sgn_squared_ereal[OF m ≠ 0]
        mult.assoc mult.commute)
  ultimately show ?thesis by auto
qed

```

```

lemma tendsto_cmult_ereal_general [tendsto_intros]:
  fixes  $f::\_ \Rightarrow \text{ereal}$  and  $c::\text{ereal}$ 
  assumes  $(f \longrightarrow l) F \neg(l=0 \wedge \text{abs}(c) = \infty)$ 
  shows  $((\lambda x. c * f x) \longrightarrow c * l) F$ 
by (cases  $c = 0$ , auto simp add: assms tendsto_mult_ereal)

```

Continuity of division

```

lemma tendsto_inverse_ereal_PInf:
  fixes  $u::\_ \Rightarrow \text{ereal}$ 
  assumes  $(u \longrightarrow \infty) F$ 
  shows  $((\lambda x. 1 / u x) \longrightarrow 0) F$ 

```

```

proof –
{
  fix  $e::\text{real}$  assume  $e > 0$ 
  have  $1/e < \infty$  by auto
  then have eventually  $(\lambda n. u\ n > 1/e)$   $F$  using assms(1) by (simp add:
tendsto_PInfty)
  moreover
  {
    fix  $z::\text{ereal}$  assume  $z > 1/e$ 
    then have  $z > 0$  using  $\langle e > 0 \rangle$  using less_le_trans not_le by fastforce
    then have  $1/z \geq 0$  by auto
    moreover have  $1/z < e$  using  $\langle e > 0 \rangle$   $\langle z > 1/e \rangle$ 
    apply (cases z) apply auto
    by (metis (mono_tags, opaque_lifting) less_ereal.simps(2) less_ereal.simps(4)
divide_less_eq_ereal_divide_less_pos_ereal_less(4)
ereal_less_eq(4) less_le_trans mult_eq_0_iff not_le not_one_less_zero
times_ereal.simps(1))
    ultimately have  $1/z \geq 0$   $1/z < e$  by auto
  }
  ultimately have eventually  $(\lambda n. 1/u\ n < e)$   $F$  eventually  $(\lambda n. 1/u\ n \geq 0)$   $F$  by
(auto simp add: eventually_mono)
} note  $*$  = this
show ?thesis
proof (subst order_tendsto_iff, auto)
  fix  $a::\text{ereal}$  assume  $a < 0$ 
  then show eventually  $(\lambda n. 1/u\ n > a)$   $F$  using  $*(2)$  eventually_mono
less_le_trans linordered_field_no_ub by fastforce
next
  fix  $a::\text{ereal}$  assume  $a > 0$ 
  then obtain  $e::\text{real}$  where  $e > 0$   $a > e$  using ereal_dense2 ereal_less(2) by blast
  then have eventually  $(\lambda n. 1/u\ n < e)$   $F$  using  $*(1)$  by auto
  then show eventually  $(\lambda n. 1/u\ n < a)$   $F$  using  $\langle a > e \rangle$  by (metis (mono_tags,
lifting) eventually_mono less_trans)
qed
qed

```

The next lemma deserves to exist by itself, as it is so common and useful.

```

lemma tendsto_inverse_real [tendsto_intros]:
  fixes  $u::\_ \Rightarrow \text{real}$ 
  shows  $(u \longrightarrow l)$   $F \implies l \neq 0 \implies ((\lambda x. 1/u\ x) \longrightarrow 1/l)$   $F$ 
  using tendsto_inverse unfolding inverse_eq_divide .

```

```

lemma tendsto_inverse_ereal [tendsto_intros]:
  fixes  $u::\_ \Rightarrow \text{ereal}$ 
  assumes  $(u \longrightarrow l)$   $F$   $l \neq 0$ 
  shows  $((\lambda x. 1/u\ x) \longrightarrow 1/l)$   $F$ 
proof (cases l)
  case (real r)
  then have  $r \neq 0$  using assms(2) by auto

```

```

then have  $1/l = \text{ereal}(1/r)$  using real by (simp add: one_ereal_def)
define v where  $v = (\lambda n. \text{real\_of\_ereal}(u\ n))$ 
have ureal: eventually  $(\lambda n. u\ n = \text{ereal}(v\ n))$  F unfolding v_def using real_lim_then_eventually_real
assms(1) real by auto
then have  $((\lambda n. \text{ereal}(v\ n)) \longrightarrow \text{ereal } r)$  F using assms real v_def by auto
then have  $*$ :  $((\lambda n. v\ n) \longrightarrow r)$  F by auto
then have  $((\lambda n. 1/v\ n) \longrightarrow 1/r)$  F using  $\langle r \neq 0 \rangle$  tendsto_inverse_real by
auto
then have lim:  $((\lambda n. \text{ereal}(1/v\ n)) \longrightarrow 1/l)$  F using  $\langle 1/l = \text{ereal}(1/r) \rangle$  by
auto

have  $r \in -\{0\}$  open  $(-\{(0::\text{real})\})$  using  $\langle r \neq 0 \rangle$  by auto
then have eventually  $(\lambda n. v\ n \in -\{0\})$  F using  $*$  using topological_tendstoD
by blast
then have eventually  $(\lambda n. v\ n \neq 0)$  F by auto
moreover
{
  fix n assume H:  $v\ n \neq 0$   $u\ n = \text{ereal}(v\ n)$ 
  then have  $\text{ereal}(1/v\ n) = 1/\text{ereal}(v\ n)$  by (simp add: one_ereal_def)
  then have  $\text{ereal}(1/v\ n) = 1/u\ n$  using H(2) by simp
}
ultimately have eventually  $(\lambda n. \text{ereal}(1/v\ n) = 1/u\ n)$  F using ureal eventu-
ally_elim2 by force
with Lim_transform_eventually[OF lim this] show ?thesis by simp
next
case (PInf)
then have  $1/l = 0$  by auto
then show ?thesis using tendsto_inverse_ereal_PInf assms PInf by auto
next
case (MInf)
then have  $1/l = 0$  by auto
have  $1/z = -1/-z$  if  $z < 0$  for  $z::\text{ereal}$ 
apply (cases z) using divide_ereal_def  $\langle z < 0 \rangle$  by auto
moreover have eventually  $(\lambda n. u\ n < 0)$  F by (metis (no_types) MInf assms(1)
tendsto_MInf zero_ereal_def)
ultimately have  $*$ : eventually  $(\lambda n. -1/-u\ n = 1/u\ n)$  F by (simp add: eventu-
ally_mono)

define v where  $v = (\lambda n. -u\ n)$ 
have  $(v \longrightarrow \infty)$  F unfolding v_def using MInf assms(1) tendsto_uminus_ereal
by fastforce
then have  $((\lambda n. 1/v\ n) \longrightarrow 0)$  F using tendsto_inverse_ereal_PInf by auto
then have  $((\lambda n. -1/v\ n) \longrightarrow 0)$  F using tendsto_uminus_ereal by fastforce
then show ?thesis unfolding v_def using Lim_transform_eventually[OF _ *]
 $\langle 1/l = 0 \rangle$  by auto
qed

```

lemma *tendsto_divide_ereal* [*tendsto_intros*]:
fixes $f\ g::\_ \Rightarrow \text{ereal}$

```

    assumes  $(f \longrightarrow l) F (g \longrightarrow m) F m \neq 0 \neg(abs(l) = \infty \wedge abs(m) = \infty)$ 
    shows  $((\lambda x. f x / g x) \longrightarrow l / m) F$ 
  proof -
    define  $h$  where  $h = (\lambda x. 1 / g x)$ 
    have *:  $(h \longrightarrow 1/m) F$  unfolding  $h\_def$  using  $assms(2) assms(3) tendsto\_inverse\_ereal$  by auto
    have  $((\lambda x. f x * h x) \longrightarrow l * (1/m)) F$ 
    apply (rule  $tendsto\_mult\_ereal[OF assms(1) *]$ ) using  $assms(3) assms(4)$  by
    (auto simp add:  $divide\_ereal\_def$ )
    moreover have  $f x * h x = f x / g x$  for  $x$  unfolding  $h\_def$  by (simp add:  $divide\_ereal\_def$ )
    moreover have  $l * (1/m) = l/m$  by (simp add:  $divide\_ereal\_def$ )
    ultimately show  $?thesis$  unfolding  $h\_def$  using  $Lim\_transform\_eventually$ 
  by auto
qed

```

Further limits

The assumptions of $\llbracket |?x| \neq \infty; |?y| \neq \infty; (?f \longrightarrow ?x) ?F; (?g \longrightarrow ?y) ?F \rrbracket \implies ((\lambda x. ?f x - ?g x) \longrightarrow ?x - ?y) ?F$ are too strong, we weaken them here.

```

lemma  $tendsto\_diff\_ereal\_general$  [ $tendsto\_intros$ ]:
  fixes  $u v::'a \Rightarrow ereal$ 
  assumes  $(u \longrightarrow l) F (v \longrightarrow m) F \neg((l = \infty \wedge m = \infty) \vee (l = -\infty \wedge m = -\infty))$ 
  shows  $((\lambda n. u n - v n) \longrightarrow l - m) F$ 
  proof -
    have  $((\lambda n. u n + (-v n)) \longrightarrow l + (-m)) F$ 
    apply (intro  $tendsto\_intros$   $assms$ ) using  $assms$  by (auto simp add:  $ereal\_uminus\_eq\_reorder$ )
    then show  $?thesis$  by (simp add:  $minus\_ereal\_def$ )
  qed

```

```

lemma  $id\_nat\_ereal\_tendsto\_PInf$  [ $tendsto\_intros$ ]:
   $(\lambda n::nat. real n) \longrightarrow \infty$ 
  by (simp add:  $filterlim\_real\_sequentially tendsto\_PInf\_eq\_at\_top$ )

```

```

lemma  $tendsto\_at\_top\_pseudo\_inverse$  [ $tendsto\_intros$ ]:
  fixes  $u::nat \Rightarrow nat$ 
  assumes  $LIM\ n\ sequentially. u\ n\ :\>\ at\_top$ 
  shows  $LIM\ n\ sequentially. Inf\ \{N. u\ N\ \geq\ n\} :\>\ at\_top$ 
  proof -
    {
      fix  $C::nat$ 
      define  $M$  where  $M = Max\ \{u\ n\ |\ n. n \leq C\} + 1$ 
      {
        fix  $n$  assume  $n \geq M$ 
        have  $eventually\ (\lambda N. u\ N \geq n)\ sequentially$  using  $assms$ 
          by (simp add:  $filterlim\_at\_top$ )
        then have *:  $\{N. u\ N \geq n\} \neq \{\}$  by force
      }
    }
  qed

```

```

    have  $N > C$  if  $u\ N \geq n$  for  $N$ 
  proof (rule ccontr)
    assume  $\neg(N > C)$ 
    have  $u\ N \leq \text{Max } \{u\ n \mid n. n \leq C\}$ 
    apply (rule Max_ge) using  $\langle \neg(N > C) \rangle$  by auto
    then show False using  $\langle u\ N \geq n \rangle \langle n \geq M \rangle$  unfolding M_def by auto
  qed
  then have **:  $\{N. u\ N \geq n\} \subseteq \{C..\}$  by fastforce
  have  $\text{Inf } \{N. u\ N \geq n\} \geq C$ 
    by (metis * ** Inf_nat_def1 atLeast_iff subset_eq)
}
then have eventually  $(\lambda n. \text{Inf } \{N. u\ N \geq n\} \geq C)$  sequentially
  using eventually_sequentially by auto
}
then show ?thesis using filterlim_at_top by auto
qed

```

```

lemma pseudo_inverse_finite_set:
  fixes  $u::\text{nat} \Rightarrow \text{nat}$ 
  assumes LIM  $n$  sequentially.  $u\ n \rightarrow \text{at\_top}$ 
  shows finite  $\{N. u\ N \leq n\}$ 
proof -
  fix  $n$ 
  have eventually  $(\lambda N. u\ N \geq n+1)$  sequentially using assms
    by (simp add: filterlim_at_top)
  then obtain  $N1$  where  $N1: \bigwedge N. N \geq N1 \implies u\ N \geq n + 1$ 
    using eventually_sequentially by auto
  have  $\{N. u\ N \leq n\} \subseteq \{..<N1\}$ 
    apply auto using  $N1$  by (metis Suc_eq_plus1 not_less not_less_eq_eq)
  then show finite  $\{N. u\ N \leq n\}$  by (simp add: finite_subset)
qed

```

```

lemma tendsto_at_top_pseudo_inverse2 [tendsto_intros]:
  fixes  $u::\text{nat} \Rightarrow \text{nat}$ 
  assumes LIM  $n$  sequentially.  $u\ n \rightarrow \text{at\_top}$ 
  shows LIM  $n$  sequentially.  $\text{Max } \{N. u\ N \leq n\} \rightarrow \text{at\_top}$ 
proof -
  {
    fix  $N0::\text{nat}$ 
    have  $N0 \leq \text{Max } \{N. u\ N \leq n\}$  if  $n \geq u\ N0$  for  $n$ 
      apply (rule Max_coboundedI) using pseudo_inverse_finite_set[OF assms]
    that by auto
    then have eventually  $(\lambda n. N0 \leq \text{Max } \{N. u\ N \leq n\})$  sequentially
      using eventually_sequentially by blast
  }
  then show ?thesis using filterlim_at_top by auto
qed

```

```

lemma ereal_truncation_top [tendsto_intros]:
  fixes x::ereal
  shows ( $\lambda n::nat. \min x n$ )  $\longrightarrow$  x
proof (cases x)
  case (real r)
  then obtain K::nat where  $K > 0$   $K > \text{abs}(r)$  using reals_Archimedean2 gr0I
  by auto
  then have  $\min x n = x$  if  $n \geq K$  for n apply (subst real, subst real, auto)
  using that eq_iff by fastforce
  then have eventually ( $\lambda n. \min x n = x$ ) sequentially using eventually_at_top_linorder
  by blast
  then show ?thesis by (simp add: tendsto_eventually)
next
  case (PInf)
  then have  $\min x n = n$  for  $n::nat$  by (auto simp add: min_def)
  then show ?thesis using id_nat_ereal_tendsto_PInf PInf by auto
next
  case (MInf)
  then have  $\min x n = x$  for  $n::nat$  by (auto simp add: min_def)
  then show ?thesis by auto
qed

```

```

lemma ereal_truncation_real_top [tendsto_intros]:
  fixes x::ereal
  assumes  $x \neq -\infty$ 
  shows ( $\lambda n::nat. \text{real\_of\_ereal}(\min x n)$ )  $\longrightarrow$  x
proof (cases x)
  case (real r)
  then obtain K::nat where  $K > 0$   $K > \text{abs}(r)$  using reals_Archimedean2 gr0I
  by auto
  then have  $\min x n = x$  if  $n \geq K$  for n apply (subst real, subst real, auto)
  using that eq_iff by fastforce
  then have  $\text{real\_of\_ereal}(\min x n) = r$  if  $n \geq K$  for n using real that by auto
  then have eventually ( $\lambda n. \text{real\_of\_ereal}(\min x n) = r$ ) sequentially using eventually_at_top_linorder by blast
  then have ( $\lambda n. \text{real\_of\_ereal}(\min x n)$ )  $\longrightarrow$  r by (simp add: tendsto_eventually)
  then show ?thesis using real by auto
next
  case (PInf)
  then have  $\text{real\_of\_ereal}(\min x n) = n$  for  $n::nat$  by (auto simp add: min_def)
  then show ?thesis using id_nat_ereal_tendsto_PInf PInf by auto
qed (simp add: assms)

```

```

lemma ereal_truncation_bottom [tendsto_intros]:
  fixes x::ereal
  shows ( $\lambda n::nat. \max x (- \text{real } n)$ )  $\longrightarrow$  x
proof (cases x)
  case (real r)
  then obtain K::nat where  $K > 0$   $K > \text{abs}(r)$  using reals_Archimedean2 gr0I

```

```

by auto
  then have  $\max x (-\text{real } n) = x$  if  $n \geq K$  for  $n$  apply (subst real, subst real,
auto) using that eq_iff by fastforce
  then have eventually  $(\lambda n. \max x (-\text{real } n) = x)$  sequentially using eventu-
ally_at_top_linorder by blast
  then show ?thesis by (simp add: tendsto_eventually)
next
  case (MInf)
  then have  $\max x (-\text{real } n) = (-1) * \text{ereal}(\text{real } n)$  for  $n::\text{nat}$  by (auto simp add:
max_def)
  moreover have  $(\lambda n. (-1) * \text{ereal}(\text{real } n)) \longrightarrow -\infty$ 
  using tendsto_cmult_ereal[of -1, OF _ id_nat_ereal_tendsto_PInf] by
(simp add: one_ereal_def)
  ultimately show ?thesis using MInf by auto
next
  case (PInf)
  then have  $\max x (-\text{real } n) = x$  for  $n::\text{nat}$  by (auto simp add: max_def)
  then show ?thesis by auto
qed

```

```

lemma ereal_truncation_real_bottom [tendsto_intros]:
  fixes  $x::\text{ereal}$ 
  assumes  $x \neq \infty$ 
  shows  $(\lambda n::\text{nat}. \text{real\_of\_ereal}(\max x (-\text{real } n))) \longrightarrow x$ 
proof (cases x)
  case (real r)
  then obtain  $K::\text{nat}$  where  $K > 0$   $K > \text{abs}(r)$  using reals_Archimedean2 gr0I
  by auto
  then have  $\max x (-\text{real } n) = x$  if  $n \geq K$  for  $n$  apply (subst real, subst real,
auto) using that eq_iff by fastforce
  then have  $\text{real\_of\_ereal}(\max x (-\text{real } n)) = r$  if  $n \geq K$  for  $n$  using real that
  by auto
  then have eventually  $(\lambda n. \text{real\_of\_ereal}(\max x (-\text{real } n)) = r)$  sequentially
  using eventually_at_top_linorder by blast
  then have  $(\lambda n. \text{real\_of\_ereal}(\max x (-\text{real } n))) \longrightarrow r$  by (simp add: tend-
sto_eventually)
  then show ?thesis using real by auto
next
  case (MInf)
  then have  $\text{real\_of\_ereal}(\max x (-\text{real } n)) = (-1) * \text{ereal}(\text{real } n)$  for  $n::\text{nat}$  by
(auto simp add: max_def)
  moreover have  $(\lambda n. (-1) * \text{ereal}(\text{real } n)) \longrightarrow -\infty$ 
  using tendsto_cmult_ereal[of -1, OF _ id_nat_ereal_tendsto_PInf] by
(simp add: one_ereal_def)
  ultimately show ?thesis using MInf by auto
qed (simp add: assms)

```

the next one is copied from *tendsto_sum*.

```

lemma tendsto_sum_ereal [tendsto_intros]:

```

```

fixes  $f :: 'a \Rightarrow 'b \Rightarrow \text{ereal}$ 
assumes  $\bigwedge i. i \in S \implies (f\ i \longrightarrow a\ i)\ F$ 
           $\bigwedge i. \text{abs}(a\ i) \neq \infty$ 
shows  $((\lambda x. \sum_{i \in S}. f\ i\ x) \longrightarrow (\sum_{i \in S}. a\ i))\ F$ 
proof (cases finite  $S$ )
  assume finite  $S$  then show ?thesis using assms
    by (induct, simp, simp add: tendsto_add_ereal_general2 assms)
qed(simp)

```

```

lemma continuous_ereal_abs:
  continuous_on (UNIV::ereal set) abs
proof -
  have continuous_on  $(\{..0\} \cup \{(0::ereal).. \})\ abs$ 
    apply (rule continuous_on_closed_Un, auto)
    apply (rule iffD1[OF continuous_on_cong, of  $\{..0\} \_ \lambda x. -x$ ])
    using less_eq_ereal_def apply (auto simp add: continuous_uminus_ereal)
    apply (rule iffD1[OF continuous_on_cong, of  $\{0..\} \_ \lambda x. x$ ])
    apply (auto)
  done
  moreover have (UNIV::ereal set) =  $\{..0\} \cup \{(0::ereal).. \}$  by auto
  ultimately show ?thesis by auto
qed

```

```

lemmas continuous_on_compose_ereal_abs[continuous_intros] =
  continuous_on_compose2[OF continuous_ereal_abs _ subset_UNIV]

```

```

lemma tendsto_abs_ereal [tendsto_intros]:
  assumes  $(u \longrightarrow (l::ereal))\ F$ 
  shows  $((\lambda n. \text{abs}(u\ n)) \longrightarrow \text{abs}\ l)\ F$ 
using continuous_ereal_abs assms by (metis UNIV_I continuous_on tendsto_compose)

```

```

lemma ereal_minus_real_tendsto_MInf [tendsto_intros]:
   $(\lambda x. \text{ereal}(-\text{real}\ x)) \longrightarrow -\infty$ 
by (subst uminus_ereal.simps(1)[symmetric], intro tendsto_intros)

```

4.5.2 Extended-Nonnegative-Real.thy

```

lemma tendsto_diff_ennreal_general [tendsto_intros]:
  fixes  $u\ v :: 'a \Rightarrow \text{ennreal}$ 
  assumes  $(u \longrightarrow l)\ F\ (v \longrightarrow m)\ F\ \neg(l = \infty \wedge m = \infty)$ 
  shows  $((\lambda n. u\ n - v\ n) \longrightarrow l - m)\ F$ 
proof -
  have  $((\lambda n. e2ennreal(enn2ereal(u\ n) - enn2ereal(v\ n))) \longrightarrow e2ennreal(enn2ereal\ l - enn2ereal\ m))\ F$ 
    apply (intro tendsto_intros) using assms by auto
  then show ?thesis by auto
qed

```



```

lemma tendsto_mult_enreal [tendsto_intros]:
  fixes l m::ennreal
  assumes (u  $\longrightarrow$  l) F (v  $\longrightarrow$  m) F  $\neg((l = 0 \wedge m = \infty) \vee (l = \infty \wedge m = 0))$ 
  shows  $((\lambda n. u\ n * v\ n) \longrightarrow l * m)$  F
proof -
  have  $((\lambda n. e2ennreal(enn2ereal(u\ n) * enn2ereal(v\ n))) \longrightarrow e2ennreal(enn2ereal\ l * enn2ereal\ m))$  F
    apply (intro tendsto_intros) using assms apply auto
    using enn2ereal_inject zero_enreal.rep_eq by fastforce+
  moreover have  $e2ennreal(enn2ereal(u\ n) * enn2ereal(v\ n)) = u\ n * v\ n$  for n
    by (subst times_enreal.abs_eq[symmetric], auto simp add: eq_onp_same_args)
  moreover have  $e2ennreal(enn2ereal\ l * enn2ereal\ m) = l * m$ 
    by (subst times_enreal.abs_eq[symmetric], auto simp add: eq_onp_same_args)
  ultimately show ?thesis
    by auto
qed

```

4.5.3 monoset

definition (*in order*) *mono_set*:
 $mono\_set\ S \longleftrightarrow (\forall x\ y. x \leq y \longrightarrow x \in S \longrightarrow y \in S)$

lemma (*in order*) *mono_greaterThan* [*intro, simp*]: *mono_set* $\{B<..\}$ **unfolding** *mono_set* **by** *auto*

lemma (*in order*) *mono_atLeast* [*intro, simp*]: *mono_set* $\{B..\}$ **unfolding** *mono_set* **by** *auto*

lemma (*in order*) *mono_UNIV* [*intro, simp*]: *mono_set* *UNIV* **unfolding** *mono_set* **by** *auto*

lemma (*in order*) *mono_empty* [*intro, simp*]: *mono_set* $\{\}$ **unfolding** *mono_set* **by** *auto*

lemma (*in complete_linorder*) *mono_set_iff*:
fixes *S* :: 'a set
defines *a* $\equiv Inf\ S$
shows $mono\_set\ S \longleftrightarrow S = \{a<..\} \vee S = \{a..\}$ (*is* $\_ = ?c$)

proof
assume *mono_set S*
then have *mono*: $\bigwedge x\ y. x \leq y \implies x \in S \implies y \in S$
by (*auto simp: mono_set*)
show *?c*
proof *cases*
assume *a* $\in S$
show *?c*
using *mono[OF $\_ \langle a \in S \rangle$]*
by (*auto intro: Inf_lower simp: a_def*)
next
assume *a* $\notin S$
have $S = \{a<..\}$

```

proof safe
  fix  $x$  assume  $x \in S$ 
  then have  $a \leq x$ 
    unfolding  $a\_def$  by (rule Inf_lower)
  then show  $a < x$ 
    using  $\langle x \in S \rangle \langle a \notin S \rangle$  by (cases a = x) auto
next
  fix  $x$  assume  $a < x$ 
  then obtain  $y$  where  $y < x$   $y \in S$ 
    unfolding  $a\_def$  Inf_less_iff ..
  with mono[of y x] show  $x \in S$ 
    by auto
qed
then show  $?c$  ..
qed
qed auto

lemma ereal_open_mono_set:
  fixes  $S :: \text{ereal set}$ 
  shows  $\text{open } S \wedge \text{mono\_set } S \longleftrightarrow S = \text{UNIV} \vee S = \{\text{Inf } S <..\}$ 
  by (metis Inf_UNIV atLeast_eq_UNIV_iff eréal_open_atLeast
    ereal_open_closed mono_set_iff open_ereal_greaterThan)

lemma ereal_closed_mono_set:
  fixes  $S :: \text{ereal set}$ 
  shows  $\text{closed } S \wedge \text{mono\_set } S \longleftrightarrow S = \{\}$   $\vee S = \{\text{Inf } S ..\}$ 
  by (metis Inf_UNIV atLeast_eq_UNIV_iff closed_ereal_atLeast
    ereal_open_closed mono_empty mono_set_iff open_ereal_greaterThan)

lemma ereal_Liminf_Sup_monoset:
  fixes  $f :: 'a \Rightarrow \text{ereal}$ 
  shows  $\text{Liminf } \text{net } f =$ 
     $\text{Sup } \{l. \forall S. \text{open } S \longrightarrow \text{mono\_set } S \longrightarrow l \in S \longrightarrow \text{eventually } (\lambda x. f x \in S)$ 
 $\text{net}\}$ 
    (is _ = Sup ?A)
proof (safe intro!: Liminf_eqI complete_lattice_class.Sup_upper complete_lattice_class.Sup_least)
  fix  $P$ 
  assume  $P$ : eventually P net
  fix  $S$ 
  assume  $S$ :  $\text{mono\_set } S$   $\text{Inf } (f ` (\text{Collect } P)) \in S$ 
  {
    fix  $x$ 
    assume  $P x$ 
    then have  $\text{Inf } (f ` (\text{Collect } P)) \leq f x$ 
      by (intro complete_lattice_class.INF_lower) simp
    with  $S$  have  $f x \in S$ 
      by (simp add: mono_set)
  }
with  $P$  show eventually  $(\lambda x. f x \in S)$  net

```

```

    by (auto elim: eventually_mono)
next
  fix y l
  assume S:  $\forall S. \text{open } S \longrightarrow \text{mono\_set } S \longrightarrow l \in S \longrightarrow \text{eventually } (\lambda x. f x \in S)$ 
net
  assume P:  $\forall P. \text{eventually } P \text{ net} \longrightarrow \text{Inf } (f ' (\text{Collect } P)) \leq y$ 
  show  $l \leq y$ 
  proof (rule dense_le)
    fix B
    assume  $B < l$ 
    then have  $\text{eventually } (\lambda x. f x \in \{B <..\}) \text{ net}$ 
      by (intro S[rule_format]) auto
    then have  $\text{Inf } (f ' \{x. B < f x\}) \leq y$ 
      using P by auto
    moreover have  $B \leq \text{Inf } (f ' \{x. B < f x\})$ 
      by (intro INF_greatest) auto
    ultimately show  $B \leq y$ 
      by simp
  qed
qed

lemma ereal_Limsup_Inf_monoset:
  fixes f :: 'a  $\Rightarrow$  ereal
  shows  $\text{Limsup net } f =$ 
 $\text{Inf } \{l. \forall S. \text{open } S \longrightarrow \text{mono\_set } (\text{uminus } ' S) \longrightarrow l \in S \longrightarrow \text{eventually } (\lambda x.$ 
 $f x \in S) \text{ net}\}$ 
  (is  $\_ = \text{Inf } ?A$ )
proof (safe intro!: Limsup_eqI complete_lattice_class.Inf_lower complete_lattice_class.Inf_greatest)
  fix P
  assume P:  $\text{eventually } P \text{ net}$ 
  fix S
  assume S:  $\text{mono\_set } (\text{uminus } ' S) \text{ Sup } (f ' (\text{Collect } P)) \in S$ 
  {
    fix x
    assume P x
    then have  $f x \leq \text{Sup } (f ' (\text{Collect } P))$ 
      by (intro complete_lattice_class.SUP_upper) simp
    with S(1)[unfolded mono_set, rule_format, of  $-\text{Sup } (f ' (\text{Collect } P)) - f x$ ]
  S(2)
    have  $f x \in S$ 
      by (simp add: inj_image_mem_iff) }
  with P show  $\text{eventually } (\lambda x. f x \in S) \text{ net}$ 
    by (auto elim: eventually_mono)
next
  fix y l
  assume S:  $\forall S. \text{open } S \longrightarrow \text{mono\_set } (\text{uminus } ' S) \longrightarrow l \in S \longrightarrow \text{eventually } (\lambda x. f x \in S) \text{ net}$ 
  assume P:  $\forall P. \text{eventually } P \text{ net} \longrightarrow y \leq \text{Sup } (f ' (\text{Collect } P))$ 
  show  $y \leq l$ 

```

```

proof (rule dense_ge)
  fix  $B$ 
  assume  $l < B$ 
  then have eventually  $(\lambda x. f\ x \in \{..< B\})$  net
    by (intro  $S[\text{rule\_format}]$ ) auto
  then have  $y \leq \text{Sup } (f\ ' \{x. f\ x < B\})$ 
    using  $P$  by auto
  moreover have  $\text{Sup } (f\ ' \{x. f\ x < B\}) \leq B$ 
    by (intro  $\text{SUP\_least}$ ) auto
  ultimately show  $y \leq B$ 
    by simp
qed
qed

lemma liminf_bounded_open:
  fixes  $x :: \text{nat} \Rightarrow \text{ereal}$ 
  shows  $x0 \leq \text{liminf } x \longleftrightarrow (\forall S. \text{open } S \longrightarrow \text{mono\_set } S \longrightarrow x0 \in S \longrightarrow (\exists N. \forall n \geq N. x\ n \in S))$ 
  (is  $\_ \longleftrightarrow ?P\ x0$ )
proof
  assume  $?P\ x0$ 
  then show  $x0 \leq \text{liminf } x$ 
    unfolding  $\text{ereal\_Liminf\_Sup\_monoset eventually\_sequentially}$ 
    by (intro  $\text{complete\_lattice\_class.Sup\_upper}$ ) auto
next
  assume  $x0 \leq \text{liminf } x$ 
  {
    fix  $S :: \text{ereal set}$ 
    assume  $om: \text{open } S \text{ mono\_set } S\ x0 \in S$ 
    {
      assume  $S = \text{UNIV}$ 
      then have  $\exists N. \forall n \geq N. x\ n \in S$ 
        by auto
    }
    moreover
    {
      assume  $S \neq \text{UNIV}$ 
      then obtain  $B$  where  $B: S = \{B<..\}$ 
        using  $om$   $\text{ereal\_open\_mono\_set}$  by auto
      then have  $B < x0$ 
        using  $om$  by auto
      then have  $\exists N. \forall n \geq N. x\ n \in S$ 
        unfolding  $B$ 
        using  $\langle x0 \leq \text{liminf } x \rangle \text{liminf\_bounded\_iff}$ 
        by auto
    }
  }
  ultimately have  $\exists N. \forall n \geq N. x\ n \in S$ 
    by auto
}

```

```

    then show ?P x0
      by auto
qed

lemma limsup_finite_then_bounded:
  fixes u::nat  $\Rightarrow$  real
  assumes limsup u <  $\infty$ 
  shows  $\exists C. \forall n. u\ n \leq C$ 
proof -
  obtain C where C: limsup u < C C <  $\infty$  using assms ereal_dense2 by blast
  then have C = ereal(real_of_ereal C) using ereal_real by force
  have eventually ( $\lambda n. u\ n < C$ ) sequentially using C(1) unfolding Limsup_def
    apply (auto simp add: INF_less_iff)
    using SUP_lessD eventually_mono by fastforce
  then obtain N where N:  $\bigwedge n. n \geq N \Rightarrow u\ n < C$  using eventually_sequentially
  by auto
  define D where D = max (real_of_ereal C) (Max {u n | n. n  $\leq$  N})
  have  $\bigwedge n. u\ n \leq D$ 
  proof -
    fix n show u n  $\leq$  D
    proof (cases)
      assume *: n  $\leq$  N
      have u n  $\leq$  Max {u n | n. n  $\leq$  N} by (rule Max_ge, auto simp add: *)
      then show u n  $\leq$  D unfolding D_def by linarith
    next
      assume  $\neg(n \leq N)$ 
      then have n  $\geq$  N by simp
      then have u n < C using N by auto
      then have u n < real_of_ereal C using  $\langle C = \text{ereal}(\text{real\_of\_ereal } C) \rangle$ 
      less_ereal.simps(1) by fastforce
      then show u n  $\leq$  D unfolding D_def by linarith
    qed
  qed
  then show ?thesis by blast
qed

```

```

lemma liminf_finite_then_bounded_below:
  fixes u::nat  $\Rightarrow$  real
  assumes liminf u >  $-\infty$ 
  shows  $\exists C. \forall n. u\ n \geq C$ 
proof -
  obtain C where C: liminf u > C C >  $-\infty$  using assms using ereal_dense2
  by blast
  then have C = ereal(real_of_ereal C) using ereal_real by force
  have eventually ( $\lambda n. u\ n > C$ ) sequentially using C(1) unfolding Liminf_def
    apply (auto simp add: less_SUP_iff)
    using eventually_elim2 less_INF_D by fastforce
  then obtain N where N:  $\bigwedge n. n \geq N \Rightarrow u\ n > C$  using eventually_sequentially
  by auto

```

```

define  $D$  where  $D = \min (\text{real\_of\_ereal } C) (\text{Min } \{u \ n \mid n. n \leq N\})$ 
have  $\bigwedge n. u \ n \geq D$ 
proof –
  fix  $n$  show  $u \ n \geq D$ 
  proof (cases)
    assume  $*$ :  $n \leq N$ 
    have  $u \ n \geq \text{Min } \{u \ n \mid n. n \leq N\}$  by (rule Min_le, auto simp add: *)
    then show  $u \ n \geq D$  unfolding  $D\_def$  by linarith
  next
    assume  $\neg(n \leq N)$ 
    then have  $n \geq N$  by simp
    then have  $u \ n > C$  using  $N$  by auto
    then have  $u \ n > \text{real\_of\_ereal } C$  using  $\langle C = \text{ereal}(\text{real\_of\_ereal } C) \rangle$ 
    less_ereal.simps(1) by fastforce
    then show  $u \ n \geq D$  unfolding  $D\_def$  by linarith
  qed
qed
then show ?thesis by blast
qed

```

```

lemma liminf_upper_bound:
  fixes  $u:: \text{nat} \Rightarrow \text{ereal}$ 
  assumes  $\text{liminf } u < l$ 
  shows  $\exists N > k. u \ N < l$ 
by (metis assms gt_ex less_le_trans liminf_bounded_iff not_less)

```

```

lemma limsup_shift:
   $\text{limsup } (\lambda n. u \ (n+1)) = \text{limsup } u$ 
proof –
  have  $(\text{SUP } m \in \{n+1..\}. u \ m) = (\text{SUP } m \in \{n..\}. u \ (m+1))$  for  $n$ 
    apply (rule SUP_eq) using Suc_le_D by auto
  then have  $a: (\text{INF } n. \text{SUP } m \in \{n..\}. u \ (m+1)) = (\text{INF } n. (\text{SUP } m \in \{n+1..\}. u \ m))$  by auto
  have  $b: (\text{INF } n. (\text{SUP } m \in \{n+1..\}. u \ m)) = (\text{INF } n \in \{1..\}. (\text{SUP } m \in \{n..\}. u \ m))$ 
    apply (rule INF_eq) using Suc_le_D by auto
  have  $(\text{INF } n \in \{1..\}. v \ n) = (\text{INF } n. v \ n)$  if decseq v for  $v:: \text{nat} \Rightarrow 'a$ 
    apply (rule INF_eq) using  $\langle \text{decseq } v \rangle$  decseq_Suc_iff by auto
  moreover have  $\text{decseq } (\lambda n. (\text{SUP } m \in \{n..\}. u \ m))$  by (simp add: SUP_subset_mono decseq_def)
  ultimately have  $c: (\text{INF } n \in \{1..\}. (\text{SUP } m \in \{n..\}. u \ m)) = (\text{INF } n. (\text{SUP } m \in \{n..\}. u \ m))$  by simp
  have  $(\text{INF } n. \text{Sup } (u \ ' \{n..\})) = (\text{INF } n. \text{SUP } m \in \{n..\}. u \ (m+1))$  using  $a \ b$  by simp
  then show ?thesis by (auto cong: limsup_INF_SUP)
qed

```

```

lemma limsup_shift_k:
   $\text{limsup } (\lambda n. u \ (n+k)) = \text{limsup } u$ 

```

```

proof (induction k)
  case (Suc k)
    have  $\limsup (\lambda n. u (n+k+1)) = \limsup (\lambda n. u (n+k))$  using limsup_shift [where
     $?u = \lambda n. u (n+k)$ ] by simp
    then show  $?case$  using Suc.IH by simp
qed (auto)

lemma liminf_shift:
   $\liminf (\lambda n. u (n+1)) = \liminf u$ 
proof –
  have  $(\inf_{m \in \{n+1.. \}} u m) = (\inf_{m \in \{n.. \}} u (m + 1))$  for  $n$ 
    apply (rule INF_eq) using Suc_le_D by (auto)
  then have  $a: (\sup n. \inf_{m \in \{n.. \}} u (m + 1)) = (\sup n. (\inf_{m \in \{n+1.. \}} u m))$  by auto
  have  $b: (\sup n. (\inf_{m \in \{n+1.. \}} u m)) = (\sup_{n \in \{1.. \}} (\inf_{m \in \{n.. \}} u m))$ 
    apply (rule SUP_eq) using Suc_le_D by (auto)
  have  $(\sup_{n \in \{1.. \}} v n) = (\sup n. v n)$  if incseq v for  $v :: nat \Rightarrow 'a$ 
    apply (rule SUP_eq) using  $\langle incseq v \rangle$  incseq_Suc_iff by auto
  moreover have incseq  $(\lambda n. (\inf_{m \in \{n.. \}} u m))$  by (simp add: INF_superset_mono mono_def)
  ultimately have  $c: (\sup_{n \in \{1.. \}} (\inf_{m \in \{n.. \}} u m)) = (\sup n. (\inf_{m \in \{n.. \}} u m))$  by simp
  have  $(\sup n. \inf (u \restriction \{n.. \})) = (\sup n. \inf_{m \in \{n.. \}} u (m + 1))$  using  $a$   $b$  by simp
  then show  $?thesis$  by (auto cong: liminf_SUP_INF)
qed

lemma liminf_shift_k:
   $\liminf (\lambda n. u (n+k)) = \liminf u$ 
proof (induction k)
  case (Suc k)
    have  $\liminf (\lambda n. u (n+k+1)) = \liminf (\lambda n. u (n+k))$  using liminf_shift [where
     $?u = \lambda n. u (n+k)$ ] by simp
    then show  $?case$  using Suc.IH by simp
qed (auto)

lemma Limsup_obtain:
  fixes  $u :: \_ \Rightarrow 'a :: complete\_linorder$ 
  assumes  $Limsup F u > c$ 
  shows  $\exists i. u i > c$ 
proof –
  have  $(\inf P \in \{P. eventually P F\}. \sup x \in \{x. P x\}. u x) > c$  using assms by
  (simp add: Limsup_def)
  then show  $?thesis$  by (metis eventually_True mem_Collect_eq less_INF_D less_SUP_iff)
qed

```

The next lemma is extremely useful, as it often makes it possible to reduce

statements about limsup to statements about limits.

```

lemma limsup_subseq_lim:
  fixes  $u::nat \Rightarrow 'a :: \{complete\_linorder, linorder\_topology\}$ 
  shows  $\exists r::nat \Rightarrow nat. strict\_mono\ r \wedge (u\ o\ r) \longrightarrow limsup\ u$ 
proof (cases)
  assume  $\forall n. \exists p > n. \forall m \geq p. u\ m \leq u\ p$ 
  then have  $\exists r. \forall n. (\forall m \geq r\ n. u\ m \leq u\ (r\ n)) \wedge r\ n < r\ (Suc\ n)$ 
    by (intro dependent_nat_choice) (auto simp: conj_commute)
  then obtain  $r :: nat \Rightarrow nat$  where strict_mono r and mono:  $\bigwedge n\ m. r\ n \leq m \implies u\ m \leq u\ (r\ n)$ 
    by (auto simp: strict_mono_Suc_iff)
  define umax where  $umax = (\lambda n. (SUP\ m \in \{n..\}. u\ m))$ 
  have decseq umax unfolding umax_def by (simp add: SUP_subset_mono anti-mono_def)
  then have umax  $\longrightarrow limsup\ u$  unfolding umax_def by (metis LIMSEQ_INF limsup_INF_SUP)
  then have  $*(umax\ o\ r) \longrightarrow limsup\ u$  by (simp add: LIMSEQ_subseq_LIMSEQ  $\langle strict\_mono\ r \rangle$ )
  have  $\bigwedge n. umax(r\ n) = u(r\ n)$  unfolding umax_def using mono
    by (metis SUP_le_iff antisym atLeast_def mem_Collect_eq order_refl)
  then have  $umax\ o\ r = u\ o\ r$  unfolding o_def by simp
  then have  $(u\ o\ r) \longrightarrow limsup\ u$  using  $*$  by simp
  then show ?thesis using  $\langle strict\_mono\ r \rangle$  by blast
next
  assume  $\neg (\forall n. \exists p > n. (\forall m \geq p. u\ m \leq u\ p))$ 
  then obtain N where  $N: \bigwedge p. p > N \implies \exists m > p. u\ p < u\ m$  by (force simp: not_le le_less)
  have  $\exists r. \forall n. N < r\ n \wedge r\ n < r\ (Suc\ n) \wedge (\forall i \in \{N <..r\ (Suc\ n)\}. u\ i \leq u\ (r\ (Suc\ n)))$ 
    proof (rule dependent_nat_choice)
      fix x assume  $N < x$ 
      then have  $a: finite\ \{N <..x\}\ \{N <..x\} \neq \{\}$  by simp_all
      have  $Max\ \{u\ i\ |\ i. i \in \{N <..x\}\} \in \{u\ i\ |\ i. i \in \{N <..x\}\}$  apply (rule Max_in)
    using a by (auto)
      then obtain p where  $p \in \{N <..x\}$  and upmax:  $u\ p = Max\ \{u\ i\ |\ i. i \in \{N <..x\}\}$  by auto
      define U where  $U = \{m. m > p \wedge u\ p < u\ m\}$ 
      have  $U \neq \{\}$  unfolding U_def using  $N[of\ p]\ \langle p \in \{N <..x\} \rangle$  by auto
      define y where  $y = Inf\ U$ 
      then have  $y \in U$  using  $\langle U \neq \{\} \rangle$  by (simp add: Inf_nat_def1)
      have  $a: \bigwedge i. i \in \{N <..x\} \implies u\ i \leq u\ p$ 
      proof –
        fix i assume  $i \in \{N <..x\}$ 
        then have  $u\ i \in \{u\ i\ |\ i. i \in \{N <..x\}\}$  by blast
        then show  $u\ i \leq u\ p$  using upmax by simp
      qed
      moreover have  $u\ p < u\ y$  using  $\langle y \in U \rangle\ U\_def$  by auto
      ultimately have  $y \notin \{N <..x\}$  using not_le by blast
      moreover have  $y > N$  using  $\langle y \in U \rangle\ U\_def\ \langle p \in \{N <..x\} \rangle$  by auto

```



```

ultimately have  $y > x$  by auto

have  $\bigwedge i. i \in \{N <..y\} \implies u\ i \leq u\ y$ 
proof -
  fix i assume  $i \in \{N <..y\}$  show  $u\ i \leq u\ y$ 
  proof (cases)
    assume  $i = y$ 
    then show ?thesis by simp
  next
    assume  $\neg(i=y)$ 
    then have  $i: i \in \{N <.. $y\}$  using  $\langle i \in \{N <.. $y\}\rangle$  by simp
    have  $u\ i \leq u\ p$ 
    proof (cases)
      assume  $i \leq x$ 
      then have  $i \in \{N <.. $x\}$  using i by simp
      then show ?thesis using a by simp
    next
      assume  $\neg(i \leq x)$ 
      then have  $i > x$  by simp
      then have  $*: i > p$  using  $\langle p \in \{N <.. $x\}\rangle$  by simp
      have  $i < \text{Inf } U$  using i y_def by simp
      then have  $i \notin U$  using Inf_nat_def not_less_Least by auto
      then show ?thesis using U_def * by auto
    qed
    then show  $u\ i \leq u\ y$  using  $\langle u\ p < u\ y\rangle$  by auto
  qed
qed
qed
then have  $N < y \wedge x < y \wedge (\forall i \in \{N <.. $y\}. u\ i \leq u\ y)$  using  $\langle y > x\rangle \langle y > N\rangle$  by auto
then show  $\exists y > N. x < y \wedge (\forall i \in \{N <.. $y\}. u\ i \leq u\ y)$  by auto
qed (auto)
then obtain r where  $r: \forall n. N < r\ n \wedge r\ n < r\ (\text{Suc } n) \wedge (\forall i \in \{N <.. $r\ (\text{Suc } n)\}. u\ i \leq u\ (r\ (\text{Suc } n)))$  by auto
have strict_mono r using r by (auto simp: strict_mono_Suc_iff)
have incseq (u o r) unfolding o_def using r by (simp add: incseq_SucI order.strict_implies_order)
then have  $(u\ o\ r) \longrightarrow (\text{SUP } n. (u\ o\ r)\ n)$  using LIMSEQ_SUP by blast
then have  $\text{limsup } (u\ o\ r) = (\text{SUP } n. (u\ o\ r)\ n)$  by (simp add: lim_imp_Limsup)
moreover have  $\text{limsup } (u\ o\ r) \leq \text{limsup } u$  using  $\langle \text{strict\_mono } r\rangle$  by (simp add: limsup_subseq_mono)
ultimately have  $(\text{SUP } n. (u\ o\ r)\ n) \leq \text{limsup } u$  by simp

{
  fix i assume  $i: i \in \{N <.. $\}$ 
  obtain n where  $i < r\ (\text{Suc } n)$  using  $\langle \text{strict\_mono } r\rangle$  using Suc_le_eq seq_suble by blast
  then have  $i \in \{N <.. $r\ (\text{Suc } n)\}$  using i by simp
  then have  $u\ i \leq u\ (r\ (\text{Suc } n))$  using r by simp
  then have  $u\ i \leq (\text{SUP } n. (u\ o\ r)\ n)$  unfolding o_def by (meson SUP_upper2$$$$$$$$$ 
```

```

UNIV_I)
}
then have (SUP i∈{N<..}. u i) ≤ (SUP n. (u o r) n) using SUP_least by
blast
then have limsup u ≤ (SUP n. (u o r) n) unfolding Limsup_def
by (metis (mono_tags, lifting) INF_lower2 atLeast_Suc_greaterThan atLeast_def
eventually_ge_at_top mem_Collect_eq)
then have limsup u = (SUP n. (u o r) n) using ⟨(SUP n. (u o r) n) ≤ limsup
u⟩ by simp
then have (u o r) ⟶ limsup u using ⟨(u o r) ⟶ (SUP n. (u o r) n)⟩
by simp
then show ?thesis using ⟨strict_mono r⟩ by auto
qed

```

lemma liminf_subseq_lim:

```

fixes u::nat ⇒ 'a :: {complete_linorder, linorder_topology}
shows ∃ r::nat⇒nat. strict_mono r ∧ (u o r) ⟶ liminf u
proof (cases)
assume ∀ n. ∃ p>n. ∀ m≥p. u m ≥ u p
then have ∃ r. ∀ n. (∀ m≥r n. u m ≥ u (r n)) ∧ r n < r (Suc n)
by (intro dependent_nat_choice) (auto simp: conj_commute)
then obtain r :: nat ⇒ nat where strict_mono r and mono: ∧ n m. r n ≤ m
⇒ u m ≥ u (r n)
by (auto simp: strict_mono_Suc_iff)
define umin where umin = (λn. (INF m∈{n..}. u m))
have incseq umin unfolding umin_def by (simp add: INF_superset_mono
incseq_def)
then have umin ⟶ liminf u unfolding umin_def by (metis LIMSEQ_SUP
liminf_SUP_INF)
then have *: (umin o r) ⟶ liminf u by (simp add: LIMSEQ_subseq_LIMSEQ
⟨strict_mono r⟩)
have ∧ n. umin(r n) = u(r n) unfolding umin_def using mono
by (metis le_INF_iff antisym atLeast_def mem_Collect_eq order_refl)
then have umin o r = u o r unfolding o_def by simp
then have (u o r) ⟶ liminf u using * by simp
then show ?thesis using ⟨strict_mono r⟩ by blast
next
assume ¬ (∀ n. ∃ p>n. (∀ m≥p. u m ≥ u p))
then obtain N where N: ∧ p. p > N ⇒ ∃ m>p. u p > u m by (force simp:
not_le le_less)
have ∃ r. ∀ n. N < r n ∧ r n < r (Suc n) ∧ (∀ i ∈ {N<..r (Suc n)}. u i ≥ u (r
(Suc n)))
proof (rule dependent_nat_choice)
fix x assume N < x
then have a: finite {N<..x} {N<..x} ≠ {} by simp_all
have Min {u i | i. i ∈ {N<..x}} ∈ {u i | i. i ∈ {N<..x}} apply (rule Min_in)
using a by (auto)
then obtain p where p ∈ {N<..x} and upmin: u p = Min{u i | i. i ∈ {N<..x}}
by auto

```

```

define U where U = {m. m > p ∧ u p > u m}
have U ≠ {} unfolding U_def using N[of p] ⟨p ∈ {N<..x}⟩ by auto
define y where y = Inf U
then have y ∈ U using ⟨U ≠ {}⟩ by (simp add: Inf_nat_def1)
have a: ∧i. i ∈ {N<..x} ⇒ u i ≥ u p
proof -
  fix i assume i ∈ {N<..x}
  then have u i ∈ {u i | i. i ∈ {N<..x}} by blast
  then show u i ≥ u p using upmin by simp
qed
moreover have u p > u y using ⟨y ∈ U⟩ U_def by auto
ultimately have y ∉ {N<..x} using not_le by blast
moreover have y > N using ⟨y ∈ U⟩ U_def ⟨p ∈ {N<..x}⟩ by auto
ultimately have y > x by auto

have ∧i. i ∈ {N<..y} ⇒ u i ≥ u y
proof -
  fix i assume i ∈ {N<..y} show u i ≥ u y
  proof (cases)
    assume i = y
    then show ?thesis by simp
  next
    assume ¬(i = y)
    then have i: i ∈ {N<..} using ⟨i ∈ {N<..y}⟩ by simp
    have u i ≥ u p
    proof (cases)
      assume i ≤ x
      then have i ∈ {N<..x} using i by simp
      then show ?thesis using a by simp
    next
      assume ¬(i ≤ x)
      then have i > x by simp
      then have *: i > p using ⟨p ∈ {N<..x}⟩ by simp
      have i < Inf U using i y_def by simp
      then have i ∉ U using Inf_nat_def not_less_Least by auto
      then show ?thesis using U_def * by auto
    qed
    then show u i ≥ u y using ⟨u p > u y⟩ by auto
  qed
qed
then have N < y ∧ x < y ∧ (∀i∈{N<..y}. u i ≥ u y) using ⟨y > x⟩ ⟨y >
N⟩ by auto
then show ∃y>N. x < y ∧ (∀i∈{N<..y}. u i ≥ u y) by auto
qed (auto)
then obtain r :: nat ⇒ nat
  where r: ∀n. N < r n ∧ r n < r (Suc n) ∧ (∀i∈{N<..r (Suc n)}. u i ≥ u
(r (Suc n)))) by auto
have strict_mono r using r by (auto simp: strict_mono_Suc_iff)
have decseq (u o r) unfolding o_def using r by (simp add: decseq_SucI or-

```

```

der.strict_implies_order)
  then have  $(u \circ r) \longrightarrow (INF\ n. (u \circ r)\ n)$  using LIMSEQ_INF by blast
  then have  $\liminf (u \circ r) = (INF\ n. (u \circ r)\ n)$  by (simp add: lim_imp_Liminf)
  moreover have  $\liminf (u \circ r) \geq \liminf u$  using  $\langle strict\_mono\ r \rangle$  by (simp add:
liminf_subseq_mono)
  ultimately have  $(INF\ n. (u \circ r)\ n) \geq \liminf u$  by simp

{
  fix i assume i:  $i \in \{N<..\}$ 
  obtain n where  $i < r\ (Suc\ n)$  using  $\langle strict\_mono\ r \rangle$  using Suc_le_eq
seq_suble by blast
  then have  $i \in \{N<..r(Suc\ n)\}$  using i by simp
  then have  $u\ i \geq u\ (r(Suc\ n))$  using r by simp
  then have  $u\ i \geq (INF\ n. (u \circ r)\ n)$  unfolding o_def by (meson INF_lower2
UNIV_I)
}
then have  $(INF\ i \in \{N<..\}. u\ i) \geq (INF\ n. (u \circ r)\ n)$  using INF_greatest by
blast
then have  $\liminf u \geq (INF\ n. (u \circ r)\ n)$  unfolding Liminf_def
by (metis (mono_tags, lifting) SUP_upper2 atLeast_Suc_greaterThan atLeast_def
eventually_ge_at_top mem_Collect_eq)
then have  $\liminf u = (INF\ n. (u \circ r)\ n)$  using  $\langle (INF\ n. (u \circ r)\ n) \geq \liminf u \rangle$ 
by simp
then have  $(u \circ r) \longrightarrow \liminf u$  using  $\langle (u \circ r) \longrightarrow (INF\ n. (u \circ r)\ n) \rangle$ 
by simp
then show ?thesis using  $\langle strict\_mono\ r \rangle$  by auto
qed

```

The following statement about limsups is reduced to a statement about limits using subsequences thanks to *limsup_subseq_lim*. The statement for limits follows for instance from *tendsto_add_ereal_general*.

```

lemma ereal_limsup_add_mono:
  fixes u v::nat  $\Rightarrow$  ereal
  shows  $\limsup (\lambda n. u\ n + v\ n) \leq \limsup u + \limsup v$ 
proof (cases)
  assume  $(\limsup u = \infty) \vee (\limsup v = \infty)$ 
  then have  $\limsup u + \limsup v = \infty$  by simp
  then show ?thesis by auto
next
  assume  $\neg((\limsup u = \infty) \vee (\limsup v = \infty))$ 
  then have  $\limsup u < \infty$   $\limsup v < \infty$  by auto

  define w where  $w = (\lambda n. u\ n + v\ n)$ 
  obtain r where  $r: strict\_mono\ r\ (w \circ r) \longrightarrow \limsup w$  using limsup_subseq_lim
by auto
  obtain s where  $s: strict\_mono\ s\ (u \circ r \circ s) \longrightarrow \limsup (u \circ r)$  using
limsup_subseq_lim by auto
  obtain t where  $t: strict\_mono\ t\ (v \circ r \circ s \circ t) \longrightarrow \limsup (v \circ r \circ s)$  using
limsup_subseq_lim by auto

```

```

define a where a = r o s o t
have strict_mono a using r s t by (simp add: a_def strict_mono_o)
have l:(w o a) ⟶ limsup w
      (u o a) ⟶ limsup (u o r)
      (v o a) ⟶ limsup (v o r o s)
apply (metis (no_types, lifting) r(2) s(1) t(1) LIMSEQ_subseq_LIMSEQ a_def
comp_assoc)
apply (metis (no_types, lifting) s(2) t(1) LIMSEQ_subseq_LIMSEQ a_def
comp_assoc)
apply (metis (no_types, lifting) t(2) a_def comp_assoc)
done

have limsup (u o r) ≤ limsup u by (simp add: limsup_subseq_mono r(1))
then have a: limsup (u o r) ≠ ∞ using  $\langle \text{limsup } u < \infty \rangle$  by auto
have limsup (v o r o s) ≤ limsup v
  by (simp add: comp_assoc limsup_subseq_mono r(1) s(1) strict_mono_o)
then have b: limsup (v o r o s) ≠ ∞ using  $\langle \text{limsup } v < \infty \rangle$  by auto

have  $(\lambda n. (u o a) n + (v o a) n) \longrightarrow \text{limsup } (u o r) + \text{limsup } (v o r o s)$ 
  using l tendsto_add_ereal_general a b by fastforce
moreover have  $(\lambda n. (u o a) n + (v o a) n) = (w o a)$  unfolding w_def by
auto
ultimately have  $(w o a) \longrightarrow \text{limsup } (u o r) + \text{limsup } (v o r o s)$  by simp
then have limsup w = limsup (u o r) + limsup (v o r o s) using l(1) LIMSEQ_unique by blast
then have limsup w ≤ limsup u + limsup v
  using  $\langle \text{limsup } (u o r) \leq \text{limsup } u \rangle \langle \text{limsup } (v o r o s) \leq \text{limsup } v \rangle$  add_mono
by simp
then show ?thesis unfolding w_def by simp
qed

```

There is an asymmetry between liminfs and limsups in *ereal*, as $\infty + (-\infty) = \infty$. This explains why there are more assumptions in the next lemma dealing with liminfs than in the previous one about limsups.

```

lemma ereal_liminf_add_mono:
  fixes u v::nat ⇒ ereal
  assumes  $\neg((\text{liminf } u = \infty \wedge \text{liminf } v = -\infty) \vee (\text{liminf } u = -\infty \wedge \text{liminf } v = \infty))$ 
  shows liminf (λn. u n + v n) ≥ liminf u + liminf v
proof (cases)
  assume  $(\text{liminf } u = -\infty) \vee (\text{liminf } v = -\infty)$ 
  then have  $*$ : liminf u + liminf v = -∞ using assms by auto
  show ?thesis by (simp add: *)
next
  assume  $\neg((\text{liminf } u = -\infty) \vee (\text{liminf } v = -\infty))$ 
  then have liminf u > -∞ liminf v > -∞ by auto

  define w where w =  $(\lambda n. u n + v n)$ 

```

obtain r **where** r : $strict\_mono\ r\ (w\ o\ r) \longrightarrow \liminf\ w$ **using** $\liminf\_subseq\_lim$ **by** $auto$

obtain s **where** s : $strict\_mono\ s\ (u\ o\ r\ o\ s) \longrightarrow \liminf\ (u\ o\ r)$ **using** $\liminf\_subseq\_lim$ **by** $auto$

obtain t **where** t : $strict\_mono\ t\ (v\ o\ r\ o\ s\ o\ t) \longrightarrow \liminf\ (v\ o\ r\ o\ s)$ **using** $\liminf\_subseq\_lim$ **by** $auto$

define a **where** $a = r\ o\ s\ o\ t$

have $strict\_mono\ a$ **using** $r\ s\ t$ **by** ($simp\ add$: $a\_def\ strict\_mono\_o$)

have l : $(w\ o\ a) \longrightarrow \liminf\ w$

$(u\ o\ a) \longrightarrow \liminf\ (u\ o\ r)$

$(v\ o\ a) \longrightarrow \liminf\ (v\ o\ r\ o\ s)$

apply ($metis\ (no\_types,\ lifting)\ r(2)\ s(1)\ t(1)\ LIMSEQ\_subseq\_LIMSEQ\ a\_def\ comp\_assoc$)

apply ($metis\ (no\_types,\ lifting)\ s(2)\ t(1)\ LIMSEQ\_subseq\_LIMSEQ\ a\_def\ comp\_assoc$)

apply ($metis\ (no\_types,\ lifting)\ t(2)\ a\_def\ comp\_assoc$)

done

have $\liminf\ (u\ o\ r) \geq \liminf\ u$ **by** ($simp\ add$: $\liminf\_subseq\_mono\ r(1)$)

then have a : $\liminf\ (u\ o\ r) \neq -\infty$ **using** $\langle \liminf\ u > -\infty \rangle$ **by** $auto$

have $\liminf\ (v\ o\ r\ o\ s) \geq \liminf\ v$

by ($simp\ add$: $comp\_assoc\ \liminf\_subseq\_mono\ r(1)\ s(1)\ strict\_mono\_o$)

then have b : $\liminf\ (v\ o\ r\ o\ s) \neq -\infty$ **using** $\langle \liminf\ v > -\infty \rangle$ **by** $auto$

have $(\lambda n. (u\ o\ a)\ n + (v\ o\ a)\ n) \longrightarrow \liminf\ (u\ o\ r) + \liminf\ (v\ o\ r\ o\ s)$

using $l\ tendsto\_add\_ereal\_general\ a\ b$ **by** $fastforce$

moreover have $(\lambda n. (u\ o\ a)\ n + (v\ o\ a)\ n) = (w\ o\ a)$ **unfolding** w_def **by** $auto$

ultimately have $(w\ o\ a) \longrightarrow \liminf\ (u\ o\ r) + \liminf\ (v\ o\ r\ o\ s)$ **by** $simp$

then have $\liminf\ w = \liminf\ (u\ o\ r) + \liminf\ (v\ o\ r\ o\ s)$ **using** $l(1)\ LIMSEQ\_unique$ **by** $blast$

then have $\liminf\ w \geq \liminf\ u + \liminf\ v$

using $\langle \liminf\ (u\ o\ r) \geq \liminf\ u \rangle\ \langle \liminf\ (v\ o\ r\ o\ s) \geq \liminf\ v \rangle\ add\_mono$

by $simp$

then show $?thesis$ **unfolding** w_def **by** $simp$

qed

lemma $ereal\_limsup\_lim\_add$:

fixes $u\ v::nat \Rightarrow ereal$

assumes $u \longrightarrow a\ abs(a) \neq \infty$

shows $\limsup\ (\lambda n. u\ n + v\ n) = a + \limsup\ v$

proof $-$

have $\limsup\ u = a$ **using** $assms(1)$ **using** $tendsto\_iff\_Liminf\_eq\_Limsup\ trivial\_limit\_at\_top\_linorder$ **by** $blast$

have $(\lambda n. -u\ n) \longrightarrow -a$ **using** $assms(1)$ **by** $auto$

then have $\limsup\ (\lambda n. -u\ n) = -a$ **using** $tendsto\_iff\_Liminf\_eq\_Limsup\ trivial\_limit\_at\_top\_linorder$ **by** $blast$

```

have limsup ( $\lambda n. u\ n + v\ n$ )  $\leq$  limsup  $u$  + limsup  $v$ 
  by (rule ereal_limsup_add_mono)
then have up: limsup ( $\lambda n. u\ n + v\ n$ )  $\leq a$  + limsup  $v$  using  $\langle$ limsup  $u = a$  $\rangle$ 
by simp

have a: limsup ( $\lambda n. (u\ n + v\ n) + (-u\ n)$ )  $\leq$  limsup ( $\lambda n. u\ n + v\ n$ ) + limsup
( $\lambda n. -u\ n$ )
  by (rule ereal_limsup_add_mono)
have eventually ( $\lambda n. u\ n = \text{ereal}(\text{real\_of\_ereal}(u\ n))$ ) sequentially using assms
  real_lim_then_eventually_real by auto
moreover have  $\bigwedge x. x = \text{ereal}(\text{real\_of\_ereal}(x)) \implies x + (-x) = 0$ 
  by (metis plus_ereal.simps(1) right_minus uminus_ereal.simps(1) zero_ereal_def)
ultimately have eventually ( $\lambda n. u\ n + (-u\ n) = 0$ ) sequentially
  by (metis (mono_tags, lifting) eventually_mono)
moreover have  $\bigwedge n. u\ n + (-u\ n) = 0 \implies u\ n + v\ n + (-u\ n) = v\ n$ 
  by (metis add commute add.left_commute add.left_neutral)
ultimately have eventually ( $\lambda n. u\ n + v\ n + (-u\ n) = v\ n$ ) sequentially
  using eventually_mono by force
then have limsup  $v = \text{limsup } (\lambda n. u\ n + v\ n + (-u\ n))$  using Limsup_eq by
force
then have limsup  $v \leq \text{limsup } (\lambda n. u\ n + v\ n) - a$  using a  $\langle$ limsup ( $\lambda n. -u\ n$ )
 $= -a$  $\rangle$  by (simp add: minus_ereal_def)
then have limsup ( $\lambda n. u\ n + v\ n$ )  $\geq a$  + limsup  $v$  using assms(2) by (metis
add commute ereal_le_minus)
then show ?thesis using up by simp
qed

```

```

lemma ereal_limsup_lim_mult:
  fixes  $u\ v::\text{nat} \Rightarrow \text{ereal}$ 
  assumes  $u \longrightarrow a$   $a > 0$   $a \neq \infty$ 
  shows limsup ( $\lambda n. u\ n * v\ n$ )  $= a * \text{limsup } v$ 
proof -
  define  $w$  where  $w = (\lambda n. u\ n * v\ n)$ 
  obtain  $r$  where  $r$ : strict_mono  $r$  ( $v\ o\ r$ )  $\longrightarrow \text{limsup } v$  using limsup_subseq_lim
  by auto
  have ( $u\ o\ r$ )  $\longrightarrow a$  using assms(1) LIMSEQ_subseq_LIMSEQ  $r$  by auto
  with tendsto_mult_ereal[OF this  $r(2)$ ] have ( $\lambda n. (u\ o\ r)\ n * (v\ o\ r)\ n$ )  $\longrightarrow$ 
 $a * \text{limsup } v$  using assms(2) assms(3) by auto
  moreover have  $\bigwedge n. (w\ o\ r)\ n = (u\ o\ r)\ n * (v\ o\ r)\ n$  unfolding  $w\_def$  by
  auto
  ultimately have ( $w\ o\ r$ )  $\longrightarrow a * \text{limsup } v$  unfolding  $w\_def$  by presburger
  then have limsup ( $w\ o\ r$ )  $= a * \text{limsup } v$  by (simp add: tendsto_iff_Liminf_eq_Limsup)
  then have  $I$ : limsup  $w \geq a * \text{limsup } v$  by (metis limsup_subseq_mono  $r(1)$ )

  obtain  $s$  where  $s$ : strict_mono  $s$  ( $w\ o\ s$ )  $\longrightarrow \text{limsup } w$  using limsup_subseq_lim
  by auto
  have *: ( $u\ o\ s$ )  $\longrightarrow a$  using assms(1) LIMSEQ_subseq_LIMSEQ  $s$  by auto
  have eventually ( $\lambda n. (u\ o\ s)\ n > 0$ ) sequentially using assms(2) * order_tendsto_iff
  by blast

```

moreover have *eventually* $(\lambda n. (u \circ s) \ n < \infty)$ *sequentially* **using** *assms*(3) * *order_tendsto_iff* **by** *blast*
moreover have $(w \circ s) \ n / (u \circ s) \ n = (v \circ s) \ n$ **if** $(u \circ s) \ n > 0$ $(u \circ s) \ n < \infty$ **for** n
unfolding *w_def* **using** *that* **by** *(auto simp add: ereal_divide_eq)*
ultimately have *eventually* $(\lambda n. (w \circ s) \ n / (u \circ s) \ n = (v \circ s) \ n)$ *sequentially*
using *eventually_elim2* **by** *force*
moreover have $(\lambda n. (w \circ s) \ n / (u \circ s) \ n) \longrightarrow (\limsup w) / a$
apply *(rule tendsto_divide_ereal[OF s(2) *])* **using** *assms*(2) *assms*(3) **by** *auto*
ultimately have $(v \circ s) \longrightarrow (\limsup w) / a$ **using** *Lim_transform_eventually*
by *fastforce*
then have $\limsup (v \circ s) = (\limsup w) / a$ **by** *(simp add: tendsto_iff_Liminf_eq_Limsup)*
then have $\limsup v \geq (\limsup w) / a$ **by** *(metis limsup_subseq_mono s(1))*
then have $a * \limsup v \geq \limsup w$ **using** *assms*(2) *assms*(3) **by** *(simp add: ereal_divide_le_pos)*
then show *?thesis* **using** *I* **unfolding** *w_def* **by** *auto*
qed

lemma *ereal_liminf_lim_mult*:

fixes $u \ v :: \text{nat} \Rightarrow \text{ereal}$
assumes $u \longrightarrow a$ $a > 0$ $a \neq \infty$
shows $\liminf (\lambda n. u \ n * v \ n) = a * \liminf v$
proof –
define w **where** $w = (\lambda n. u \ n * v \ n)$
obtain r **where** r : *strict_mono* r $(v \circ r) \longrightarrow \liminf v$ **using** *liminf_subseq_lim*
by *auto*
have $(u \circ r) \longrightarrow a$ **using** *assms*(1) *LIMSEQ_subseq_LIMSEQ* r **by** *auto*
with *tendsto_mult_ereal[OF this r(2)]* **have** $(\lambda n. (u \circ r) \ n * (v \circ r) \ n) \longrightarrow a * \liminf v$ **using** *assms*(2) *assms*(3) **by** *auto*
moreover have $\bigwedge n. (w \circ r) \ n = (u \circ r) \ n * (v \circ r) \ n$ **unfolding** *w_def* **by** *auto*
ultimately have $(w \circ r) \longrightarrow a * \liminf v$ **unfolding** *w_def* **by** *presburger*
then have $\liminf (w \circ r) = a * \liminf v$ **by** *(simp add: tendsto_iff_Liminf_eq_Limsup)*
then have I : $\liminf w \leq a * \liminf v$ **by** *(metis liminf_subseq_mono r(1))*

obtain s **where** s : *strict_mono* s $(w \circ s) \longrightarrow \liminf w$ **using** *liminf_subseq_lim*
by *auto*

have $*$: $(u \circ s) \longrightarrow a$ **using** *assms*(1) *LIMSEQ_subseq_LIMSEQ* s **by** *auto*
have *eventually* $(\lambda n. (u \circ s) \ n > 0)$ *sequentially* **using** *assms*(2) * *order_tendsto_iff*
by *blast*

moreover have *eventually* $(\lambda n. (u \circ s) \ n < \infty)$ *sequentially* **using** *assms*(3) * *order_tendsto_iff* **by** *blast*

moreover have $(w \circ s) \ n / (u \circ s) \ n = (v \circ s) \ n$ **if** $(u \circ s) \ n > 0$ $(u \circ s) \ n < \infty$ **for** n

unfolding *w_def* **using** *that* **by** *(auto simp add: ereal_divide_eq)*
ultimately have *eventually* $(\lambda n. (w \circ s) \ n / (u \circ s) \ n = (v \circ s) \ n)$ *sequentially*
using *eventually_elim2* **by** *force*
moreover have $(\lambda n. (w \circ s) \ n / (u \circ s) \ n) \longrightarrow (\liminf w) / a$


```

    apply (rule tendsto_divide_ereal[OF s(2) *]) using assms(2) assms(3) by
    auto
    ultimately have (v o s) ⟶ (liminf w) / a using Lim_transform_eventually
    by fastforce
    then have liminf (v o s) = (liminf w) / a by (simp add: tendsto_iff_Liminf_eq_Limsup)
    then have liminf v ≤ (liminf w) / a by (metis liminf_subseq_mono s(1))
    then have a * liminf v ≤ liminf w using assms(2) assms(3) by (simp add:
    ereal_le_divide_pos)
    then show ?thesis using I unfolding w_def by auto
  qed

lemma ereal_liminf_lim_add:
  fixes u v::nat ⇒ ereal
  assumes u ⟶ a abs(a) ≠ ∞
  shows liminf (λn. u n + v n) = a + liminf v
proof -
  have liminf u = a using assms(1) tendsto_iff_Liminf_eq_Limsup trivial_limit_at_top_linorder
  by blast
  then have *: abs(liminf u) ≠ ∞ using assms(2) by auto
  have (λn. -u n) ⟶ -a using assms(1) by auto
  then have liminf (λn. -u n) = -a using tendsto_iff_Liminf_eq_Limsup trivial_limit_at_top_linorder by blast
  then have **: abs(liminf (λn. -u n)) ≠ ∞ using assms(2) by auto

  have liminf (λn. u n + v n) ≥ liminf u + liminf v
    apply (rule ereal_liminf_add_mono) using * by auto
  then have up: liminf (λn. u n + v n) ≥ a + liminf v using ⟨liminf u = a⟩ by
  simp

  have a: liminf (λn. (u n + v n) + (-u n)) ≥ liminf (λn. u n + v n) + liminf
  (λn. -u n)
    apply (rule ereal_liminf_add_mono) using ** by auto
  have eventually (λn. u n = ereal(real_of_ereal(u n))) sequentially using assms
    real_lim_then_eventually_real by auto
  moreover have ∧x. x = ereal(real_of_ereal(x)) ⟹ x + (-x) = 0
    by (metis plus_ereal.simps(1) right_minus uminus_ereal.simps(1) zero_ereal_def)
  ultimately have eventually (λn. u n + (-u n) = 0) sequentially
    by (metis (mono_tags, lifting) eventually_mono)
  moreover have ∧n. u n + (-u n) = 0 ⟹ u n + v n + (-u n) = v n
    by (metis add commute add.left_commute add.left_neutral)
  ultimately have eventually (λn. u n + v n + (-u n) = v n) sequentially
    using eventually_mono by force
  then have liminf v = liminf (λn. u n + v n + (-u n)) using Liminf_eq by
  force
  then have liminf v ≥ liminf (λn. u n + v n) - a using a ⟨liminf (λn. -u n) =
  -a⟩ by (simp add: minus_ereal_def)
  then have liminf (λn. u n + v n) ≤ a + liminf v using assms(2) by (metis
  add commute ereal_minus_le)
  then show ?thesis using up by simp

```

qed

```

lemma ereal_liminf_limsup_add:
  fixes u v::nat  $\Rightarrow$  ereal
  shows liminf ( $\lambda n. u\ n + v\ n$ )  $\leq$  liminf u + limsup v
proof (cases)
  assume limsup v =  $\infty$   $\vee$  liminf u =  $\infty$ 
  then show ?thesis by auto
next
  assume  $\neg$ (limsup v =  $\infty$   $\vee$  liminf u =  $\infty$ )
  then have limsup v <  $\infty$  liminf u <  $\infty$  by auto

  define w where w = ( $\lambda n. u\ n + v\ n$ )
  obtain r where r: strict_mono r (u o r)  $\longrightarrow$  liminf u using liminf_subseq_lim
  by auto
  obtain s where s: strict_mono s (w o r o s)  $\longrightarrow$  liminf (w o r) using
  liminf_subseq_lim by auto
  obtain t where t: strict_mono t (v o r o s o t)  $\longrightarrow$  limsup (v o r o s) using
  limsup_subseq_lim by auto

  define a where a = r o s o t
  have strict_mono a using r s t by (simp add: a_def strict_mono_o)
  have l:(u o a)  $\longrightarrow$  liminf u
    (w o a)  $\longrightarrow$  liminf (w o r)
    (v o a)  $\longrightarrow$  limsup (v o r o s)
  apply (metis (no_types, lifting) r(2) s(1) t(1) LIMSEQ_subseq_LIMSEQ a_def
  comp_assoc)
  apply (metis (no_types, lifting) s(2) t(1) LIMSEQ_subseq_LIMSEQ a_def
  comp_assoc)
  apply (metis (no_types, lifting) t(2) a_def comp_assoc)
  done

  have liminf (w o r)  $\geq$  liminf w by (simp add: liminf_subseq_mono r(1))
  have limsup (v o r o s)  $\leq$  limsup v
    by (simp add: comp_assoc limsup_subseq_mono r(1) s(1) strict_mono_o)
  then have b: limsup (v o r o s) <  $\infty$  using  $\langle$ limsup v <  $\infty$  $\rangle$  by auto

  have ( $\lambda n. (u\ o\ a)\ n + (v\ o\ a)\ n$ )  $\longrightarrow$  liminf u + limsup (v o r o s)
  apply (rule tendsto_add_ereal_general) using b  $\langle$ liminf u <  $\infty$  $\rangle$  l(1) l(3) by
  force+
  moreover have ( $\lambda n. (u\ o\ a)\ n + (v\ o\ a)\ n$ ) = (w o a) unfolding w_def by
  auto
  ultimately have (w o a)  $\longrightarrow$  liminf u + limsup (v o r o s) by simp
  then have liminf (w o r) = liminf u + limsup (v o r o s) using l(2) using
  LIMSEQ_unique by blast
  then have liminf w  $\leq$  liminf u + limsup v
    using  $\langle$ liminf (w o r)  $\geq$  liminf w $\rangle$   $\langle$ limsup (v o r o s)  $\leq$  limsup v $\rangle$ 
    by (metis add_mono_thms_linordered_semiring(2) le_less_trans not_less)
  then show ?thesis unfolding w_def by simp

```

qed

```
lemma ereal_liminf_limsup_minus:
  fixes u v::nat ⇒ ereal
  shows liminf (λn. u n - v n) ≤ limsup u - limsup v
  unfolding minus_ereal_def
  apply (subst add.commute)
  apply (rule order_trans[OF ereal_liminf_limsup_add])
  using ereal_Limsup_u_minus[of sequentially λn. - v n]
  apply (simp add: add.commute)
  done
```

```
lemma liminf_minus_ennreal:
  fixes u v::nat ⇒ ennreal
  shows (⋀n. v n ≤ u n) ⇒ liminf (λn. u n - v n) ≤ limsup u - limsup v
  unfolding liminf_SUP_INF limsup_INF_SUP
  including ennreal.lifting
proof (transfer, clarsimp)
  fix v u :: nat ⇒ ereal assume *: ∀x. 0 ≤ v x ∀x. 0 ≤ u x ⋀n. v n ≤ u n
  moreover have 0 ≤ limsup u - limsup v
    using * by (intro ereal_diff_positive Limsup_mono always_eventually) simp
  moreover have 0 ≤ Sup (u ‘ {x..}) for x
    using * by (intro SUP_upper2[of x]) auto
  moreover have 0 ≤ Sup (v ‘ {x..}) for x
    using * by (intro SUP_upper2[of x]) auto
  ultimately show (SUP n. INF n∈{n..}. max 0 (u n - v n))
    ≤ max 0 ((INF x. max 0 (Sup (u ‘ {x..}))) - (INF x. max 0 (Sup (v ‘
{x..}))))))
    by (auto simp: * ereal_diff_positive max.absorb2 liminf_SUP_INF[symmetric]
limsup_INF_SUP[symmetric] ereal_liminf_limsup_minus)
qed
```

4.5.4 Relate extended reals and the indicator function

```
lemma ereal_indicator_le_0: (indicator S x::ereal) ≤ 0 ⟷ x ∉ S
  by (auto split: split_indicator simp: one_ereal_def)
```

```
lemma ereal_indicator: ereal (indicator A x) = indicator A x
  by (auto simp: indicator_def one_ereal_def)
```

```
lemma ereal_mult_indicator: ereal (x * indicator A y) = ereal x * indicator A y
  by (simp split: split_indicator)
```

```
lemma ereal_indicator_mult: ereal (indicator A y * x) = indicator A y * ereal x
  by (simp split: split_indicator)
```

```
lemma ereal_indicator_nonneg[simp, intro]: 0 ≤ (indicator A x::ereal)
  unfolding indicator_def by auto
```

```

lemma indicator_inter_arith_ereal: indicator A x * indicator B x = (indicator
(A ∩ B) x :: ereal)
  by (simp split: split_indicator)

end

```

4.6 Radius of Convergence and Summation Tests

```

theory Summation_Tests
imports
  Complex_Main
  HOL-Library.Discrete
  HOL-Library.Extended_Real
  HOL-Library.Liminf_Limsup
  Extended_Real_Limits
begin

```

The definition of the radius of convergence of a power series, various summability tests, lemmas to compute the radius of convergence etc.

4.6.1 Convergence tests for infinite sums

Root test

```

lemma limsup_root_powser:
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  shows limsup ( $\lambda n.$  ereal (root n (norm (f n * z n)))) =
    limsup ( $\lambda n.$  ereal (root n (norm (f n)))) * ereal (norm z)
proof -
  have A: ( $\lambda n.$  ereal (root n (norm (f n * z n)))) =
    ( $\lambda n.$  ereal (root n (norm (f n)))) * ereal (norm z) (is ?g = ?h)
  proof
    fix n show ?g n = ?h n
    by (cases n = 0) (simp_all add: norm_mult real_root_mult real_root_pos2
norm_power)
  qed
  show ?thesis by (subst A, subst limsup_ereal_mult_right) simp_all
qed

```

```

lemma limsup_root_limit:
  assumes ( $\lambda n.$  ereal (root n (norm (f n))))  $\longrightarrow$  l (is ?g  $\longrightarrow$   $\_$ )
  shows limsup ( $\lambda n.$  ereal (root n (norm (f n)))) = l
proof -
  from assms have convergent ?g lim ?g = l
    unfolding convergent_def by (blast intro: limI)+
  with convergent_limsup_cl show ?thesis by force
qed

```

```

lemma limsup_root_limit':
  assumes  $(\lambda n. \text{root } n \text{ (norm (f n))}) \longrightarrow l$ 
  shows  $\text{limsup } (\lambda n. \text{ereal } (\text{root } n \text{ (norm (f n))})) = \text{ereal } l$ 
  by (intro limsup_root_limit tendsto_ereal assms)

theorem root_test_convergence':
  fixes  $f :: \text{nat} \Rightarrow 'a :: \text{banach}$ 
  defines  $l \equiv \text{limsup } (\lambda n. \text{ereal } (\text{root } n \text{ (norm (f n))}))$ 
  assumes  $l: l < 1$ 
  shows summable f
proof -
  have  $0 = \text{limsup } (\lambda n. 0)$  by (simp add: Limsup_const)
  also have  $\dots \leq l$  unfolding l_def by (intro Limsup_mono) (simp_all add:
real_root_ge_zero)
  finally have  $l \geq 0$  by simp
  with l obtain  $l'$  where  $l': l = \text{ereal } l'$  by (cases l) simp_all

  define c where  $c = (1 - l') / 2$ 
  from l and  $\langle l \geq 0 \rangle$  have  $c: l + c > l' + c \geq 0 \quad l' + c < 1$  unfolding c_def
  by (simp_all add: field_simps l')
  have  $\forall C > l. \text{eventually } (\lambda n. \text{ereal } (\text{root } n \text{ (norm (f n))})) < C$  sequentially
  by (subst Limsup_le_iff[symmetric]) (simp add: l_def)
  with c have  $\text{eventually } (\lambda n. \text{ereal } (\text{root } n \text{ (norm (f n))})) < l + \text{ereal } c$  sequentially
by simp
  with eventually_gt_at_top[of 0::nat]
  have  $\text{eventually } (\lambda n. \text{norm } (f n) \leq (l' + c) ^ n)$  sequentially
proof eventually_elim
  fix  $n :: \text{nat}$  assume  $n: n > 0$ 
  assume  $\text{ereal } (\text{root } n \text{ (norm (f n))}) < l + \text{ereal } c$ 
  hence  $\text{root } n \text{ (norm (f n))} \leq l' + c$  by (simp add: l')
  with c n have  $\text{root } n \text{ (norm (f n))} ^ n \leq (l' + c) ^ n$ 
  by (intro power_mono) (simp_all add: real_root_ge_zero)
  also from n have  $\text{root } n \text{ (norm (f n))} ^ n = \text{norm } (f n)$  by simp
  finally show  $\text{norm } (f n) \leq (l' + c) ^ n$  by simp
qed
thus ?thesis
  by (rule summable_comparison_test_ev[OF _ summable_geometric]) (simp
add: c)
qed

theorem root_test_divergence:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \text{banach}$ 
  defines  $l \equiv \text{limsup } (\lambda n. \text{ereal } (\text{root } n \text{ (norm (f n))}))$ 
  assumes  $l: l > 1$ 
  shows  $\neg \text{summable } f$ 
proof
  assume summable f
  hence bounded: Bseq f by (simp add: summable_imp_Bseq)

```

have $0 = \text{limsup } (\lambda n. 0)$ by (simp add: Limsup_const)
 also have $\dots \leq l$ unfolding l_def by (intro Limsup_mono) (simp_all add: real_root_ge_zero)

finally have $l_{\text{nonneg}}: l \geq 0$ by simp

define c where $c = (\text{if } l = \infty \text{ then } 2 \text{ else } 1 + (\text{real\_of\_ereal } l - 1) / 2)$
 from l_{nonneg} consider $l = \infty \mid \exists l'. l = \text{ereal } l'$ by (cases l) simp_all
 hence $c: c > 1 \wedge \text{ereal } c < l$ by cases (insert l, auto simp: c_def field_simps)

have $\text{unbounded}: \neg \text{bdd\_above } \{n. \text{root } n (\text{norm } (f \ n)) > c\}$

proof

assume $\text{bdd\_above } \{n. \text{root } n (\text{norm } (f \ n)) > c\}$

then obtain N where $\forall n. \text{root } n (\text{norm } (f \ n)) > c \longrightarrow n \leq N$ unfolding bdd_above_def by blast

hence $\exists N. \forall n \geq N. \text{root } n (\text{norm } (f \ n)) \leq c$

by (intro exI[of _ N + 1]) (force simp: not_less_eq_eq[symmetric])

hence $\text{eventually } (\lambda n. \text{root } n (\text{norm } (f \ n)) \leq c)$ sequentially

by (auto simp: eventually_at_top_linorder)

hence $l \leq c$ unfolding l_def by (intro Limsup_bounded) simp_all

with c show False by auto

qed

from bounded obtain K where $K: K > 0 \wedge n. \text{norm } (f \ n) \leq K$ using BseqE
 by blast

define n where $n = \text{nat } \lceil \log c \ K \rceil$

from unbounded have $\exists m > n. c < \text{root } m (\text{norm } (f \ m))$ unfolding bdd_above_def
 by (auto simp: not_le)

then obtain m where $m: n < m \wedge c < \text{root } m (\text{norm } (f \ m))$ by auto

from $c \ K$ have $K = c \text{ powr } \log c \ K$ by (simp add: powr_def log_def)

also from c have $c \text{ powr } \log c \ K \leq c \text{ powr } \text{real } n$ unfolding n_def

by (intro powr_mono, linarith, simp)

finally have $K \leq c ^ n$ using c by (simp add: powr_realpow)

also from $c \ m$ have $c ^ n < c ^ m$ by simp

also from $c \ m$ have $c ^ m < \text{root } m (\text{norm } (f \ m)) ^ m$ by (intro power_strict_mono) simp_all

also from m have $\dots = \text{norm } (f \ m)$ by simp

finally show False using $K(2)[\text{of } m]$ by simp

qed

Cauchy's condensation test

context

fixes $f :: \text{nat} \Rightarrow \text{real}$

begin

private lemma *condensation_inequality*:

assumes $\text{mono}: \bigwedge m \ n. 0 < m \implies m \leq n \implies f \ n \leq f \ m$

shows $(\sum_{k=1..<n} f \ k) \geq (\sum_{k=1..<n} f \ (2 * 2 ^ \text{Discrete.log } k))$ (is ?thesis1)
 $(\sum_{k=1..<n} f \ k) \leq (\sum_{k=1..<n} f \ (2 ^ \text{Discrete.log } k))$ (is ?thesis2)

by (intro sum_mono mono Discrete.log_exp2_ge Discrete.log_exp2_le, simp, simp)+

private lemma condensation_condense1: $(\sum_{k=1..<2^n} f (2^k \wedge \text{Discrete.log } k)) = (\sum_{k < n} 2^k * f (2^k))$

proof (induction n)

case (Suc n)

have $\{1..<2^{\text{Suc } n}\} = \{1..<2^n\} \cup \{2^n..<(2^{\text{Suc } n} :: \text{nat})\}$ by auto

also have $(\sum_{k \in \dots} f (2^k \wedge \text{Discrete.log } k)) = (\sum_{k < n} 2^k * f (2^k)) + (\sum_{k = 2^n..<2^{\text{Suc } n}} f (2^{\text{Discrete.log } k}))$

by (subst sum.union_disjoint) (insert Suc, auto)

also have $\text{Discrete.log } k = n$ if $k \in \{2^n..<2^{\text{Suc } n}\}$ for k using that by (intro Discrete.log_eqI) simp_all

hence $(\sum_{k = 2^n..<2^{\text{Suc } n}} f (2^{\text{Discrete.log } k})) = (\sum_{\_ :: \text{nat} = 2^n..<2^{\text{Suc } n}} f (2^n))$

by (intro sum.cong) simp_all

also have $\dots = 2^n * f (2^n)$ by (simp)

finally show ?case by simp

qed simp

private lemma condensation_condense2: $(\sum_{k=1..<2^n} f (2 * 2^k \wedge \text{Discrete.log } k)) = (\sum_{k < n} 2^k * f (2^{\text{Suc } k}))$

proof (induction n)

case (Suc n)

have $\{1..<2^{\text{Suc } n}\} = \{1..<2^n\} \cup \{2^n..<(2^{\text{Suc } n} :: \text{nat})\}$ by auto

also have $(\sum_{k \in \dots} f (2 * 2^k \wedge \text{Discrete.log } k)) = (\sum_{k < n} 2^k * f (2^{\text{Suc } k})) + (\sum_{k = 2^n..<2^{\text{Suc } n}} f (2 * 2^{\text{Discrete.log } k}))$

by (subst sum.union_disjoint) (insert Suc, auto)

also have $\text{Discrete.log } k = n$ if $k \in \{2^n..<2^{\text{Suc } n}\}$ for k using that by (intro Discrete.log_eqI) simp_all

hence $(\sum_{k = 2^n..<2^{\text{Suc } n}} f (2 * 2^{\text{Discrete.log } k})) = (\sum_{\_ :: \text{nat} = 2^n..<2^{\text{Suc } n}} f (2^{\text{Suc } n}))$

by (intro sum.cong) simp_all

also have $\dots = 2^n * f (2^{\text{Suc } n})$ by (simp)

finally show ?case by simp

qed simp

theorem condensation_test:

assumes mono: $\bigwedge m. 0 < m \implies f (\text{Suc } m) \leq f m$

assumes nonneg: $\bigwedge n. f n \geq 0$

shows summable f $\longleftrightarrow$ summable $(\lambda n. 2^n * f (2^n))$

proof —

define f' where $f' n = (\text{if } n = 0 \text{ then } 0 \text{ else } f n)$ for n

from mono have mono': decseq $(\lambda n. f (\text{Suc } n))$ by (intro decseq_SucI) simp

hence mono': $f n \leq f m$ if $m \leq n$ $m > 0$ for m n

using that decseqD[OF mono', of m - 1 n - 1] by simp

have $(\lambda n. f (Suc\ n)) = (\lambda n. f' (Suc\ n))$ **by** (intro ext) (simp add: f'_def)
 hence summable $f \longleftrightarrow$ summable f'
 by (subst (1 2) summable_Suc_iff [symmetric]) (simp only:)
 also have $\dots \longleftrightarrow$ convergent $(\lambda n. \sum_{k < n}. f' k)$ **unfolding** summable_iff_convergent
 ..
 also have monoseq $(\lambda n. \sum_{k < n}. f' k)$ **unfolding** f'_def
 by (intro mono_SucI1) (auto intro!: mult_nonneg_nonneg nonneg)
 hence convergent $(\lambda n. \sum_{k < n}. f' k) \longleftrightarrow$ Bseq $(\lambda n. \sum_{k < n}. f' k)$
 by (rule monoseq_imp_convergent_iff_Bseq)
 also have $\dots \longleftrightarrow$ Bseq $(\lambda n. \sum_{k=1..<n}. f' k)$ **unfolding** One_nat_def
 by (subst sum_shift_lb_Suc0_0_upt) (simp_all add: f'_def atLeast0LessThan)
 also have $\dots \longleftrightarrow$ Bseq $(\lambda n. \sum_{k=1..<n}. f k)$ **unfolding** f'_def **by** simp
 also have $\dots \longleftrightarrow$ Bseq $(\lambda n. \sum_{k=1..<2^n}. f k)$
 by (rule nonneg_incseq_Bseq_subseq_iff [symmetric])
 (auto intro!: sum_nonneg incseq_SucI nonneg simp: strict_mono_def)
 also have $\dots \longleftrightarrow$ Bseq $(\lambda n. \sum_{k < n}. 2^k * f (2^k))$
proof (intro iffI)
 assume A: Bseq $(\lambda n. \sum_{k=1..<2^n}. f k)$
 have eventually $(\lambda n. norm (\sum_{k < n}. 2^k * f (2^{Suc\ k})) \leq norm (\sum_{k=1..<2^n}. f k))$ sequentially
proof (intro always_eventually allI)
 fix n :: nat
 have norm $(\sum_{k < n}. 2^k * f (2^{Suc\ k})) = (\sum_{k < n}. 2^k * f (2^{Suc\ k}))$
unfolding real_norm_def
 by (intro abs_of_nonneg sum_nonneg ballI mult_nonneg_nonneg nonneg)
 simp_all
 also have $\dots \leq (\sum_{k=1..<2^n}. f k)$
 by (subst condensation_condense2 [symmetric]) (intro condensation_inequality mono')
 also have $\dots = norm \dots$ **unfolding** real_norm_def
 by (intro abs_of_nonneg [symmetric] sum_nonneg ballI mult_nonneg_nonneg nonneg)
 finally show norm $(\sum_{k < n}. 2^k * f (2^{Suc\ k})) \leq norm (\sum_{k=1..<2^n}. f k)$.
qed
 from this and A have Bseq $(\lambda n. \sum_{k < n}. 2^k * f (2^{Suc\ k}))$ **by** (rule Bseq_eventually_mono)
 from Bseq_mult[OF Bfun_const[of 2] this] have Bseq $(\lambda n. \sum_{k < n}. 2^{Suc\ k} * f (2^{Suc\ k}))$
 by (simp add: sum_distrib_left sum_distrib_right mult_ac)
 hence Bseq $(\lambda n. (\sum_{k=Suc\ 0..<Suc\ n}. 2^k * f (2^k)) + f 1)$
 by (intro Bseq_add, subst sum.shift_bounds_Suc_ivl) (simp add: atLeast0LessThan)
 hence Bseq $(\lambda n. (\sum_{k=0..<Suc\ n}. 2^k * f (2^k)))$
 by (subst sum.atLeast_Suc_lessThan) (simp_all add: add_ac)
 thus Bseq $(\lambda n. (\sum_{k < n}. 2^k * f (2^k)))$
 by (subst (asm) Bseq_Suc_iff) (simp add: atLeast0LessThan)
next
 assume A: Bseq $(\lambda n. (\sum_{k < n}. 2^k * f (2^k)))$
 have eventually $(\lambda n. norm (\sum_{k=1..<2^n}. f k) \leq norm (\sum_{k < n}. 2^k * f (2^k)))$


```

(2^k))) sequentially
proof (intro always_eventually allI)
  fix n :: nat
  have norm ( $\sum_{k=1..2^n} f k$ ) = ( $\sum_{k=1..2^n} f k$ ) unfolding real_norm_def
    by (intro abs_of_nonneg sum_nonneg ballI mult_nonneg_nonneg nonneg)
  also have ...  $\leq (\sum_{k<n} 2^k * f (2^k))$ 
  by (subst condensation_condense1 [symmetric]) (intro condensation_inequality
mono')
  also have ... = norm ... unfolding real_norm_def
  by (intro abs_of_nonneg [symmetric] sum_nonneg ballI mult_nonneg_nonneg
nonneg) simp_all
  finally show norm ( $\sum_{k=1..2^n} f k$ )  $\leq$  norm ( $\sum_{k<n} 2^k * f (2^k)$ ) .
qed
from this and A show Bseq ( $\lambda n. \sum_{k=1..2^n} f k$ ) by (rule Bseq_eventually_mono)
qed
also have monoseq ( $\lambda n. (\sum_{k<n} 2^k * f (2^k))$ )
  by (intro mono_SucI1) (auto intro!: mult_nonneg_nonneg nonneg)
hence Bseq ( $\lambda n. (\sum_{k<n} 2^k * f (2^k))$ )  $\longleftrightarrow$  convergent ( $\lambda n. (\sum_{k<n} 2^k * f (2^k))$ )
by (rule monoseq_imp_convergent_iff Bseq [symmetric])
also have ...  $\longleftrightarrow$  summable ( $\lambda k. 2^k * f (2^k)$ ) by (simp only: summable_iff_convergent)
finally show ?thesis .
qed
end

```

Summability of powers

```

lemma abs_summable_complex_powr_iff:
  summable ( $\lambda n. \text{norm} (\exp (\text{of\_real} (\ln (\text{of\_nat } n)) * s))$ )  $\longleftrightarrow$  Re s < -1
proof (cases Re s  $\leq$  0)
  let ?l =  $\lambda n. \text{complex\_of\_real} (\ln (\text{of\_nat } n))$ 
  case False
  have eventually ( $\lambda n. \text{norm} (1 :: \text{real}) \leq \text{norm} (\exp (?l n * s))$ ) sequentially
    apply (rule eventually_mono [OF eventually_gt_at_top[of 0::nat]])
    using False ge_one_powr_ge_zero by auto
  from summable_comparison_test_ev[OF this] False show ?thesis by (auto simp:
summable_const_iff)
  next
  let ?l =  $\lambda n. \text{complex\_of\_real} (\ln (\text{of\_nat } n))$ 
  case True
  hence summable ( $\lambda n. \text{norm} (\exp (?l n * s))$ )  $\longleftrightarrow$  summable ( $\lambda n. 2^n * \text{norm} (\exp (?l (2^n) * s))$ )
    by (intro condensation_test) (auto intro!: mult_right_mono_neg)
  also have ( $\lambda n. 2^n * \text{norm} (\exp (?l (2^n) * s))$ ) = ( $\lambda n. (2^{\text{powr} (\text{Re } s + 1)}) ^ n$ )
  proof
    fix n :: nat
    have  $2^n * \text{norm} (\exp (?l (2^n) * s)) = \exp (\text{real } n * \ln 2) * \exp (\text{real } n * \ln$ 

```

```

2 * Re s)
  using True by (subst exp_of_nat_mult) (simp add: ln_realpow algebra_simps)
  also have ... = exp (real n * (ln 2 * (Re s + 1)))
    by (simp add: algebra_simps exp_add)
  also have ... = exp (ln 2 * (Re s + 1)) ^ n by (subst exp_of_nat_mult) simp
  also have exp (ln 2 * (Re s + 1)) = 2 powr (Re s + 1) by (simp add:
powr_def)
  finally show 2^n * norm (exp (?l (2^n) * s)) = (2 powr (Re s + 1)) ^ n .
qed
also have summable ...  $\longleftrightarrow$  2 powr (Re s + 1) < 2 powr 0
  by (subst summable_geometric_iff) simp
also have ...  $\longleftrightarrow$  Re s < -1 by (subst powr_less_cancel_iff) (simp, linarith)
finally show ?thesis .
qed

```

```

theorem summable_complex_powr_iff:
  assumes Re s < -1
  shows summable (λn. exp (of_real (ln (of_nat n)) * s))
  by (rule summable_norm_cancel, subst abs_summable_complex_powr_iff) fact

```

```

lemma summable_real_powr_iff: summable (λn. of_nat n powr s :: real)  $\longleftrightarrow$  s
< -1
proof -
  from eventually_gt_at_top[of 0::nat]
  have summable (λn. of_nat n powr s)  $\longleftrightarrow$  summable (λn. exp (ln (of_nat n)
* s))
  by (intro summable_cong) (auto elim!: eventually_mono simp: powr_def)
  also have ...  $\longleftrightarrow$  s < -1 using abs_summable_complex_powr_iff[of of_real
s] by simp
  finally show ?thesis .
qed

```

```

lemma inverse_power_summable:
  assumes s: s ≥ 2
  shows summable (λn. inverse (of_nat n ^ s :: 'a :: {real_normed_div_algebra,banach}))
proof (rule summable_norm_cancel, subst summable_cong)
  from eventually_gt_at_top[of 0::nat]
  show eventually (λn. norm (inverse (of_nat n ^ s :: 'a)) = real_of_nat n powr
(-real s)) at_top
  by eventually_elim (simp add: norm_inverse norm_power powr_minus powr_realpow)
qed (insert s summable_real_powr_iff[of -s], simp_all)

```

```

lemma not_summable_harmonic: ¬summable (λn. inverse (of_nat n) :: 'a ::
real_normed_field)
proof
  assume summable (λn. inverse (of_nat n) :: 'a)
  hence convergent (λn. norm (of_real (∑ k<n. inverse (of_nat k)) :: 'a))
  by (simp add: summable_iff_convergent convergent_norm)
  hence convergent (λn. abs (∑ k<n. inverse (of_nat k)) :: real) by (simp only:

```

```

norm_of_real)
  also have  $(\lambda n. \text{abs } (\sum k < n. \text{inverse } (\text{of\_nat } k)) :: \text{real}) = (\lambda n. \sum k < n. \text{inverse } (\text{of\_nat } k))$ 
  by (intro ext abs_of_nonneg sum_nonneg) auto
  also have  $\text{convergent } \dots \longleftrightarrow \text{summable } (\lambda k. \text{inverse } (\text{of\_nat } k) :: \text{real})$ 
  by (simp add: summable_iff_convergent)
  finally show False using summable_real_pour_iff[of -1] by (simp add: pour_minus)
qed

```

Kummer's test

theorem kummers_test_convergence:

```

  fixes  $f\ p :: \text{nat} \Rightarrow \text{real}$ 
  assumes pos_f:  $\text{eventually } (\lambda n. f\ n > 0)$  sequentially
  assumes nonneg_p:  $\text{eventually } (\lambda n. p\ n \geq 0)$  sequentially
  defines  $l \equiv \liminf (\lambda n. \text{ereal } (p\ n * f\ n / f\ (\text{Suc } n) - p\ (\text{Suc } n)))$ 
  assumes l:  $l > 0$ 
  shows summable f
  unfolding summable_iff_convergent'
proof -
  define r where  $r = (\text{if } l = \infty \text{ then } 1 \text{ else } \text{real\_of\_ereal } l / 2)$ 
  from l have  $r > 0 \wedge \text{of\_real } r < l$  by (cases l) (simp_all add: r_def)
  hence r:  $r > 0 \wedge \text{of\_real } r < l$  by simp_all
  hence  $\text{eventually } (\lambda n. p\ n * f\ n / f\ (\text{Suc } n) - p\ (\text{Suc } n) > r)$  sequentially
    unfolding l_def by (force dest: less_LiminfD)
  moreover from pos_f have  $\text{eventually } (\lambda n. f\ (\text{Suc } n) > 0)$  sequentially
    by (subst eventually_sequentially_Suc)
  ultimately have  $\text{eventually } (\lambda n. p\ n * f\ n - p\ (\text{Suc } n) * f\ (\text{Suc } n) > r * f\ (\text{Suc } n))$  sequentially
    by eventually_elim (simp add: field_simps)
  from eventually_conj[OF pos_f eventually_conj[OF nonneg_p this]]
  obtain m where  $m: \bigwedge n. n \geq m \implies f\ n > 0 \wedge \bigwedge n. n \geq m \implies p\ n \geq 0$ 
     $\bigwedge n. n \geq m \implies p\ n * f\ n - p\ (\text{Suc } n) * f\ (\text{Suc } n) > r * f\ (\text{Suc } n)$ 
  unfolding eventually_at_top_linorder by blast

```

let $?c = (\text{norm } (\sum k \leq m. r * f\ k) + p\ m * f\ m) / r$

have $Bseq\ (\lambda n. (\sum k \leq n + \text{Suc } m. f\ k))$

proof (rule BseqI')

fix $k :: \text{nat}$

define n where $n = k + \text{Suc } m$

have $n: n > m$ by (simp add: n_def)

from r have $r * \text{norm } (\sum k \leq n. f\ k) = \text{norm } (\sum k \leq n. r * f\ k)$

by (simp add: sum_distrib_left[symmetric] abs_mult)

also from n have $\{..n\} = \{..m\} \cup \{\text{Suc } m..n\}$ by auto

hence $(\sum k \leq n. r * f\ k) = (\sum k \in \{..m\} \cup \{\text{Suc } m..n\}. r * f\ k)$ by (simp only:)

also have $\dots = (\sum k \leq m. r * f\ k) + (\sum k = \text{Suc } m..n. r * f\ k)$

by (subst sum.union_disjoint) auto

also have $\text{norm } \dots \leq \text{norm } (\sum k \leq m. r * f\ k) + \text{norm } (\sum k = \text{Suc } m..n. r * f\ k)$

k)
 by (rule norm_triangle_ineq)
 also from r less_imp_le[OF $m(1)$] have $(\sum_{k=\text{Suc } m..n}. r * f k) \geq 0$
 by (intro sum_nonneg) auto
 hence norm $(\sum_{k=\text{Suc } m..n}. r * f k) = (\sum_{k=\text{Suc } m..n}. r * f k)$ by simp
 also have $(\sum_{k=\text{Suc } m..n}. r * f k) = (\sum_{k=m..<n}. r * f (\text{Suc } k))$
 by (subst sum.shift_bounds_Suc_ivl [symmetric])
 (simp only: atLeastLessThanSuc_atLeastAtMost)
 also from m have $\dots \leq (\sum_{k=m..<n}. p k * f k - p (\text{Suc } k) * f (\text{Suc } k))$
 by (intro sum_mono[OF less_imp_le]) simp_all
 also have $\dots = -(\sum_{k=m..<n}. p (\text{Suc } k) * f (\text{Suc } k) - p k * f k)$
 by (simp add: sum_negf [symmetric] algebra_simps)
 also from n have $\dots = p m * f m - p n * f n$
 by (cases n , simp, simp only: atLeastLessThanSuc_atLeastAtMost, subst
 sum_Suc_diff) simp_all
 also from less_imp_le[OF $m(1)$] $m(2)$ n have $\dots \leq p m * f m$ by simp
 finally show norm $(\sum_{k \leq n}. f k) \leq (\text{norm } (\sum_{k \leq m}. r * f k) + p m * f m) /$
 r using r
 by (subst pos_le_divide_eq[OF $r(1)$]) (simp only: mult_ac)
 qed
 moreover have $(\sum_{k \leq n}. f k) \leq (\sum_{k \leq n'}. f k)$ if $\text{Suc } m \leq n \leq n'$ for $n n'$
 using less_imp_le[OF $m(1)$] that by (intro sum_mono2) auto
 ultimately show convergent $(\lambda n. \sum_{k \leq n}. f k)$ by (rule Bseq_monoseq_convergent'_inc)
 qed

theorem kummers_test_divergence:

fixes $f p :: \text{nat} \Rightarrow \text{real}$
 assumes pos_f: eventually $(\lambda n. f n > 0)$ sequentially
 assumes pos_p: eventually $(\lambda n. p n > 0)$ sequentially
 assumes divergent_p: $\neg \text{summable } (\lambda n. \text{inverse } (p n))$
 defines $l \equiv \text{limsup } (\lambda n. \text{ereal } (p n * f n / f (\text{Suc } n) - p (\text{Suc } n)))$
 assumes l: $l < 0$
 shows $\neg \text{summable } f$
 proof
 assume summable f
 from eventually_conj[OF pos_f eventually_conj[OF pos_p Limsup_lessD[OF
 l[unfolded l_def]]]]
 obtain N where $N: \bigwedge n. n \geq N \implies p n > 0 \bigwedge n. n \geq N \implies f n > 0$
 $\bigwedge n. n \geq N \implies p n * f n / f (\text{Suc } n) - p (\text{Suc } n) < 0$
 by (auto simp: eventually_at_top_linorder)
 hence A: $p n * f n < p (\text{Suc } n) * f (\text{Suc } n)$ if $n \geq N$ for n using that $N[\text{of } n]$
 $N[\text{of } \text{Suc } n]$
 by (simp add: field_simps)
 have B: $p n * f n \geq p N * f N$ if $n \geq N$ for n using that and A
 by (induction n rule: dec_induct) (auto intro!: less_imp_le elim!: order.trans)
 have eventually $(\lambda n. \text{norm } (p N * f N * \text{inverse } (p n)) \leq f n)$ sequentially
 apply (rule eventually_mono [OF eventually_ge_at_top[of N]])
 using B N by (auto simp: field_simps abs_of_pos)

```

from this and  $\langle \text{summable } f \rangle$  have summable  $(\lambda n. p\ N * f\ N * \text{inverse } (p\ n))$ 
  by (rule summable_comparison_test_ev)
from summable_mult[OF this, of inverse  $(p\ N * f\ N)$ ] N[OF le_refl]
  have summable  $(\lambda n. \text{inverse } (p\ n))$  by (simp add: field_split_simps)
with divergent_p show False by contradiction
qed

```

Ratio test

theorem *ratio_test_convergence*:

```

fixes f :: nat  $\Rightarrow$  real
assumes pos_f: eventually  $(\lambda n. f\ n > 0)$  sequentially
defines l  $\equiv \text{liminf } (\lambda n. \text{ereal } (f\ n / f\ (\text{Suc } n)))$ 
assumes l: l > 1
shows summable f
proof (rule kummers_test_convergence[OF pos_f])
  note l
  also have l = liminf  $(\lambda n. \text{ereal } (f\ n / f\ (\text{Suc } n) - 1)) + 1$ 
    by (subst Liminf_add_ereal_right[symmetric]) (simp_all add: minus_ereal_def
l_def one_ereal_def)
  finally show liminf  $(\lambda n. \text{ereal } (1 * f\ n / f\ (\text{Suc } n) - 1)) > 0$ 
    by (cases liminf  $(\lambda n. \text{ereal } (1 * f\ n / f\ (\text{Suc } n) - 1)))$  simp_all
qed simp

```

theorem *ratio_test_divergence*:

```

fixes f :: nat  $\Rightarrow$  real
assumes pos_f: eventually  $(\lambda n. f\ n > 0)$  sequentially
defines l  $\equiv \text{limsup } (\lambda n. \text{ereal } (f\ n / f\ (\text{Suc } n)))$ 
assumes l: l < 1
shows  $\neg \text{summable } f$ 
proof (rule kummers_test_divergence[OF pos_f])
  have limsup  $(\lambda n. \text{ereal } (f\ n / f\ (\text{Suc } n) - 1)) + 1 = l$ 
    by (subst Limsup_add_ereal_right[symmetric]) (simp_all add: minus_ereal_def
l_def one_ereal_def)
  also note l
  finally show limsup  $(\lambda n. \text{ereal } (1 * f\ n / f\ (\text{Suc } n) - 1)) < 0$ 
    by (cases limsup  $(\lambda n. \text{ereal } (1 * f\ n / f\ (\text{Suc } n) - 1)))$  simp_all
qed (simp_all add: summable_const_iff)

```

Raabe's test

theorem *raabes_test_convergence*:

```

fixes f :: nat  $\Rightarrow$  real
assumes pos: eventually  $(\lambda n. f\ n > 0)$  sequentially
defines l  $\equiv \text{liminf } (\lambda n. \text{ereal } (\text{of\_nat } n * (f\ n / f\ (\text{Suc } n) - 1)))$ 
assumes l: l > 1
shows summable f
proof (rule kummers_test_convergence)
  let ?l' = liminf  $(\lambda n. \text{ereal } (\text{of\_nat } n * f\ n / f\ (\text{Suc } n) - \text{of\_nat } (\text{Suc } n)))$ 
  have 1 < l by fact

```

```

    also have  $l = \liminf (\lambda n. \text{ereal} (\text{of\_nat } n * f \ n / f \ (\text{Suc } n) - \text{of\_nat} (\text{Suc } n))$ 
    + 1)
    by (simp add: l_def algebra_simps)
    also have  $\dots = ?l' + 1$  by (subst Liminf_add_ereal_right) simp_all
    finally show  $?l' > 0$  by (cases ?l') (simp_all add: algebra_simps)
qed (simp_all add: pos)

```

theorem raabes_test_divergence:

```

fixes  $f :: \text{nat} \Rightarrow \text{real}$ 
assumes pos: eventually  $(\lambda n. f \ n > 0)$  sequentially
defines  $l \equiv \limsup (\lambda n. \text{ereal} (\text{of\_nat } n * (f \ n / f \ (\text{Suc } n) - 1)))$ 
assumes  $l: l < 1$ 
shows  $\neg \text{summable } f$ 
proof (rule kummers_test_divergence)
  let  $?l' = \limsup (\lambda n. \text{ereal} (\text{of\_nat } n * f \ n / f \ (\text{Suc } n) - \text{of\_nat} (\text{Suc } n)))$ 
  note l
  also have  $l = \limsup (\lambda n. \text{ereal} (\text{of\_nat } n * f \ n / f \ (\text{Suc } n) - \text{of\_nat} (\text{Suc } n))$ 
  + 1)
  by (simp add: l_def algebra_simps)
  also have  $\dots = ?l' + 1$  by (subst Limsup_add_ereal_right) simp_all
  finally show  $?l' < 0$  by (cases ?l') (simp_all add: algebra_simps)
qed (insert pos eventually_gt_at_top[of 0::nat] not_summable_harmonic, simp_all)

```

4.6.2 Radius of convergence

The radius of convergence of a power series. This value always exists, ranges from 0 to ∞ , and the power series is guaranteed to converge for all inputs with a norm that is smaller than that radius and to diverge for all inputs with a norm that is greater.

definition $\text{conv\_radius} :: (\text{nat} \Rightarrow 'a :: \text{banach}) \Rightarrow \text{ereal}$ **where**
 $\text{conv\_radius } f = \text{inverse} (\limsup (\lambda n. \text{ereal} (\text{root } n (\text{norm } (f \ n))))))$

lemma $\text{conv\_radius\_cong\_weak}$ [cong]: $(\bigwedge n. f \ n = g \ n) \implies \text{conv\_radius } f = \text{conv\_radius } g$
 by (drule ext) simp_all

lemma $\text{conv\_radius\_nonneg}$: $\text{conv\_radius } f \geq 0$

```

proof -
  have  $0 = \limsup (\lambda n. 0)$  by (subst Limsup_const) simp_all
  also have  $\dots \leq \limsup (\lambda n. \text{ereal} (\text{root } n (\text{norm } (f \ n))))$ 
  by (intro Limsup_mono) (simp_all add: real_root_ge_zero)
  finally show ?thesis
  unfolding conv_radius_def by (auto simp: ereal_inverse_nonneg_iff)
qed

```

lemma conv_radius_zero [simp]: $\text{conv\_radius } (\lambda_. 0) = \infty$
 by (auto simp: conv_radius_def zero_ereal_def [symmetric] Limsup_const)

```

lemma conv_radius_altdef:
  conv_radius f = liminf ( $\lambda n. \text{inverse } (\text{ereal } (\text{root } n \ (\text{norm } (f \ n))))$ )
by (subst Liminf_inverse_ereal) (simp_all add: real_root_ge_zero conv_radius_def)

```

```

lemma conv_radius_cong':
  assumes eventually ( $\lambda x. f \ x = g \ x$ ) sequentially
  shows conv_radius f = conv_radius g
  unfolding conv_radius_altdef by (intro Liminf_eq eventually_mono [OF assms])
auto

```

```

lemma conv_radius_cong:
  assumes eventually ( $\lambda x. \text{norm } (f \ x) = \text{norm } (g \ x)$ ) sequentially
  shows conv_radius f = conv_radius g
  unfolding conv_radius_altdef by (intro Liminf_eq eventually_mono [OF assms])
auto

```

```

theorem abs_summable_in_conv_radius:
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes ereal (norm z) < conv_radius f
  shows summable ( $\lambda n. \text{norm } (f \ n * z^{\wedge} n)$ )
proof (rule root_test_convergence')
  define l where l = limsup ( $\lambda n. \text{ereal } (\text{root } n \ (\text{norm } (f \ n))))$ 
  have 0 = limsup ( $\lambda n. 0$ ) by (simp add: Limsup_const)
  also have ...  $\leq$  l unfolding l_def by (intro Limsup_mono) (simp_all add:
    real_root_ge_zero)
  finally have l_nonneg: l  $\geq$  0 .

```

```

  have limsup ( $\lambda n. \text{root } n \ (\text{norm } (f \ n * z^{\wedge} n))$ ) = l * ereal (norm z) unfolding
    l_def
  by (rule limsup_root_powser)
  also from l_nonneg consider l = 0 | l =  $\infty$  |  $\exists l'. l = \text{ereal } l' \wedge l' > 0$ 
  by (cases l) (auto simp: less_le)
  hence l * ereal (norm z) < 1
  proof cases
  assume l =  $\infty$ 
  hence conv_radius f = 0 unfolding conv_radius_def l_def by simp
  with assms show ?thesis by simp
  next
  assume  $\exists l'. l = \text{ereal } l' \wedge l' > 0$ 
  then obtain l' where l': l = ereal l' 0 < l' by auto
  hence l  $\neq \infty$  by auto
  have l * ereal (norm z) < l * conv_radius f
  by (intro ereal_mult_strict_left_mono) (simp_all add: l' assms)
  also have conv_radius f = inverse l by (simp add: conv_radius_def l_def)
  also from l' have l * inverse l = 1 by simp
  finally show ?thesis .
qed simp_all
  finally show limsup ( $\lambda n. \text{ereal } (\text{root } n \ (\text{norm } (\text{norm } (f \ n * z^{\wedge} n))))$ ) < 1 by
    simp

```

qed

```

lemma summable_in_conv_radius:
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  assumes ereal (norm z) < conv_radius f
  shows summable (λn. f n * z ^ n)
  by (rule summable_norm_cancel, rule abs_summable_in_conv_radius) fact+

theorem not_summable_outside_conv_radius:
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  assumes ereal (norm z) > conv_radius f
  shows ¬summable (λn. f n * z ^ n)
proof (rule root_test_divergence)
  define l where l = limsup (λn. ereal (root n (norm (f n))))
  have 0 = limsup (λn. 0) by (simp add: Limsup_const)
  also have ... ≤ l unfolding l_def by (intro Limsup_mono) (simp_all add:
    real_root_ge_zero)
  finally have l_nonneg: l ≥ 0 .
  from assms have l_nz: l ≠ 0 unfolding conv_radius_def l_def by auto

  have limsup (λn. ereal (root n (norm (f n * z ^ n)))) = l * ereal (norm z)
    unfolding l_def by (rule limsup_root_powser)
  also have ... > 1
  proof (cases l)
    assume l = ∞
    with assms conv_radius_nonneg[of f] show ?thesis
      by (auto simp: zero_ereal_def[symmetric])
  next
    fix l' assume l': l = ereal l'
    from l_nonneg l_nz have 1 = l * inverse l by (auto simp: l' field_simps)
    also from l_nz have inverse l = conv_radius f
      unfolding l_def conv_radius_def by auto
    also from l' l_nz l_nonneg assms have l * ... < l * ereal (norm z)
      by (intro ereal_mult_strict_left_mono) (auto simp: l')
    finally show ?thesis .
  qed (insert l_nonneg, simp_all)
  finally show limsup (λn. ereal (root n (norm (f n * z ^ n)))) > 1 .
qed

```

```

lemma conv_radius_geI:
  assumes summable (λn. f n * z ^ n :: 'a :: {banach, real_normed_div_algebra})
  shows conv_radius f ≥ norm z
  using not_summable_outside_conv_radius[of f z] assms by (force simp: not_le[symmetric])

```

```

lemma conv_radius_leI:
  assumes ¬summable (λn. norm (f n * z ^ n :: 'a :: {banach, real_normed_div_algebra}))
  shows conv_radius f ≤ norm z
  using abs_summable_in_conv_radius[of z f] assms by (force simp: not_le[symmetric])

```



```

lemma conv_radius_leI':
  assumes  $\neg$ summable  $(\lambda n. f\ n * z^{\wedge} n :: 'a :: \{banach, real\_normed\_div\_algebra\})$ 
  shows conv_radius f  $\leq$  norm z
  using summable_in_conv_radius[of z f] assms by (force simp: not_le[symmetric])

```

```

lemma conv_radius_geI_ex:
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes  $\bigwedge r. 0 < r \implies \text{ereal } r < R \implies \exists z. \text{norm } z = r \wedge \text{summable } (\lambda n. f\ n * z^{\wedge} n)$ 
  shows conv_radius f  $\geq$  R
proof (rule linorder_cases[of conv_radius f R])
  assume R: conv_radius f < R
  with conv_radius_nonneg[of f] obtain conv_radius'
    where [simp]: conv_radius f = ereal conv_radius'
    by (cases conv_radius f) simp_all
  define r where r = (if R =  $\infty$  then conv_radius' + 1 else (real_of_ereal R + conv_radius') / 2)
  from R conv_radius_nonneg[of f] have 0 < r  $\wedge$  ereal r < R  $\wedge$  ereal r > conv_radius f
  unfolding r_def by (cases R) (auto simp: r_def field_simps)
  with assms(1)[of r] obtain z where norm z > conv_radius f summable  $(\lambda n. f\ n * z^{\wedge} n)$  by auto
  with not_summable_outside_conv_radius[of f z] show ?thesis by simp
qed simp_all

```

```

lemma conv_radius_geI_ex':
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes  $\bigwedge r. 0 < r \implies \text{ereal } r < R \implies \text{summable } (\lambda n. f\ n * \text{of\_real } r^{\wedge} n)$ 
  shows conv_radius f  $\geq$  R
proof (rule conv_radius_geI_ex)
  fix r assume 0 < r ereal r < R
  with assms[of r] show  $\exists z. \text{norm } z = r \wedge \text{summable } (\lambda n. f\ n * z^{\wedge} n)$ 
    by (intro exI[of _ of_real r :: 'a]) auto
qed

```

```

lemma conv_radius_leI_ex:
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes R  $\geq$  0
  assumes  $\bigwedge r. 0 < r \implies \text{ereal } r > R \implies \exists z. \text{norm } z = r \wedge \neg \text{summable } (\lambda n. \text{norm } (f\ n * z^{\wedge} n))$ 
  shows conv_radius f  $\leq$  R
proof (rule linorder_cases[of conv_radius f R])
  assume R: conv_radius f > R
  from R assms(1) obtain R' where R': R = ereal R' by (cases R) simp_all
  define r where
    r = (if conv_radius f =  $\infty$  then R' + 1 else (R' + real_of_ereal (conv_radius f)) / 2)
  from R conv_radius_nonneg[of f] have r > R  $\wedge$  r < conv_radius f unfolding

```

```

r_def
  by (cases conv_radius f) (auto simp: r_def field_simps R')
  with assms(1) assms(2)[of r] R'
  obtain z where norm z < conv_radius f  $\neg$  summable ( $\lambda n. \text{norm } (f n * z^n)$ )
by auto
  with abs_summable_in_conv_radius[of z f] show ?thesis by auto
qed simp_all

```

```

lemma conv_radius_leI_ex':
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes R  $\geq$  0
  assumes  $\bigwedge r. 0 < r \implies \text{ereal } r > R \implies \neg \text{summable } (\lambda n. f n * \text{of\_real } r^n)$ 
  shows conv_radius f  $\leq$  R
proof (rule conv_radius_leI_ex)
  fix r assume 0 < r  $\text{ereal } r > R$ 
  with assms(2)[of r] show  $\exists z. \text{norm } z = r \wedge \neg \text{summable } (\lambda n. \text{norm } (f n * z^n))$ 
  by (intro exI[of _ of_real r :: 'a]) (auto dest: summable_norm_cancel)
qed fact+

```

```

lemma conv_radius_eqI:
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes R  $\geq$  0
  assumes  $\bigwedge r. 0 < r \implies \text{ereal } r < R \implies \exists z. \text{norm } z = r \wedge \text{summable } (\lambda n. f n * z^n)$ 
  assumes  $\bigwedge r. 0 < r \implies \text{ereal } r > R \implies \exists z. \text{norm } z = r \wedge \neg \text{summable } (\lambda n. \text{norm } (f n * z^n))$ 
  shows conv_radius f = R
  by (intro antisym conv_radius_geI_ex conv_radius_leI_ex assms)

```

```

lemma conv_radius_eqI':
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes R  $\geq$  0
  assumes  $\bigwedge r. 0 < r \implies \text{ereal } r < R \implies \text{summable } (\lambda n. f n * (\text{of\_real } r)^n)$ 
  assumes  $\bigwedge r. 0 < r \implies \text{ereal } r > R \implies \neg \text{summable } (\lambda n. \text{norm } (f n * (\text{of\_real } r)^n))$ 
  shows conv_radius f = R
proof (intro conv_radius_eqI[OF assms(1)])
  fix r assume 0 < r  $\text{ereal } r < R$  with assms(2)[OF this]
  show  $\exists z. \text{norm } z = r \wedge \text{summable } (\lambda n. f n * z^n)$  by force
next
  fix r assume 0 < r  $\text{ereal } r > R$  with assms(3)[OF this]
  show  $\exists z. \text{norm } z = r \wedge \neg \text{summable } (\lambda n. \text{norm } (f n * z^n))$  by force
qed

```

```

lemma conv_radius_zeroI:
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes  $\bigwedge z. z \neq 0 \implies \neg \text{summable } (\lambda n. f n * z^n)$ 
  shows conv_radius f = 0

```

```

proof (rule ccontr)
  assume conv_radius f  $\neq 0$ 
  with conv_radius_nonneg[of f] have pos: conv_radius f > 0 by simp
  define r where r = (if conv_radius f =  $\infty$  then 1 else real_of_ereal (conv_radius f) / 2)
  from pos have r: ereal r > 0  $\wedge$  ereal r < conv_radius f
    by (cases conv_radius f) (simp_all add: r_def)
  hence summable ( $\lambda n. f\ n * \text{of\_real } r^n$ ) by (intro summable_in_conv_radius)
  simp
  moreover from r and assms[of of_real r] have  $\neg$ summable ( $\lambda n. f\ n * \text{of\_real } r^n$ ) by simp
  ultimately show False by contradiction
qed

```

```

lemma conv_radius_inftyI':
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes  $\bigwedge r. r > c \implies \exists z. \text{norm } z = r \wedge \text{summable } (\lambda n. f\ n * z^n)$ 
  shows conv_radius f =  $\infty$ 
proof -
  {
    fix r :: real
    have max_r (c + 1) > c by (auto simp: max_def)
    from assms[OF this] obtain z where norm z = max_r (c + 1) summable ( $\lambda n. f\ n * z^n$ ) by blast
    from conv_radius_geI[OF this(2)] this(1) have conv_radius f  $\geq$  r by simp
  }
  from this[of real_of_ereal (conv_radius f + 1)] show conv_radius f =  $\infty$ 
  by (cases conv_radius f) simp_all
qed

```

```

lemma conv_radius_inftyI:
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes  $\bigwedge r. \exists z. \text{norm } z = r \wedge \text{summable } (\lambda n. f\ n * z^n)$ 
  shows conv_radius f =  $\infty$ 
  using assms by (rule conv_radius_inftyI')

```

```

lemma conv_radius_inftyI'':
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes  $\bigwedge z. \text{summable } (\lambda n. f\ n * z^n)$ 
  shows conv_radius f =  $\infty$ 
proof (rule conv_radius_inftyI')
  fix r :: real assume r > 0
  with assms show  $\exists z. \text{norm } z = r \wedge \text{summable } (\lambda n. f\ n * z^n)$ 
  by (intro exI[of _ of_real r]) simp
qed

```

```

lemma conv_radius_conv_Sup:
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  shows conv_radius f = Sup {r.  $\forall z. \text{ereal } (\text{norm } z) < r \implies \text{summable } (\lambda n. f\ n$ 

```

```

* z ^ n)}
proof (rule Sup_eqI [symmetric], goal_cases)
  case (1 r)
  thus ?case
    by (intro conv_radius_geI_ex') auto
next
  case (2 r)
  from 2[of 0] have r: r ≥ 0 by auto
  show ?case
    proof (intro conv_radius_leI_ex' r)
      fix R assume R: R > 0 ereal R > r
      with r obtain r' where [simp]: r = ereal r' by (cases r) auto
      show ¬summable (λn. f n * of_real R ^ n)
        proof
          assume *: summable (λn. f n * of_real R ^ n)
          define R' where R' = (R + r') / 2
          from R have R': R' > r' R' < R by (simp_all add: R'_def)
          hence ∀ z. norm z < R' ⟶ summable (λn. f n * z ^ n)
            using powser_inside[OF *] by auto
          from 2[of R'] and this have R' ≤ r' by auto
          with ⟨R' > r'⟩ show False by simp
        qed
      qed
    qed

lemma conv_radius_shift:
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  shows conv_radius (λn. f (n + m)) = conv_radius f
  unfolding conv_radius_conv_Sup summable_powser_ignore_initial_segment ..

lemma conv_radius_norm [simp]: conv_radius (λx. norm (f x)) = conv_radius f
  by (simp add: conv_radius_def)

lemma conv_radius_ratio_limit_ereal:
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  assumes nz: eventually (λn. f n ≠ 0) sequentially
  assumes lim: (λn. ereal (norm (f n) / norm (f (Suc n)))) ⟶ c
  shows conv_radius f = c
proof (rule conv_radius_eqI')
  show c ≥ 0 by (intro Lim_bounded2[OF lim]) simp_all
next
  fix r assume r: 0 < r ereal r < c
  let ?l = liminf (λn. ereal (norm (f n * of_real r ^ n) / norm (f (Suc n) * of_real
    r ^ Suc n)))
  have ?l = liminf (λn. ereal (norm (f n) / (norm (f (Suc n)))) * ereal (inverse
    r))
    using r by (simp add: norm_mult norm_power field_split_simps)
  also from r have ... = liminf (λn. ereal (norm (f n) / (norm (f (Suc n))))) *
    ereal (inverse r)

```

```

    by (intro Liminf_ereal_mult_right) simp_all
  also have liminf ( $\lambda n. \text{ereal} (\text{norm} (f\ n) / (\text{norm} (f\ (\text{Suc}\ n)))) = c$ 
    by (intro lim_imp_Liminf lim) simp
  finally have l:  $?l = c * \text{ereal} (\text{inverse}\ r)$  by simp
  from r have l':  $c * \text{ereal} (\text{inverse}\ r) > 1$  by (cases c) (simp_all add: field_simps)
  show summable ( $\lambda n. f\ n * \text{of\_real}\ r^{\wedge} n$ )
    by (rule summable_norm_cancel, rule ratio_test_convergence)
      (insert r nz l l', auto elim!: eventually_mono)
next
  fix r assume r:  $0 < r \text{ereal}\ r > c$ 
  let ?l = limsup ( $\lambda n. \text{ereal} (\text{norm} (f\ n * \text{of\_real}\ r^{\wedge} n) / \text{norm} (f\ (\text{Suc}\ n) * \text{of\_real}\ r^{\wedge} \text{Suc}\ n)))$ 
  have ?l = limsup ( $\lambda n. \text{ereal} (\text{norm} (f\ n) / (\text{norm} (f\ (\text{Suc}\ n)))) * \text{ereal} (\text{inverse}\ r)$ )
    using r by (simp add: norm_mult norm_power field_split_simps)
  also from r have ... = limsup ( $\lambda n. \text{ereal} (\text{norm} (f\ n) / (\text{norm} (f\ (\text{Suc}\ n)))) * \text{ereal} (\text{inverse}\ r)$ )
    by (intro Limsup_ereal_mult_right) simp_all
  also have limsup ( $\lambda n. \text{ereal} (\text{norm} (f\ n) / (\text{norm} (f\ (\text{Suc}\ n)))) = c$ 
    by (intro lim_imp_Limsup lim) simp
  finally have l:  $?l = c * \text{ereal} (\text{inverse}\ r)$  by simp
  from r have l':  $c * \text{ereal} (\text{inverse}\ r) < 1$  by (cases c) (simp_all add: field_simps)
  show  $\neg \text{summable} (\lambda n. \text{norm} (f\ n * \text{of\_real}\ r^{\wedge} n))$ 
    by (rule ratio_test_divergence) (insert r nz l l', auto elim!: eventually_mono)
qed

```

```

lemma conv_radius_ratio_limit_ereal_nonzero:
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes nz:  $c \neq 0$ 
  assumes lim: ( $\lambda n. \text{ereal} (\text{norm} (f\ n) / \text{norm} (f\ (\text{Suc}\ n)))) \longrightarrow c$ 
  shows conv_radius f = c
proof (rule conv_radius_ratio_limit_ereal[OF _ lim], rule ccontr)
  assume  $\neg \text{eventually} (\lambda n. f\ n \neq 0)$  sequentially
  hence frequently ( $\lambda n. f\ n = 0$ ) sequentially by (simp add: frequently_def)
  hence frequently ( $\lambda n. \text{ereal} (\text{norm} (f\ n) / \text{norm} (f\ (\text{Suc}\ n))) = 0$ ) sequentially
    by (force elim!: frequently_elim1)
  hence  $c = 0$  by (intro limit_frequently_eq[OF _ _ lim]) auto
  with nz show False by contradiction
qed

```

```

lemma conv_radius_ratio_limit:
  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes c' =  $\text{ereal}\ c$ 
  assumes nz: eventually ( $\lambda n. f\ n \neq 0$ ) sequentially
  assumes lim: ( $\lambda n. \text{norm} (f\ n) / \text{norm} (f\ (\text{Suc}\ n))) \longrightarrow c$ 
  shows conv_radius f =  $c'$ 
  using assms by (intro conv_radius_ratio_limit_ereal) simp_all

```

```

lemma conv_radius_ratio_limit_nonzero:

```

```

fixes f :: nat => 'a :: {banach, real_normed_div_algebra}
assumes c' = ereal c
assumes nz: c ≠ 0
assumes lim: (λn. norm (f n) / norm (f (Suc n))) ⟶ c
shows conv_radius f = c'
using assms by (intro conv_radius_ratio_limit_ereal_nonzero) simp_all

```

```

lemma conv_radius_cmult_left:
  assumes c ≠ (0 :: 'a :: {banach, real_normed_div_algebra})
  shows conv_radius (λn. c * f n) = conv_radius f
proof -
  have conv_radius (λn. c * f n) =
    inverse (limsup (λn. ereal (root n (norm (c * f n)))))
  unfolding conv_radius_def ..
  also have (λn. ereal (root n (norm (c * f n)))) =
    (λn. ereal (root n (norm c)) * ereal (root n (norm (f n))))
  by (rule ext) (auto simp: norm_mult real_root_mult)
  also have limsup ... = ereal 1 * limsup (λn. ereal (root n (norm (f n))))
  using assms by (intro ereal_limsup_lim_mult tendsto_ereal LIMSEQ_root_const)
  auto
  also have inverse ... = conv_radius f by (simp add: conv_radius_def)
  finally show ?thesis .
qed

```

```

lemma conv_radius_cmult_right:
  assumes c ≠ (0 :: 'a :: {banach, real_normed_div_algebra})
  shows conv_radius (λn. f n * c) = conv_radius f
proof -
  have conv_radius (λn. f n * c) = conv_radius (λn. c * f n)
  by (simp add: conv_radius_def norm_mult mult.commute)
  with conv_radius_cmult_left[OF assms, of f] show ?thesis by simp
qed

```

```

lemma conv_radius_mult_power:
  assumes c ≠ (0 :: 'a :: {real_normed_div_algebra, banach})
  shows conv_radius (λn. c ^ n * f n) = conv_radius f / ereal (norm c)
proof -
  have limsup (λn. ereal (root n (norm (c ^ n * f n)))) =
    limsup (λn. ereal (norm c) * ereal (root n (norm (f n))))
  by (intro Limsup_eq eventually_mono [OF eventually_gt_at_top[of 0::nat]])
    (auto simp: norm_mult norm_power real_root_mult real_root_power)
  also have ... = ereal (norm c) * limsup (λn. ereal (root n (norm (f n))))
  using assms by (subst Limsup_ereal_mult_left[symmetric]) simp_all
  finally have A: limsup (λn. ereal (root n (norm (c ^ n * f n)))) =
    ereal (norm c) * limsup (λn. ereal (root n (norm (f n)))) .
  show ?thesis using assms
  apply (cases limsup (λn. ereal (root n (norm (f n)))) = 0)
  apply (simp add: A conv_radius_def)
  apply (unfold conv_radius_def A divide_ereal_def, simp add: mult.commute)

```

```

ereal inverse_mult)
  done
qed

```

```

lemma conv_radius_mult_power_right:
  assumes  $c \neq (0 :: 'a :: \{\text{real\_normed\_div\_algebra}, \text{banach}\})$ 
  shows  $\text{conv\_radius } (\lambda n. f\ n * c^{\wedge} n) = \text{conv\_radius } f / \text{ereal } (\text{norm } c)$ 
  using conv_radius_mult_power[OF assms, of f]
  unfolding conv_radius_def by (simp add: mult.commute norm_mult)

```

```

lemma conv_radius_divide_power:
  assumes  $c \neq (0 :: 'a :: \{\text{real\_normed\_div\_algebra}, \text{banach}\})$ 
  shows  $\text{conv\_radius } (\lambda n. f\ n / c^{\wedge} n) = \text{conv\_radius } f * \text{ereal } (\text{norm } c)$ 
proof -
  from assms have inverse  $c \neq 0$  by simp
  from conv_radius_mult_power_right[OF this, of f] show ?thesis
  by (simp add: divide_inverse divide_ereal_def assms norm_inverse power_inverse)
qed

```

```

lemma conv_radius_add_ge:
  min (conv_radius f) (conv_radius g)  $\leq$ 
  conv_radius  $(\lambda x. f\ x + g\ x :: 'a :: \{\text{banach}, \text{real\_normed\_div\_algebra}\})$ 
  by (rule conv_radius_geI_ex')
  (auto simp: algebra_simps intro!: summable_add summable_in_conv_radius)

```

```

lemma conv_radius_mult_ge:
  fixes  $f\ g :: \text{nat} \Rightarrow ('a :: \{\text{banach}, \text{real\_normed\_div\_algebra}\})$ 
  shows  $\text{conv\_radius } (\lambda x. \sum i \leq x. f\ i * g\ (x - i)) \geq \min (\text{conv\_radius } f) (\text{conv\_radius } g)$ 
proof (rule conv_radius_geI_ex')
  fix  $r$  assume  $r: r > 0 \text{ereal } r < \min (\text{conv\_radius } f) (\text{conv\_radius } g)$ 
  from  $r$  have summable  $(\lambda n. (\sum i \leq n. (f\ i * \text{of\_real } r^{\wedge} i) * (g\ (n - i) * \text{of\_real } r^{\wedge} (n - i))))$ 
  by (intro summable_Cauchy_product abs_summable_in_conv_radius) simp_all
  thus summable  $(\lambda n. (\sum i \leq n. f\ i * g\ (n - i)) * \text{of\_real } r^{\wedge} n)$ 
  by (simp add: algebra_simps of_real_def power_add [symmetric] scaleR_sum_right)
qed

```

```

lemma le_conv_radius_iff:
  fixes  $a :: \text{nat} \Rightarrow 'a :: \{\text{real\_normed\_div\_algebra}, \text{banach}\}$ 
  shows  $r \leq \text{conv\_radius } a \iff (\forall x. \text{norm } (x - \xi) < r \longrightarrow \text{summable } (\lambda i. a\ i * (x - \xi)^{\wedge} i))$ 
  apply (intro iffI allI impI summable_in_conv_radius conv_radius_geI_ex)
  apply (meson less_ereal.simps(1) not_le order_trans)
  apply (rule_tac  $x = \text{of\_real } ra$  in exI, simp)
  apply (metis abs_of_nonneg add_diff_cancel_left' less_eq_real_def norm_of_real)
  done

```

end

4.7 Uniform Limit and Uniform Convergence

theory *Uniform_Limit*
imports *Connected_Summation_Tests*
begin

4.7.1 Definition

definition *uniformly_on* :: 'a set $\Rightarrow$ ('a $\Rightarrow$ 'b::metric_space) $\Rightarrow$ ('a $\Rightarrow$ 'b) filter
where *uniformly_on* S l = (INF e $\in$ {0 <.. ∞ }. principal {f. $\forall x \in S. \text{dist } (f \ x) \ (l \ x) < e$ })

abbreviation

uniform_limit S f l $\equiv$ filterlim f (*uniformly_on* S l)

definition *uniformly_convergent_on* **where**

uniformly_convergent_on X f $\longleftrightarrow$ ($\exists l. \text{uniform\_limit } X \ f \ l \text{ sequentially}$)

definition *uniformly_Cauchy_on* **where**

uniformly_Cauchy_on X f $\longleftrightarrow$ ($\forall e > 0. \exists M. \forall x \in X. \forall (m::nat) \geq M. \forall n \geq M. \text{dist } (f \ m \ x) \ (f \ n \ x) < e$)

proposition *uniform_limit_iff*:

uniform_limit S f l F $\longleftrightarrow$ ($\forall e > 0. \forall_F n \text{ in } F. \forall x \in S. \text{dist } (f \ n \ x) \ (l \ x) < e$)

unfolding filterlim_iff *uniformly_on_def*

by (subst eventually_INF_base)

(fastforce

simp: eventually_principal *uniformly_on_def*

intro: bexI[**where** x=min a b **for** a b]

elim: eventually_mono)+

lemma *uniform_limitD*:

uniform_limit S f l F $\Longrightarrow e > 0 \Longrightarrow \forall_F n \text{ in } F. \forall x \in S. \text{dist } (f \ n \ x) \ (l \ x) < e$

by (simp add: *uniform_limit_iff*)

lemma *uniform_limitI*:

($\bigwedge e. e > 0 \Longrightarrow \forall_F n \text{ in } F. \forall x \in S. \text{dist } (f \ n \ x) \ (l \ x) < e$) $\Longrightarrow \text{uniform\_limit } S \ f \ l \ F$

by (simp add: *uniform_limit_iff*)

lemma *uniform_limit_sequentially_iff*:

uniform_limit S f l sequentially $\longleftrightarrow$ ($\forall e > 0. \exists N. \forall n \geq N. \forall x \in S. \text{dist } (f \ n \ x) \ (l \ x) < e$)

unfolding *uniform_limit_iff* eventually_sequentially ..

lemma *uniform_limit_at_iff*:

uniform_limit S f l (at x) $\longleftrightarrow$

$(\forall e > 0. \exists d > 0. \forall z. 0 < \text{dist } z \ x \wedge \text{dist } z \ x < d \longrightarrow (\forall x \in S. \text{dist } (f \ z \ x) \ (l \ x) < e))$

unfolding *uniform_limit_iff eventually_at by simp*

lemma *uniform_limit_at_le_iff*:

uniform_limit S f l (at x) $\longleftrightarrow$

$(\forall e > 0. \exists d > 0. \forall z. 0 < \text{dist } z \ x \wedge \text{dist } z \ x < d \longrightarrow (\forall x \in S. \text{dist } (f \ z \ x) \ (l \ x) \leq e))$

unfolding *uniform_limit_iff eventually_at*

by (*fastforce dest: spec[where x = e / 2 for e]*)

lemma *metric_uniform_limit_imp_uniform_limit*:

assumes *f: uniform_limit S f a F*

assumes *le: eventually $(\lambda x. \forall y \in S. \text{dist } (g \ x \ y) \ (b \ y) \leq \text{dist } (f \ x \ y) \ (a \ y))$ F*

shows *uniform_limit S g b F*

proof (*rule uniform_limitI*)

fix *e :: real* **assume** $0 < e$

from *uniform_limitD[OF f this] le*

show $\forall_F x \text{ in } F. \forall y \in S. \text{dist } (g \ x \ y) \ (b \ y) < e$

by *eventually_elim force*

qed

4.7.2 Exchange limits

proposition *swap_uniform_limit*:

assumes *f: $\forall_F n \text{ in } F. (f \ n \longrightarrow g \ n) \text{ (at } x \text{ within } S)$*

assumes *g: $(g \longrightarrow l) \ F$*

assumes *uc: uniform_limit S f h F*

assumes $\neg \text{trivial\_limit } F$

shows $(h \longrightarrow l) \text{ (at } x \text{ within } S)$

proof (*rule tendstoI*)

fix *e :: real*

define *e'* **where** $e' = e/3$

assume $0 < e$

then have $0 < e'$ **by** (*simp add: e'_def*)

from *uniform_limitD[OF uc $\langle 0 < e' \rangle$]*

have $\forall_F n \text{ in } F. \forall x \in S. \text{dist } (h \ x) \ (f \ n \ x) < e'$

by (*simp add: dist_commute*)

moreover

from *f*

have $\forall_F n \text{ in } F. \forall_F x \text{ in at } x \text{ within } S. \text{dist } (g \ n) \ (f \ n \ x) < e'$

by *eventually_elim (auto dest!: tendstoD[OF _ $\langle 0 < e' \rangle$] simp: dist_commute)*

moreover

from *tendstoD[OF g $\langle 0 < e' \rangle$]* **have** $\forall_F x \text{ in } F. \text{dist } l \ (g \ x) < e'$

by (*simp add: dist_commute*)

ultimately

have $\forall_F \_ \text{ in } F. \forall_F x \text{ in at } x \text{ within } S. \text{dist } (h \ x) \ l < e$

proof *eventually_elim*

case (*elim n*)

```

note fh = elim(1)
note gl = elim(3)
have  $\forall_F x$  in at x within S.  $x \in S$ 
  by (auto simp: eventually_at_filter)
with elim(2)
show ?case
proof eventually_elim
  case (elim x)
  from fh[rule_format, OF  $\langle x \in S \rangle$ ] elim(1)
  have dist (h x) (g n) <  $e' + e'$ 
    by (rule dist_triangle_lt[OF add_strict_mono])
  from dist_triangle_lt[OF add_strict_mono, OF this gl]
  show ?case by (simp add: e'_def)
qed
qed
thus  $\forall_F x$  in at x within S. dist (h x) l < e
  using eventually_happens by (metis  $\langle \neg \text{trivial\_limit } F \rangle$ )
qed

```

4.7.3 Uniform limit theorem

```

lemma tendsto_uniform_limitI:
  assumes uniform_limit S f l F
  assumes  $x \in S$ 
  shows  $((\lambda y. f y x) \longrightarrow l x) F$ 
  using assms
  by (auto intro!: tendstoI simp: eventually_mono dest!: uniform_limitD)

theorem uniform_limit_theorem:
  assumes c:  $\forall_F n$  in F. continuous_on A (f n)
  assumes ul: uniform_limit A f l F
  assumes  $\neg \text{trivial\_limit } F$ 
  shows continuous_on A l
  unfolding continuous_on_def
proof safe
  fix x assume  $x \in A$ 
  then have  $\forall_F n$  in F.  $(f n \longrightarrow f n x) \text{ (at } x \text{ within } A) \text{ } ((\lambda n. f n x) \longrightarrow l x) F$ 
    using c ul
    by (auto simp: continuous_on_def eventually_mono tendsto_uniform_limitI)
  then show  $(l \longrightarrow l x) \text{ (at } x \text{ within } A)$ 
    by (rule swap_uniform_limit) fact+
qed

lemma uniformly_Cauchy_onI:
  assumes  $\bigwedge e. e > 0 \implies \exists M. \forall x \in X. \forall m \geq M. \forall n \geq M. \text{dist } (f m x) (f n x) < e$ 
  shows uniformly_Cauchy_on X f
  using assms unfolding uniformly_Cauchy_on_def by blast

lemma uniformly_Cauchy_onI':

```

```

  assumes  $\bigwedge e. e > 0 \implies \exists M. \forall x \in X. \forall m \geq M. \forall n > m. \text{dist } (f \ m \ x) \ (f \ n \ x) < e$ 
  shows uniformly_Cauchy_on X f
proof (rule uniformly_Cauchy_onI)
  fix e :: real assume e:  $e > 0$ 
  from assms[OF this] obtain M
  where M:  $\bigwedge x \ m \ n. x \in X \implies m \geq M \implies n > m \implies \text{dist } (f \ m \ x) \ (f \ n \ x) < e$ 
  by fast
  {
    fix x m n assume x:  $x \in X$  and m:  $m \geq M$  and n:  $n \geq M$ 
    with M[OF this(1,2), of n] M[OF this(1,3), of m] e have  $\text{dist } (f \ m \ x) \ (f \ n \ x) < e$ 
  }
  by (cases m n rule: linorder_cases) (simp_all add: dist_commute)
  thus  $\exists M. \forall x \in X. \forall m \geq M. \forall n \geq M. \text{dist } (f \ m \ x) \ (f \ n \ x) < e$  by fast
qed

```

```

lemma uniformly_Cauchy_imp_Cauchy:
  uniformly_Cauchy_on X f  $\implies x \in X \implies \text{Cauchy } (\lambda n. f \ n \ x)$ 
  unfolding Cauchy_def uniformly_Cauchy_on_def by fast

```

```

lemma uniform_limit_cong:
  fixes f g :: 'a  $\Rightarrow$  'b  $\Rightarrow$  ('c :: metric_space) and h i :: 'b  $\Rightarrow$  'c
  assumes eventually  $(\lambda y. \forall x \in X. f \ y \ x = g \ y \ x) \ F$ 
  assumes  $\bigwedge x. x \in X \implies h \ x = i \ x$ 
  shows uniform_limit X f h F  $\longleftrightarrow$  uniform_limit X g i F
proof -
  {
    fix f g :: 'a  $\Rightarrow$  'b  $\Rightarrow$  'c and h i :: 'b  $\Rightarrow$  'c
    assume C: uniform_limit X f h F and A: eventually  $(\lambda y. \forall x \in X. f \ y \ x = g \ y \ x) \ F$ 
    and B:  $\bigwedge x. x \in X \implies h \ x = i \ x$ 
    {
      fix e :: real assume e > 0
      with C have eventually  $(\lambda y. \forall x \in X. \text{dist } (f \ y \ x) \ (h \ x) < e) \ F$ 
      unfolding uniform_limit_iff by blast
      with A have eventually  $(\lambda y. \forall x \in X. \text{dist } (g \ y \ x) \ (i \ x) < e) \ F$ 
      by eventually_elim (insert B, simp_all)
    }
    hence uniform_limit X g i F unfolding uniform_limit_iff by blast
  }
  note A = this
  show ?thesis by (rule iffI) (erule A; insert assms; simp add: eq_commute)+
qed

```

```

lemma uniform_limit_cong':
  fixes f g :: 'a  $\Rightarrow$  'b  $\Rightarrow$  ('c :: metric_space) and h i :: 'b  $\Rightarrow$  'c
  assumes  $\bigwedge y \ x. x \in X \implies f \ y \ x = g \ y \ x$ 
  assumes  $\bigwedge x. x \in X \implies h \ x = i \ x$ 
  shows uniform_limit X f h F  $\longleftrightarrow$  uniform_limit X g i F
  using assms by (intro uniform_limit_cong always_eventually) blast+

```

```

lemma uniformly_convergent_cong:
  assumes eventually ( $\lambda x. \forall y \in A. f\ x\ y = g\ x\ y$ ) sequentially  $A = B$ 
  shows uniformly_convergent_on  $A\ f \longleftrightarrow$  uniformly_convergent_on  $B\ g$ 
  unfolding uniformly_convergent_on_def assms(2) [symmetric]
  by (intro iff_exI uniform_limit_cong eventually_mono [OF assms(1)]) auto

```

```

lemma uniformly_convergent_uniform_limit_iff:
  uniformly_convergent_on  $X\ f \longleftrightarrow$  uniform_limit  $X\ f\ (\lambda x. \lim (\lambda n. f\ n\ x))$ 
sequentially
proof
  assume uniformly_convergent_on  $X\ f$ 
  then obtain  $l$  where  $l$ : uniform_limit  $X\ f\ l$  sequentially
  unfolding uniformly_convergent_on_def by blast
  from  $l$  have uniform_limit  $X\ f\ (\lambda x. \lim (\lambda n. f\ n\ x))$  sequentially  $\longleftrightarrow$ 
    uniform_limit  $X\ f\ l$  sequentially
  by (intro uniform_limit_cong' limI tendsto_uniform_limitI[of f X l]) simp_all
  also note  $l$ 
  finally show uniform_limit  $X\ f\ (\lambda x. \lim (\lambda n. f\ n\ x))$  sequentially .
qed (auto simp: uniformly_convergent_on_def)

```

```

lemma uniformly_convergentI: uniform_limit  $X\ f\ l$  sequentially  $\implies$  uniformly_convergent_on
 $X\ f$ 
  unfolding uniformly_convergent_on_def by blast

```

```

lemma uniformly_convergent_on_empty [iff]: uniformly_convergent_on  $\{\}$   $f$ 
  by (simp add: uniformly_convergent_on_def uniform_limit_sequentially_iff)

```

```

lemma uniformly_convergent_on_const [simp,intro]:
  uniformly_convergent_on  $A\ (\lambda \_. c)$ 
  by (auto simp: uniformly_convergent_on_def uniform_limit_iff intro!: exI[of _ c])

```

Cauchy-type criteria for uniform convergence.

```

lemma Cauchy_uniformly_convergent:
  fixes  $f :: \text{nat} \Rightarrow 'a \Rightarrow 'b :: \text{complete\_space}$ 
  assumes uniformly_Cauchy_on  $X\ f$ 
  shows uniformly_convergent_on  $X\ f$ 
unfolding uniformly_convergent_uniform_limit_iff uniform_limit_iff
proof safe
  let  $?f = \lambda x. \lim (\lambda n. f\ n\ x)$ 
  fix  $e :: \text{real}$  assume  $e: e > 0$ 
  hence  $e/2 > 0$  by simp
  with assms obtain  $N$  where  $N: \bigwedge x\ m\ n. x \in X \implies m \geq N \implies n \geq N \implies$ 
 $\text{dist } (f\ m\ x) (f\ n\ x) < e/2$ 
  unfolding uniformly_Cauchy_on_def by fast
  show eventually  $(\lambda n. \forall x \in X. \text{dist } (f\ n\ x) (?f\ x) < e)$  sequentially
  using eventually_ge_at_top[of  $N$ ]
  proof eventually_elim

```

```

fix n assume n: n ≥ N
show  $\forall x \in X. \text{dist } (f \ n \ x) \ (\ ?f \ x) < e$ 
proof
  fix x assume x: x ∈ X
  with assms have  $(\lambda n. f \ n \ x) \longrightarrow \ ?f \ x$ 
    by (auto dest!: Cauchy_convergent_uniformly_Cauchy_imp_Cauchy_simp:
convergent_LIMSEQ_iff)
  with  $\langle e/2 > 0 \rangle$  have eventually  $(\lambda m. m \geq N \wedge \text{dist } (f \ m \ x) \ (\ ?f \ x) < e/2)$ 
sequentially
    by (intro tendstoD eventually_conj eventually_ge_at_top)
  then obtain m where m: m ≥ N  $\text{dist } (f \ m \ x) \ (\ ?f \ x) < e/2$ 
    unfolding eventually_at_top_linorder by blast
  have  $\text{dist } (f \ n \ x) \ (\ ?f \ x) \leq \text{dist } (f \ n \ x) \ (f \ m \ x) + \text{dist } (f \ m \ x) \ (\ ?f \ x)$ 
    by (rule dist_triangle)
  also from x n have ... < e/2 + e/2 by (intro add_strict_mono N m)
  finally show  $\text{dist } (f \ n \ x) \ (\ ?f \ x) < e$  by simp
qed
qed
qed

lemma uniformly_convergent_Cauchy:
  assumes uniformly_convergent_on X f
  shows uniformly_Cauchy_on X f
proof (rule uniformly_Cauchy_onI)
  fix e::real assume e > 0
  then have 0 < e / 2 by simp
  with assms[unfolded uniformly_convergent_on_def uniform_limit_sequentially_iff]
  obtain l N where l: x ∈ X  $\implies n \geq N \implies \text{dist } (f \ n \ x) \ (l \ x) < e / 2$  for n x
    by metis
  from l l have x ∈ X  $\implies n \geq N \implies m \geq N \implies \text{dist } (f \ n \ x) \ (f \ m \ x) < e$  for n
m x
    by (rule dist_triangle_half_l)
  then show  $\exists M. \forall x \in X. \forall m \geq M. \forall n \geq M. \text{dist } (f \ m \ x) \ (f \ n \ x) < e$  by blast
qed

lemma uniformly_convergent_eq_Cauchy:
  uniformly_convergent_on X f = uniformly_Cauchy_on X f for f::nat  $\Rightarrow$  'b  $\Rightarrow$ 
'a::complete_space
  using Cauchy_uniformly_convergent_uniformly_convergent_Cauchy by blast

lemma uniformly_convergent_eq_cauchy:
  fixes s::nat  $\Rightarrow$  'b  $\Rightarrow$  'a::complete_space
  shows
     $(\exists l. \forall e > 0. \exists N. \forall n \ x. N \leq n \wedge P \ x \longrightarrow \text{dist}(s \ n \ x)(l \ x) < e) \longleftrightarrow$ 
     $(\forall e > 0. \exists N. \forall m \ n \ x. N \leq m \wedge N \leq n \wedge P \ x \longrightarrow \text{dist } (s \ m \ x) \ (s \ n \ x) < e)$ 
proof -
  have *:  $(\forall n \geq N. \forall x. Q \ x \longrightarrow R \ n \ x) \longleftrightarrow (\forall n \ x. N \leq n \wedge Q \ x \longrightarrow R \ n \ x)$ 
     $(\forall x. Q \ x \longrightarrow (\forall m \geq N. \forall n \geq N. S \ n \ m \ x)) \longleftrightarrow (\forall m \ n \ x. N \leq m \wedge N \leq n \wedge$ 
     $Q \ x \longrightarrow S \ n \ m \ x)$ 

```

```

    for  $N::nat$  and  $Q::'b \Rightarrow bool$  and  $R\ S$ 
    by blast+
  show ?thesis
    using uniformly_convergent_eq_Cauchy[of Collect  $P\ s$ ]
    unfolding uniformly_convergent_on_def uniformly_Cauchy_on_def uniform_limit_sequentially_iff
    by (simp add: *)
qed

```

```

lemma uniformly_cauchy_imp_uniformly_convergent:
  fixes  $s :: nat \Rightarrow 'a \Rightarrow 'b::complete\_space$ 
  assumes  $\forall e>0. \exists N. \forall m\ (n::nat)\ x. N \leq m \wedge N \leq n \wedge P\ x \longrightarrow dist(s\ m\ x)(s\ n\ x) < e$ 
  and  $\forall x. P\ x \longrightarrow (\forall e>0. \exists N. \forall n. N \leq n \longrightarrow dist(s\ n\ x)(l\ x) < e)$ 
  shows  $\forall e>0. \exists N. \forall n\ x. N \leq n \wedge P\ x \longrightarrow dist(s\ n\ x)(l\ x) < e$ 
proof -
  obtain  $l'$  where  $l:\forall e>0. \exists N. \forall n\ x. N \leq n \wedge P\ x \longrightarrow dist(s\ n\ x)(l'\ x) < e$ 
    using assms(1) unfolding uniformly_convergent_eq_cauchy[symmetric] by
  auto
  moreover
  {
    fix  $x$ 
    assume  $P\ x$ 
    then have  $l\ x = l'\ x$ 
      using tendsto_unique[OF trivial_limit_sequentially, of  $\lambda n. s\ n\ x\ l\ x\ l'\ x$ ]
      using  $l$  and assms(2) unfolding lim_sequentially by blast
  }
  ultimately show ?thesis by auto
qed

```

TODO: remove explicit formulations $(\exists l. \forall e>0. \exists N. \forall n\ x. N \leq n \wedge ?P\ x \longrightarrow dist(s\ n\ x)(l\ x) < e) = (\forall e>0. \exists N. \forall m\ n\ x. N \leq m \wedge N \leq n \wedge ?P\ x \longrightarrow dist(s\ m\ x)(s\ n\ x) < e)$

$\llbracket \forall e>0. \exists N. \forall m\ n\ x. N \leq m \wedge N \leq n \wedge ?P\ x \longrightarrow dist(s\ m\ x)(s\ n\ x) < e; \forall x. ?P\ x \longrightarrow (\forall e>0. \exists N. \forall n \geq N. dist(s\ n\ x)(l\ x) < e) \rrbracket \implies \forall e>0. \exists N. \forall n\ x. N \leq n \wedge ?P\ x \longrightarrow dist(s\ n\ x)(l\ x) < e$!

```

lemma uniformly_convergent_imp_convergent:
  uniformly_convergent_on  $X\ f \implies x \in X \implies convergent\ (\lambda n. f\ n\ x)$ 
  unfolding uniformly_convergent_on_def convergent_def
  by (auto dest: tendsto_uniform_limitI)

```

4.7.4 Weierstrass M-Test

```

proposition Weierstrass_m_test_ev:
  fixes  $f :: \_ \Rightarrow \_ \Rightarrow \_ :: banach$ 
  assumes eventually  $(\lambda n. \forall x \in A. norm(f\ n\ x) \leq M\ n)$  sequentially
  assumes summable  $M$ 
  shows uniform_limit  $A\ (\lambda n\ x. \sum_{i<n}. f\ i\ x)$   $(\lambda x. \text{suminf } (\lambda i. f\ i\ x))$  sequentially
proof (rule uniform_limitI)
  fix  $e :: real$ 

```

```

assume 0 < e
from suminf_exist_split[OF ‹0 < e› ‹summable M›]
have  $\forall_F k$  in sequentially. norm  $(\sum i. M (i + k)) < e$ 
  by (auto simp: eventually_sequentially)
with eventually_all_ge_at_top[OF assms(1)]
  show  $\forall_F n$  in sequentially.  $\forall x \in A. \text{dist } (\sum_{i < n} f i x) (\sum i. f i x) < e$ 
proof eventually_elim
  case (elim k)
  show ?case
  proof safe
    fix x assume x ∈ A
    have  $\exists N. \forall n \geq N. \text{norm } (f n x) \leq M n$ 
      using assms(1) ‹x ∈ A› by (force simp: eventually_at_top_linorder)
    hence summable_norm_f: summable  $(\lambda n. \text{norm } (f n x))$ 
      by (rule summable_norm_comparison_test[OF ‹summable M›])
    have summable_f: summable  $(\lambda n. f n x)$ 
      using summable_norm_cancel[OF summable_norm_f] .
    have summable_norm_f_plus_k: summable  $(\lambda i. \text{norm } (f (i + k) x))$ 
      using summable_ignore_initial_segment[OF summable_norm_f]
      by auto
    have summable_M_plus_k: summable  $(\lambda i. M (i + k))$ 
      using summable_ignore_initial_segment[OF ‹summable M›]
      by auto

    have  $\text{dist } (\sum_{i < k} f i x) (\sum i. f i x) = \text{norm } ((\sum i. f i x) - (\sum_{i < k} f i x))$ 
      using dist_norm dist_commute by (subst dist_commute)
    also have ... = norm  $(\sum i. f (i + k) x)$ 
      using suminf_minus_initial_segment[OF summable_f, where k=k] by
simp
    also have ...  $\leq (\sum i. \text{norm } (f (i + k) x))$ 
      using summable_norm[OF summable_norm_f_plus_k] .
    also have ...  $\leq (\sum i. M (i + k))$ 
      by (rule suminf_le[OF summable_norm_f_plus_k summable_M_plus_k])
      (insert elim(1) ‹x ∈ A›, simp)
    finally show  $\text{dist } (\sum_{i < k} f i x) (\sum i. f i x) < e$ 
      using elim by auto
  qed
qed
qed

```

Alternative version, formulated as in HOL Light

corollary *series_comparison_uniform*:

```

fixes f ::  $\_ \Rightarrow \text{nat} \Rightarrow \_ :: \text{banach}$ 
assumes g: summable g and le:  $\bigwedge n x. N \leq n \wedge x \in A \implies \text{norm}(f x n) \leq g n$ 
shows  $\exists l. \forall e. 0 < e \longrightarrow (\exists N. \forall n x. N \leq n \wedge x \in A \longrightarrow \text{dist}(\text{sum } (f x) \{..<n\}) (l x) < e)$ 
proof -
  have 1:  $\forall_F n$  in sequentially.  $\forall x \in A. \text{norm } (f x n) \leq g n$ 
    using le eventually_sequentially by auto

```

```

show ?thesis
  apply (rule_tac x=( $\lambda x. \sum i. f x i$ ) in exI)
  apply (metis (no_types, lifting) eventually_sequentially uniform_limitD [OF
Weierstrass_m_test_ev [OF 1 g]])
  done
qed

```

```

corollary Weierstrass_m_test:
  fixes f ::  $\_ \Rightarrow \_ \Rightarrow \_ :: \text{banach}$ 
  assumes  $\bigwedge n x. x \in A \implies \text{norm } (f n x) \leq M n$ 
  assumes summable M
  shows uniform_limit A ( $\lambda n x. \sum i < n. f i x$ ) ( $\lambda x. \text{suminf } (\lambda i. f i x)$ ) sequentially
  using assms by (intro Weierstrass_m_test_ev always_eventually) auto

```

```

corollary Weierstrass_m_test'_ev:
  fixes f ::  $\_ \Rightarrow \_ \Rightarrow \_ :: \text{banach}$ 
  assumes eventually ( $\lambda n. \forall x \in A. \text{norm } (f n x) \leq M n$ ) sequentially summable M
  shows uniformly_convergent_on A ( $\lambda n x. \sum i < n. f i x$ )
  unfolding uniformly_convergent_on_def by (rule exI, rule Weierstrass_m_test_ev[OF
assms])

```

```

corollary Weierstrass_m_test':
  fixes f ::  $\_ \Rightarrow \_ \Rightarrow \_ :: \text{banach}$ 
  assumes  $\bigwedge n x. x \in A \implies \text{norm } (f n x) \leq M n$  summable M
  shows uniformly_convergent_on A ( $\lambda n x. \sum i < n. f i x$ )
  unfolding uniformly_convergent_on_def by (rule exI, rule Weierstrass_m_test[OF
assms])

```

```

lemma uniform_limit_eq_rhs: uniform_limit X f l F  $\implies l = m \implies$  uniform_limit
X f m F
  by simp

```

4.7.5 Structural introduction rules

```

named_theorems uniform_limit_intros introduction rules for uniform_limit
setup <
  Global_Theory.add_thms_dynamic (binding <uniform_limit_eq_intros>,
    fn context =>
      Named_Theorems.get (Context.proof_of context) named_theorems <uniform_limit_intros>
        |> map_filter (try (fn thm => @{thm uniform_limit_eq_rhs} OF [thm])))
  >

```

```

lemma (in bounded_linear) uniform_limit[uniform_limit_intros]:
  assumes uniform_limit X g l F
  shows uniform_limit X ( $\lambda a b. f (g a b)$ ) ( $\lambda a. f (l a)$ ) F
proof (rule uniform_limitI)
  fix e::real
  from pos_bounded obtain K
  where K:  $\bigwedge x y. \text{dist } (f x) (f y) \leq K * \text{dist } x y$  K > 0

```



```

  by (auto simp: ac_simps dist_norm diff[symmetric])
  assume  $0 < e$  with  $\langle K > 0 \rangle$  have  $e / K > 0$  by simp
  from uniform_limitD[OF assms this]
  show  $\forall_F n \text{ in } F. \forall x \in X. \text{dist } (f (g n x)) (f (l x)) < e$ 
    by eventually_elim (metis le_less_trans mult.commute pos_less_divide_eq K)
qed

```

```

lemma (in bounded_linear) uniformly_convergent_on:
  assumes uniformly_convergent_on A g
  shows uniformly_convergent_on A  $(\lambda x y. f (g x y))$ 
proof -
  from assms obtain l where uniform_limit A g l sequentially
  unfolding uniformly_convergent_on_def by blast
  hence uniform_limit A  $(\lambda x y. f (g x y))$   $(\lambda x. f (l x))$  sequentially
  by (rule uniform_limit)
  thus ?thesis unfolding uniformly_convergent_on_def by blast
qed

```

```

lemmas bounded_linear_uniform_limit_intros[uniform_limit_intros] =
  bounded_linear.uniform_limit[OF bounded_linear_Im]
  bounded_linear.uniform_limit[OF bounded_linear_Re]
  bounded_linear.uniform_limit[OF bounded_linear_cnj]
  bounded_linear.uniform_limit[OF bounded_linearfst]
  bounded_linear.uniform_limit[OF bounded_linear_snd]
  bounded_linear.uniform_limit[OF bounded_linear_zero]
  bounded_linear.uniform_limit[OF bounded_linear_of_real]
  bounded_linear.uniform_limit[OF bounded_linear_inner_left]
  bounded_linear.uniform_limit[OF bounded_linear_inner_right]
  bounded_linear.uniform_limit[OF bounded_linear_divide]
  bounded_linear.uniform_limit[OF bounded_linear_scaleR_right]
  bounded_linear.uniform_limit[OF bounded_linear_mult_left]
  bounded_linear.uniform_limit[OF bounded_linear_mult_right]
  bounded_linear.uniform_limit[OF bounded_linear_scaleR_left]

```

```

lemmas uniform_limit_uminus[uniform_limit_intros] =
  bounded_linear.uniform_limit[OF bounded_linear_minus[OF bounded_linear_ident]]

```

```

lemma uniform_limit_const[uniform_limit_intros]: uniform_limit S  $(\lambda x. c)$  c f
  by (auto intro!: uniform_limitI)

```

```

lemma uniform_limit_add[uniform_limit_intros]:
  fixes f g::' $a \Rightarrow 'b \Rightarrow 'c::\text{real\_normed\_vector}$ '
  assumes uniform_limit X f l F
  assumes uniform_limit X g m F
  shows uniform_limit X  $(\lambda a b. f a b + g a b)$   $(\lambda a. l a + m a)$  F
proof (rule uniform_limitI)
  fix e::real
  assume  $0 < e$ 

```

hence $0 < e / 2$ by *simp*
 from
 uniform_limitD [*OF* *assms*(1) *this*]
 uniform_limitD [*OF* *assms*(2) *this*]
 show $\forall_F n \text{ in } F. \forall x \in X. \text{dist } (f \ n \ x + g \ n \ x) \ (l \ x + m \ x) < e$
 by *eventually_elim* (*simp* *add*: *dist_triangle_add_half*)
 qed

lemma *uniform_limit_minus* [*uniform_limit_intros*]:
 fixes $f \ g :: 'a \Rightarrow 'b \Rightarrow 'c :: \text{real\_normed\_vector}$
 assumes *uniform_limit* $X \ f \ l \ F$
 assumes *uniform_limit* $X \ g \ m \ F$
 shows *uniform_limit* $X \ (\lambda a \ b. f \ a \ b - g \ a \ b) \ (\lambda a. l \ a - m \ a) \ F$
 unfolding *diff_conv_add_uminus*
 by (rule *uniform_limit_intros* *assms*) +

lemma *uniform_limit_norm* [*uniform_limit_intros*]:
 assumes *uniform_limit* $S \ g \ l \ f$
 shows *uniform_limit* $S \ (\lambda x \ y. \text{norm } (g \ x \ y)) \ (\lambda x. \text{norm } (l \ x)) \ f$
 using *assms*
 apply (rule *metric_uniform_limit_imp_uniform_limit*)
 apply (rule *eventuallyI*)
 by (metis *dist_norm norm_triangle_ineq3 real_norm_def*)

lemma (in *bounded_bilinear*) *bounded_uniform_limit* [*uniform_limit_intros*]:
 assumes *uniform_limit* $X \ f \ l \ F$
 assumes *uniform_limit* $X \ g \ m \ F$
 assumes *bounded* ($m \ ' X$)
 assumes *bounded* ($l \ ' X$)
 shows *uniform_limit* $X \ (\lambda a \ b. \text{prod } (f \ a \ b) \ (g \ a \ b)) \ (\lambda a. \text{prod } (l \ a) \ (m \ a)) \ F$
proof (rule *uniform_limitI*)
 fix $e :: \text{real}$
 from *pos_bounded* **obtain** K **where** K :
 $0 < K \wedge a \ b. \text{norm } (\text{prod } a \ b) \leq \text{norm } a * \text{norm } b * K$
 by *auto*
 hence $\text{sqrt } (K * 4) > 0$ by *simp*

from *assms* **obtain** $Km \ Kl$
where Km : $Km > 0 \wedge x. x \in X \implies \text{norm } (m \ x) \leq Km$
and Kl : $Kl > 0 \wedge x. x \in X \implies \text{norm } (l \ x) \leq Kl$
 by (*auto simp*: *bounded_pos*)
 hence $K * Km * 4 > 0 \ K * Kl * 4 > 0$
 using $\langle K > 0 \rangle$
 by *simp_all*
 assume $0 < e$

hence $\text{sqrt } e > 0$ by *simp*
from *uniform_limitD* [*OF* *assms*(1) *divide_pos_pos* [*OF* *this* $\langle \text{sqrt } (K * 4) > 0 \rangle$]]
 uniform_limitD [*OF* *assms*(2) *divide_pos_pos* [*OF* *this* $\langle \text{sqrt } (K * 4) > 0 \rangle$]]

```

uniform_limitD[OF assms(1) divide_pos_pos[OF ‹e > 0› ‹K * Km * 4 > 0›]]
uniform_limitD[OF assms(2) divide_pos_pos[OF ‹e > 0› ‹K * Kl * 4 > 0›]]
show  $\forall_F n \text{ in } F. \forall x \in X. \text{dist} (\text{prod} (f \ n \ x) (g \ n \ x)) (\text{prod} (l \ x) (m \ x)) < e$ 
proof eventually_elim
  case (elim n)
  show ?case
  proof safe
    fix x assume x  $\in$  X
    have  $\text{dist} (\text{prod} (f \ n \ x) (g \ n \ x)) (\text{prod} (l \ x) (m \ x)) \leq$ 
       $\text{norm} (\text{prod} (f \ n \ x - l \ x) (g \ n \ x - m \ x)) +$ 
       $\text{norm} (\text{prod} (f \ n \ x - l \ x) (m \ x)) +$ 
       $\text{norm} (\text{prod} (l \ x) (g \ n \ x - m \ x))$ 
    by (auto simp: dist_norm prod_diff_prod intro: order_trans norm_triangle_ineq
      add_mono)
    also note K(2)[of f n x - l x g n x - m x]
    also from elim(1)[THEN bspec, OF ‹_  $\in$  X›, unfolded dist_norm]
    have  $\text{norm} (f \ n \ x - l \ x) \leq \text{sqrt } e / \text{sqrt} (K * 4)$ 
    by simp
    also from elim(2)[THEN bspec, OF ‹_  $\in$  X›, unfolded dist_norm]
    have  $\text{norm} (g \ n \ x - m \ x) \leq \text{sqrt } e / \text{sqrt} (K * 4)$ 
    by simp
    also have  $\text{sqrt } e / \text{sqrt} (K * 4) * (\text{sqrt } e / \text{sqrt} (K * 4)) * K = e / 4$ 
    using ‹K > 0› ‹e > 0› by auto
    also note K(2)[of f n x - l x m x]
    also note K(2)[of l x g n x - m x]
    also from elim(3)[THEN bspec, OF ‹_  $\in$  X›, unfolded dist_norm]
    have  $\text{norm} (f \ n \ x - l \ x) \leq e / (K * Km * 4)$ 
    by simp
    also from elim(4)[THEN bspec, OF ‹_  $\in$  X›, unfolded dist_norm]
    have  $\text{norm} (g \ n \ x - m \ x) \leq e / (K * Kl * 4)$ 
    by simp
    also note Kl(2)[OF ‹_  $\in$  X›]
    also note Km(2)[OF ‹_  $\in$  X›]
    also have  $e / (K * Km * 4) * Km * K = e / 4$ 
    using ‹K > 0› ‹Km > 0› by simp
    also have  $Kl * (e / (K * Kl * 4)) * K = e / 4$ 
    using ‹K > 0› ‹Kl > 0› by simp
    also have  $e / 4 + e / 4 + e / 4 < e$  using ‹e > 0› by simp
    finally show  $\text{dist} (\text{prod} (f \ n \ x) (g \ n \ x)) (\text{prod} (l \ x) (m \ x)) < e$ 
    using ‹K > 0› ‹Kl > 0› ‹Km > 0› ‹e > 0›
    by (simp add: algebra_simps mult_right_mono divide_right_mono)
  qed
qed
qed

```

lemmas bounded_bilinear_bounded_uniform_limit_intros[uniform_limit_intros]

=

bounded_bilinear.bounded_uniform_limit[OF Inner_Product.bounded_bilinear_inner]

bounded_bilinear.bounded_uniform_limit[OF Real_Vector_Spaces.bounded_bilinear_mult]

bounded_bilinear.bounded_uniform_limit[*OF Real_Vector_Spaces.bounded_bilinear_scaleR*]

lemma *uniform_lim_mult*:

fixes *f* :: 'a $\Rightarrow$ 'b $\Rightarrow$ 'c::*real_normed_algebra*
assumes *f*: *uniform_limit* *S* *f* *l* *F*
and *g*: *uniform_limit* *S* *g* *m* *F*
and *l*: *bounded* (*l* ' *S*)
and *m*: *bounded* (*m* ' *S*)
shows *uniform_limit* *S* ($\lambda a\ b. f\ a\ b * g\ a\ b$) ($\lambda a. l\ a * m\ a$) *F*
by (*intro bounded_bilinear_bounded_uniform_limit_intros assms*)

lemma *uniform_lim_inverse*:

fixes *f* :: 'a $\Rightarrow$ 'b $\Rightarrow$ 'c::*real_normed_field*
assumes *f*: *uniform_limit* *S* *f* *l* *F*
and *b*: $\bigwedge x. x \in S \implies b \leq \text{norm}(l\ x)$
and *b* > 0
shows *uniform_limit* *S* ($\lambda x\ y. \text{inverse}\ (f\ x\ y)$) (*inverse* $\circ$ *l*) *F*

proof (*rule uniform_limitI*)

fix *e*::*real*

assume *e* > 0

have *lte*: *dist* (*inverse* (*f* *x* *y*)) ((*inverse* $\circ$ *l*) *y*) < *e*

if *b*/2 $\leq$ *norm* (*f* *x* *y*) *norm* (*f* *x* *y* - *l* *y*) < *e* * *b*² / 2 *y* $\in$ *S*

for *x* *y*

proof -

have [*simp*]: *l* *y* $\neq$ 0 *f* *x* *y* $\neq$ 0

using $\langle b > 0 \rangle$ *that* *b* [*OF* $\langle y \in S \rangle$] **by** *fastforce*+

have *norm* (*l* *y* - *f* *x* *y*) < *e* * *b*² / 2

by (*metis norm_minus_commute that*(2))

also have ... $\leq e * (\text{norm}\ (f\ x\ y) * \text{norm}\ (l\ y))$

using $\langle e > 0 \rangle$ *that* *b* [*OF* $\langle y \in S \rangle$] **apply** (*simp add: power2_eq_square*)

by (*metis* $\langle b > 0 \rangle$ *less_eq_real_def mult.left_commute mult_mono*)

finally show ?*thesis*

by (*auto simp: dist_norm field_split_simps norm_mult norm_divide*)

qed

have $\forall_F n\ \text{in}\ F. \forall x \in S. \text{dist}\ (f\ n\ x)\ (l\ x) < b/2$

using *uniform_limitD* [*OF* *f*, *of* *b*/2] **by** (*simp add:* $\langle b > 0 \rangle$)

then have $\forall_F x\ \text{in}\ F. \forall y \in S. b/2 \leq \text{norm}\ (f\ x\ y)$

apply (*rule eventually_mono*)

using *b* **apply** (*simp only: dist_norm*)

by (*metis* (*no_types*, *opaque_lifting*) *diff_zero dist_commute dist_norm norm_triangle_half_l not_less*)

then have $\forall_F x\ \text{in}\ F. \forall y \in S. b/2 \leq \text{norm}\ (f\ x\ y) \wedge \text{norm}\ (f\ x\ y - l\ y) < e * b^2 / 2$

apply (*simp only: ball_conj_distrib dist_norm [symmetric]*)

apply (*rule eventually_conj, assumption*)

apply (*rule uniform_limitD* [*OF* *f*, *of* $e * b^2 / 2$])

using $\langle b > 0 \rangle$ $\langle e > 0 \rangle$ **by** *auto*

then show $\forall_F x\ \text{in}\ F. \forall y \in S. \text{dist}\ (\text{inverse}\ (f\ x\ y))\ ((\text{inverse} \circ l)\ y) < e$

using *lte* **by** (*force intro: eventually_mono*)

qed

```

lemma uniform_lim_divide:
  fixes f :: 'a  $\Rightarrow$  'b  $\Rightarrow$  'c::real_normed_field
  assumes f: uniform_limit S f l F
    and g: uniform_limit S g m F
    and l: bounded (l ' S)
    and b:  $\bigwedge x. x \in S \implies b \leq \text{norm}(m\ x)$ 
    and b > 0
  shows uniform_limit S ( $\lambda a\ b. f\ a\ b / g\ a\ b$ ) ( $\lambda a. l\ a / m\ a$ ) F
proof -
  have m: bounded ((inverse  $\circ$  m) ' S)
    using b < b > 0
  apply (simp add: bounded_iff)
  by (metis le_imp_inverse_le norm_inverse)
  have uniform_limit S ( $\lambda a\ b. f\ a\ b * \text{inverse}(g\ a\ b)$ )
    ( $\lambda a. l\ a * (\text{inverse} \circ m)\ a$ ) F
  by (rule uniform_lim_mult [OF f uniform_lim_inverse [OF g b < b > 0]] l m)
  then show ?thesis
  by (simp add: field_class.field_divide_inverse)
qed

```

```

lemma uniform_limit_null_comparison:
  assumes  $\forall_F\ x\ \text{in}\ F. \forall a \in S. \text{norm}(f\ x\ a) \leq g\ x\ a$ 
  assumes uniform_limit S g ( $\lambda_. 0$ ) F
  shows uniform_limit S f ( $\lambda_. 0$ ) F
  using assms(2)
proof (rule metric_uniform_limit_imp_uniform_limit)
  show  $\forall_F\ x\ \text{in}\ F. \forall y \in S. \text{dist}(f\ x\ y)\ 0 \leq \text{dist}(g\ x\ y)\ 0$ 
  using assms(1) by (rule eventually_mono) (force simp add: dist_norm)
qed

```

```

lemma uniform_limit_on_Un:
  uniform_limit I f g F  $\implies$  uniform_limit J f g F  $\implies$  uniform_limit (I  $\cup$  J) f g F
by (auto intro!: uniform_limitI dest!: uniform_limitD elim: eventually_elim2)

```

```

lemma uniform_limit_on_empty [iff]:
  uniform_limit {} f g F
by (auto intro!: uniform_limitI)

```

```

lemma uniform_limit_on_UNION:
  assumes finite S
  assumes  $\bigwedge s. s \in S \implies \text{uniform_limit}(h\ s)\ f\ g\ F$ 
  shows uniform_limit ( $\bigcup (h\ ' S)$ ) f g F
  using assms
by induct (auto intro: uniform_limit_on_empty uniform_limit_on_Un)

```

```

lemma uniform_limit_on_Union:

```

```

assumes finite I
assumes  $\bigwedge J. J \in I \implies \text{uniform\_limit } J f g F$ 
shows  $\text{uniform\_limit } (\text{Union } I) f g F$ 
by (metis SUP_identity_eq assms uniform_limit_on_UNION)

```

```

lemma uniform_limit_on_subset:
   $\text{uniform\_limit } J f g F \implies I \subseteq J \implies \text{uniform\_limit } I f g F$ 
  by (auto intro!: uniform_limitI dest!: uniform_limitD intro: eventually_mono)

```

```

lemma uniform_limit_bounded:
  fixes f::'i  $\Rightarrow$  'a::topological_space  $\Rightarrow$  'b::metric_space
  assumes l:  $\text{uniform\_limit } S f l F$ 
  assumes bnd:  $\forall_F i \text{ in } F. \text{bounded } (f i ' S)$ 
  assumes F  $\neq \text{bot}$ 
  shows  $\text{bounded } (l ' S)$ 
proof -
  from l have  $\forall_F n \text{ in } F. \forall x \in S. \text{dist } (l x) (f n x) < 1$ 
    by (auto simp: uniform_limit_iff dist_commute dest!: spec[where x=1])
  with bnd
  have  $\forall_F n \text{ in } F. \exists M. \forall x \in S. \text{dist undefined } (l x) \leq M + 1$ 
    by eventually_elim
    (auto intro!: order_trans[OF dist_triangle2 add_mono] intro: less_imp_le
      simp: bounded_any_center[where a=undefined])
  then show ?thesis using assms
    by (auto simp: bounded_any_center[where a=undefined] dest!: eventually_happens)
qed

```

```

lemma uniformly_convergent_add:
   $\text{uniformly\_convergent\_on } A f \implies \text{uniformly\_convergent\_on } A g \implies$ 
   $\text{uniformly\_convergent\_on } A (\lambda k x. f k x + g k x :: 'a :: \{\text{real\_normed\_algebra}\})$ 
  unfolding uniformly_convergent_on_def by (blast dest: uniform_limit_add)

```

```

lemma uniformly_convergent_minus:
   $\text{uniformly\_convergent\_on } A f \implies \text{uniformly\_convergent\_on } A g \implies$ 
   $\text{uniformly\_convergent\_on } A (\lambda k x. f k x - g k x :: 'a :: \{\text{real\_normed\_algebra}\})$ 
  unfolding uniformly_convergent_on_def by (blast dest: uniform_limit_minus)

```

```

lemma uniformly_convergent_mult:
   $\text{uniformly\_convergent\_on } A f \implies$ 
   $\text{uniformly\_convergent\_on } A (\lambda k x. c * f k x :: 'a :: \{\text{real\_normed\_algebra}\})$ 
  unfolding uniformly_convergent_on_def
  by (blast dest: bounded_linear_uniform_limit_intros(13))

```

4.7.6 Power series and uniform convergence

```

proposition powser_uniformly_convergent:
  fixes a :: nat  $\Rightarrow$  'a::{\text{real\_normed\_div\_algebra},banach}
  assumes  $r < \text{conv\_radius } a$ 
  shows  $\text{uniformly\_convergent\_on } (\text{cball } \xi r) (\lambda n x. \sum i < n. a i * (x - \xi) ^ i)$ 

```

```

proof (cases  $0 \leq r$ )
  case True
  then have *: summable  $(\lambda n. \text{norm } (a \ n) * r ^ n)$ 
    using abs_summable_in_conv_radius [of of_real r a] assms
  by (simp add: norm_mult norm_power)
  show ?thesis
    by (simp add: Weierstrass_m_test'_ev [OF _ *] norm_mult norm_power
      mult_left_mono power_mono dist_norm norm_minus_commute)
next
  case False then show ?thesis by (simp add: not_le)
qed

lemma powser_uniform_limit:
  fixes  $a :: \text{nat} \Rightarrow 'a :: \{\text{real\_normed\_div\_algebra}, \text{banach}\}$ 
  assumes  $r < \text{conv\_radius } a$ 
  shows uniform_limit (cball  $\xi$  r)  $(\lambda n \ x. \sum_{i < n}. a \ i * (x - \xi) ^ i)$   $(\lambda x. \text{suminf}$ 
 $(\lambda i. a \ i * (x - \xi) ^ i))$  sequentially
  using powser_uniformly_convergent [OF assms]
  by (simp add: Uniform_Limit.uniformly_convergent_uniform_limit_iff Series.suminf_eq_lim)

lemma powser_continuous_suminf:
  fixes  $a :: \text{nat} \Rightarrow 'a :: \{\text{real\_normed\_div\_algebra}, \text{banach}\}$ 
  assumes  $r < \text{conv\_radius } a$ 
  shows continuous_on (cball  $\xi$  r)  $(\lambda x. \text{suminf } (\lambda i. a \ i * (x - \xi) ^ i))$ 
apply (rule uniform_limit_theorem [OF _ powser_uniform_limit])
apply (rule eventuallyI continuous_intros assms)+
apply (simp add:)
done

lemma powser_continuous_sums:
  fixes  $a :: \text{nat} \Rightarrow 'a :: \{\text{real\_normed\_div\_algebra}, \text{banach}\}$ 
  assumes  $r: r < \text{conv\_radius } a$ 
  and  $sm: \bigwedge x. x \in \text{cball } \xi \ r \implies (\lambda n. a \ n * (x - \xi) ^ n) \text{ sums } (f \ x)$ 
  shows continuous_on (cball  $\xi$  r) f
apply (rule continuous_on_cong [THEN iffD1, OF refl _ powser_continuous_suminf
  [OF r]])
using sm sums_unique by fastforce

lemmas uniform_limit_subset_union = uniform_limit_on_subset[OF uniform_limit_on_Union]

end

theory Function_Topology
imports
  Elementary_Topology
  Abstract_Limits
  Connected
begin

```

4.8 Function Topology

We want to define the general product topology.

The product topology on a product of topological spaces is generated by the sets which are products of open sets along finitely many coordinates, and the whole space along the other coordinates. This is the coarsest topology for which the projection to each factor is continuous.

To form a product of objects in Isabelle/HOL, all these objects should be subsets of a common type 'a. The product is then $\prod_{i \in I} X_i$, the set of elements from 'a such that the i -th coordinate belongs to X_i for all $i \in I$.

Hence, to form a product of topological spaces, all these spaces should be subsets of a common type. This means that type classes can not be used to define such a product if one wants to take the product of different topological spaces (as the type 'a can only be given one structure of topological space using type classes). On the other hand, one can define different topologies (as introduced in *thy*) on one type, and these topologies do not need to share the same maximal open set. Hence, one can form a product of topologies in this sense, and this works well. The big caveat is that it does not interact well with the main body of topology in Isabelle/HOL defined in terms of type classes... For instance, continuity of maps is not defined in this setting. As the product of different topological spaces is very important in several areas of mathematics (for instance adeles), I introduce below the product topology in terms of topologies, and reformulate afterwards the consequences in terms of type classes (which are of course very handy for applications).

Given this limitation, it looks to me that it would be very beneficial to revamp the theory of topological spaces in Isabelle/HOL in terms of topologies, and keep the statements involving type classes as consequences of more general statements in terms of topologies (but I am probably too naive here).

Here is an example of a reformulation using topologies. Let

$$\begin{aligned} \text{continuous\_map } T1\ T2\ f = \\ ((\forall\ U.\ \text{openin } T2\ U \longrightarrow \text{openin } T1\ (f^{-1}U \cap \text{topspace}(T1))) \\ \wedge (f^{-1}(\text{topspace } T1) \subseteq (\text{topspace } T2))) \end{aligned}$$

be the natural continuity definition of a map from the topology $T1$ to the topology $T2$. Then the current *continuous_on* (with type classes) can be redefined as

$$\begin{aligned} \text{continuous\_on } s\ f = \\ \text{continuous\_map } (\text{top\_of\_set } s)\ (\text{topology euclidean})\ f \end{aligned}$$

In fact, I need *continuous_map* to express the continuity of the projection on subfactors for the product topology, in Lemma *continuous_on_restrict_product_topology*, and I show the above equivalence in Lemma *continuous_map_iff_continuous*.

I only develop the basics of the product topology in this theory. The most important missing piece is Tychonov theorem, stating that a product of compact spaces is always compact for the product topology, even when the product is not finite (or even countable).

I realized afterwards that this theory has a lot in common with `~~/src/HOL/Library/Finite_Map.thy`.

4.8.1 The product topology

We can now define the product topology, as generated by the sets which are products of open sets along finitely many coordinates, and the whole space along the other coordinates. Equivalently, it is generated by sets which are one open set along one single coordinate, and the whole space along other coordinates. In fact, this is only equivalent for nonempty products, but for the empty product the first formulation is better (the second one gives an empty product space, while an empty product should have exactly one point, equal to *undefined* along all coordinates).

So, we use the first formulation, which moreover seems to give rise to more straightforward proofs.

definition *product_topology* :: $('i \Rightarrow ('a \text{ topology})) \Rightarrow ('i \text{ set}) \Rightarrow (('i \Rightarrow 'a) \text{ topology})$
where *product_topology* $T \ I =$
topology_generated_by $\{(\Pi_E \ i \in I. \ X \ i) \mid X. (\forall i. \text{openin } (T \ i) \ (X \ i)) \wedge \text{finite } \{i. X \ i \neq \text{topspace } (T \ i)\}\}$

abbreviation *powertop_real* :: $'a \text{ set} \Rightarrow ('a \Rightarrow \text{real}) \text{ topology}$
where *powertop_real* $\equiv \text{product\_topology } (\lambda i. \text{euclideanreal})$

The total set of the product topology is the product of the total sets along each coordinate.

proposition *product_topology*:

product_topology $X \ I =$
topology
(arbitrary union_of
((finite_intersection_of
($\lambda F. \exists i \ U. F = \{f. f \ i \in U\} \wedge i \in I \wedge \text{openin } (X \ i) \ U$))
relative_to $(\Pi_E \ i \in I. \ \text{topspace } (X \ i))$ *)))*
(is $\_ = \text{topology } (\_ \text{ union\_of } ((\_ \text{ intersection\_of } ?\Psi) \text{ relative\_to } ?TOP))$ *))*

proof –

let $? \Omega = (\lambda F. \exists Y. F = \Pi_E \ I \ Y \wedge (\forall i. \text{openin } (X \ i) \ (Y \ i)) \wedge \text{finite } \{i. Y \ i \neq \text{topspace } (X \ i)\})$

have $*$: $(\text{finite}' \text{ intersection\_of } ? \Omega) \ A = (\text{finite\_intersection\_of } ? \Psi \text{ relative\_to } ?TOP) \ A$ **for** A

proof –

have 1 : $\exists U. (\exists \mathcal{U}. \text{finite } \mathcal{U} \wedge \mathcal{U} \subseteq \text{Collect } ? \Psi \wedge \bigcap \mathcal{U} = U) \wedge ?TOP \cap U = \bigcap \mathcal{U}$

if $\mathcal{U}: \mathcal{U} \subseteq \text{Collect } ? \Omega$ **and** $\text{finite}' \mathcal{U} \ A = \bigcap \mathcal{U} \ U \neq \{\}$ **for** $\mathcal{U}$

```

proof –
  have  $\forall U \in \mathcal{U}. \exists Y. U = \text{Pi}_E \ I \ Y \wedge (\forall i. \text{openin} \ (X \ i) \ (Y \ i)) \wedge \text{finite} \ \{i. Y$ 
i  $\neq \text{topspace} \ (X \ i)\}$ 
  using  $\mathcal{U}$  by auto
  then obtain  $Y$  where  $Y: \bigwedge U. U \in \mathcal{U} \implies U = \text{Pi}_E \ I \ (Y \ U) \wedge (\forall i. \text{openin}$ 
 $(X \ i) \ (Y \ U \ i)) \wedge \text{finite} \ \{i. (Y \ U) \ i \neq \text{topspace} \ (X \ i)\}$ 
  by metis
  obtain  $U$  where  $U \in \mathcal{U}$ 
  using  $\langle \mathcal{U} \neq \{\} \rangle$  by blast
  let  $?F = \lambda U. (\lambda i. \{f. f \ i \in Y \ U \ i\}) \text{ ‘ } \{i \in I. Y \ U \ i \neq \text{topspace} \ (X \ i)\}$ 
  show ?thesis
  proof (intro conjI exI)
    show  $\text{finite} \ (\bigcup U \in \mathcal{U}. ?F \ U)$ 
    using  $Y \ \langle \text{finite}' \ \mathcal{U} \rangle$  by auto
    show  $?TOP \cap \bigcap (\bigcup U \in \mathcal{U}. ?F \ U) = \bigcap \mathcal{U}$ 
    proof
      have  $*$ :  $f \in U$ 
      if  $U \in \mathcal{U}$  and  $\forall V \in \mathcal{U}. \forall i. i \in I \wedge Y \ V \ i \neq \text{topspace} \ (X \ i) \longrightarrow f \ i \in Y$ 
V i
      and  $\forall i \in I. f \ i \in \text{topspace} \ (X \ i)$  and  $f \in \text{extensional} \ I$  for  $f \ U$ 
      proof –
        have  $\text{Pi}_E \ I \ (Y \ U) = U$ 
        using  $Y \ \langle U \in \mathcal{U} \rangle$  by blast
        then show  $f \in U$ 
        using that unfolding  $\text{Pi}_E\_def \ \text{Pi\_def}$  by blast
      qed
      show  $?TOP \cap \bigcap (\bigcup U \in \mathcal{U}. ?F \ U) \subseteq \bigcap \mathcal{U}$ 
      by (auto simp: PiE_iff *)
      show  $\bigcap \mathcal{U} \subseteq ?TOP \cap \bigcap (\bigcup U \in \mathcal{U}. ?F \ U)$ 
      using  $Y \ \text{openin\_subset} \ \langle \text{finite}' \ \mathcal{U} \rangle$  by fastforce
    qed
  qed (use  $Y \ \text{openin\_subset}$  in  $\langle \text{blast+} \rangle$ )
qed
have  $2$ :  $\exists \mathcal{U}'. \text{finite}' \ \mathcal{U}' \wedge \mathcal{U}' \subseteq \text{Collect} \ ?\Omega \wedge \bigcap \mathcal{U}' = ?TOP \cap \bigcap \mathcal{U}$ 
if  $\mathcal{U}: \mathcal{U} \subseteq \text{Collect} \ ?\Psi$  and  $\text{finite} \ \mathcal{U}$  for  $\mathcal{U}$ 
proof (cases  $\mathcal{U} = \{\}$ )
  case True
    then show ?thesis
    apply (rule_tac  $x = \{?TOP\}$  in exI, simp)
    apply (rule_tac  $x = \lambda i. \text{topspace} \ (X \ i)$  in exI)
    apply force
    done
  next
    case False
    then obtain  $U$  where  $U \in \mathcal{U}$ 
    by blast
    have  $\forall U \in \mathcal{U}. \exists i \ Y. U = \{f. f \ i \in Y\} \wedge i \in I \wedge \text{openin} \ (X \ i) \ Y$ 
    using  $\mathcal{U}$  by auto
    then obtain  $J \ Y$  where

```

```

    Y:  $\bigwedge U. U \in \mathcal{U} \implies U = \{f. f (J U) \in Y U\} \wedge J U \in I \wedge \text{openin } (X (J U)) (Y U)$ 
  by metis
  let ?G =  $\lambda U. \Pi_E i \in I. \text{if } i = J U \text{ then } Y U \text{ else } \text{topspace } (X i)$ 
  show ?thesis
  proof (intro conjI exI)
    show finite (?G 'U) ?G 'U  $\neq \{\}$ 
    using <finite U> <U  $\in \mathcal{U}$ > by blast+
    have *:  $\bigwedge U. U \in \mathcal{U} \implies \text{openin } (X (J U)) (Y U)$ 
    using Y by force
    show ?G 'U  $\subseteq \{Pi_E I Y \mid Y. (\forall i. \text{openin } (X i) (Y i)) \wedge \text{finite } \{i. Y i \neq \text{topspace } (X i)\}\}$ 
    apply clarsimp
    apply (rule_tac x=  $(\lambda i. \text{if } i = J U \text{ then } Y U \text{ else } \text{topspace } (X i))$  in exI)
    apply (auto simp: *)
    done
  next
    show  $(\bigcap U \in \mathcal{U}. ?G U) = ?TOP \cap \bigcap \mathcal{U}$ 
    proof
      have  $(\Pi_E i \in I. \text{if } i = J U \text{ then } Y U \text{ else } \text{topspace } (X i)) \subseteq (\Pi_E i \in I. \text{topspace } (X i))$ 
    apply (clarsimp simp: PiE_iff Ball_def all_conj_distrib split: if_split_asm)
    using Y <U  $\in \mathcal{U}$ > openin_subset by blast+
    then have  $(\bigcap U \in \mathcal{U}. ?G U) \subseteq ?TOP$ 
    using <U  $\in \mathcal{U}$ >
    by fastforce
    moreover have  $(\bigcap U \in \mathcal{U}. ?G U) \subseteq \bigcap \mathcal{U}$ 
    using PiE_mem Y by fastforce
    ultimately show  $(\bigcap U \in \mathcal{U}. ?G U) \subseteq ?TOP \cap \bigcap \mathcal{U}$ 
    by auto
    qed (use Y in fastforce)
  qed
  qed
  show ?thesis
  unfolding relative_to_def intersection_of_def
  by (safe; blast dest!: 1 2)
  qed
  show ?thesis
  unfolding product_topology_def generate_topology_on_eq
  apply (rule arg_cong [where f = topology])
  apply (rule arg_cong [where f = (union_of)arbitrary])
  apply (force simp: *)
  done
  qed

lemma topspace_product_topology [simp]:
   $\text{topspace } (\text{product\_topology } T I) = (\Pi_E i \in I. \text{topspace } (T i))$ 
proof
  show  $\text{topspace } (\text{product\_topology } T I) \subseteq (\Pi_E i \in I. \text{topspace } (T i))$ 

```

```

    unfolding product_topology_def topology_generated_by_topspace
    unfolding topspace_def by auto
    have  $(\prod_E i \in I. \text{topspace } (T i)) \in \{(\prod_E i \in I. X i) \mid X. (\forall i. \text{openin } (T i) (X i)) \wedge \text{finite } \{i. X i \neq \text{topspace } (T i)\}\}$ 
    using openin_topspace_not_finite_existsD by auto
    then show  $(\prod_E i \in I. \text{topspace } (T i)) \subseteq \text{topspace } (\text{product\_topology } T I)$ 
    unfolding product_topology_def using PiE_def by (auto)
qed

```

```

lemma product_topology_basis:
  assumes  $\bigwedge i. \text{openin } (T i) (X i) \text{ finite } \{i. X i \neq \text{topspace } (T i)\}$ 
  shows  $\text{openin } (\text{product\_topology } T I) (\prod_E i \in I. X i)$ 
  unfolding product_topology_def
  by (rule topology_generated_by_Basis) (use assms in auto)

```

```

proposition product_topology_open_contains_basis:
  assumes  $\text{openin } (\text{product\_topology } T I) U$   $x \in U$ 
  shows  $\exists X. x \in (\prod_E i \in I. X i) \wedge (\forall i. \text{openin } (T i) (X i)) \wedge \text{finite } \{i. X i \neq \text{topspace } (T i)\} \wedge (\prod_E i \in I. X i) \subseteq U$ 
proof -
  have  $\text{generate\_topology\_on } \{(\prod_E i \in I. X i) \mid X. (\forall i. \text{openin } (T i) (X i)) \wedge \text{finite } \{i. X i \neq \text{topspace } (T i)\}\} U$ 
  using assms unfolding product_topology_def by (intro openin_toplogy_generated_by)
  auto
  then have  $\bigwedge x. x \in U \implies \exists X. x \in (\prod_E i \in I. X i) \wedge (\forall i. \text{openin } (T i) (X i)) \wedge \text{finite } \{i. X i \neq \text{topspace } (T i)\} \wedge (\prod_E i \in I. X i) \subseteq U$ 
  proof induction
    case (Int U V x)
    then obtain XU XV where H:
       $x \in \text{Pi}_E I XU$   $(\forall i. \text{openin } (T i) (XU i))$   $\text{finite } \{i. XU i \neq \text{topspace } (T i)\}$ 
       $\text{Pi}_E I XU \subseteq U$ 
       $x \in \text{Pi}_E I XV$   $(\forall i. \text{openin } (T i) (XV i))$   $\text{finite } \{i. XV i \neq \text{topspace } (T i)\}$ 
       $\text{Pi}_E I XV \subseteq V$ 
    by auto meson
    define X where  $X = (\lambda i. XU i \cap XV i)$ 
    have  $\text{Pi}_E I X \subseteq \text{Pi}_E I XU \cap \text{Pi}_E I XV$ 
    unfolding X_def by (auto simp add: PiE_iff)
    then have  $\text{Pi}_E I X \subseteq U \cap V$  using H by auto
    moreover have  $\forall i. \text{openin } (T i) (X i)$ 
    unfolding X_def using H by auto
    moreover have  $\text{finite } \{i. X i \neq \text{topspace } (T i)\}$ 
    apply (rule rev_finite_subset[of  $\{i. XU i \neq \text{topspace } (T i)\} \cup \{i. XV i \neq \text{topspace } (T i)\}$ ])
    unfolding X_def using H by auto
    moreover have  $x \in \text{Pi}_E I X$ 
    unfolding X_def using H by auto
    ultimately show ?case
    by auto
  next

```

```

    case (UN K x)
    then obtain k where k ∈ K x ∈ k by auto
    with UN have ∃ X. x ∈ Pi_E I X ∧ (∀ i. openin (T i) (X i)) ∧ finite {i. X i
≠ topspace (T i)} ∧ Pi_E I X ⊆ k
    by simp
    then obtain X where x ∈ Pi_E I X ∧ (∀ i. openin (T i) (X i)) ∧ finite {i. X
i ≠ topspace (T i)} ∧ Pi_E I X ⊆ k
    by blast
    then have x ∈ Pi_E I X ∧ (∀ i. openin (T i) (X i)) ∧ finite {i. X i ≠ topspace
(T i)} ∧ Pi_E I X ⊆ (⋃ K)
    using ⟨k ∈ K⟩ by auto
    then show ?case
    by auto
  qed auto
  then show ?thesis using ⟨x ∈ U⟩ by auto
qed

```

lemma *product_topology_empty_discrete:*

```

  product_topology T {} = discrete_topology {(λx. undefined)}
  by (simp add: subtopology_eq_discrete_topology_sing)

```

lemma *openin_product_topology:*

```

  openin (product_topology X I) =
    arbitrary union_of
      ((finite_intersection_of (λF. (∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ openin
(X i) U)))
        relative_to topspace (product_topology X I))
  apply (subst product_topology)
  apply (simp add: topology_inverse' [OF istopology_subbase])
  done

```

lemma *subtopology_PiE:*

```

  subtopology (product_topology X I) (Π_E i∈I. (S i)) = product_topology (λi.
subtopology (X i) (S i)) I

```

proof –

```

  let ?P = λF. ∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ openin (X i) U
  let ?X = Π_E i∈I. topspace (X i)
  have finite_intersection_of ?P relative_to ?X ∩ Pi_E I S =
    finite_intersection_of (?P relative_to ?X ∩ Pi_E I S) relative_to ?X ∩ Pi_E
I S
  by (rule finite_intersection_of_relative_to)
  also have ... = finite_intersection_of
    ((λF. ∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ (openin (X i) relative_to
S i) U)
      relative_to ?X ∩ Pi_E I S)
    relative_to ?X ∩ Pi_E I S
  apply (rule arg_cong2 [where f = (relative_to)])
  apply (rule arg_cong [where f = (intersection_of)finite])
  apply (rule ext)

```

```

    apply (auto simp: relative_to_def intersection_of_def)
  done
finally
have finite_intersection_of ?P relative_to ?X  $\cap$  PiE I S =
  finite_intersection_of
    ( $\lambda F. \exists i U. F = \{f. f i \in U\} \wedge i \in I \wedge (\text{openin } (X i) \text{ relative\_to } S i) U$ )
    relative_to ?X  $\cap$  PiE I S
  by (metis finite_intersection_of_relative_to)
then show ?thesis
unfolding topology_eq
apply clarify
apply (simp add: openin_product_topology flip: openin_relative_to)
apply (simp add: arbitrary_union_of_relative_to flip: PiE_Int)
done
qed

lemma product_topology_base_alt:
  finite_intersection_of ( $\lambda F. (\exists i U. F = \{f. f i \in U\} \wedge i \in I \wedge \text{openin } (X i) U)$ )
    relative_to ( $\Pi_E i \in I. \text{topspace } (X i)$ ) =
    ( $\lambda F. (\exists U. F = \text{PiE } I U \wedge \text{finite } \{i \in I. U i \neq \text{topspace}(X i)\} \wedge (\forall i \in I. \text{openin } (X i) (U i)))$ )
  (is ?lhs = ?rhs)
proof -
  have ( $\forall F. ?lhs F \longrightarrow ?rhs F$ )
  unfolding all_relative_to all_intersection_of topspace_product_topology
proof clarify
  fix  $\mathcal{F}$ 
  assume finite  $\mathcal{F}$  and  $\mathcal{F} \subseteq \{\{f. f i \in U\} \mid i U. i \in I \wedge \text{openin } (X i) U\}$ 
  then show  $\exists U. (\Pi_E i \in I. \text{topspace } (X i)) \cap \bigcap \mathcal{F} = \text{PiE } I U \wedge$ 
    finite  $\{i \in I. U i \neq \text{topspace } (X i)\} \wedge (\forall i \in I. \text{openin } (X i) (U i))$ 
  proof (induction)
    case (insert F  $\mathcal{F}$ )
    then obtain U where eq:  $(\Pi_E i \in I. \text{topspace } (X i)) \cap \bigcap \mathcal{F} = \text{PiE } I U$ 
      and fin: finite  $\{i \in I. U i \neq \text{topspace } (X i)\}$ 
      and ope:  $\bigwedge i. i \in I \implies \text{openin } (X i) (U i)$ 
      by auto
    obtain i V where  $F = \{f. f i \in V\}$   $i \in I$   $\text{openin } (X i) V$ 
      using insert by auto
    let ?U =  $\lambda j. U j \cap (\text{if } j = i \text{ then } V \text{ else } \text{topspace}(X j))$ 
    show ?case
    proof (intro exI conjI)
      show  $(\Pi_E i \in I. \text{topspace } (X i)) \cap \bigcap (\text{insert } F \mathcal{F}) = \text{PiE } I ?U$ 
      using eq PiE_mem  $\langle i \in I \rangle$  by (auto simp:  $\langle F = \{f. f i \in V\} \rangle$ ) fastforce
    next
      show finite  $\{i \in I. ?U i \neq \text{topspace } (X i)\}$ 
      by (rule rev_finite_subset [OF finite.insertI [OF fin]]) auto
    next
      show  $\forall i \in I. \text{openin } (X i) (?U i)$ 
      by (simp add:  $\langle \text{openin } (X i) V \rangle$  ope openin_Int)
    end
  end
end

```

```

    qed
  qed (auto intro: dest: not_finite_existsD)
  qed
  moreover have  $(\forall F. ?rhs F \longrightarrow ?lhs F)$ 
  proof clarify
    fix  $U :: 'a \Rightarrow 'b \text{ set}$ 
    assume  $fin: \text{finite } \{i \in I. U\ i \neq \text{topspace } (X\ i)\}$  and  $ope: \forall i \in I. \text{openin } (X\ i)$ 
    ( $U\ i$ )
    let  $?U = \bigcap_{i \in \{i \in I. U\ i \neq \text{topspace } (X\ i)\}} \{x. x\ i \in U\ i\}$ 
    show  $?lhs (Pi_E\ I\ U)$ 
      unfolding relative_to_def topspace_product_topology
    proof (intro exI conjI)
      show  $(\text{finite\_intersection\_of } (\lambda F. \exists i\ U. F = \{f. f\ i \in U\} \wedge i \in I \wedge \text{openin } (X\ i)\ U))\ ?U$ 
      using  $fin\ ope$  by (intro finite_intersection_of_Inter finite_intersection_of_inc)
    auto
      show  $(\Pi_E\ i \in I. \text{topspace } (X\ i)) \cap ?U = Pi_E\ I\ U$ 
      using  $ope\ \text{openin\_subset}$  by fastforce
    qed
  qed
  ultimately show  $?thesis$ 
    by meson
  qed

corollary openin_product_topology_alt:
  openin (product_topology X I) S  $\longleftrightarrow$ 
   $(\forall x \in S. \exists U. \text{finite } \{i \in I. U\ i \neq \text{topspace } (X\ i)\} \wedge$ 
   $(\forall i \in I. \text{openin } (X\ i)\ (U\ i)) \wedge x \in Pi_E\ I\ U \wedge Pi_E\ I\ U \subseteq S)$ 
  unfolding openin_product_topology arbitrary_union_of_alt product_topology_base_alt
  topspace_product_topology
  apply safe
  apply (drule bspec; blast)+
  done

lemma closure_of_product_topology:
  (product_topology X I) closure_of (Pi_E I S) = Pi_E I  $(\lambda i. (X\ i)\ \text{closure\_of } (S\ i))$ 
  proof -
    have *:  $(\forall T. f \in T \wedge \text{openin } (product\_topology\ X\ I)\ T \longrightarrow (\exists y \in Pi_E\ I\ S. y \in T))$ 
       $\longleftrightarrow (\forall i \in I. \forall T. f\ i \in T \wedge \text{openin } (X\ i)\ T \longrightarrow S\ i \cap T \neq \{\})$ 
    (is  $?lhs = ?rhs$ )
    if top:  $\bigwedge i. i \in I \implies f\ i \in \text{topspace } (X\ i)$  and ext:  $f \in \text{extensional } I$  for  $f$ 
  proof
    assume  $L[\text{rule\_format}]: ?lhs$ 
    show  $?rhs$ 
    proof clarify
      fix  $i\ T$ 
      assume  $i \in I\ f\ i \in T\ \text{openin } (X\ i)\ T\ S\ i \cap T = \{\}$ 

```

```

then have openin (product_topology X I) (( $\Pi_{E \ i \in I} \text{topspace } (X \ i)$ )  $\cap \{x. x \ i \in T\}$ )
by (force simp: openin_product_topology intro: arbitrary_union_of_inc
relative_to_inc finite_intersection_of_inc)
then show False
using L [of topspace (product_topology X I)  $\cap \{f. f \ i \in T\}$ ]  $\langle S \ i \cap T = \{ \} \rangle$ 
 $\langle f \ i \in T \rangle \langle i \in I \rangle$ 
by (auto simp: top ext PiE_iff)
qed
next
assume R [rule_format]: ?rhs
show ?lhs
proof (clarsimp simp: openin_product_topology union_of_def arbitrary_def)
fix U U
assume
  U:  $\mathcal{U} \subseteq \text{Collect}$ 
  (finite_intersection_of ( $\lambda F. \exists i \ U. F = \{x. x \ i \in U\} \wedge i \in I \wedge \text{openin } (X \ i) \ U$ )
relative_to
    ( $\Pi_{E \ i \in I} \text{topspace } (X \ i)$ )) and
  f  $\in U \ U \in \mathcal{U}$ 
then have (finite_intersection_of ( $\lambda F. \exists i \ U. F = \{x. x \ i \in U\} \wedge i \in I \wedge$ 
openin (X i) U)
    relative_to ( $\Pi_{E \ i \in I} \text{topspace } (X \ i)$ )) U
by blast
with  $\langle f \in U \rangle \langle U \in \mathcal{U} \rangle$ 
obtain  $\mathcal{T}$  where finite  $\mathcal{T}$ 
and  $\mathcal{T}$ :  $\bigwedge C. C \in \mathcal{T} \implies \exists i \in I. \exists V. \text{openin } (X \ i) \ V \wedge C = \{x. x \ i \in V\}$ 
and topspace (product_topology X I)  $\cap \bigcap \mathcal{T} \subseteq U$  f  $\in \text{topspace } (\text{product\_topology}$ 
X I)  $\cap \bigcap \mathcal{T}$ 
apply (clarsimp simp add: relative_to_def intersection_of_def)
apply (rule that, auto dest!: subsetD)
done
then have f  $\in \text{PiE } I \ (\text{topspace} \circ X) \ f \in \bigcap \mathcal{T}$  and subU: PiE I (topspace  $\circ$ 
X)  $\cap \bigcap \mathcal{T} \subseteq U$ 
by (auto simp: PiE_iff)
have *: f  $\ i \in \text{topspace } (X \ i) \cap \bigcap \{U. \text{openin } (X \ i) \ U \wedge \{x. x \ i \in U\} \in \mathcal{T}\}$ 
 $\wedge \text{openin } (X \ i) \ (\text{topspace } (X \ i) \cap \bigcap \{U. \text{openin } (X \ i) \ U \wedge \{x. x \ i \in U\}$ 
 $\in \mathcal{T}\})$ 
if i  $\in I$  for i
proof –
have finite ( $\lambda U. \{x. x \ i \in U\}$ ) – ‘ $\mathcal{T}$ ’
proof (rule finite_vimageI [OF  $\langle \text{finite } \mathcal{T} \rangle$ ])
show inj ( $\lambda U. \{x. x \ i \in U\}$ )
by (auto simp: inj_on_def)
qed
then have fin: finite  $\{U. \text{openin } (X \ i) \ U \wedge \{x. x \ i \in U\} \in \mathcal{T}\}$ 
by (rule rev_finite_subset) auto
have openin (X i) ( $\bigcap (\text{insert } (\text{topspace } (X \ i)) \ \{U. \text{openin } (X \ i) \ U \wedge \{x. x$ 
 $i \in U\} \in \mathcal{T}\})$ )

```



```

    by (rule openin_Inter) (auto simp: fin)
  then show ?thesis
    using ⟨f ∈ ⋂ T⟩ by (fastforce simp: that top)
qed
define Φ where Φ ≡ λi. topspace (X i) ∩ ⋂ {U. openin (X i) U ∧ {f. f i ∈
U} ∈ T}
  have ∀i ∈ I. ∃x. x ∈ S i ∩ Φ i
    using R [OF _ *] unfolding Φ_def by blast
  then obtain ∅ where ∅ [rule_format]: ∀i ∈ I. ∅ i ∈ S i ∩ Φ i
    by metis
  show ∃y ∈ PiE I S. ∃x ∈ U. y ∈ x
  proof
    show ∃U ∈ U. (λi ∈ I. ∅ i) ∈ U
  proof
    have restrict ∅ I ∈ PiE I (topspace ∘ X) ∩ ⋂ T
      using T by (fastforce simp: Φ_def PiE_def dest: ∅)
    then show restrict ∅ I ∈ U
      using subU by blast
    qed (rule ⟨U ∈ U⟩)
  next
    show (λi ∈ I. ∅ i) ∈ PiE I S
      using ∅ by simp
    qed
  qed
qed
show ?thesis
  apply (simp add: * closure_of_def PiE_iff set_eq_iff cong: conj_cong)
  by metis
qed

corollary closedin_product_topology:
  closedin (product_topology X I) (PiE I S) ⟷ PiE I S = {} ∨ (∀i ∈ I. closedin
(X i) (S i))
  apply (simp add: PiE_eq PiE_eq_empty_iff closure_of_product_topology flip:
closure_of_eq)
  apply (metis closure_of_empty)
  done

corollary closedin_product_topology_singleton:
  f ∈ extensional I ⟹ closedin (product_topology X I) {f} ⟷ (∀i ∈ I. closedin
(X i) {f i})
  using PiE_singleton closedin_product_topology [of X I]
  by (metis (no_types, lifting) all_not_in_conv insertI1)

lemma product_topology_empty:
  product_topology X {} = topology (λS. S ∈ {{}}, {λk. undefined})
  unfolding product_topology union_of_def intersection_of_def arbitrary_def rel-
ative_to_def
  by (auto intro: arg_cong [where f=topology])

```

```

lemma openin_product_topology_empty: openin (product_topology X {}) S  $\longleftrightarrow$ 
S  $\in$  { {}, { $\lambda k.$  undefined} }
  unfolding union_of_def intersection_of_def arbitrary_def relative_to_def openin_product_topology
  by auto

```

The basic property of the product topology is the continuity of projections:

```

lemma continuous_map_product_coordinates [simp]:
  assumes i  $\in$  I
  shows continuous_map (product_topology T I) (T i) ( $\lambda x.$  x i)
proof –
  {
    fix U assume openin (T i) U
    define X where X = ( $\lambda j.$  if j = i then U else topspace (T j))
    then have *: ( $\lambda x.$  x i)  $-'$  U  $\cap$  ( $\Pi_E$  i  $\in$  I. topspace (T i)) = ( $\Pi_E$  j  $\in$  I. X j)
      unfolding X_def using assms openin_subset [OF  $\langle$  openin (T i) U  $\rangle$ ]
      by (auto simp add: PiE_iff, auto, metis subsetCE)
    have **: ( $\forall i.$  openin (T i) (X i))  $\wedge$  finite {i. X i  $\neq$  topspace (T i)}
      unfolding X_def using  $\langle$  openin (T i) U  $\rangle$  by auto
    have openin (product_topology T I) (( $\lambda x.$  x i)  $-'$  U  $\cap$  ( $\Pi_E$  i  $\in$  I. topspace (T
i)))
      unfolding product_topology_def
      apply (rule topology_generated_by_Basis)
      apply (subst *)
      using ** by auto
  }
  then show ?thesis unfolding continuous_map_alt
    by (auto simp add: assms PiE_iff)
qed

```

```

lemma continuous_map_coordinatewise_then_product [intro]:
  assumes  $\bigwedge i.$  i  $\in$  I  $\implies$  continuous_map T1 (T i) ( $\lambda x.$  f x i)
     $\bigwedge i$  x. i  $\notin$  I  $\implies$  x  $\in$  topspace T1  $\implies$  f x i = undefined
  shows continuous_map T1 (product_topology T I) f
unfolding product_topology_def
proof (rule continuous_on_generated_topo)
  fix U assume U  $\in$  {PiE I X | X. ( $\forall i.$  openin (T i) (X i))  $\wedge$  finite {i. X i  $\neq$ 
topspace (T i)}}
  then obtain X where H: U = PiE I X  $\bigwedge i.$  openin (T i) (X i) finite {i. X i  $\neq$ 
topspace (T i)}
    by blast
  define J where J = {i  $\in$  I. X i  $\neq$  topspace (T i)}
  have finite J J  $\subseteq$  I unfolding J_def using H (3) by auto
  have ( $\lambda x.$  f x i)  $-'$  (topspace (T i))  $\cap$  topspace T1 = topspace T1 if i  $\in$  I for i
    using that assms (1) by (simp add: continuous_map_preimage_topspace)
  then have *: ( $\lambda x.$  f x i)  $-'$  (X i)  $\cap$  topspace T1 = topspace T1 if i  $\in$  I  $-$  J for i
    using that unfolding J_def by auto

```

```

  have  $f - 'U \cap \text{topspace } T1 = (\bigcap_{i \in I}. (\lambda x. f x i) - '(X i) \cap \text{topspace } T1) \cap$ 
    ( $\text{topspace } T1$ )
  by (subst  $H(1)$ , auto simp add:  $\text{PiE\_iff}$  assms)
  also have  $\dots = (\bigcap_{i \in J}. (\lambda x. f x i) - '(X i) \cap \text{topspace } T1) \cap (\text{topspace } T1)$ 
  using  $* \langle J \subseteq I \rangle$  by auto
  also have  $\text{openin } T1 \dots$ 
  apply (rule  $\text{openin\_INT}$ )
  apply (simp add:  $\langle \text{finite } J \rangle$ )
  using  $H(2)$  assms(1)  $\langle J \subseteq I \rangle$  by auto
  ultimately show  $\text{openin } T1 (f - 'U \cap \text{topspace } T1)$  by simp
next
  show  $f - ' \text{topspace } T1 \subseteq \bigcup \{ \text{PiE } I X \mid X. (\forall i. \text{openin } (T i) (X i)) \wedge \text{finite } \{i. X i \neq \text{topspace } (T i)\} \}$ 
  apply (subst  $\text{topology\_generated\_by\_topspace}$ [ $\text{symmetric}$ ])
  apply (subst  $\text{product\_topology\_def}$ [ $\text{symmetric}$ ])
  apply (simp only:  $\text{topspace\_product\_topology}$ )
  apply (auto simp add:  $\text{PiE\_iff}$ )
  using assms unfolding  $\text{continuous\_map\_def}$  by auto
qed

```

lemma $\text{continuous\_map\_product\_then\_coordinatewise}$ [intro]:

```

  assumes  $\text{continuous\_map } T1 (\text{product\_topology } T I) f$ 
  shows  $\bigwedge i. i \in I \implies \text{continuous\_map } T1 (T i) (\lambda x. f x i)$ 
   $\bigwedge i x. i \notin I \implies x \in \text{topspace } T1 \implies f x i = \text{undefined}$ 

```

proof –

```

  fix  $i$  assume  $i \in I$ 
  have  $(\lambda x. f x i) = (\lambda y. y i) \circ f$  by auto
  also have  $\text{continuous\_map } T1 (T i) \dots$ 
  apply (rule  $\text{continuous\_map\_compose}$ [of  $\_ \text{product\_topology } T I$ ])
  using assms  $\langle i \in I \rangle$  by auto
  ultimately show  $\text{continuous\_map } T1 (T i) (\lambda x. f x i)$ 
  by simp

```

next

```

  fix  $i x$  assume  $i \notin I$   $x \in \text{topspace } T1$ 
  have  $f x \in \text{topspace } (\text{product\_topology } T I)$ 
  using assms  $\langle x \in \text{topspace } T1 \rangle$  unfolding  $\text{continuous\_map\_def}$  by auto
  then have  $f x \in (\prod_{i \in I}. \text{topspace } (T i))$ 
  using  $\text{topspace\_product\_topology}$  by metis
  then show  $f x i = \text{undefined}$ 
  using  $\langle i \notin I \rangle$  by (auto simp add:  $\text{PiE\_iff}$   $\text{extensional\_def}$ )
qed

```

lemma $\text{continuous\_on\_restrict}$:

```

  assumes  $J \subseteq I$ 
  shows  $\text{continuous\_map } (\text{product\_topology } T I) (\text{product\_topology } T J) (\lambda x. \text{restrict } x J)$ 
proof (rule  $\text{continuous\_map\_coordinatewise\_then\_product}$ )
  fix  $i$  assume  $i \in J$ 
  then have  $(\lambda x. \text{restrict } x J i) = (\lambda x. x i)$  unfolding  $\text{restrict\_def}$  by auto

```

```

    then show continuous_map (product_topology T I) (T i) (λx. restrict x J i)
      using ⟨i ∈ J⟩ ⟨J ⊆ I⟩ by auto
  next
    fix i assume i ∉ J
    then show restrict x J i = undefined for x::'a ⇒ 'b
      unfolding restrict_def by auto
qed

```

Powers of a single topological space as a topological space, using type classes

```

instantiation fun :: (type, topological_space) topological_space
begin

```

```

definition open_fun_def:
  open U = openin (product_topology (λi. euclidean) UNIV) U

```

```

instance proof
  have topspace (product_topology (λ(i::'a). euclidean::('b topology)) UNIV) =
    UNIV
    unfolding topspace_product_topology topspace_euclidean by auto
  then show open (UNIV::('a ⇒ 'b) set)
    unfolding open_fun_def by (metis openin_topspace)
qed (auto simp add: open_fun_def)

end

```

```

lemma open_PiE [intro?]:
  fixes X::'i ⇒ ('b::topological_space) set
  assumes ∧i. open (X i) finite {i. X i ≠ UNIV}
  shows open (PiE UNIV X)
    by (simp add: assms open_fun_def product_topology_basis)

```

```

lemma euclidean_product_topology:
  product_topology (λi. euclidean::('b::topological_space) topology) UNIV = eu-
    clidean
by (metis open_openin topology_eq open_fun_def)

```

```

proposition product_topology_basis':
  fixes x::'i ⇒ 'a and U::'i ⇒ ('b::topological_space) set
  assumes finite I ∧i. i ∈ I ⇒ open (U i)
  shows open {f. ∀ i∈I. f (x i) ∈ U i}
proof –
  define J where J = x`I
  define V where V = (λy. if y ∈ J then ⋂ {U i | i. i ∈ I ∧ x i = y} else UNIV)
  define X where X = (λy. if y ∈ J then V y else UNIV)
  have *: open (X i) for i
    unfolding X_def V_def using assms by auto
  have **: finite {i. X i ≠ UNIV}

```

```

  unfolding X_def V_def J_def using assms(1) by auto
  have open (PiE UNIV X)
  by (simp add: * ** open_PiE)
  moreover have PiE UNIV X = {f.  $\forall i \in I. f(x\ i) \in U\ i$ }
  apply (auto simp add: PiE_iff) unfolding X_def V_def J_def
  proof (auto)
    fix f :: 'a  $\Rightarrow$  'b and i :: 'i
    assume a1: i  $\in$  I
    assume a2:  $\forall i. f\ i \in (if\ i \in x'I\ then\ if\ i \in x'I\ then\ \bigcap\ \{U\ ia\ |\ ia. ia \in I \wedge x\ ia = i\}\ else\ UNIV\ else\ UNIV)$ 
    have f3: x i  $\in$  x'I
    using a1 by blast
    have U i  $\in$  {U ia | ia. ia  $\in$  I  $\wedge$  x ia = x i}
    using a1 by blast
    then show f(x i)  $\in$  U i
    using f3 a2 by (meson Inter_iff)
  qed
  ultimately show ?thesis by simp
qed

```

The results proved in the general situation of products of possibly different spaces have their counterparts in this simpler setting.

```

lemma continuous_on_product_coordinates [simp]:
  continuous_on UNIV ( $\lambda x. x\ i :: ('b :: topological\_space)$ )
  using continuous_map_product_coordinates [of _ UNIV  $\lambda i. euclidean$ ]
  by (metis (no_types) continuous_map_iff continuous_euclidean_product_topology
  iso_tuple_UNIV_I subtopology_UNIV)

```

```

lemma continuous_on_coordinatewise_then_product [continuous_intros]:
  fixes f :: 'a :: topological_space  $\Rightarrow$  'b  $\Rightarrow$  'c :: topological_space
  assumes  $\bigwedge i. continuous\_on\ S\ (\lambda x. f\ x\ i)$ 
  shows continuous_on S f
  using continuous_map_coordinatewise_then_product [of UNIV, where T =
 $\lambda i. euclidean$ ]
  by (metis UNIV_I assms continuous_map_iff continuous_euclidean_product_topology)

```

```

lemma continuous_on_product_then_coordinatewise_UNIV:
  assumes continuous_on UNIV f
  shows continuous_on UNIV ( $\lambda x. f\ x\ i$ )
  unfolding continuous_map_iff_continuous2 [symmetric]
  by (rule continuous_map_product_then_coordinatewise [where I=UNIV]) (use
  assms in <auto simp: euclidean_product_topology>)

```

```

lemma continuous_on_product_then_coordinatewise:
  assumes continuous_on S f
  shows continuous_on S ( $\lambda x. f\ x\ i$ )
proof -
  have continuous_on S (( $\lambda q. q\ i$ )  $\circ$  f)
  by (metis assms continuous_on_compose continuous_on_id)

```

```

      continuous_on_product_then_coordinatewise_UNIV continuous_on_subset
subset_UNIV)
  then show ?thesis
    by auto
qed

```

```

lemma continuous_on_coordinatewise_iff:
  fixes f :: ('a  $\Rightarrow$  real)  $\Rightarrow$  'b  $\Rightarrow$  real
  shows continuous_on (A  $\cap$  S) f  $\longleftrightarrow$  ( $\forall i$ . continuous_on (A  $\cap$  S) ( $\lambda x$ . f x i))
  by (auto simp: continuous_on_product_then_coordinatewise continuous_on_coordinatewise_then_pro

```

```

lemma continuous_map_span_sum:
  fixes B :: 'a::real_normed_vector set
  assumes biB:  $\bigwedge i$ .  $i \in I \implies b\ i \in B$ 
  shows continuous_map euclidean (top_of_set (span B)) ( $\lambda x$ .  $\sum_{i \in I} x\ i *_{\mathbb{R}} b\ i$ )
i)
proof (rule continuous_map_euclidean_top_of_set)
  show ( $\lambda x$ .  $\sum_{i \in I} x\ i *_{\mathbb{R}} b\ i$ ) - 'span B = UNIV
    by auto (meson biB lessThan_iff span_base span_scale span_sum)
  show continuous_on UNIV ( $\lambda x$ .  $\sum_{i \in I} x\ i *_{\mathbb{R}} b\ i$ )
    by (intro continuous_intros) auto
qed

```

Topological countability for product spaces

The next two lemmas are useful to prove first or second countability of product spaces, but they have more to do with countability and could be put in the corresponding theory.

```

lemma countable_nat_product_event_const:
  fixes F :: 'a set and a :: 'a
  assumes a  $\in$  F countable F
  shows countable {x :: (nat  $\Rightarrow$  'a). ( $\forall i$ .  $x\ i \in F$ )  $\wedge$  finite {i.  $x\ i \neq a$ }}
proof -
  have *: {x :: (nat  $\Rightarrow$  'a). ( $\forall i$ .  $x\ i \in F$ )  $\wedge$  finite {i.  $x\ i \neq a$ }}
     $\subseteq$  ( $\bigcup N$ . {x. ( $\forall i$ .  $x\ i \in F$ )  $\wedge$  ( $\forall i \geq N$ .  $x\ i = a$ )})
  using infinite_nat_iff_unbounded_le by fastforce
  have countable {x. ( $\forall i$ .  $x\ i \in F$ )  $\wedge$  ( $\forall i \geq N$ .  $x\ i = a$ )} for N :: nat
  proof (induction N)
    case 0
    have {x. ( $\forall i$ .  $x\ i \in F$ )  $\wedge$  ( $\forall i \geq (0 :: nat)$ .  $x\ i = a$ )} = {( $\lambda i$ . a)}
      using  $\langle a \in F \rangle$  by auto
    then show ?case by auto
  next
    case (Suc N)
    define f :: ((nat  $\Rightarrow$  'a)  $\times$  'a)  $\Rightarrow$  (nat  $\Rightarrow$  'a)
      where f = ( $\lambda (x, b)$ . ( $\lambda i$ . if  $i = N$  then b else  $x\ i$ ))
    have {x. ( $\forall i$ .  $x\ i \in F$ )  $\wedge$  ( $\forall i \geq \text{Suc } N$ .  $x\ i = a$ )}  $\subseteq$  f'({x. ( $\forall i$ .  $x\ i \in F$ )  $\wedge$ 
( $\forall i \geq N$ .  $x\ i = a$ )}  $\times$  F)
    proof (auto)

```

```

fix x assume H:  $\forall i::nat. x\ i \in F \ \forall i \geq Suc\ N. x\ i = a$ 
define y where  $y = (\lambda i. \text{if } i = N \text{ then } a \text{ else } x\ i)$ 
have f (y, x N) = x
  unfolding f_def y_def by auto
moreover have  $(y, x\ N) \in \{x. (\forall i. x\ i \in F) \wedge (\forall i \geq N. x\ i = a)\} \times F$ 
  unfolding y_def using H  $\langle a \in F \rangle$  by auto
ultimately show  $x \in f'(\{x. (\forall i. x\ i \in F) \wedge (\forall i \geq N. x\ i = a)\} \times F)$ 
  by (metis (no_types, lifting) image_eqI)
qed
moreover have countable  $(\{x. (\forall i. x\ i \in F) \wedge (\forall i \geq N. x\ i = a)\} \times F)$ 
  using Suc.IH assms(2) by auto
ultimately show ?case
  by (meson countable_image countable_subset)
qed
then show ?thesis using countable_subset[OF *] by auto
qed

lemma countable_product_event_const:
  fixes F::('a::countable)  $\Rightarrow$  'b set and b::'b
  assumes  $\bigwedge i. \text{countable } (F\ i)$ 
  shows countable  $\{f::('a \Rightarrow 'b). (\forall i. f\ i \in F\ i) \wedge (\text{finite } \{i. f\ i \neq b\})\}$ 
proof -
  define G where  $G = (\bigcup i. F\ i) \cup \{b\}$ 
  have countable G unfolding G_def using assms by auto
  have b  $\in G$  unfolding G_def by auto
  define pi where  $pi = (\lambda(x::(\text{nat} \Rightarrow 'b)). (\lambda i::'a. x\ ((\text{to\_nat}::('a \Rightarrow \text{nat}))\ i)))$ 
  have  $\{f::('a \Rightarrow 'b). (\forall i. f\ i \in F\ i) \wedge (\text{finite } \{i. f\ i \neq b\})\}$ 
     $\subseteq pi'\{g::(\text{nat} \Rightarrow 'b). (\forall j. g\ j \in G) \wedge (\text{finite } \{j. g\ j \neq b\})\}$ 
  proof (auto)
    fix f assume H:  $\forall i. f\ i \in F\ i \ \text{finite } \{i. f\ i \neq b\}$ 
    define I where  $I = \{i. f\ i \neq b\}$ 
    define g where  $g = (\lambda j. \text{if } j \in \text{to\_nat } I \text{ then } f\ (\text{from\_nat } j) \text{ else } b)$ 
    have  $\{j. g\ j \neq b\} \subseteq \text{to\_nat } I$  unfolding g_def by auto
    then have finite  $\{j. g\ j \neq b\}$ 
      unfolding I_def using H(2) using finite_surj by blast
    moreover have  $g\ j \in G$  for j
      unfolding g_def G_def using H by auto
    ultimately have  $g \in \{g::(\text{nat} \Rightarrow 'b). (\forall j. g\ j \in G) \wedge (\text{finite } \{j. g\ j \neq b\})\}$ 
      by auto
    moreover have  $f = pi\ g$ 
      unfolding pi_def g_def I_def using H by fastforce
    ultimately show  $f \in pi'\{g. (\forall j. g\ j \in G) \wedge \text{finite } \{j. g\ j \neq b\}\}$ 
      by auto
  qed
qed
then show ?thesis
  using countable_nat_product_event_const[OF  $\langle b \in G \rangle \langle \text{countable } G \rangle$ ]
  by (meson countable_image countable_subset)
qed

```

```

instance fun :: (countable, first_countable_topology) first_countable_topology
proof
  fix x::'a  $\Rightarrow$  'b
  have  $\exists A::('b \Rightarrow \text{nat} \Rightarrow 'b \text{ set}). \forall x. (\forall i. x \in A \ x \ i \wedge \text{open} \ (A \ x \ i)) \wedge (\forall S. \text{open} \ S \wedge x \in S \longrightarrow (\exists i. A \ x \ i \subseteq S))$ 
  apply (rule choice) using first_countable_basis by auto
  then obtain A::('b  $\Rightarrow$  nat  $\Rightarrow$  'b set) where A:  $\bigwedge x \ i. x \in A \ x \ i$ 
   $\bigwedge x \ i. \text{open} \ (A \ x \ i)$ 
   $\bigwedge x \ S. \text{open} \ S \Longrightarrow x \in S \Longrightarrow (\exists i. A \ x \ i \subseteq S)$ 
  by metis

  B i is a countable basis of neighborhoods of  $x_i$ .

  define B where B =  $(\lambda i. (A \ (x \ i))'UNIV \cup \{UNIV\})$ 
  have countable (B i) for i unfolding B_def by auto
  have open_B:  $\bigwedge X \ i. X \in B \ i \Longrightarrow \text{open} \ X$ 
  by (auto simp: B_def A)
  define K where K =  $\{Pi_E \ UNIV \ X \mid X. (\forall i. X \ i \in B \ i) \wedge \text{finite} \ \{i. X \ i \neq UNIV\}\}$ 
  have  $Pi_E \ UNIV \ (\lambda i. UNIV) \in K$ 
  unfolding K_def B_def by auto
  then have  $K \neq \{\}$  by auto
  have countable  $\{X. (\forall i. X \ i \in B \ i) \wedge \text{finite} \ \{i. X \ i \neq UNIV\}\}$ 
  apply (rule countable_product_event_const) using  $\langle \bigwedge i. \text{countable} \ (B \ i) \rangle$  by auto
  moreover have  $K = (\lambda X. Pi_E \ UNIV \ X)'\{X. (\forall i. X \ i \in B \ i) \wedge \text{finite} \ \{i. X \ i \neq UNIV\}\}$ 
  unfolding K_def by auto
  ultimately have countable K by auto
  have  $x \in k$  if  $k \in K$  for k
  using that unfolding K_def B_def apply auto using A(1) by auto
  have open k if  $k \in K$  for k
  using that unfolding K_def by (blast intro: open_B open_PiE elim: )
  have Inc:  $\exists k \in K. k \subseteq U$  if open U  $\wedge x \in U$  for U
  proof –
    have openin (product_topology  $(\lambda i. \text{euclidean}) \ UNIV$ ) U  $x \in U$ 
    using  $\langle \text{open} \ U \wedge x \in U \rangle$  unfolding open_fun_def by auto
    with product_topology_open_contains_basis[OF this]
    have  $\exists X. x \in (\Pi_E \ i \in UNIV. X \ i) \wedge (\forall i. \text{open} \ (X \ i)) \wedge \text{finite} \ \{i. X \ i \neq UNIV\}$ 
   $\wedge (\Pi_E \ i \in UNIV. X \ i) \subseteq U$ 
    by simp
    then obtain X where H:  $x \in (\Pi_E \ i \in UNIV. X \ i)$ 
     $\bigwedge i. \text{open} \ (X \ i)$ 
     $\text{finite} \ \{i. X \ i \neq UNIV\}$ 
     $(\Pi_E \ i \in UNIV. X \ i) \subseteq U$ 
    by auto
    define I where I =  $\{i. X \ i \neq UNIV\}$ 
    define Y where Y =  $(\lambda i. \text{if } i \in I \text{ then } (SOME \ y. y \in B \ i \wedge y \subseteq X \ i) \text{ else } UNIV)$ 
    have *:  $\exists y. y \in B \ i \wedge y \subseteq X \ i$  for i

```



```

    unfolding B_def using A(3)[OF H(2)] H(1) by (metis PiE_E UNIV_I
UnCI image_iff)
  have **:  $Y\ i \in B\ i \wedge Y\ i \subseteq X\ i$  for  $i$ 
  apply (cases  $i \in I$ )
  unfolding Y_def using * that apply (auto)
  apply (metis (no_types, lifting) someI, metis (no_types, lifting) someI_ex
subset_iff)
  unfolding B_def apply simp
  unfolding I_def apply auto
  done
  have  $\{i. Y\ i \neq UNIV\} \subseteq I$ 
  unfolding Y_def by auto
  then have ***:  $finite\ \{i. Y\ i \neq UNIV\}$ 
  unfolding I_def using H(3) rev_finite_subset by blast
  have  $(\forall i. Y\ i \in B\ i) \wedge finite\ \{i. Y\ i \neq UNIV\}$ 
  using ** *** by auto
  then have  $Pi_E\ UNIV\ Y \in K$ 
  unfolding K_def by auto

  have  $Y\ i \subseteq X\ i$  for  $i$ 
  apply (cases  $i \in I$ ) using ** apply simp unfolding Y_def I_def by auto
  then have  $Pi_E\ UNIV\ Y \subseteq Pi_E\ UNIV\ X$  by auto
  then have  $Pi_E\ UNIV\ Y \subseteq U$  using H(4) by auto
  then show ?thesis using  $\langle Pi_E\ UNIV\ Y \in K \rangle$  by auto
qed

  show  $\exists L. (\forall (i::nat). x \in L\ i \wedge open\ (L\ i)) \wedge (\forall U. open\ U \wedge x \in U \longrightarrow (\exists i. L\ i \subseteq U))$ 
  apply (rule first_countableI[of K])
  using  $\langle countable\ K \rangle \langle \bigwedge k. k \in K \implies x \in k \rangle \langle \bigwedge k. k \in K \implies open\ k \rangle Inc$  by
auto
qed

proposition product_topology_countable_basis:
  shows  $\exists K::('a::countable \Rightarrow 'b::second\_countable\_topology)\ set\ set).$ 
 $topological\_basis\ K \wedge countable\ K \wedge$ 
 $(\forall k \in K. \exists X. (k = Pi_E\ UNIV\ X) \wedge (\forall i. open\ (X\ i)) \wedge finite\ \{i. X\ i \neq UNIV\})$ 
proof –
  obtain  $B::'b\ set\ set$  where  $B: countable\ B \wedge topological\_basis\ B$ 
  using ex_countable_basis by auto
  then have  $B \neq \{\}$  by (meson UNIV_I empty_iff open_UNIV topological_basisE)
  define B2 where  $B2 = B \cup \{UNIV\}$ 
  have countable B2
  unfolding B2_def using B by auto
  have open U if  $U \in B2$  for U
  using that unfolding B2_def using B topological_basis_open by auto

  define K where  $K = \{Pi_E\ UNIV\ X \mid X. (\forall i::'a. X\ i \in B2) \wedge finite\ \{i. X\ i \neq$ 

```

```

UNIV}}
  have i:  $\forall k \in K. \exists X. (k = \text{Pi}_E \text{ UNIV } X) \wedge (\forall i. \text{open } (X \ i)) \wedge \text{finite } \{i. X \ i \neq \text{UNIV}\}$ 
  unfolding K_def using  $\langle \bigwedge U. U \in B2 \implies \text{open } U \rangle$  by auto

  have countable  $\{X. (\forall (i::'a). X \ i \in B2) \wedge \text{finite } \{i. X \ i \neq \text{UNIV}\}\}$ 
  apply (rule countable_product_event_const) using  $\langle \text{countable } B2 \rangle$  by auto
  moreover have  $K = (\lambda X. \text{Pi}_E \text{ UNIV } X) \{X. (\forall i. X \ i \in B2) \wedge \text{finite } \{i. X \ i \neq \text{UNIV}\}\}$ 
  unfolding K_def by auto
  ultimately have ii: countable K by auto

  have iii: topological_basis K
  proof (rule topological_basisI)
    fix U and x::'a  $\Rightarrow$  'b assume open U  $x \in U$ 
    then have openin (product_topology ( $\lambda i. \text{euclidean}$ ) UNIV) U
    unfolding open_fun_def by auto
    with product_topology_open_contains_basis[OF this  $\langle x \in U \rangle$ ]
    have  $\exists X. x \in (\Pi_E i \in \text{UNIV}. X \ i) \wedge (\forall i. \text{open } (X \ i)) \wedge \text{finite } \{i. X \ i \neq \text{UNIV}\}$ 
     $\wedge (\Pi_E i \in \text{UNIV}. X \ i) \subseteq U$ 
    by simp
    then obtain X where  $H: x \in (\Pi_E i \in \text{UNIV}. X \ i)$ 
       $\bigwedge i. \text{open } (X \ i)$ 
       $\text{finite } \{i. X \ i \neq \text{UNIV}\}$ 
       $(\Pi_E i \in \text{UNIV}. X \ i) \subseteq U$ 
    by auto
    then have  $x \ i \in X \ i$  for i by auto
    define I where  $I = \{i. X \ i \neq \text{UNIV}\}$ 
    define Y where  $Y = (\lambda i. \text{if } i \in I \text{ then } (\text{SOME } y. y \in B2 \wedge y \subseteq X \ i \wedge x \ i \in y) \text{ else UNIV})$ 
    have *:  $\exists y. y \in B2 \wedge y \subseteq X \ i \wedge x \ i \in y$  for i
    unfolding B2_def using B  $\langle \text{open } (X \ i) \rangle$   $\langle x \ i \in X \ i \rangle$  by (meson UnCI topological_basisE)
    have **:  $Y \ i \in B2 \wedge Y \ i \subseteq X \ i \wedge x \ i \in Y \ i$  for i
    using someI_ex[OF *]
    apply (cases i  $\in I$ )
    unfolding Y_def using * apply (auto)
    unfolding B2_def I_def by auto
    have  $\{i. Y \ i \neq \text{UNIV}\} \subseteq I$ 
    unfolding Y_def by auto
    then have ***:  $\text{finite } \{i. Y \ i \neq \text{UNIV}\}$ 
    unfolding I_def using H(3) rev_finite_subset by blast
    have  $(\forall i. Y \ i \in B2) \wedge \text{finite } \{i. Y \ i \neq \text{UNIV}\}$ 
    using ** *** by auto
    then have  $\text{Pi}_E \text{ UNIV } Y \in K$ 
    unfolding K_def by auto

  have  $Y \ i \subseteq X \ i$  for i
  apply (cases i  $\in I$ ) using ** apply simp unfolding Y_def I_def by auto

```

```

then have  $Pi_E \text{ UNIV } Y \subseteq Pi_E \text{ UNIV } X$  by auto
then have  $Pi_E \text{ UNIV } Y \subseteq U$  using  $H(4)$  by auto

have  $x \in Pi_E \text{ UNIV } Y$ 
  using ** by auto

show  $\exists V \in K. x \in V \wedge V \subseteq U$ 
  using  $\langle Pi_E \text{ UNIV } Y \in K \rangle \langle Pi_E \text{ UNIV } Y \subseteq U \rangle \langle x \in Pi_E \text{ UNIV } Y \rangle$  by auto
next
fix  $U$  assume  $U \in K$ 
show open  $U$ 
  using  $\langle U \in K \rangle$  unfolding open_fun_def K_def by clarify (metis  $\langle U \in K \rangle$ 
i open_PiE open_fun_def)
qed

show ?thesis using i ii iii by auto
qed

instance fun :: (countable, second_countable_topology) second_countable_topology
  apply standard
  using product_topology_countable_basis topological_basis_imp_subbasis by auto

```

4.8.2 The Alexander subbase theorem

theorem *Alexander_subbase:*

assumes X : topology (arbitrary union_of (finite intersection_of ($\lambda x. x \in \mathcal{B}$) relative_to $\bigcup \mathcal{B}$)) = X

and fin: $\bigwedge C. \llbracket C \subseteq \mathcal{B}; \bigcup C = \text{topspace } X \rrbracket \implies \exists C'. \text{finite } C' \wedge C' \subseteq C \wedge \bigcup C' = \text{topspace } X$

shows compact_space X

proof –

have UB : $\bigcup \mathcal{B} = \text{topspace } X$

by (simp flip: X)

have $False$ **if** $\mathcal{U}$: $\forall U \in \mathcal{U}. \text{openin } X \ U$ **and** sub : $\text{topspace } X \subseteq \bigcup \mathcal{U}$

and neg: $\bigwedge \mathcal{F}. \llbracket \mathcal{F} \subseteq \mathcal{U}; \text{finite } \mathcal{F} \rrbracket \implies \neg \text{topspace } X \subseteq \bigcup \mathcal{F}$ **for** $\mathcal{U}$

proof –

define $\mathcal{A}$ **where** $\mathcal{A} \equiv \{C. (\forall U \in C. \text{openin } X \ U) \wedge \text{topspace } X \subseteq \bigcup C \wedge (\forall \mathcal{F}. \text{finite } \mathcal{F} \longrightarrow \mathcal{F} \subseteq C \longrightarrow \sim(\text{topspace } X \subseteq \bigcup \mathcal{F}))\}$

have 1: $\mathcal{A} \neq \{\}$

unfolding $\mathcal{A\_def}$ **using** $sub \ \mathcal{U} \ neg$ **by** force

have 2: $\bigcup \mathcal{C} \in \mathcal{A}$ **if** $\mathcal{C} \neq \{\}$ **and** $\mathcal{C}$: subset.chain $\mathcal{A} \ \mathcal{C}$ **for** $\mathcal{C}$

unfolding $\mathcal{A\_def}$

proof (intro CollectI conjI ballI allI impI notI)

show openin $X \ U$ **if** U : $U \in \bigcup \mathcal{C}$ **for** U

using $U \ \mathcal{C}$ **unfolding** $\mathcal{A\_def}$ subset_chain_def **by** force

have $\mathcal{C} \subseteq \mathcal{A}$

using subset_chain_def $\mathcal{C}$ **by** blast

with that $\mathcal{A\_def}$ **show** UUC : $\text{topspace } X \subseteq \bigcup (\bigcup \mathcal{C})$

by blast

```

show False if finite  $\mathcal{F}$  and  $\mathcal{F} \subseteq \bigcup \mathcal{C}$  and  $\text{topspace } X \subseteq \bigcup \mathcal{F}$  for  $\mathcal{F}$ 
proof -
  obtain  $\mathcal{B}$  where  $\mathcal{B} \in \mathcal{C}$   $\mathcal{F} \subseteq \mathcal{B}$ 
    by (metis Sup_empty  $\mathcal{C}$   $\langle \mathcal{F} \subseteq \bigcup \mathcal{C} \rangle$   $\langle \text{finite } \mathcal{F} \rangle$  UUC empty_subsetI
    finite.emptyI finite_subset_Union_chain neg)
  then show False
    using  $\mathcal{A\_def}$   $\langle \mathcal{C} \subseteq \mathcal{A} \rangle$   $\langle \text{finite } \mathcal{F} \rangle$   $\langle \text{topspace } X \subseteq \bigcup \mathcal{F} \rangle$  by blast
qed
qed
obtain  $\mathcal{K}$  where  $\mathcal{K} \in \mathcal{A}$  and  $\bigwedge X. \llbracket X \in \mathcal{A}; \mathcal{K} \subseteq X \rrbracket \implies X = \mathcal{K}$ 
  using subset_Zorn_nonempty [OF 1 2] by metis
then have *:  $\bigwedge \mathcal{W}. \llbracket \bigwedge W. W \in \mathcal{W} \implies \text{openin } X \ W; \text{topspace } X \subseteq \bigcup \mathcal{W}; \mathcal{K} \subseteq$ 
 $\mathcal{W};$ 
 $\bigwedge \mathcal{F}. \llbracket \text{finite } \mathcal{F}; \mathcal{F} \subseteq \mathcal{W}; \text{topspace } X \subseteq \bigcup \mathcal{F} \rrbracket \implies \text{False} \rrbracket$ 
 $\implies \mathcal{W} = \mathcal{K}$ 
  and ope:  $\forall U \in \mathcal{K}. \text{openin } X \ U$  and top:  $\text{topspace } X \subseteq \bigcup \mathcal{K}$ 
  and non:  $\bigwedge \mathcal{F}. \llbracket \text{finite } \mathcal{F}; \mathcal{F} \subseteq \mathcal{K}; \text{topspace } X \subseteq \bigcup \mathcal{F} \rrbracket \implies \text{False}$ 
  unfolding  $\mathcal{A\_def}$  by simp_all metis+
then obtain  $x$  where  $x \in \text{topspace } X$   $x \notin \bigcup (\mathcal{B} \cap \mathcal{K})$ 
proof -
  have  $\bigcup (\mathcal{B} \cap \mathcal{K}) \neq \bigcup \mathcal{B}$ 
    by (metis  $\langle \bigcup \mathcal{B} = \text{topspace } X \rangle$  fin inf.bounded_iff non order_refl)
  then have  $\exists a. a \notin \bigcup (\mathcal{B} \cap \mathcal{K}) \wedge a \in \bigcup \mathcal{B}$ 
    by blast
  then show ?thesis
    using that by (metis  $UB$ )
qed
obtain  $C$  where  $C: \text{openin } X \ C$   $C \in \mathcal{K}$   $x \in C$ 
  using  $\langle x \in \text{topspace } X \rangle$  ope top by auto
then have  $C \subseteq \text{topspace } X$ 
  by (metis openin_subset)
then have (arbitrary union_of (finite intersection_of ( $\lambda x. x \in \mathcal{B}$ ) relative_to
 $\bigcup \mathcal{B}$ ))  $C$ 
  using openin_subbase  $C$  unfolding  $X$  [symmetric] by blast
moreover have  $C \neq \text{topspace } X$ 
  using  $\langle \mathcal{K} \in \mathcal{A} \rangle$   $\langle C \in \mathcal{K} \rangle$  unfolding  $\mathcal{A\_def}$  by blast
ultimately obtain  $\mathcal{V}$   $W$  where  $W: (\text{finite intersection\_of } (\lambda x. x \in \mathcal{B}) \text{ rela-}$ 
 $\text{tive\_to } \text{topspace } X) \ W$ 
  and  $x \in W$   $W \in \mathcal{V}$   $\bigcup \mathcal{V} \neq \text{topspace } X$   $C = \bigcup \mathcal{V}$ 
  using  $C$  by (auto simp: union_of_def  $UB$ )
then have  $\bigcup \mathcal{V} \subseteq \text{topspace } X$ 
  by (metis  $\langle C \subseteq \text{topspace } X \rangle$ )
then have  $\text{topspace } X \notin \mathcal{V}$ 
  using  $\langle \bigcup \mathcal{V} \neq \text{topspace } X \rangle$  by blast
then obtain  $\mathcal{B}'$  where  $\mathcal{B}': \text{finite } \mathcal{B}'$   $\mathcal{B}' \subseteq \mathcal{B}$   $x \in \bigcap \mathcal{B}'$   $W = \text{topspace } X \cap \bigcap \mathcal{B}'$ 
  using  $W$   $\langle x \in W \rangle$  unfolding relative_to_def intersection_of_def by auto
then have  $\bigcap \mathcal{B}' \subseteq \bigcup \mathcal{B}$ 
  using  $\langle W \in \mathcal{V} \rangle$   $\langle \bigcup \mathcal{V} \neq \text{topspace } X \rangle$   $\langle \bigcup \mathcal{V} \subseteq \text{topspace } X \rangle$  by blast
then have  $\bigcap \mathcal{B}' \subseteq C$ 

```

```

    using  $UB \langle C = \bigcup \mathcal{V} \rangle \langle W = \text{topspace } X \cap \bigcap \mathcal{B}' \rangle \langle W \in \mathcal{V} \rangle$  by auto
  have  $\forall b \in \mathcal{B}'. \exists C'. \text{finite } C' \wedge C' \subseteq \mathcal{K} \wedge \text{topspace } X \subseteq \bigcup (\text{insert } b \ C')$ 
proof
  fix b
  assume  $b \in \mathcal{B}'$ 
  have  $\text{insert } b \ \mathcal{K} = \mathcal{K}$  if neg:  $\neg (\exists C'. \text{finite } C' \wedge C' \subseteq \mathcal{K} \wedge \text{topspace } X \subseteq \bigcup (\text{insert } b \ C'))$ 
proof (rule *)
  show  $\text{openin } X \ W$  if  $W \in \text{insert } b \ \mathcal{K}$  for  $W$ 
  using that
proof
  have  $b \in \mathcal{B}$ 
  using  $\langle b \in \mathcal{B}' \rangle \langle \mathcal{B}' \subseteq \mathcal{B} \rangle$  by blast
  then have  $\exists \mathcal{U}. \text{finite } \mathcal{U} \wedge \mathcal{U} \subseteq \mathcal{B} \wedge \bigcap \mathcal{U} = b$ 
  by (rule_tac  $x=\{b\}$  in exI) auto
  moreover have  $\bigcup \mathcal{B} \cap b = b$ 
  using  $\mathcal{B}'(2) \langle b \in \mathcal{B}' \rangle$  by auto
  ultimately show  $\text{openin } X \ W$  if  $W = b$ 
  using that  $\langle b \in \mathcal{B}' \rangle$ 
  apply (simp add: openin_subbase flip: X)
  apply (auto simp: arbitrary_def intersection_of_def relative_to_def
intro!: union_of_inc)
  done
  show  $\text{openin } X \ W$  if  $W \in \mathcal{K}$ 
  by (simp add:  $\langle W \in \mathcal{K} \rangle$  ope)
qed
next
show  $\text{topspace } X \subseteq \bigcup (\text{insert } b \ \mathcal{K})$ 
using top by auto
next
show False if  $\text{finite } \mathcal{F}$  and  $\mathcal{F} \subseteq \text{insert } b \ \mathcal{K}$   $\text{topspace } X \subseteq \bigcup \mathcal{F}$  for  $\mathcal{F}$ 
proof -
  have  $\text{insert } b \ (\mathcal{F} \cap \mathcal{K}) = \mathcal{F}$ 
  using non that by blast
  then show False
  by (metis Int_lower2 finite_insert neg that(1) that(3))
qed
qed auto
then show  $\exists C'. \text{finite } C' \wedge C' \subseteq \mathcal{K} \wedge \text{topspace } X \subseteq \bigcup (\text{insert } b \ C')$ 
using  $\langle b \in \mathcal{B}' \rangle \langle x \notin \bigcup (\mathcal{B} \cap \mathcal{K}) \rangle \mathcal{B}'$ 
by (metis IntI InterE Union_iff subsetD insertI1)
qed
then obtain  $F$  where  $F: \forall b \in \mathcal{B}'. \text{finite } (F \ b) \wedge F \ b \subseteq \mathcal{K} \wedge \text{topspace } X \subseteq \bigcup (\text{insert } b \ (F \ b))$ 
by metis
let  $?D = \text{insert } C \ (\bigcup (F \ \mathcal{B}'))$ 
show False
proof (rule non)
  have  $\text{topspace } X \subseteq (\bigcap b \in \mathcal{B}'. \bigcup (\text{insert } b \ (F \ b)))$ 

```

```

    using F by (simp add: INT_greatest)
  also have ...  $\subseteq \bigcup ?\mathcal{D}$ 
    using  $\langle \bigcap \mathcal{B}' \subseteq C \rangle$  by force
  finally show topspace  $X \subseteq \bigcup ?\mathcal{D}$  .
  show  $?\mathcal{D} \subseteq \mathcal{K}$ 
    using  $\langle C \in \mathcal{K} \rangle$  F by auto
  show finite  $?\mathcal{D}$ 
    using  $\langle \text{finite } \mathcal{B}' \rangle$  F by auto
qed
qed
then show ?thesis
  by (force simp: compact_space_def compactin_def)
qed

corollary Alexander_subbase_alt:
  assumes  $U \subseteq \bigcup \mathcal{B}$ 
  and fin:  $\bigwedge C. \llbracket C \subseteq \mathcal{B}; U \subseteq \bigcup C \rrbracket \implies \exists C'. \text{finite } C' \wedge C' \subseteq C \wedge U \subseteq \bigcup C'$ 
  and X: topology
    (arbitrary union_of
      (finite intersection_of  $(\lambda x. x \in \mathcal{B})$  relative_to U)) = X
  shows compact_space X
proof -
  have topspace  $X = U$ 
    using X topspace_subbase by fastforce
  have eq:  $\bigcup (\text{Collect } ((\lambda x. x \in \mathcal{B}) \text{ relative\_to } U)) = U$ 
    unfolding relative_to_def
    using  $\langle U \subseteq \bigcup \mathcal{B} \rangle$  by blast
  have *:  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{C} \wedge \bigcup \mathcal{F} = \text{topspace } X$ 
    if  $C \subseteq \text{Collect } ((\lambda x. x \in \mathcal{B}) \text{ relative\_to } \text{topspace } X)$  and UC:  $\bigcup \mathcal{C} = \text{topspace } X$  for C
  proof -
    have  $\mathcal{C} \subseteq (\lambda U. \text{topspace } X \cap U) \text{ ' } \mathcal{B}$ 
      using that by (auto simp: relative_to_def)
    then obtain  $\mathcal{B}'$  where  $\mathcal{B}' \subseteq \mathcal{B}$  and  $\mathcal{B}': \mathcal{C} = (\cap) (\text{topspace } X) \text{ ' } \mathcal{B}'$ 
      by (auto simp: subset_image_iff)
    moreover have  $U \subseteq \bigcup \mathcal{B}'$ 
      using  $\mathcal{B}' \langle \text{topspace } X = U \rangle UC$  by auto
    ultimately obtain  $\mathcal{C}'$  where finite  $\mathcal{C}'$   $\mathcal{C}' \subseteq \mathcal{B}'$   $U \subseteq \bigcup \mathcal{C}'$ 
      using fin [of  $\mathcal{B}'$ ]  $\langle \text{topspace } X = U \rangle \langle U \subseteq \bigcup \mathcal{B} \rangle$  by blast
    then show ?thesis
      unfolding  $\mathcal{B}' \text{ ex\_finite\_subset\_image } \langle \text{topspace } X = U \rangle$  by auto
  qed
qed
show ?thesis
  apply (rule Alexander_subbase [where  $\mathcal{B} = \text{Collect } ((\lambda x. x \in \mathcal{B}) \text{ relative\_to } (\text{topspace } X))$ ])
  apply (simp flip: X)
  apply (metis finite_intersection_of_relative_to eq)
  apply (blast intro: *)

```

done
qed

proposition *continuous_map_componentwise:*

continuous_map X (product_topology Y I) f $\longleftrightarrow$
f ' (topspace X) $\subseteq$ extensional I $\wedge$ ($\forall k \in I$. continuous_map X (Y k) (λx . f x
k))
(is ?lhs $\longleftrightarrow$ _ $\wedge$?rhs)

proof (cases $\forall x \in \text{topspace } X. f x \in \text{extensional } I$)

case True

then have *f ' (topspace X) $\subseteq$ extensional I*

by force

moreover have *?rhs* if *L*: *?lhs*

proof –

have *openin X {x $\in$ topspace X. f x k $\in$ U} if k $\in$ I and openin (Y k) U for*
k U

proof –

have *openin (product_topology Y I) ({Y. Y k $\in$ U} $\cap$ ($\Pi_E i \in I$. topspace (Y*
i)))

apply (simp add: openin_product_topology flip: arbitrary_union_of_relative_to)

apply (simp add: relative_to_def)

using that apply (blast intro: arbitrary_union_of_inc finite_intersection_of_inc)

done

with that have *openin X {x $\in$ topspace X. f x $\in$ ({Y. Y k $\in$ U} $\cap$ ($\Pi_E i \in I$.*
topspace (Y i)))}

using *L* unfolding continuous_map_def by blast

moreover have *{x $\in$ topspace X. f x $\in$ ({Y. Y k $\in$ U} $\cap$ ($\Pi_E i \in I$. topspace*
(Y i)))} = {x $\in$ topspace X. f x k $\in$ U}

using *L* by (auto simp: continuous_map_def)

ultimately show *?thesis*

by metis

qed

with that

show *?thesis*

by (auto simp: continuous_map_def)

qed

moreover have *?lhs* if *?rhs*

proof –

have *1: $\bigwedge x. x \in \text{topspace } X \implies f x \in (\Pi_E i \in I. \text{topspace } (Y i))$*

using that True by (auto simp: continuous_map_def PiE_iff)

have *2: {x $\in$ S. $\exists T \in \mathcal{T}. f x \in T$ } = ($\bigcup T \in \mathcal{T}. \{x \in S. f x \in T\}$) for S $\mathcal{T}$*

by blast

have *3: {x $\in$ S. $\forall U \in \mathcal{U}. f x \in U$ } = ($\bigcap (\text{insert } S ((\lambda U. \{x \in S. f x \in U\})$ ‘
*U))) for S $\mathcal{U}$**

by blast

show *?thesis*

unfolding continuous_map_def openin_product_topology arbitrary_def

proof (clarsimp simp: all_union_of 1 2)

fix $\mathcal{T}$

```

    assume  $\mathcal{T}$ :  $\mathcal{T} \subseteq \text{Collect } (\text{finite\_intersection\_of } (\lambda F. \exists i \ U. F = \{f. f\ i \in U\} \\
\wedge i \in I \wedge \text{openin } (Y\ i) \ U) \\
      \text{relative\_to } (\Pi_{E\ i \in I. \text{topspace } (Y\ i)})) \\
    \text{show } \text{openin } X \ (\bigcup_{T \in \mathcal{T}. \{x \in \text{topspace } X. f\ x \in T\}}) \\
    \text{proof } (\text{rule } \text{openin\_Union}; \text{clarify}) \\
      \text{fix } S\ T \\
      \text{assume } T \in \mathcal{T} \\
      \text{obtain } \mathcal{U} \text{ where } T = (\Pi_{E\ i \in I. \text{topspace } (Y\ i)}) \cap \bigcap \mathcal{U} \text{ and finite } \mathcal{U} \\
      \mathcal{U} \subseteq \{\{f. f\ i \in U\} \mid i \ U. i \in I \wedge \text{openin } (Y\ i) \ U\} \\
      \text{using } \text{subsetD } [OF\ \mathcal{T} \langle T \in \mathcal{T} \rangle] \text{ by } (\text{auto simp: intersection\_of\_def} \\
\text{relative\_to\_def}) \\
      \text{with that show } \text{openin } X \ \{x \in \text{topspace } X. f\ x \in T\} \\
      \text{apply } (\text{simp add: continuous\_map\_def 1 cong: conj\_cong}) \\
      \text{unfolding } \mathfrak{J} \\
      \text{apply } (\text{rule } \text{openin\_Inter}; \text{auto}) \\
      \text{done} \\
    \text{qed} \\
  \text{qed} \\
\text{qed} \\
\text{ultimately show } ?thesis \\
  \text{by } \text{metis} \\
\text{next} \\
  \text{case } False \\
  \text{then show } ?thesis \\
    \text{by } (\text{auto simp: continuous\_map\_def PiE\_def}) \\
\text{qed}$ 
```

```

lemma continuous_map_componentwise_UNIV:
  continuous_map  $X$  (product_topology  $Y$   $UNIV$ )  $f \longleftrightarrow (\forall k. \text{continuous\_map } X \\
(Y\ k) \ (\lambda x. f\ x\ k)) \\
  \text{by } (\text{simp add: continuous\_map\_componentwise})$ 
```

```

lemma continuous_map_product_projection [continuous_intros]:
   $k \in I \implies \text{continuous\_map } (\text{product\_topology } X\ I) \ (X\ k) \ (\lambda x. x\ k) \\
  \text{using } \text{continuous\_map\_componentwise } [\text{of } \text{product\_topology } X\ I\ X\ I\ \text{id}] \text{ by } \text{simp}$ 
```

```

declare continuous_map_from_subtopology [OF continuous_map_product_projection,
continuous_intros]

```

```

proposition open_map_product_projection:
  assumes  $i \in I$ 
  shows open_map (product_topology  $Y\ I$ ) ( $Y\ i$ )  $(\lambda f. f\ i)$ 
  unfolding openin_product_topology_all_union_of_arbitrary_def open_map_def
image_Union
proof clarify
  fix  $\mathcal{V}$ 
  assume  $\mathcal{V}$ :  $\mathcal{V} \subseteq \text{Collect}$ 
    (finite_intersection_of

```



```

      (λF. ∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ openin (Y i) U) relative_to
      topspace (product_topology Y I))
show openin (Y i) (⋃ x ∈ V. (λf. f i) ' x)
proof (rule openin_Union, clarify)
  fix S V
  assume V ∈ V
  obtain F where finite F
  and V: V = (ΠE i ∈ I. topspace (Y i)) ∩ ⋂ F
  and F: F ⊆ {{f. f i ∈ U} | i U. i ∈ I ∧ openin (Y i) U}
  using subsetD [OF V ‹V ∈ V›]
  by (auto simp: intersection_of_def relative_to_def)
show openin (Y i) ((λf. f i) ' V)
proof (subst openin_subopen; clarify)
  fix x f
  assume f ∈ V
  let ?T = {a ∈ topspace (Y i).
    (λj. if j = i then a
      else if j ∈ I then f j else undefined) ∈ (ΠE i ∈ I. topspace (Y i))
    ∩ ⋂ F}
  show ∃ T. openin (Y i) T ∧ f i ∈ T ∧ T ⊆ (λf. f i) ' V
  proof (intro exI conjI)
    show openin (Y i) ?T
    proof (rule openin_continuous_map_preimage)
      have continuous_map (Y i) (Y k) (λx. if k = i then x else f k) if k ∈ I
    for k
    proof (cases k=i)
      case True
      then show ?thesis
      by (metis (mono_tags) continuous_map_id eq_id_iff)
    next
      case False
      then show ?thesis
      by simp (metis IntD1 PiE_iff V ‹f ∈ V› that)
    qed
    then show continuous_map (Y i) (product_topology Y I)
      (λx j. if j = i then x else if j ∈ I then f j else undefined)
    by (auto simp: continuous_map_componentwise assms extensional_def)
  next
    have openin (product_topology Y I) (ΠE i ∈ I. topspace (Y i))
    by (metis openin_topspace topspace_product_topology)
    moreover have openin (product_topology Y I) (⋂ B ∈ F. (ΠE i ∈ I. topspace
      (Y i)) ∩ B)
      if F ≠ {}
    proof -
      show ?thesis
      proof (rule openin_Inter)
        show ⋀ X. X ∈ (⋂) (ΠE i ∈ I. topspace (Y i)) ' F ⇒ openin
          (product_topology Y I) X
        unfolding openin_product_topology relative_to_def

```

```

    apply (clarify intro!: arbitrary_union_of_inc)
    apply (rename_tac F)
    apply (rule_tac x=F in exI)
    using subsetD [OF F]
    apply (force intro: finite_intersection_of_inc)
    done
  qed (use ⟨finite F⟩ ⟨F ≠ {}⟩ in auto)
qed
ultimately show openin (product_topology Y I) (( $\prod_E i \in I$ . topspace (Y
i))  $\cap \bigcap \mathcal{F}$ )
  by (auto simp only: Int_Inter_eq split: if_split)
qed
next
have eqf: ( $\lambda j$ . if  $j = i$  then  $f\ i$  else if  $j \in I$  then  $f\ j$  else undefined) =  $f$ 
  using PiE_arb V ⟨ $f \in V$ ⟩ by force
show  $f\ i \in ?T$ 
  using V assms ⟨ $f \in V$ ⟩ by (auto simp: PiE_iff eqf)
next
show  $?T \subseteq (\lambda f$ .  $f\ i$ ) ‘  $V$ 
  unfolding V by (auto simp: intro!: rev_image_eqI)
qed
qed
qed
qed

lemma retraction_map_product_projection:
  assumes  $i \in I$ 
  shows (retraction_map (product_topology X I) (X i) ( $\lambda x$ .  $x\ i$ )  $\longleftrightarrow$ 
    (topspace (product_topology X I) =  $\{\}$ )  $\longrightarrow$  topspace (X i) =  $\{\}$ )
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    using retraction_imp_surjective_map by force
next
  assume R: ?rhs
  show ?lhs
  proof (cases topspace (product_topology X I) =  $\{\}$ )
    case True
    then show ?thesis
      using R by (auto simp: retraction_map_def retraction_maps_def continuous_map_on_empty)
    next
    case False
    have *:  $\exists g$ . continuous_map (X i) (product_topology X I)  $g \wedge (\forall x \in \text{topspace}$ 
      (X i).  $g\ x\ i = x$ )
    if  $z$ :  $z \in (\prod_E i \in I$ . topspace (X i)) for  $z$ 
  proof -
    have cm: continuous_map (X i) (X j) ( $\lambda x$ . if  $j = i$  then  $x$  else  $z\ j$ ) if  $j \in I$ 

```

```

for j
  using ⟨j ∈ I⟩ z by (case_tac j = i) auto
show ?thesis
  using ⟨i ∈ I⟩ that
  by (rule_tac x=λx j. if j = i then x else z j in exI) (auto simp: continu-
ous_map_componentwise PiE_iff extensional_def cm)
qed
show ?thesis
  using ⟨i ∈ I⟩ False
  by (auto simp: retraction_map_def retraction_maps_def assms continu-
ous_map_product_projection *)
qed
qed

```

4.8.3 Open Pi-sets in the product topology

proposition *openin_PiE_gen:*

$$\text{openin } (\text{product\_topology } X \ I) \ (PiE \ I \ S) \longleftrightarrow$$

$$PiE \ I \ S = \{\} \vee$$

$$\text{finite } \{i \in I. \sim(S \ i = \text{topspace}(X \ i))\} \wedge (\forall i \in I. \text{openin } (X \ i) \ (S \ i))$$

(is ?lhs $\longleftrightarrow$ _ $\vee$?rhs)

proof (cases $PiE \ I \ S = \{\}$)

case False

moreover have ?lhs = ?rhs

proof

assume L: ?lhs

moreover

obtain z where z: z ∈ $PiE \ I \ S$

using False by blast

ultimately obtain U where fin: $\text{finite } \{i \in I. U \ i \neq \text{topspace } (X \ i)\}$

and $PiE \ I \ U \neq \{\}$

and sub: $PiE \ I \ U \subseteq PiE \ I \ S$

by (fastforce simp add: openin_product_topology_alt)

then have *: $\bigwedge i. i \in I \implies U \ i \subseteq S \ i$

by (simp add: subset_PiE)

show ?rhs

proof (intro conjI ballI)

show $\text{finite } \{i \in I. S \ i \neq \text{topspace } (X \ i)\}$

apply (rule finite_subset [OF _ fin], clarify)

using *

by (metis False L openin_subset topspace_product_topology subset_PiE subset_antisym)

next

fix i :: 'a

assume i ∈ I

then show $\text{openin } (X \ i) \ (S \ i)$

using open_map_product_projection [of i I X] L

apply (simp add: open_map_def)

apply (drule_tac x= $PiE \ I \ S$ in spec)

```

    apply (simp add: False image_projection_PiE split: if_split_asm)
  done
qed
next
  assume ?rhs
  then show ?lhs
    apply (simp only: openin_product_topology)
    apply (rule arbitrary_union_of_inc)
    apply (auto simp: product_topology_base_alt)
  done
qed
ultimately show ?thesis
  by simp
qed simp

corollary openin_PiE:
  finite I  $\implies$  openin (product_topology X I) (PiE I S)  $\longleftrightarrow$  PiE I S = {}  $\vee$  ( $\forall i \in I$ . openin (X i) (S i))
  by (simp add: openin_PiE_gen)

proposition compact_space_product_topology:
  compact_space(product_topology X I)  $\longleftrightarrow$ 
  topspace(product_topology X I) = {}  $\vee$  ( $\forall i \in I$ . compact_space(X i))
  (is ?lhs = ?rhs)
proof (cases topspace(product_topology X I) = {})
  case False
  then obtain z where z: z  $\in$  ( $\Pi_E i \in I$ . topspace(X i))
  by auto
  show ?thesis
  proof
    assume L: ?lhs
    show ?rhs
    proof (clarsimp simp add: False compact_space_def)
      fix i
      assume i  $\in$  I
      with L have continuous_map (product_topology X I) (X i) ( $\lambda f$ . f i)
      by (simp add: continuous_map_product_projection)
      moreover
      have  $\bigwedge x. x \in \text{topspace } (X i) \implies x \in (\lambda f. f i) \text{ ' } (\Pi_E i \in I. \text{topspace } (X i))$ 
      using  $\langle i \in I \rangle z$ 
      apply (rule_tac x= $\lambda j. \text{if } j = i \text{ then } x \text{ else if } j \in I \text{ then } z j \text{ else undefined}$  in
      image_eqI, auto)
    done
    then have ( $\lambda f. f i$ ) ' ( $\Pi_E i \in I. \text{topspace } (X i)$ ) = topspace (X i)
    using  $\langle i \in I \rangle z$  by auto
    ultimately show compactin (X i) (topspace (X i))
    by (metis L compact_space_def image_compactin topspace_product_topology)
  qed

```

```

next
  assume R: ?rhs
  show ?lhs
  proof (cases I = {})
    case True
    with R show ?thesis
    by (simp add: compact_space_def)
  next
    case False
    then obtain i where i ∈ I
    by blast
    show ?thesis
    using R
    proof
      assume com [rule_format]: ∀ i ∈ I. compact_space (X i)
      let ?C = {{f. f i ∈ U} | i U. i ∈ I ∧ openin (X i) U}
      show compact_space (product_topology X I)
      proof (rule Alexander_subbase_alt)
        show topspace (product_topology X I) ⊆ ⋃ ?C
        unfolding topspace_product_topology using ⟨i ∈ I⟩ by blast
      next
        fix C
        assume Csub: C ⊆ ?C and UC: topspace (product_topology X I) ⊆ ⋃ C
        define D where D ≡ λi. {U. openin (X i) U ∧ {f. f i ∈ U} ∈ C}
        show ∃ C'. finite C' ∧ C' ⊆ C ∧ topspace (product_topology X I) ⊆ ⋃ C'
        proof (cases ∃ i. i ∈ I ∧ topspace (X i) ⊆ ⋃ (D i))
          case True
          then obtain i where i ∈ I
          and i: topspace (X i) ⊆ ⋃ (D i)
          unfolding D_def by blast
          then have *: ⋀U. [Ball U (openin (X i)); topspace (X i) ⊆ ⋃ U] ⇒
            ∃F. finite F ∧ F ⊆ U ∧ topspace (X i) ⊆ ⋃ F
          using com [OF ⟨i ∈ I⟩] by (auto simp: compact_space_def com-
            pactin_def)
          have topspace (X i) ⊆ ⋃ (D i)
          using i by auto
          with * obtain F where finite F ∧ F ⊆ (D i) ∧ topspace (X i) ⊆ ⋃ F
          unfolding D_def by fastforce
          with ⟨i ∈ I⟩ show ?thesis
          unfolding D_def
          by (rule_tac x=(λU. {x. x i ∈ U}) 'F in exI) auto
        next
          case False
          then have ∀ i ∈ I. ∃ y. y ∈ topspace (X i) ∧ y ∉ ⋃ (D i)
          by force
          then obtain g where g: ⋀i. i ∈ I ⇒ g i ∈ topspace (X i) ∧ g i ∉
            ⋃ (D i)
          by metis
          then have (λi. if i ∈ I then g i else undefined) ∈ topspace (product_topology

```

```

X I)
  by (simp add: PiE_I)
  moreover have  $(\lambda i. \text{if } i \in I \text{ then } g \ i \text{ else undefined}) \notin \bigcup C$ 
  using Csub g unfolding D_def by force
  ultimately show ?thesis
  using UC by blast
qed
qed (simp add: product_topology)
qed (simp add: compact_space_topspace_empty)
qed
qed
qed (simp add: compact_space_topspace_empty)

corollary compactin_PiE:
  compactin (product_topology X I) (PiE I S)  $\longleftrightarrow$ 
  PiE I S =  $\{\}$   $\vee$   $(\forall i \in I. \text{compactin } (X \ i) \ (S \ i))$ 
by (auto simp: compactin_subspace subtopology_PiE subset_PiE compact_space_product_topology
  PiE_eq_empty_iff)

lemma in_product_topology_closure_of:
  z  $\in$  (product_topology X I) closure_of S
   $\implies i \in I \implies z \ i \in ((X \ i) \text{ closure\_of } ((\lambda x. x \ i) \text{ ` } S))$ 
using continuous_map_product_projection
by (force simp: continuous_map_eq_image_closure_subset image_subset_iff)

lemma homeomorphic_space_singleton_product:
  product_topology X {k} homeomorphic_space (X k)
unfolding homeomorphic_space
apply (rule_tac x= $\lambda x. x \ k$  in exI)
apply (rule bijective_open_imp_homeomorphic_map)
  apply (simp_all add: continuous_map_product_projection open_map_product_projection)
unfolding PiE_over_singleton_iff
  apply (auto simp: image_iff inj_on_def)
done

```

4.8.4 Relationship with connected spaces, paths, etc.

```

proposition connected_space_product_topology:
  connected_space(product_topology X I)  $\longleftrightarrow$ 
   $(\Pi_{E \ i \in I. \text{topspace } (X \ i)) = \{\}$   $\vee$   $(\forall i \in I. \text{connected\_space}(X \ i))$ 
  (is ?lhs  $\longleftrightarrow$  ?eq  $\vee$  ?rhs)
proof (cases ?eq)
  case False
  moreover have ?lhs = ?rhs
  proof
    assume ?lhs
    moreover
    have connectedin(X i) (topspace(X i))
    if i  $\in$  I and ci: connectedin(product_topology X I) (topspace(product_topology

```

```

X I)) for i
proof -
  have cm: continuous_map (product_topology X I) (X i) (λf. f i)
    by (simp add: ⟨i ∈ I⟩ continuous_map_product_projection)
  show ?thesis
    using connectedin_continuous_map_image [OF cm ci] ⟨i ∈ I⟩
    by (simp add: False image_projection_PiE)
qed
ultimately show ?rhs
  by (meson connectedin_topspace)
next
assume cs [rule_format]: ?rhs
have False
  if disj:  $U \cap V = \{\}$  and subUV:  $(\Pi_E i \in I. \text{topspace } (X i)) \subseteq U \cup V$ 
  and U:  $U \text{ openin } (\text{product\_topology } X I)$ 
  and V:  $V \text{ openin } (\text{product\_topology } X I)$ 
  and  $U \neq \{\}$   $V \neq \{\}$ 
  for U V
proof -
  obtain f where  $f \in U$ 
    using ⟨ $U \neq \{\}$ ⟩ by blast
  then have f:  $f \in (\Pi_E i \in I. \text{topspace } (X i))$ 
    using U openin_subset by fastforce
  have  $U \subseteq \text{topspace}(\text{product\_topology } X I)$   $V \subseteq \text{topspace}(\text{product\_topology } X$ 
I)
    using U V openin_subset by blast+
  moreover have  $(\Pi_E i \in I. \text{topspace } (X i)) \subseteq U$ 
  proof -
    obtain C where (finite_intersection_of (λF.  $\exists i U. F = \{x. x i \in U\} \wedge i$ 
 $\in I \wedge \text{openin } (X i) U$ ) relative_to
      ( $\Pi_E i \in I. \text{topspace } (X i)$ )) C  $C \subseteq U$   $f \in C$ 
    using U ⟨ $f \in U$ ⟩ unfolding openin_product_topology_union_of_def by
auto
    then obtain  $\mathcal{T}$  where finite  $\mathcal{T}$ 
    and t:  $\bigwedge C. C \in \mathcal{T} \implies \exists i u. (i \in I \wedge \text{openin } (X i) u) \wedge C = \{x. x i \in u\}$ 
    and subU:  $\text{topspace } (\text{product\_topology } X I) \cap \bigcap \mathcal{T} \subseteq U$ 
    and ftop:  $f \in \text{topspace } (\text{product\_topology } X I)$ 
    and fint:  $f \in \bigcap \mathcal{T}$ 
    by (fastforce simp: relative_to_def intersection_of_def subset_iff)
    let ?L =  $\bigcup C \in \mathcal{T}. \{i. (\lambda x. x i) \text{ ' } C \subset \text{topspace } (X i)\}$ 
    obtain L where finite L
    and L:  $\bigwedge i U. \llbracket i \in I; \text{openin } (X i) U; U \subset \text{topspace}(X i); \{x. x i \in U\} \in$ 
 $\mathcal{T} \rrbracket \implies i \in L$ 
  proof
    show finite ?L
    proof (rule finite_Union)
      fix M
      assume  $M \in (\lambda C. \{i. (\lambda x. x i) \text{ ' } C \subset \text{topspace } (X i)\}) \text{ ' } \mathcal{T}$ 
      then obtain C where  $C \in \mathcal{T}$  and  $C: M = \{i. (\lambda x. x i) \text{ ' } C \subset \text{topspace}$ 

```

```

(X i)}
  by blast
  then obtain j V where j ∈ I and ope: openin (X j) V and Ceq: C =
{x. x j ∈ V}
    using t by meson
  then have f j ∈ V
    using ⟨C ∈ T⟩ fint by force
  then have (λx. x k) ‘ {x. x j ∈ V} = UNIV if k ≠ j for k
    using that
  apply (clarsimp simp add: set_eq_iff)
  apply (rule_tac x=λm. if m = k then x else f m in image_eqI, auto)
  done
  then have {i. (λx. x i) ‘ C ⊂ topspace (X i)} ⊆ {j}
    using Ceq by auto
  then show finite M
    using C finite_subset by fastforce
qed (use ⟨finite T⟩ in blast)
next
fix i U
  assume i ∈ I and ope: openin (X i) U and psub: U ⊂ topspace (X i)
and int: {x. x i ∈ U} ∈ T
  then show i ∈ ?L
    by (rule_tac a={x. x i ∈ U} in UN_I) (force+)
qed
show ?thesis
proof
fix h
  assume h: h ∈ (ΠE i∈I. topspace (X i))
  define g where g ≡ λi. if i ∈ L then f i else h i
  have gin: g ∈ (ΠE i∈I. topspace (X i))
    unfolding g_def using f h by auto
  moreover have g ∈ X if X ∈ T for X
    using fint openin_subset t [OF that] L g_def h that by fastforce
  ultimately have g ∈ U
    using subU by auto
  have h ∈ U if finite M h ∈ PiE I (topspace ∘ X) {i ∈ I. h i ≠ g i} ⊆ M
for M h
    using that
  proof (induction arbitrary: h)
    case empty
    then show ?case
      using PiE_ext ⟨g ∈ U⟩ gin by force
  next
    case (insert i M)
    define f where f ≡ λj. if j = i then g i else h j
    have fin: f ∈ PiE I (topspace ∘ X)
      unfolding f_def using gin insert.prem1 by auto
    have subM: {j ∈ I. f j ≠ g j} ⊆ M
      unfolding f_def using insert.prem2 by auto

```



```

have f ∈ U
  using insert.IH [OF fin subM] .
show ?case
proof (cases h ∈ V)
  case True
  show ?thesis
  proof (cases i ∈ I)
    case True
    let ?U = {x ∈ topspace(X i). (λj. if j = i then x else h j) ∈ U}
    let ?V = {x ∈ topspace(X i). (λj. if j = i then x else h j) ∈ V}
    have False
    proof (rule connected_spaceD [OF cs [OF ⟨i ∈ I⟩]])
      have ∧k. k ∈ I ⇒ continuous_map (X i) (X k) (λx. if k = i then
x else h k)
      using continuous_map_eq_topcontinuous_at insert.premis(1)
topcontinuous_at_def by fastforce
      then have cm: continuous_map (X i) (product_topology X I) (λx
j. if j = i then x else h j)
      using ⟨i ∈ I⟩ insert.premis(1)
      by (auto simp: continuous_map_componentwise extensional_def)
    show openin (X i) ?U
      by (rule openin_continuous_map_preimage [OF cm U])
    show openin (X i) ?V
      by (rule openin_continuous_map_preimage [OF cm V])
    show topspace (X i) ⊆ ?U ∪ ?V
  proof clarsimp
    fix x
    assume x ∈ topspace (X i) and (λj. if j = i then x else h j) ∉ V
    with True subUV ⟨h ∈ PiE I (topspace ∘ X)⟩
    show (λj. if j = i then x else h j) ∈ U
      by (drule_tac c=(λj. if j = i then x else h j) in subsetD) auto
  qed
  show ?U ∩ ?V = {}
    using disj by blast
  show ?U ≠ {}
    using ⟨U ≠ {}⟩ f_def
  proof -
    have (λj. if j = i then g i else h j) ∈ U
      using ⟨f ∈ U⟩ f_def by blast
    moreover have f i ∈ topspace (X i)
      by (metis PiE_iff True comp_apply fin)
    ultimately have ∃ b. b ∈ topspace (X i) ∧ (λa. if a = i then b
else h a) ∈ U
      using f_def by auto
    then show ?thesis
      by blast
  qed
  have (λj. if j = i then h i else h j) = h
    by force

```

```

    moreover have  $h \ i \in \text{topspace } (X \ i)$ 
      using  $\text{True insert.prem}(1)$  by auto
    ultimately show  $?V \neq \{\}$ 
      using  $\langle h \in V \rangle$  by force
  qed
  then show  $?thesis \ ..$ 
next
  case  $\text{False}$ 
  show  $?thesis$ 
  proof (cases  $h = f$ )
    case  $\text{True}$ 
    show  $?thesis$ 
      by (rule  $\text{insert.IH } [OF \ \text{insert.prem}(1)]$ ) (simp add:  $\text{True subM}$ )
  next
    case  $\text{False}$ 
    then show  $?thesis$ 
      using  $\text{gin insert.prem}(1) \ \langle i \notin I \rangle$  unfolding  $f\_def$  by fastforce
  qed
qed
next
  case  $\text{False}$ 
  then show  $?thesis$ 
    using  $\text{subUV insert.prem}(1)$  by auto
  qed
qed
then show  $h \in U$ 
  unfolding  $g\_def$  using  $\text{PiE\_iff } \langle \text{finite } L \rangle \ h$  by fastforce
qed
qed
ultimately show  $?thesis$ 
  using  $\text{disj inf\_absorb2 } \langle V \neq \{\} \rangle$  by fastforce
qed
then show  $?lhs$ 
  unfolding  $\text{connected\_space\_def}$ 
  by auto
qed
ultimately show  $?thesis$ 
  by simp
qed (simp add:  $\text{connected\_space\_topspace\_empty}$ )

```

lemma connectedin_PiE :

```

   $\text{connectedin } (\text{product\_topology } X \ I) \ (\text{PiE } I \ S) \longleftrightarrow$ 
     $\text{PiE } I \ S = \{\} \vee (\forall i \in I. \text{connectedin } (X \ i) \ (S \ i))$ 
  by (fastforce simp add:  $\text{connectedin\_def subtopology\_PiE connected\_space\_product\_topology}$ 
     $\text{subset\_PiE PiE\_eq\_empty\_iff}$ )

```

lemma $\text{path\_connected\_space\_product\_topology}$:

```

   $\text{path\_connected\_space}(\text{product\_topology } X \ I) \longleftrightarrow$ 

```

```

    topspace(product_topology X I) = {} ∨ (∀ i ∈ I. path_connected_space(X i))
  (is ?lhs ⟷ ?eq ∨ ?rhs)
proof (cases ?eq)
  case False
  moreover have ?lhs = ?rhs
  proof
    assume L: ?lhs
    show ?rhs
    proof (clarsimp simp flip: path_connectedin_topospace)
      fix i :: 'a
      assume i ∈ I
      have cm: continuous_map (product_topology X I) (X i) (λf. f i)
        by (simp add: ⟨i ∈ I⟩ continuous_map_product_projection)
      show path_connectedin (X i) (topspace (X i))
        using path_connectedin_continuous_map_image [OF cm L [unfolded
path_connectedin_topospace [symmetric]]]
        by (metis ⟨i ∈ I⟩ False retraction_imp_surjective_map retraction_map_product_projection)
      qed
    next
      assume R [rule_format]: ?rhs
      show ?lhs
      unfolding path_connected_space_def topspace_product_topology
      proof clarify
        fix x y
        assume x: x ∈ (ΠE i ∈ I. topspace (X i)) and y: y ∈ (ΠE i ∈ I. topspace (X
i))
        have ∀ i. ∃ g. i ∈ I ⟶ pathin (X i) g ∧ g 0 = x i ∧ g 1 = y i
          using PiE_mem R path_connected_space_def x y by force
        then obtain g where g: ∧ i. i ∈ I ⟶ pathin (X i) (g i) ∧ g i 0 = x i ∧ g
i 1 = y i
          by metis
        with x y show ∃ g. pathin (product_topology X I) g ∧ g 0 = x ∧ g 1 = y
          apply (rule_tac x=λa. λi ∈ I. g i a in exI)
          apply (force simp: pathin_def continuous_map_componentwise)
          done
        qed
      qed
      ultimately show ?thesis
      by simp
    qed (simp add: path_connected_space_topospace_empty)

lemma path_connectedin_PiE:
  path_connectedin (product_topology X I) (PiE I S) ⟷
  PiE I S = {} ∨ (∀ i ∈ I. path_connectedin (X i) (S i))
  by (fastforce simp add: path_connectedin_def subtopology_PiE path_connected_space_product_topology
subset_PiE PiE_eq_empty_iff topspace_subtopology_subset)

```

4.8.5 Projections from a function topology to a component

lemma *quotient_map_product_projection*:

assumes $i \in I$

shows $\text{quotient\_map}(\text{product\_topology } X \ I) \ (X \ i) \ (\lambda x. x \ i) \longleftrightarrow$
 $(\text{topspace}(\text{product\_topology } X \ I) = \{\} \longrightarrow \text{topspace}(X \ i) = \{\})$
(is $?lhs = ?rhs)$

proof

assume $?lhs$ **with** *assms* **show** $?rhs$

by (*auto simp: continuous_open_quotient_map open_map_product_projection*)

next

assume $?rhs$ **with** *assms* **show** $?lhs$

by (*auto simp: Abstract_Topology.retraction_imp_quotient_map retraction_map_product_projection*)

qed

lemma *product_topology_homeomorphic_component*:

assumes $i \in I \wedge j. \llbracket j \in I; j \neq i \rrbracket \implies \exists a. \text{topspace}(X \ j) = \{a\}$

shows $\text{product\_topology } X \ I \ \text{homeomorphic\_space } (X \ i)$

proof –

have $\text{quotient\_map} \ (\text{product\_topology } X \ I) \ (X \ i) \ (\lambda x. x \ i)$

using *assms* **by** (*force simp add: quotient_map_product_projection PiE_eq_empty_iff*)

moreover

have $\text{inj\_on} \ (\lambda x. x \ i) \ (\Pi_E \ i \in I. \text{topspace } (X \ i))$

using *assms* **by** (*auto simp: inj_on_def PiE_iff*) (*metis extensionalityI singletonD*)

ultimately show $?thesis$

unfolding *homeomorphic_space_def*

by (*rule_tac x = $\lambda x. x \ i$ in exI*) (*simp add: homeomorphic_map_def flip: homeomorphic_map_maps*)

qed

lemma *topological_property_of_product_component*:

assumes *major*: $P \ (\text{product\_topology } X \ I)$

and *minor*: $\bigwedge z \ i. \llbracket z \in (\Pi_E \ i \in I. \text{topspace}(X \ i)); P(\text{product\_topology } X \ I); i \in I \rrbracket$

$\implies P(\text{subtopology} \ (\text{product\_topology } X \ I) \ (PiE \ I \ (\lambda j. \text{if } j = i \text{ then } \text{topspace}(X \ i) \text{ else } \{z \ j\})))$

(is $\bigwedge z \ i. \llbracket \_; \_; \_ \rrbracket \implies P \ (?SX \ z \ i)$)

and $PQ: \bigwedge X \ X'. X \ \text{homeomorphic\_space } X' \implies (P \ X \longleftrightarrow Q \ X')$

shows $(\Pi_E \ i \in I. \text{topspace}(X \ i)) = \{\} \vee (\forall i \in I. Q(X \ i))$

proof –

have $Q(X \ i) \text{ if } (\Pi_E \ i \in I. \text{topspace}(X \ i)) \neq \{\} \ i \in I \text{ for } i$

proof –

from that obtain f **where** $f: f \in (\Pi_E \ i \in I. \text{topspace } (X \ i))$

by *force*

have $?SX \ f \ i \ \text{homeomorphic\_space } X \ i$

apply (*simp add: subtopology_PiE*)

using *product_topology_homeomorphic_component* [*OF* $\langle i \in I \rangle$, *of* $\lambda j. \text{subtopology } (X \ j) \ (\text{if } j = i \text{ then } \text{topspace } (X \ i) \text{ else } \{f \ j\})$]

using f **by** *fastforce*

```

    then show ?thesis
      using minor [OF f major ‹i ∈ I›] PQ by auto
  qed
  then show ?thesis by metis
qed
end

```

4.9 Bounded Linear Function

theory *Bounded_Linear_Function*

imports

Topology_Euclidean_Space
Operator_Norm
Uniform_Limit
Function_Topology

begin

lemma *onorm_componentwise*:

assumes *bounded_linear* *f*

shows $\text{onorm } f \leq (\sum i \in \text{Basis}. \text{norm } (f i))$

proof –

```

{
  fix i::'a
  assume i ∈ Basis
  hence onorm (λx. (x • i) *R f i) ≤ onorm (λx. (x • i)) * norm (f i)
    by (auto intro!: onorm_scaleR_left_lemma bounded_linear_inner_left)
  also have ... ≤ norm i * norm (f i)
    by (rule mult_right_mono)
    (auto simp: ac_simps Cauchy_Schwarz_ineq2 intro!: onorm_le)
  finally have onorm (λx. (x • i) *R f i) ≤ norm (f i) using ‹i ∈ Basis›
    by simp
} hence onorm (λx. ∑ i ∈ Basis. (x • i) *R f i) ≤ (∑ i ∈ Basis. norm (f i))
  by (auto intro!: order_trans[OF onorm_sum_le] bounded_linear_scaleR_const
    sum_mono bounded_linear_inner_left)
also have (λx. ∑ i ∈ Basis. (x • i) *R f i) = (λx. f (∑ i ∈ Basis. (x • i) *R i))
  by (simp add: linear_sum bounded_linear.linear assms linear_simps)
also have ... = f
  by (simp add: euclidean_representation)
finally show ?thesis .

```

qed

lemmas *onorm_componentwise_le* = *order_trans*[*OF onorm_componentwise*]

4.9.1 Intro rules for *bounded_linear*

named_theorems *bounded_linear_intros*

```

lemma onorm_inner_left:
  assumes bounded_linear r
  shows onorm ( $\lambda x. r\ x \cdot f$ )  $\leq$  onorm r * norm f
proof (rule onorm_bound)
  fix x
  have norm (r x  $\cdot$  f)  $\leq$  norm (r x) * norm f
    by (simp add: Cauchy_Schwarz_ineq2)
  also have ...  $\leq$  onorm r * norm x * norm f
    by (intro mult_right_mono onorm assms norm_ge_zero)
  finally show norm (r x  $\cdot$  f)  $\leq$  onorm r * norm f * norm x
    by (simp add: ac_simps)
qed (intro mult_nonneg_nonneg norm_ge_zero onorm_pos_le assms)

```

```

lemma onorm_inner_right:
  assumes bounded_linear r
  shows onorm ( $\lambda x. f \cdot r\ x$ )  $\leq$  norm f * onorm r
  apply (subst inner_commute)
  apply (rule onorm_inner_left[OF assms, THEN order_trans])
  apply simp
  done

```

```

lemmas [bounded_linear_intros] =
  bounded_linear_zero
  bounded_linear_add
  bounded_linear_const_mult
  bounded_linear_mult_const
  bounded_linear_scaleR_const
  bounded_linear_const_scaleR
  bounded_linear_ident
  bounded_linear_sum
  bounded_linear_Pair
  bounded_linear_sub
  bounded_linearfst_comp
  bounded_linearsnd_comp
  bounded_linear_inner_left_comp
  bounded_linear_inner_right_comp

```

4.9.2 declaration of derivative/continuous/tendsto introduction rules for bounded linear functions

```

attribute_setup bounded_linear =
  <Scan.succeed (Thm.declaration_attribute (fn thm =>
    fold (fn (r, s) => Named_Theorems.add_thm s (thm RS r))
    [
      (@{thm bounded_linear.has_derivative}, named_theorems <derivative_intros>),
      (@{thm bounded_linear.tendsto}, named_theorems <tendsto_intros>),
      (@{thm bounded_linear.continuous}, named_theorems <continuous_intros>),
      (@{thm bounded_linear.continuous_on}, named_theorems <continuous_intros>),
      (@{thm bounded_linear.uniformly_continuous_on}, named_theorems <continuous_intros>),
    ]
  )

```

```

    (@{thm bounded_linear_compose}, named_theorems ⟨bounded_linear_intros⟩)
  )))
attribute_setup bounded_bilinear =
  ⟨Scan.succeed (Thm.declaration_attribute (fn thm =>
    fold (fn (r, s) => Named_Theorems.add_thm s (thm RS r))
      [
        (@{thm bounded_bilinear.FDERIV}, named_theorems ⟨derivative_intros⟩),
        (@{thm bounded_bilinear.tendsto}, named_theorems ⟨tendsto_intros⟩),
        (@{thm bounded_bilinear.continuous}, named_theorems ⟨continuous_intros⟩),
        (@{thm bounded_bilinear.continuous_on}, named_theorems ⟨continuous_intros⟩),
        (@{thm bounded_linear_compose[OF bounded_bilinear.bounded_linear_left]},
          named_theorems ⟨bounded_linear_intros⟩),
        (@{thm bounded_linear_compose[OF bounded_bilinear.bounded_linear_right]},
          named_theorems ⟨bounded_linear_intros⟩),
        (@{thm bounded_linear.uniformly_continuous_on[OF bounded_bilinear.bounded_linear_left]},
          named_theorems ⟨continuous_intros⟩),
        (@{thm bounded_linear.uniformly_continuous_on[OF bounded_bilinear.bounded_linear_right]},
          named_theorems ⟨continuous_intros⟩)
      ]))
  ⟩

```

4.9.3 Type of bounded linear functions

```

typedef (overloaded) ('a, 'b) blinfun (( $\_ \Rightarrow_L \_$ ) [22, 21] 21) =
  {f::'a::real_normed_vector $\Rightarrow$ 'b::real_normed_vector. bounded_linear f}
morphisms blinfun_apply Blinfun
by (blast intro: bounded_linear_intros)

```

```

declare [[coercion
  blinfun_apply :: ('a::real_normed_vector  $\Rightarrow_L$  'b::real_normed_vector)  $\Rightarrow$  'a  $\Rightarrow$ 
  'b]]

```

```

lemma bounded_linear_blinfun_apply[bounded_linear_intros]:
  bounded_linear g  $\implies$  bounded_linear ( $\lambda x$ . blinfun_apply f (g x))
by (metis blinfun_apply mem_Collect_eq bounded_linear_compose)

```

```

setup_lifting type_definition_blinfun

```

```

lemma blinfun_eqI: ( $\bigwedge i$ . blinfun_apply x i = blinfun_apply y i)  $\implies$  x = y
by transfer auto

```

```

lemma bounded_linear_Blinfun_apply: bounded_linear f  $\implies$  blinfun_apply (Blinfun
f) = f
by (auto simp: Blinfun_inverse)

```

4.9.4 Type class instantiations

```

instantiation blinfun :: (real_normed_vector, real_normed_vector) real_normed_vector
begin

```

lift_definition *norm_blinfun* :: $'a \Rightarrow_L 'b \Rightarrow \text{real}$ **is** *onorm* .

lift_definition *minus_blinfun* :: $'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b$
is $\lambda f g x. f x - g x$
by (*rule bounded_linear_sub*)

definition *dist_blinfun* :: $'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b \Rightarrow \text{real}$
where *dist_blinfun* *a b* = *norm* (*a* - *b*)

definition [*code del*]:
(*uniformity* :: $(('a \Rightarrow_L 'b) \times ('a \Rightarrow_L 'b)) \text{ filter}) = (\text{INF } e \in \{0 < ..\}. \text{principal } \{(x, y). \text{dist } x y < e\})$

definition *open_blinfun* :: $('a \Rightarrow_L 'b) \text{ set} \Rightarrow \text{bool}$
where [*code del*]: *open_blinfun* *S* = $(\forall x \in S. \forall_F (x', y) \text{ in } \text{uniformity}. x' = x \longrightarrow y \in S)$

lift_definition *uminus_blinfun* :: $'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b$ **is** $\lambda f x. - f x$
by (*rule bounded_linear_minus*)

lift_definition *zero_blinfun* :: $'a \Rightarrow_L 'b$ **is** $\lambda x. 0$
by (*rule bounded_linear_zero*)

lift_definition *plus_blinfun* :: $'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b$
is $\lambda f g x. f x + g x$
by (*metis bounded_linear_add*)

lift_definition *scaleR_blinfun* :: $\text{real} \Rightarrow 'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b$ **is** $\lambda r f x. r *_R f x$
by (*metis bounded_linear_compose bounded_linear_scaleR_right*)

definition *sgn_blinfun* :: $'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b$
where *sgn_blinfun* *x* = *scaleR* (*inverse* (*norm* *x*)) *x*

instance

apply *standard*

unfolding *dist_blinfun_def open_blinfun_def sgn_blinfun_def uniformity_blinfun_def*

apply (*rule refl* | (*transfer*, *force simp: onorm_triangle onorm_scaleR onorm_eq_0 algebra_simps*))+

done

end

declare *uniformity_Abort* [**where** $'a = ('a :: \text{real\_normed\_vector}) \Rightarrow_L ('b :: \text{real\_normed\_vector})$,
code]

lemma *norm_blinfun_eqI*:

assumes $n \leq \text{norm } (\text{blinfun\_apply } f x) / \text{norm } x$

assumes $\bigwedge x. \text{norm } (\text{blinfun\_apply } f x) \leq n * \text{norm } x$

assumes $0 \leq n$


```

shows norm f = n
by (auto simp: norm_blinfun_def
  intro!: antisym onorm_bound assms order_trans[OF le_onorm]
  bounded_linear_intros)

lemma norm_blinfun: norm (blinfun_apply f x) ≤ norm f * norm x
by transfer (rule onorm)

lemma norm_blinfun_bound: 0 ≤ b ⇒ (∧x. norm (blinfun_apply f x) ≤ b *
norm x) ⇒ norm f ≤ b
by transfer (rule onorm_bound)

lemma bounded_bilinear_blinfun_apply[bounded_bilinear]: bounded_bilinear blin-
fun_apply
proof
  fix f g::'a ⇒L 'b and a b::'a and r::real
  show (f + g) a = f a + g a (r *R f) a = r *R f a
    by (transfer, simp)+
  interpret bounded_linear f for f::'a ⇒L 'b
    by (auto intro!: bounded_linear_intros)
  show f (a + b) = f a + f b f (r *R a) = r *R f a
    by (simp_all add: add scaleR)
  show ∃ K. ∀ a b. norm (blinfun_apply a b) ≤ norm a * norm b * K
    by (auto intro!: exI[where x=1] norm_blinfun)
qed

interpretation blinfun: bounded_bilinear blinfun_apply
by (rule bounded_bilinear_blinfun_apply)

lemmas bounded_linear_apply_blinfun[intro, simp] = blinfun.bounded_linear_left

declare blinfun.zero_left [simp] blinfun.zero_right [simp]

context bounded_bilinear
begin

named_theorems bilinear_simps

lemmas [bilinear_simps] =
  add_left
  add_right
  diff_left
  diff_right
  minus_left
  minus_right
  scaleR_left
  scaleR_right
  zero_left

```

$zero\_right$
 sum_left
 sum_right

end

```

instance blinfun :: (real_normed_vector, banach) banach
proof
  fix X::nat => 'a =>_L 'b
  assume Cauchy X
  {
    fix x::'a
    {
      fix x::'a
      assume norm x ≤ 1
      have Cauchy (λn. X n x)
      proof (rule CauchyI)
        fix e::real
        assume 0 < e
        from CauchyD[OF ⟨Cauchy X⟩ ⟨0 < e⟩] obtain M
          where M: ∧m n. m ≥ M => n ≥ M => norm (X m - X n) < e
        by auto
        show ∃ M. ∀ m ≥ M. ∀ n ≥ M. norm (X m x - X n x) < e
        proof (safe intro!: exI[where x=M])
          fix m n
          assume le: M ≤ m M ≤ n
          have norm (X m x - X n x) = norm ((X m - X n) x)
            by (simp add: blinfun.bilinear_simps)
          also have ... ≤ norm (X m - X n) * norm x
            by (rule norm_blinfun)
          also have ... ≤ norm (X m - X n) * 1
            using ⟨norm x ≤ 1⟩ norm_ge_zero by (rule mult_left_mono)
          also have ... = norm (X m - X n) by simp
          also have ... < e using le by fact
          finally show norm (X m x - X n x) < e .
        qed
        qed
      hence convergent (λn. X n x)
        by (metis Cauchy_convergent_iff)
    } note convergent_norm1 = this
    define y where y = x /_R norm x
    have y: norm y ≤ 1 and xy: x = norm x *_R y
      by (simp_all add: y_def inverse_eq_divide)
    have convergent (λn. norm x *_R X n y)
      by (intro bounded_bilinear.convergent[OF bounded_bilinear_scaleR] conver-
gent_const
      convergent_norm1 y)
    also have (λn. norm x *_R X n y) = (λn. X n x)

```

```

    by (subst xy) (simp add: blinfun.bilinear_simps)
  finally have convergent ( $\lambda n. X\ n\ x$ ) .
}
then obtain v where v:  $\bigwedge x. (\lambda n. X\ n\ x) \longrightarrow v\ x$ 
  unfolding convergent_def
  by metis

have Cauchy ( $\lambda n. norm\ (X\ n)$ )
proof (rule CauchyI)
  fix e::real
  assume e > 0
  from CauchyD[OF  $\langle Cauchy\ X \rangle \langle 0 < e \rangle$ ] obtain M
    where M:  $\bigwedge m\ n. m \geq M \implies n \geq M \implies norm\ (X\ m - X\ n) < e$ 
    by auto
  show  $\exists M. \forall m \geq M. \forall n \geq M. norm\ (norm\ (X\ m) - norm\ (X\ n)) < e$ 
  proof (safe intro!: exI[where x=M])
    fix m n assume mn:  $m \geq M\ n \geq M$ 
    have  $norm\ (norm\ (X\ m) - norm\ (X\ n)) \leq norm\ (X\ m - X\ n)$ 
      by (metis norm_triangle_ineq3 real_norm_def)
    also have  $\dots < e$  using mn by fact
    finally show  $norm\ (norm\ (X\ m) - norm\ (X\ n)) < e$  .
  qed
qed
then obtain K where K: ( $\lambda n. norm\ (X\ n)$ )  $\longrightarrow K$ 
  unfolding Cauchy_convergent_iff convergent_def
  by metis

have bounded_linear v
proof
  fix x y and r::real
  from tendsto_add[OF v[of x] v [of y]] v[of x + y, unfolded blinfun.bilinear_simps]
    tendsto_scaleR[OF tendsto_const[of r] v[of x]] v[of r *R x, unfolded blinfun.bilinear_simps]
  show  $v\ (x + y) = v\ x + v\ y$ 
  by (metis (poly_guards_query) LIMSEQ_unique)
  show  $\exists K. \forall x. norm\ (v\ x) \leq norm\ x * K$ 
  proof (safe intro!: exI[where x=K])
    fix x
    have  $norm\ (v\ x) \leq K * norm\ x$ 
    by (rule tendsto_le[OF tendsto_mult[OF K tendsto_const] tendsto_norm[OF v]])
    (auto simp: norm_blinfun)
    thus  $norm\ (v\ x) \leq norm\ x * K$ 
    by (simp add: ac_simps)
  qed
qed
hence Bv:  $\bigwedge x. (\lambda n. X\ n\ x) \longrightarrow Blinfun\ v\ x$ 
  by (auto simp: bounded_linear_Blinfun_apply v)

```

```

have X ⟶ Blinfun v
proof (rule LIMSEQ_I)
  fix r::real assume r > 0
  define r' where r' = r / 2
  have 0 < r' r' < r using ⟨r > 0⟩ by (simp_all add: r'_def)
  from CauchyD[OF ⟨Cauchy X⟩ ⟨r' > 0⟩]
  obtain M where M: ∧m n. m ≥ M ⟹ n ≥ M ⟹ norm (X m - X n) < r'
    by metis
  show ∃ no. ∀ n ≥ no. norm (X n - Blinfun v) < r
  proof (safe intro!: exI[where x=M])
    fix n assume n: M ≤ n
    have norm (X n - Blinfun v) ≤ r'
    proof (rule norm_blinfun_bound)
      fix x
      have eventually (λm. m ≥ M) sequentially
      by (metis eventually_ge_at_top)
      hence ev_le: eventually (λm. norm (X n x - X m x) ≤ r' * norm x)
        sequentially
      proof eventually_elim
        case (elim m)
        have norm (X n x - X m x) = norm ((X n - X m) x)
          by (simp add: blinfun.bilinear_simps)
        also have ... ≤ norm ((X n - X m)) * norm x
          by (rule norm_blinfun)
        also have ... ≤ r' * norm x
          using M[OF n elim] by (simp add: mult_right_mono)
        finally show ?case .
      qed
      have tendsto_v: (λm. norm (X n x - X m x)) ⟶ norm (X n x -
        Blinfun v x)
      by (auto intro!: tendsto_intros Bv)
      show norm ((X n - Blinfun v) x) ≤ r' * norm x
      by (auto intro!: tendsto_upperbound tendsto_v ev_le simp: blinfun.bilinear_simps)
    qed (simp add: ⟨0 < r'⟩ less_imp_le)
    thus norm (X n - Blinfun v) < r
      by (metis ⟨r' < r⟩ le_less_trans)
    qed
  qed
  thus convergent X
    by (rule convergentI)
qed

```

4.9.5 On Euclidean Space

```

lemma Zfun_sum:
  assumes finite s
  assumes f: ∧i. i ∈ s ⟹ Zfun (f i) F
  shows Zfun (λx. sum (λi. f i x) s) F
  using assms by induct (auto intro!: Zfun_zero Zfun_add)

```

```

lemma norm_blinfun_euclidean_le:
  fixes a::'a::euclidean_space  $\Rightarrow_L$  'b::real_normed_vector
  shows norm a  $\leq$  sum ( $\lambda x.$  norm (a x)) Basis
  apply (rule norm_blinfun_bound)
  apply (simp add: sum_nonneg)
  apply (subst euclidean_representation[symmetric, where 'a='a])
  apply (simp only: blinfun.bilinear_simps sum_distrib_right)
  apply (rule order.trans[OF norm_sum sum_mono])
  apply (simp add: abs_mult mult_right_mono ac_simps Basis_le_norm)
  done

lemma tendsto_componentwise1:
  fixes a::'a::euclidean_space  $\Rightarrow_L$  'b::real_normed_vector
  and b::'c  $\Rightarrow$  'a  $\Rightarrow_L$  'b
  assumes ( $\bigwedge j. j \in \text{Basis} \implies ((\lambda n. b\ n\ j) \longrightarrow a\ j)\ F$ )
  shows (b  $\longrightarrow$  a) F
proof -
  have  $\bigwedge j. j \in \text{Basis} \implies \text{Zfun } (\lambda x. \text{norm } (b\ x\ j - a\ j))\ F$ 
  using assms unfolding tendsto_Zfun_iff Zfun_norm_iff .
  hence  $\text{Zfun } (\lambda x. \sum_{j \in \text{Basis}} \text{norm } (b\ x\ j - a\ j))\ F$ 
  by (auto intro!: Zfun_sum)
  thus ?thesis
  unfolding tendsto_Zfun_iff
  by (rule Zfun_le)
  (auto intro!: order_trans[OF norm_blinfun_euclidean_le] simp: blinfun.bilinear_simps)
qed

lift_definition
  blinfun_of_matrix::('b::euclidean_space  $\Rightarrow$  'a::euclidean_space  $\Rightarrow$  real)  $\Rightarrow$  'a  $\Rightarrow_L$  'b
  is  $\lambda a\ x. \sum_{i \in \text{Basis}} \sum_{j \in \text{Basis}} ((x \cdot j) * a\ i\ j) *_R i$ 
  by (intro bounded_linear_intros)

lemma blinfun_of_matrix_works:
  fixes f::'a::euclidean_space  $\Rightarrow_L$  'b::euclidean_space
  shows blinfun_of_matrix ( $\lambda i\ j. (f\ j) \cdot i$ ) = f
proof (transfer, rule, rule euclidean_eqI)
  fix f::'a  $\Rightarrow$  'b and x::'a and b::'b assume bounded_linear f and b: b  $\in$  Basis
  then interpret bounded_linear f by simp
  have ( $\sum_{j \in \text{Basis}} \sum_{i \in \text{Basis}} (x \cdot i * (f\ i \cdot j)) *_R j$ )  $\cdot$  b
  = ( $\sum_{j \in \text{Basis}} \text{if } j = b \text{ then } (\sum_{i \in \text{Basis}} (x \cdot i * (f\ i \cdot j))) \text{ else } 0$ )
  using b
  by (simp add: inner_sum_left inner_Basis if_distrib cong: if_cong) (simp add: sum.swap)
  also have ... = ( $\sum_{i \in \text{Basis}} (x \cdot i * (f\ i \cdot b))$ )
  using b by (simp)
  also have ... = f x  $\cdot$  b
  by (metis (mono_tags, lifting) Linear_Algebra.linear_componentwise linear_axioms)

```

finally show $(\sum_{j \in \text{Basis}} \sum_{i \in \text{Basis}} (x \cdot i * (f i \cdot j)) *_R j) \cdot b = f x \cdot b$.
qed

lemma *blinfun_of_matrix_apply*:
 $\text{blinfun\_of\_matrix } a \ x = (\sum_{i \in \text{Basis}} \sum_{j \in \text{Basis}} ((x \cdot j) * a \ i \ j) *_R i)$
by *transfer simp*

lemma *blinfun_of_matrix_minus*: $\text{blinfun\_of\_matrix } x - \text{blinfun\_of\_matrix } y$
 $= \text{blinfun\_of\_matrix } (x - y)$
by *transfer (auto simp: algebra_simps sum_subtractf)*

lemma *norm_blinfun_of_matrix*:
 $\text{norm } (\text{blinfun\_of\_matrix } a) \leq (\sum_{i \in \text{Basis}} \sum_{j \in \text{Basis}} |a \ i \ j|)$
apply (*rule norm_blinfun_bound*)
apply (*simp add: sum_nonneg*)
apply (*simp only: blinfun_of_matrix_apply sum_distrib_right*)
apply (*rule order_trans[OF norm_sum sum_mono]*)
apply (*rule order_trans[OF norm_sum sum_mono]*)
apply (*simp add: abs_mult mult_right_mono ac_simps Basis_le_norm*)
done

lemma *tendsto_blinfun_of_matrix*:
assumes $\bigwedge i \ j. i \in \text{Basis} \implies j \in \text{Basis} \implies ((\lambda n. b \ n \ i \ j) \longrightarrow a \ i \ j) \ F$
shows $((\lambda n. \text{blinfun\_of\_matrix } (b \ n)) \longrightarrow \text{blinfun\_of\_matrix } a) \ F$
proof –
have $\bigwedge i \ j. i \in \text{Basis} \implies j \in \text{Basis} \implies Zfun (\lambda x. \text{norm } (b \ x \ i \ j - a \ i \ j)) \ F$
using *assms unfolding tendsto_Zfun_iff Zfun_norm_iff* .
hence $Zfun (\lambda x. (\sum_{i \in \text{Basis}} \sum_{j \in \text{Basis}} |b \ x \ i \ j - a \ i \ j|)) \ F$
by (*auto intro!: Zfun_sum*)
thus *?thesis*
unfolding *tendsto_Zfun_iff blinfun_of_matrix_minus*
by (*rule Zfun_le*) (*auto intro!: order_trans[OF norm_blinfun_of_matrix]*)
qed

lemma *tendsto_componentwise*:
fixes $a::'a::\text{euclidean\_space} \Rightarrow_L 'b::\text{euclidean\_space}$
and $b::'c \Rightarrow 'a \Rightarrow_L 'b$
shows $(\bigwedge i \ j. i \in \text{Basis} \implies j \in \text{Basis} \implies ((\lambda n. b \ n \ j \cdot i) \longrightarrow a \ j \cdot i) \ F) \implies$
 $(b \longrightarrow a) \ F$
apply (*subst blinfun_of_matrix_works[of a, symmetric]*)
apply (*subst blinfun_of_matrix_works[of b x for x, symmetric, abs_def]*)
by (*rule tendsto_blinfun_of_matrix*)

lemma
continuous_blinfun_componentwiseI:
fixes $f::'b::t2\_space \Rightarrow 'a::\text{euclidean\_space} \Rightarrow_L 'c::\text{euclidean\_space}$
assumes $\bigwedge i \ j. i \in \text{Basis} \implies j \in \text{Basis} \implies \text{continuous } F \ (\lambda x. (f \ x) \ j \cdot i)$
shows *continuous F f*
using *assms by (auto simp: continuous_def intro!: tendsto_componentwise)*

lemma

continuous_blinfun_componentwiseI1:
fixes $f :: 'b :: t2\_space \Rightarrow 'a :: euclidean\_space \Rightarrow_L 'c :: real\_normed\_vector$
assumes $\bigwedge i. i \in Basis \implies continuous\ F\ (\lambda x. f\ x\ i)$
shows $continuous\ F\ f$
using *assms* **by** (*auto simp: continuous_def intro!: tendsto_componentwise1*)

lemma

continuous_on_blinfun_componentwise:
fixes $f :: 'd :: t2\_space \Rightarrow 'e :: euclidean\_space \Rightarrow_L 'f :: real\_normed\_vector$
assumes $\bigwedge i. i \in Basis \implies continuous\_on\ s\ (\lambda x. f\ x\ i)$
shows $continuous\_on\ s\ f$
using *assms*
by (*auto intro!: continuous_at_imp_continuous_on intro!: tendsto_componentwise1 simp: continuous_on_eq_continuous_within continuous_def*)

lemma *bounded_linear_blinfun_matrix:* $bounded\_linear\ (\lambda x. (x :: \_ \Rightarrow_L \_) j \cdot i)$
by (*auto intro!: bounded_linearI' bounded_linear_intros*)

lemma *continuous_blinfun_matrix:*

fixes $f :: 'b :: t2\_space \Rightarrow 'a :: real\_normed\_vector \Rightarrow_L 'c :: real\_inner$
assumes $continuous\ F\ f$
shows $continuous\ F\ (\lambda x. (f\ x)\ j \cdot i)$
by (*rule bounded_linear.continuous[OF bounded_linear_blinfun_matrix assms]*)

lemma *continuous_on_blinfun_matrix:*

fixes $f :: 'a :: t2\_space \Rightarrow 'b :: real\_normed\_vector \Rightarrow_L 'c :: real\_inner$
assumes $continuous\_on\ S\ f$
shows $continuous\_on\ S\ (\lambda x. (f\ x)\ j \cdot i)$
using *assms*
by (*auto simp: continuous_on_eq_continuous_within continuous_blinfun_matrix*)

lemma *continuous_on_blinfun_of_matrix[continuous_intros]:*

assumes $\bigwedge i\ j. i \in Basis \implies j \in Basis \implies continuous\_on\ S\ (\lambda s. g\ s\ i\ j)$
shows $continuous\_on\ S\ (\lambda s. blinfun\_of\_matrix\ (g\ s))$
using *assms*
by (*auto simp: continuous_on intro!: tendsto_blinfun_of_matrix*)

lemma *mult_if_delta:*

*(if P then (1 :: 'a :: comm_semiring_1) else 0) * q = (if P then q else 0)*
by *auto*

lemma *compact_blinfun_lemma:*

fixes $f :: nat \Rightarrow 'a :: euclidean\_space \Rightarrow_L 'b :: euclidean\_space$
assumes $bounded\ (range\ f)$
shows $\forall d \subseteq Basis. \exists l :: 'a \Rightarrow_L 'b. \exists r :: nat \Rightarrow nat.$
 $strict\_mono\ r \wedge (\forall e > 0. eventually\ (\lambda n. \forall i \in d. dist\ (f\ (r\ n)\ i)\ (l\ i) < e)$
sequentially)

```

by (rule compact_lemma_general[where unproj =  $\lambda e. \text{blinfun\_of\_matrix } (\lambda i j. e j \cdot i)$ ])
  (auto intro!: euclidean_eqI[where 'a='b] bounded_linear_image assms
    simp: blinfun_of_matrix_works blinfun_of_matrix_apply inner_Basis mult_if_delta
    sum.delta'
    scaleR_sum_left[symmetric])

```

```

lemma blinfun_euclidean_eqI: ( $\bigwedge i. i \in \text{Basis} \implies \text{blinfun\_apply } x \ i = \text{blinfun\_apply } y \ i$ )  $\implies x = y$ 
apply (auto intro!: blinfun_eqI)
apply (subst (2) euclidean_representation[symmetric, where 'a='a])
apply (subst (1) euclidean_representation[symmetric, where 'a='a])
apply (simp add: blinfun.bilinear_simps)
done

```

```

lemma Blinfun_eq_matrix: bounded_linear f  $\implies \text{Blinfun } f = \text{blinfun\_of\_matrix } (\lambda i j. f j \cdot i)$ 
by (intro blinfun_euclidean_eqI)
  (auto simp: blinfun_of_matrix_apply bounded_linear_Blinfun_apply inner_Basis
    if_distrib
    if_distribR sum.delta' euclidean_representation
    cong: if_cong)

```

TODO: generalize (via *compact_cball*)?

```

instance blinfun :: (euclidean_space, euclidean_space) heine_borel
proof
  fix f :: nat  $\Rightarrow$  'a  $\Rightarrow_L$  'b
  assume f: bounded (range f)
  then obtain l::'a  $\Rightarrow_L$  'b and r where r: strict_mono r
  and l:  $\forall e > 0. \text{eventually } (\lambda n. \forall i \in \text{Basis}. \text{dist } (f \ (r \ n) \ i) \ (l \ i) < e) \text{ sequentially}$ 
  using compact_blinfun_lemma [OF f] by blast
  {
    fix e::real
    let ?d = real_of_nat DIM('a) * real_of_nat DIM('b)
    assume e > 0
    hence e / ?d > 0 by (simp)
    with l have eventually ( $\lambda n. \forall i \in \text{Basis}. \text{dist } (f \ (r \ n) \ i) \ (l \ i) < e / ?d$ ) sequentially
    by simp
    moreover
    {
      fix n
      assume n:  $\forall i \in \text{Basis}. \text{dist } (f \ (r \ n) \ i) \ (l \ i) < e / ?d$ 
      have norm (f (r n) - l) = norm (blinfun_of_matrix ( $\lambda i j. (f \ (r \ n) - l) \ j \cdot i$ ))
      unfolding blinfun_of_matrix_works ..
      also note norm_blinfun_of_matrix
      also have ( $\sum i \in \text{Basis}. \sum j \in \text{Basis}. |(f \ (r \ n) - l) \ j \cdot i|$ ) <
        ( $\sum i \in (\text{Basis}::'b \text{ set}). e / \text{real\_of\_nat } \text{DIM}('b)$ )
      proof (rule sum_strict_mono)

```



```

fix i::'b assume i: i ∈ Basis
have (∑ j::'a ∈ Basis. |(f (r n) - l) j · i|) < (∑ j::'a ∈ Basis. e / ?d)
proof (rule sum_strict_mono)
  fix j::'a assume j: j ∈ Basis
  have |(f (r n) - l) j · i| ≤ norm ((f (r n) - l) j)
    by (simp add: Basis_le_norm i)
  also have ... < e / ?d
    using n i j by (auto simp: dist_norm blinfun.bilinear_simps)
  finally show |(f (r n) - l) j · i| < e / ?d by simp
qed simp_all
also have ... ≤ e / real_of_nat DIM('b)
  by simp
finally show (∑ j ∈ Basis. |(f (r n) - l) j · i|) < e / real_of_nat DIM('b)
  by simp
qed simp_all
also have ... ≤ e by simp
finally have dist (f (r n)) l < e
  by (auto simp: dist_norm)
}
ultimately have eventually (λn. dist (f (r n)) l < e) sequentially
  using eventually_elim2 by force
}
then have *: ((f ∘ r) ⟶ l) sequentially
  unfolding o_def tendsto_iff by simp
with r show ∃ l r. strict_mono r ∧ ((f ∘ r) ⟶ l) sequentially
  by auto
qed

```

4.9.6 concrete bounded linear functions

lemma *transfer_bounded_bilinear_bounded_linearI*:

assumes $g = (\lambda i x. (blinfun\_apply (f i) x))$
 shows $bounded\_bilinear\ g = bounded\_linear\ f$

proof

assume $bounded\_bilinear\ g$

then interpret $bounded\_bilinear\ f$ by (simp add: assms)

show $bounded\_linear\ f$

proof (unfold locales, safe intro!: blinfun_eqI)

fix i

show $f (x + y) i = (f x + f y) i$ $f (r *_{\mathbb{R}} x) i = (r *_{\mathbb{R}} f x) i$ for $r x y$

by (auto intro!: blinfun_eqI simp: blinfun.bilinear_simps)

from $\_nonneg\_bounded$ show $\exists K. \forall x. norm (f x) \leq norm x * K$

by (rule ex_reg) (auto intro!: onorm_bound simp: norm_blinfun.rep_eq ac_simps)

qed

qed (auto simp: assms intro!: blinfun.comp)

lemma *transfer_bounded_bilinear_bounded_linear[transfer_rule]*:

$(rel\_fun (rel\_fun (=) (pcr\_blinfun (=) (=))) (=))\ bounded\_bilinear\ bounded\_linear$

by (*auto simp: pcr_blinfun_def cr_blinfun_def rel_fun_def OO_def*
intro!: transfer_bounded_bilinear_bounded_linearI)

context *bounded_bilinear*
begin

lift_definition *prod_left*:: $'b \Rightarrow 'a \Rightarrow_L 'c$ **is** $(\lambda b\ a.\ \text{prod}\ a\ b)$
by (*rule bounded_linear_left*)
declare *prod_left.rep_eq*[*simp*]

lemma *bounded_linear_prod_left*[*bounded_linear*]: *bounded_linear prod_left*
by *transfer (rule flip)*

lift_definition *prod_right*:: $'a \Rightarrow 'b \Rightarrow_L 'c$ **is** $(\lambda a\ b.\ \text{prod}\ a\ b)$
by (*rule bounded_linear_right*)
declare *prod_right.rep_eq*[*simp*]

lemma *bounded_linear_prod_right*[*bounded_linear*]: *bounded_linear prod_right*
by *transfer (rule bounded_bilinear_axioms)*

end

lift_definition *id_blinfun*:: $'a::\text{real\_normed\_vector} \Rightarrow_L 'a$ **is** $\lambda x.\ x$
by (*rule bounded_linear_ident*)

lemmas *blinfun_apply_id_blinfun*[*simp*] = *id_blinfun.rep_eq*

lemma *norm_blinfun_id*[*simp*]:
norm (id_blinfun::' $a::\{\text{real\_normed\_vector}, \text{perfect\_space}\} \Rightarrow_L 'a$) = 1
by *transfer (auto simp: onorm_id)*

lemma *norm_blinfun_id_le*:
norm (id_blinfun::' $a::\text{real\_normed\_vector} \Rightarrow_L 'a$) ≤ 1
by *transfer (auto simp: onorm_id_le)*

lift_definition *fst_blinfun*::($'a::\text{real\_normed\_vector} \times 'b::\text{real\_normed\_vector}$)
 $\Rightarrow_L 'a$ **is** *fst*
by (*rule bounded_linear_fst*)

lemma *blinfun_apply_fst_blinfun*[*simp*]: *blinfun_apply fst_blinfun = fst*
by *transfer (rule refl)*

lift_definition *snd_blinfun*::($'a::\text{real\_normed\_vector} \times 'b::\text{real\_normed\_vector}$)
 $\Rightarrow_L 'b$ **is** *snd*
by (*rule bounded_linear_snd*)

lemma *blinfun_apply_snd_blinfun*[*simp*]: *blinfun_apply snd_blinfun = snd*

by *transfer* (rule *refl*)

lift_definition *blinfun_compose*::

'a::real_normed_vector $\Rightarrow_L$ *'b::real_normed_vector* $\Rightarrow$

'c::real_normed_vector $\Rightarrow_L$ *'a* $\Rightarrow$

'c $\Rightarrow_L$ *'b* (**infixl** *o_L* 55) **is** (*o*)

parametric *comp_transfer*

unfolding *o_def*

by (rule *bounded_linear_compose*)

lemma *blinfun_apply_blinfun_compose*[*simp*]: (*a o_L b*) *c* = *a* (*b c*)

by (*simp add: blinfun_compose.rep_eq*)

lemma *norm_blinfun_compose*:

norm (*f o_L g*) $\leq$ *norm f* * *norm g*

by *transfer* (rule *onorm_compose*)

lemma *bounded_bilinear_blinfun_compose*[*bounded_bilinear*]: *bounded_bilinear* (*o_L*)

by *unfold_locales*

(*auto intro!: blinfun_eqI exI*[**where** *x=1*] *simp: blinfun.bilinear_simps norm_blinfun_compose*)

lemma *blinfun_compose_zero*[*simp*]:

blinfun_compose 0 = (λ _. 0)

blinfun_compose x 0 = 0

by (*auto simp: blinfun.bilinear_simps intro!: blinfun_eqI*)

lemma *blinfun_bij2*:

fixes *f::'a* $\Rightarrow_L$ *'a::euclidean_space*

assumes *f o_L g* = *id_blinfun*

shows *bij* (*blinfun_apply g*)

proof (rule *bijI*)

show *inj g*

using *assms*

by (*metis blinfun_apply_id_blinfun blinfun_compose.rep_eq injI inj_on_imageI2*)

then show *surj g*

using *blinfun.bounded_linear_right bounded_linear_def linear_inj_imp_surj*

by *blast*

qed

lemma *blinfun_bij1*:

fixes *f::'a* $\Rightarrow_L$ *'a::euclidean_space*

assumes *f o_L g* = *id_blinfun*

shows *bij* (*blinfun_apply f*)

proof (rule *bijI*)

show *surj* (*blinfun_apply f*)

by (*metis assms blinfun_apply_blinfun_compose blinfun_apply_id_blinfun surjI*)

then show *inj* (*blinfun_apply f*)

using *blinfun.bounded_linear_right* *bounded_linear_def* *linear_surj_imp_inj*
by *blast*
qed

lift_definition *blinfun_inner_right*::*'a::real_inner* $\Rightarrow$ *'a* $\Rightarrow_L$ *real* **is** $(\cdot)$
by (*rule bounded_linear_inner_right*)
declare *blinfun_inner_right.rep_eq*[*simp*]

lemma *bounded_linear_blinfun_inner_right*[*bounded_linear*]: *bounded_linear blinfun_inner_right*
by *transfer* (*rule bounded_bilinear_inner*)

lift_definition *blinfun_inner_left*::*'a::real_inner* $\Rightarrow$ *'a* $\Rightarrow_L$ *real* **is** $\lambda x y. y \cdot x$
by (*rule bounded_linear_inner_left*)
declare *blinfun_inner_left.rep_eq*[*simp*]

lemma *bounded_linear_blinfun_inner_left*[*bounded_linear*]: *bounded_linear blinfun_inner_left*
by *transfer* (*rule bounded_bilinear_flip*[*OF* *bounded_bilinear_inner*])

lift_definition *blinfun_scaleR_right*::*real* $\Rightarrow$ *'a* $\Rightarrow_L$ *'a::real_normed_vector* **is** $(*_R)$
by (*rule bounded_linear_scaleR_right*)
declare *blinfun_scaleR_right.rep_eq*[*simp*]

lemma *bounded_linear_blinfun_scaleR_right*[*bounded_linear*]: *bounded_linear blinfun_scaleR_right*
by *transfer* (*rule bounded_bilinear_scaleR*)

lift_definition *blinfun_scaleR_left*::*'a::real_normed_vector* $\Rightarrow$ *real* $\Rightarrow_L$ *'a* **is** $\lambda x y. y *_R x$
by (*rule bounded_linear_scaleR_left*)
lemmas [*simp*] = *blinfun_scaleR_left.rep_eq*

lemma *bounded_linear_blinfun_scaleR_left*[*bounded_linear*]: *bounded_linear blinfun_scaleR_left*
by *transfer* (*rule bounded_bilinear_flip*[*OF* *bounded_bilinear_scaleR*])

lift_definition *blinfun_mult_right*::*'a* $\Rightarrow$ *'a* $\Rightarrow_L$ *'a::real_normed_algebra* **is** $(*)$
by (*rule bounded_linear_mult_right*)
declare *blinfun_mult_right.rep_eq*[*simp*]

lemma *bounded_linear_blinfun_mult_right*[*bounded_linear*]: *bounded_linear blinfun_mult_right*
by *transfer* (*rule bounded_bilinear_mult*)

lift_definition *blinfun_mult_left*:: $'a::\text{real\_normed\_algebra} \Rightarrow 'a \Rightarrow_L 'a$ **is** $\lambda x y.$
 $y * x$
by (rule *bounded_linear_mult_left*)
lemmas [*simp*] = *blinfun_mult_left.rep_eq*

lemma *bounded_linear_blinfun_mult_left*[*bounded_linear*]: *bounded_linear blinfun_mult_left*
by transfer (rule *bounded_bilinear.flip*[*OF* *bounded_bilinear_mult*])

lemmas *bounded_linear_function_uniform_limit_intros*[*uniform_limit_intros*] =
bounded_linear.uniform_limit[*OF* *bounded_linear_apply_blinfun*]
bounded_linear.uniform_limit[*OF* *bounded_linear_blinfun_apply*]
bounded_linear.uniform_limit[*OF* *bounded_linear_blinfun_matrix*]

4.9.7 The strong operator topology on continuous linear operators

Let $'a$ and $'b$ be two normed real vector spaces. Then the space of linear continuous operators from $'a$ to $'b$ has a canonical norm, and therefore a canonical corresponding topology (the type classes instantiation are given in *Bounded_Linear_Function.thy*).

However, there is another topology on this space, the strong operator topology, where T_n tends to T iff, for all x in $'a$, then $T_n x$ tends to $T x$. This is precisely the product topology where the target space is endowed with the norm topology. It is especially useful when $'b$ is the set of real numbers, since then this topology is compact.

We can not implement it using type classes as there is already a topology, but at least we can define it as a topology.

Note that there is yet another (common and useful) topology on operator spaces, the weak operator topology, defined analogously using the product topology, but where the target space is given the weak-* topology, i.e., the pullback of the weak topology on the bidual of the space under the canonical embedding of a space into its bidual. We do not define it there, although it could also be defined analogously.

definition *strong_operator_topology*::($'a::\text{real\_normed\_vector} \Rightarrow_L 'b::\text{real\_normed\_vector}$)
topology
where *strong_operator_topology* = *pullback_topology UNIV blinfun_apply euclidean*

lemma *strong_operator_topology_topspace*:
topspace strong_operator_topology = *UNIV*
unfolding *strong_operator_topology_def* *topspace_pullback_topology* *topspace_euclidean*
by *auto*

lemma *strong_operator_topology_basis*:

```

    fixes f::('a::real_normed_vector  $\Rightarrow_L$  'b::real_normed_vector) and U::'i  $\Rightarrow$  'b set
  and x::'i  $\Rightarrow$  'a
    assumes finite I  $\wedge i. i \in I \implies \text{open } (U\ i)$ 
    shows  $\text{openin strong\_operator\_topology } \{f. \forall i \in I. \text{blinfun\_apply } f\ (x\ i) \in U\ i\}$ 
  proof -
    have  $\text{open } \{g::('a \Rightarrow 'b). \forall i \in I. g\ (x\ i) \in U\ i\}$ 
      by (rule product_topology_basis'[OF assms])
    moreover have  $\{f. \forall i \in I. \text{blinfun\_apply } f\ (x\ i) \in U\ i\}$ 
      =  $\text{blinfun\_apply- } \{g::('a \Rightarrow 'b). \forall i \in I. g\ (x\ i) \in U\ i\} \cap \text{UNIV}$ 
      by auto
    ultimately show ?thesis
      unfolding strong_operator_topology_def by (subst openin_pullback_topology)
  auto
qed

```

```

lemma strong_operator_topology_continuous_evaluation:
  continuous_map strong_operator_topology euclidean  $(\lambda f. \text{blinfun\_apply } f\ x)$ 
proof -
  have continuous_map strong_operator_topology euclidean  $((\lambda f. f\ x) \circ \text{blinfun\_apply})$ 
    unfolding strong_operator_topology_def apply (rule continuous_map_pullback)
    using continuous_on_product_coordinates by fastforce
  then show ?thesis unfolding comp_def by simp
qed

```

```

lemma continuous_on_strong_operator_topo_iff_coordinatewise:
  continuous_map T strong_operator_topology f
 $\iff (\forall x. \text{continuous\_map } T\ \text{euclidean } (\lambda y. \text{blinfun\_apply } (f\ y)\ x))$ 
proof (auto)
  fix x::'b
  assume continuous_map T strong_operator_topology f
  with continuous_map_compose[OF this strong_operator_topology_continuous_evaluation]
  have continuous_map T euclidean  $((\lambda z. \text{blinfun\_apply } z\ x) \circ f)$ 
    by simp
  then show continuous_map T euclidean  $(\lambda y. \text{blinfun\_apply } (f\ y)\ x)$ 
    unfolding comp_def by auto
next
  assume *:  $\forall x. \text{continuous\_map } T\ \text{euclidean } (\lambda y. \text{blinfun\_apply } (f\ y)\ x)$ 
  have  $\wedge i. \text{continuous\_map } T\ \text{euclidean } (\lambda x. \text{blinfun\_apply } (f\ x)\ i)$ 
    using * unfolding comp_def by auto
  then have continuous_map T euclidean  $(\text{blinfun\_apply } \circ f)$ 
    unfolding o_def
    by (metis (no_types) continuous_map_componentwise_UNIV euclidean_product_topology)
  show continuous_map T strong_operator_topology f
    unfolding strong_operator_topology_def
    apply (rule continuous_map_pullback')
    by (auto simp add:  $\langle \text{continuous\_map } T\ \text{euclidean } (\text{blinfun\_apply } \circ f) \rangle$ )
qed

```

```

lemma strong_operator_topology_weaker_than_euclidean:
  continuous_map euclidean strong_operator_topology ( $\lambda f. f$ )
  by (subst continuous_on_strong_operator_topo_iff_coordinatewise,
    auto simp add: linear_continuous_on)

end

```

4.10 Derivative

```

theory Derivative
  imports
    Bounded_Linear_Function
    Line_Segment
    Convex_Euclidean_Space
begin

declare bounded_linear_inner_left [intro]

declare has_derivative_bounded_linear [dest]

```

4.10.1 Derivatives

```

lemma has_derivative_add_const:
  (f has_derivative f') net  $\implies ((\lambda x. f\ x + c)$  has_derivative f') net
  by (intro derivative_eq_intros) auto

```

4.10.2 Derivative with composed bilinear function

More explicit epsilon-delta forms.

```

proposition has_derivative_within':
  (f has_derivative f') (at x within s)  $\longleftrightarrow$ 
    bounded_linear f'  $\wedge$ 
    ( $\forall e > 0. \exists d > 0. \forall x' \in s. 0 < \text{norm } (x' - x) \wedge \text{norm } (x' - x) < d \implies$ 
       $\text{norm } (f\ x' - f\ x - f'(x' - x)) / \text{norm } (x' - x) < e$ )
  unfolding has_derivative_within Lim_within dist_norm
  by (simp add: diff_diff_eq)

```

```

lemma has_derivative_at':
  (f has_derivative f') (at x)
   $\longleftrightarrow$  bounded_linear f'  $\wedge$ 
    ( $\forall e > 0. \exists d > 0. \forall x'. 0 < \text{norm } (x' - x) \wedge \text{norm } (x' - x) < d \implies$ 
       $\text{norm } (f\ x' - f\ x - f'(x' - x)) / \text{norm } (x' - x) < e$ )
  using has_derivative_within' [of f f' x UNIV] by simp

```

```

lemma has_derivative_componentwise_within:
  (f has_derivative f') (at a within S)  $\longleftrightarrow$ 
    ( $\forall i \in \text{Basis}. ((\lambda x. f\ x \cdot i)$  has_derivative ( $\lambda x. f'\ x \cdot i$ )) (at a within S))
  apply (simp add: has_derivative_within)

```

```

apply (subst tendsto_componentwise_iff)
apply (simp add: bounded_linear_componentwise_iff [symmetric] ball_conj_distrib)
apply (simp add: algebra_simps)
done

```

```

lemma has_derivative_at_withinI:
  (f has_derivative f') (at x)  $\implies$  (f has_derivative f') (at x within s)
unfolding has_derivative_within' has_derivative_at'
by blast

```

```

lemma has_derivative_right:
  fixes f :: real  $\Rightarrow$  real
  and y :: real
  shows (f has_derivative ((*) y)) (at x within ({x <..}  $\cap$  I))  $\longleftrightarrow$ 
    (( $\lambda t$ . (f x - f t) / (x - t))  $\longrightarrow$  y) (at x within ({x <..}  $\cap$  I))
proof -
  have (( $\lambda t$ . (f t - (f x + y * (t - x))) / |t - x|)  $\longrightarrow$  0) (at x within ({x <..}
 $\cap$  I))  $\longleftrightarrow$ 
    (( $\lambda t$ . (f t - f x) / (t - x) - y)  $\longrightarrow$  0) (at x within ({x <..}  $\cap$  I))
  by (intro Lim_cong_within) (auto simp add: diff_divide_distrib add_divide_distrib)
  also have ...  $\longleftrightarrow$  (( $\lambda t$ . (f t - f x) / (t - x))  $\longrightarrow$  y) (at x within ({x <..}  $\cap$ 
I))
  by (simp add: Lim_null[symmetric])
  also have ...  $\longleftrightarrow$  (( $\lambda t$ . (f x - f t) / (x - t))  $\longrightarrow$  y) (at x within ({x <..}  $\cap$ 
I))
  by (intro Lim_cong_within) (simp_all add: field_simps)
  finally show ?thesis
  by (simp add: bounded_linear_mult_right has_derivative_within)
qed

```

Caratheodory characterization

```

lemma DERIV_caratheodory_within:
  (f has_field_derivative l) (at x within S)  $\longleftrightarrow$ 
    ( $\exists g$ . ( $\forall z$ . f z - f x = g z * (z - x))  $\wedge$  continuous (at x within S) g  $\wedge$  g x = l)
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  show ?rhs
  proof (intro exI conjI)
    let ?g = ( $\%z$ . if z = x then l else (f z - f x) / (z - x))
    show  $\forall z$ . f z - f x = ?g z * (z - x) by simp
    show continuous (at x within S) ?g using <?lhs>
    by (auto simp add: continuous_within has_field_derivative_iff cong: Lim_cong_within)
    show ?g x = l by simp
  qed
next
  assume ?rhs
  then obtain g where

```



```

  (∀ z. f z - f x = g z * (z - x)) and continuous (at x within S) g and g x = l
by blast
  thus ?lhs
  by (auto simp add: continuous_within has_field_derivative_iff cong: Lim_cong_within)
qed

```

4.10.3 Differentiability

definition

```

differentiable_on :: ('a::real_normed_vector ⇒ 'b::real_normed_vector) ⇒ 'a set
⇒ bool
  (infix differentiable'_on 50)
  where f differentiable'_on s ⟷ (∀ x ∈ s. f differentiable (at x within s))

```

```

lemma differentiableI: (f has_derivative f') net ⟹ f differentiable net
  unfolding differentiable_def
  by auto

```

```

lemma differentiable_onD: [f differentiable_on S; x ∈ S] ⟹ f differentiable (at
x within S)
  using differentiable_on_def by blast

```

```

lemma differentiable_at_withinI: f differentiable (at x) ⟹ f differentiable (at x
within s)
  unfolding differentiable_def
  using has_derivative_at_withinI
  by blast

```

```

lemma differentiable_at_imp_differentiable_on:
  (⋀ x. x ∈ s ⟹ f differentiable at x) ⟹ f differentiable_on s
  by (metis differentiable_at_withinI differentiable_on_def)

```

```

corollary differentiable_iff_scaleR:
  fixes f :: real ⇒ 'a::real_normed_vector
  shows f differentiable F ⟷ (∃ d. (f has_derivative (λx. x *R d)) F)
  by (auto simp: differentiable_def dest: has_derivative_linear linear_imp_scaleR)

```

```

lemma differentiable_on_eq_differentiable_at:
  open s ⟹ f differentiable_on s ⟷ (∀ x ∈ s. f differentiable at x)
  unfolding differentiable_on_def
  by (metis at_within_interior interior_open)

```

```

lemma differentiable_transform_within:
  assumes f differentiable (at x within s)
  and 0 < d
  and x ∈ s
  and ⋀ x'. [x' ∈ s; dist x' x < d] ⟹ f x' = g x'
  shows g differentiable (at x within s)
  using assms has_derivative_transform_within unfolding differentiable_def

```

by *blast*

lemma *differentiable_on_ident* [*simp*, *derivative_intros*]: $(\lambda x. x)$ *differentiable_on* S
by (*simp add: differentiable_at_imp_differentiable_on*)

lemma *differentiable_on_id* [*simp*, *derivative_intros*]: *id* *differentiable_on* S
by (*simp add: id_def*)

lemma *differentiable_on_const* [*simp*, *derivative_intros*]: $(\lambda z. c)$ *differentiable_on* S
by (*simp add: differentiable_on_def*)

lemma *differentiable_on_mult* [*simp*, *derivative_intros*]:
fixes $f :: 'M::\text{real\_normed\_vector} \Rightarrow 'a::\text{real\_normed\_algebra}$
shows $\llbracket f \text{ differentiable\_on } S; g \text{ differentiable\_on } S \rrbracket \Longrightarrow (\lambda z. f\ z * g\ z) \text{ differentiable\_on } S$
unfolding *differentiable_on_def* *differentiable_def*
using *differentiable_def* *differentiable_mult* **by** *blast*

lemma *differentiable_on_compose*:
 $\llbracket g \text{ differentiable\_on } S; f \text{ differentiable\_on } (g\ 'S) \rrbracket \Longrightarrow (\lambda x. f\ (g\ x)) \text{ differentiable\_on } S$
by (*simp add: differentiable_in_compose differentiable_on_def*)

lemma *bounded_linear_imp_differentiable_on*: *bounded_linear* $f \Longrightarrow f$ *differentiable_on* S
by (*simp add: differentiable_on_def bounded_linear_imp_differentiable*)

lemma *linear_imp_differentiable_on*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{real\_normed\_vector}$
shows *linear* $f \Longrightarrow f$ *differentiable_on* S
by (*simp add: differentiable_on_def linear_imp_differentiable*)

lemma *differentiable_on_minus* [*simp*, *derivative_intros*]:
 $f \text{ differentiable\_on } S \Longrightarrow (\lambda z. -(f\ z)) \text{ differentiable\_on } S$
by (*simp add: differentiable_on_def*)

lemma *differentiable_on_add* [*simp*, *derivative_intros*]:
 $\llbracket f \text{ differentiable\_on } S; g \text{ differentiable\_on } S \rrbracket \Longrightarrow (\lambda z. f\ z + g\ z) \text{ differentiable\_on } S$
by (*simp add: differentiable_on_def*)

lemma *differentiable_on_diff* [*simp*, *derivative_intros*]:
 $\llbracket f \text{ differentiable\_on } S; g \text{ differentiable\_on } S \rrbracket \Longrightarrow (\lambda z. f\ z - g\ z) \text{ differentiable\_on } S$
by (*simp add: differentiable_on_def*)

lemma *differentiable_on_inverse* [*simp*, *derivative_intros*]:

```

fixes f :: 'a :: real_normed_vector  $\Rightarrow$  'b :: real_normed_field
shows f differentiable_on S  $\implies$  ( $\bigwedge x. x \in S \implies f\ x \neq 0$ )  $\implies$  ( $\lambda x. \text{inverse } (f\ x)$ ) differentiable_on S
by (simp add: differentiable_on_def)

```

```

lemma differentiable_on_scaleR [derivative_intros, simp]:
   $\llbracket f \text{ differentiable\_on } S; g \text{ differentiable\_on } S \rrbracket \implies (\lambda x. f\ x *_R g\ x) \text{ differentiable\_on } S$ 
unfolding differentiable_on_def
by (blast intro: differentiable_scaleR)

```

```

lemma has_derivative_sqnorm_at [derivative_intros, simp]:
  (( $\lambda x. (\text{norm } x)^2$ ) has_derivative ( $\lambda x. 2 *_R (a \cdot x)$ )) (at a)
using bounded_bilinear.FDERIV [of ( $\cdot$ ) id id a _ id id]
by (auto simp: inner_commute dot_square_norm bounded_bilinear_inner)

```

```

lemma differentiable_sqnorm_at [derivative_intros, simp]:
fixes a :: 'a :: {real_normed_vector, real_inner}
shows ( $\lambda x. (\text{norm } x)^2$ ) differentiable (at a)
by (force simp add: differentiable_def intro: has_derivative_sqnorm_at)

```

```

lemma differentiable_on_sqnorm [derivative_intros, simp]:
fixes S :: 'a :: {real_normed_vector, real_inner} set
shows ( $\lambda x. (\text{norm } x)^2$ ) differentiable_on S
by (simp add: differentiable_at_imp_differentiable_on)

```

```

lemma differentiable_norm_at [derivative_intros, simp]:
fixes a :: 'a :: {real_normed_vector, real_inner}
shows a  $\neq 0 \implies$  norm differentiable (at a)
using differentiableI has_derivative_norm by blast

```

```

lemma differentiable_on_norm [derivative_intros, simp]:
fixes S :: 'a :: {real_normed_vector, real_inner} set
shows 0  $\notin S \implies$  norm differentiable_on S
by (metis differentiable_at_imp_differentiable_on differentiable_norm_at)

```

4.10.4 Frechet derivative and Jacobian matrix

```

definition frechet_derivative f net = (SOME f'. (f has_derivative f') net)

```

```

proposition frechet_derivative_works:
  f differentiable net  $\longleftrightarrow$  (f has_derivative (frechet_derivative f net)) net
unfolding frechet_derivative_def differentiable_def
unfolding some_eq_ex[of  $\lambda f'. (f \text{ has\_derivative } f') \text{ net}$ ] ..

```

```

lemma linear_frechet_derivative: f differentiable net  $\implies$  linear (frechet_derivative f net)
unfolding frechet_derivative_works has_derivative_def
by (auto intro: bounded_linear.linear)

```

```

lemma frechet_derivative_const [simp]: frechet_derivative ( $\lambda x. c$ ) (at  $a$ ) = ( $\lambda x.$ 
0)
using differentiable_const frechet_derivative_works has_derivative_const has_derivative_unique
by blast

```

```

lemma frechet_derivative_id [simp]: frechet_derivative id (at  $a$ ) = id
using differentiable_def frechet_derivative_works has_derivative_id has_derivative_unique
by blast

```

```

lemma frechet_derivative_ident [simp]: frechet_derivative ( $\lambda x. x$ ) (at  $a$ ) = ( $\lambda x.$ 
 $x$ )
by (metis eq_id_iff frechet_derivative_id)

```

4.10.5 Differentiability implies continuity

```

proposition differentiable_imp_continuous_within:
 $f$  differentiable (at  $x$  within  $s$ )  $\implies$  continuous (at  $x$  within  $s$ )  $f$ 
by (auto simp: differentiable_def intro: has_derivative_continuous)

```

```

lemma differentiable_imp_continuous_on:
 $f$  differentiable_on  $s \implies$  continuous_on  $s$   $f$ 
unfolding differentiable_on_def continuous_on_eq_continuous_within
using differentiable_imp_continuous_within by blast

```

```

lemma differentiable_on_subset:
 $f$  differentiable_on  $t \implies s \subseteq t \implies f$  differentiable_on  $s$ 
unfolding differentiable_on_def
using differentiable_within_subset
by blast

```

```

lemma differentiable_on_empty:  $f$  differentiable_on {}
unfolding differentiable_on_def
by auto

```

```

lemma has_derivative_continuous_on:
 $(\bigwedge x. x \in s \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } s)) \implies$  continuous_on  $s$   $f$ 
by (auto intro!: differentiable_imp_continuous_on differentiableI simp: differentiable_on_def)

```

Results about neighborhoods filter.

```

lemma eventually_nhds_metric_le:
eventually  $P$  (nhds  $a$ ) =  $(\exists d > 0. \forall x. \text{dist } x a \leq d \longrightarrow P x)$ 
unfolding eventually_nhds_metric by (safe, rule_tac x=d / 2 in exI, auto)

```

```

lemma le_nhds:  $F \leq \text{nhds } a \longleftrightarrow (\forall S. \text{open } S \wedge a \in S \longrightarrow \text{eventually } (\lambda x. x \in S) F)$ 
unfolding le_filter_def eventually_nhds by (fast elim: eventually_mono)

```

lemma *le_nhds_metric*: $F \leq \text{nhds } a \iff (\forall e > 0. \text{eventually } (\lambda x. \text{dist } x \ a < e) \ F)$

unfolding *le_filter_def eventually_nhds_metric* **by** (*fast elim: eventually_mono*)

lemma *le_nhds_metric_le*: $F \leq \text{nhds } a \iff (\forall e > 0. \text{eventually } (\lambda x. \text{dist } x \ a \leq e) \ F)$

unfolding *le_filter_def eventually_nhds_metric_le* **by** (*fast elim: eventually_mono*)

Several results are easier using a "multiplied-out" variant. (I got this idea from Dieudonne's proof of the chain rule).

lemma *has_derivative_within_alt*:

$(f \text{ has\_derivative } f') \ (at \ x \ \text{within } s) \iff \text{bounded\_linear } f' \wedge$
 $(\forall e > 0. \exists d > 0. \forall y \in s. \text{norm}(y - x) < d \longrightarrow \text{norm}(f \ y - f \ x - f' (y - x)) \leq$
 $e * \text{norm}(y - x))$

unfolding *has_derivative_within filterlim_def le_nhds_metric_le eventually_filtermap eventually_at dist_norm diff_diff_eq*

by (*force simp add: linear_0 bounded_linear.linear pos_divide_le_eq*)

lemma *has_derivative_within_alt2*:

$(f \text{ has\_derivative } f') \ (at \ x \ \text{within } s) \iff \text{bounded\_linear } f' \wedge$
 $(\forall e > 0. \text{eventually } (\lambda y. \text{norm}(f \ y - f \ x - f' (y - x)) \leq e * \text{norm}(y - x))$
 $(at \ x \ \text{within } s))$

unfolding *has_derivative_within filterlim_def le_nhds_metric_le eventually_filtermap eventually_at dist_norm diff_diff_eq*

by (*force simp add: linear_0 bounded_linear.linear pos_divide_le_eq*)

lemma *has_derivative_at_alt*:

$(f \text{ has\_derivative } f') \ (at \ x) \iff$
 $\text{bounded\_linear } f' \wedge$
 $(\forall e > 0. \exists d > 0. \forall y. \text{norm}(y - x) < d \longrightarrow \text{norm}(f \ y - f \ x - f'(y - x)) \leq e * \text{norm}(y - x))$

using *has_derivative_within_alt* [**where** $s = \text{UNIV}$]

by *simp*

4.10.6 The chain rule

proposition *diff_chain_within* [*derivative_intros*]:

assumes $(f \text{ has\_derivative } f') \ (at \ x \ \text{within } s)$

and $(g \text{ has\_derivative } g') \ (at \ (f \ x) \ \text{within } (f' \ s))$

shows $((g \circ f) \text{ has\_derivative } (g' \circ f')) (at \ x \ \text{within } s)$

using *has_derivative_in_compose* [*OF assms*]

by (*simp add: comp_def*)

lemma *diff_chain_at* [*derivative_intros*]:

$(f \text{ has\_derivative } f') \ (at \ x) \implies$

$(g \text{ has\_derivative } g') \ (at \ (f \ x)) \implies ((g \circ f) \text{ has\_derivative } (g' \circ f')) \ (at \ x)$

using *has_derivative_compose* [*of f f' x UNIV g g'*]

by (*simp add: comp_def*)

lemma *has_vector_derivative_within_open*:
 $a \in S \implies \text{open } S \implies$
 $(f \text{ has\_vector\_derivative } f') \text{ (at } a \text{ within } S) \longleftrightarrow (f \text{ has\_vector\_derivative } f') \text{ (at } a)$
by (*simp only: at_within_interior interior_open*)

lemma *field_vector_diff_chain_within*:
assumes $Df: (f \text{ has\_vector\_derivative } f') \text{ (at } x \text{ within } S)$
and $Dg: (g \text{ has\_field\_derivative } g') \text{ (at } (f \ x) \text{ within } f' \ S)$
shows $((g \circ f) \text{ has\_vector\_derivative } (f' * g')) \text{ (at } x \text{ within } S)$
using *diff_chain_within[OF Df[unfolded has_vector_derivative_def]*
 $Dg \text{ [unfolded has\_field\_derivative\_def]}$
by (*auto simp: o_def mult.commute has_vector_derivative_def*)

lemma *vector_derivative_diff_chain_within*:
assumes $Df: (f \text{ has\_vector\_derivative } f') \text{ (at } x \text{ within } S)$
and $Dg: (g \text{ has\_derivative } g') \text{ (at } (f \ x) \text{ within } f' S)$
shows $((g \circ f) \text{ has\_vector\_derivative } (g' f')) \text{ (at } x \text{ within } S)$
using *diff_chain_within[OF Df[unfolded has_vector_derivative_def] Dg]*
 $\text{linear.scaleR[OF has\_derivative\_linear[OF Dg]]}$
unfolding *has_vector_derivative_def o_def*
by (*auto simp: o_def mult.commute has_vector_derivative_def*)

4.10.7 Composition rules stated just for differentiability

lemma *differentiable_chain_at*:
 $f \text{ differentiable (at } x) \implies$
 $g \text{ differentiable (at } (f \ x)) \implies (g \circ f) \text{ differentiable (at } x)$
unfolding *differentiable_def*
by (*meson diff_chain_at*)

lemma *differentiable_chain_within*:
 $f \text{ differentiable (at } x \text{ within } S) \implies$
 $g \text{ differentiable (at } (f \ x) \text{ within } (f' \ S)) \implies (g \circ f) \text{ differentiable (at } x \text{ within } S)$
unfolding *differentiable_def*
by (*meson diff_chain_within*)

4.10.8 Uniqueness of derivative

The general result is a bit messy because we need approachability of the limit point from any direction. But OK for nontrivial intervals etc.

proposition *frechet_derivative_unique_within*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{real\_normed\_vector}$
assumes $1: (f \text{ has\_derivative } f') \text{ (at } x \text{ within } S)$
and $2: (f \text{ has\_derivative } f'') \text{ (at } x \text{ within } S)$
and $S: \bigwedge i \ e. \llbracket i \in \text{Basis}; e > 0 \rrbracket \implies \exists d. 0 < |d| \wedge |d| < e \wedge (x + d *_{\mathbb{R}} i) \in S$
shows $f' = f''$
proof –
note $as = \text{assms}(1,2)[\text{unfolded has\_derivative\_def}]$

```

then interpret f': bounded_linear f' by auto
from as interpret f'': bounded_linear f'' by auto
have x islimpt S unfolding islimpt_approachable
proof (intro allI impI)
  fix e :: real
  assume e > 0
  obtain d where 0 < |d| and |d| < e and x + d *R (SOME i. i ∈ Basis) ∈ S
    using assms(3) SOME_Basis ⟨e>0 by blast
  then show ∃ x' ∈ S. x' ≠ x ∧ dist x' x < e
    by (rule_tac x=x + d *R (SOME i. i ∈ Basis) in bexI) (auto simp: dist_norm
SOME_Basis nonzero_Basis) qed
  then have *: netlimit (at x within S) = x
    by (simp add: Lim_ident_at trivial_limit_within)
show ?thesis
proof (rule linear_eq_stdbasis)
  show linear f' linear f''
    unfolding linear_conv_bounded_linear using as by auto
next
  fix i :: 'a
  assume i: i ∈ Basis
  define e where e = norm (f' i - f'' i)
  show f' i = f'' i
  proof (rule ccontr)
    assume f' i ≠ f'' i
    then have e > 0
      unfolding e_def by auto
    obtain d where d:
      0 < d
      (∧ y. y ∈ S ⟶ 0 < dist y x ∧ dist y x < d ⟶
        dist ((f y - f x - f' (y - x)) /R norm (y - x) -
          (f y - f x - f'' (y - x)) /R norm (y - x)) (0 - 0) < e)
      using tendsto_diff [OF as(1,2)[THEN conjunct2]]
      unfolding * Lim_within
      using ⟨e>0 by blast
    obtain c where c: 0 < |c| |c| < d ∧ x + c *R i ∈ S
      using assms(3) i d(1) by blast
    have *: norm (- ((1 / |c|) *R f' (c *R i)) + (1 / |c|) *R f'' (c *R i)) =
      norm ((1 / |c|) *R (- (f' (c *R i)) + f'' (c *R i)))
      unfolding scaleR_right_distrib by auto
    also have ... = norm ((1 / |c|) *R (c *R (- (f' i) + f'' i)))
      unfolding f'.scaleR f''.scaleR
      unfolding scaleR_right_distrib scaleR_minus_right
      by auto
    also have ... = e
      unfolding e_def
      using c(1)
      using norm_minus_cancel[of f' i - f'' i]
      by auto
  finally show False

```

```

    using c
    using d(2)[of x + c *R i]
    unfolding dist_norm
    unfolding f'.scaleR f''.scaleR f'.add f''.add f'.diff f''.diff
      scaleR_scaleR scaleR_right_diff_distrib scaleR_right_distrib
    using i
    by (auto simp: inverse_eq_divide)
  qed
qed
qed

proposition frechet_derivative_unique_within_closed_interval:
  fixes f::'a::euclidean_space  $\Rightarrow$  'b::real_normed_vector
  assumes ab:  $\bigwedge i. i \in \text{Basis} \implies a \cdot i < b \cdot i$ 
    and x:  $x \in \text{cbox } a \ b$ 
    and (f has_derivative f') (at x within cbox a b)
    and (f has_derivative f'') (at x within cbox a b)
  shows f' = f''
proof (rule frechet_derivative_unique_within)
  fix e :: real
  fix i :: 'a
  assume e > 0 and i: i  $\in$  Basis
  then show  $\exists d. 0 < |d| \wedge |d| < e \wedge x + d *_{\mathbb{R}} i \in \text{cbox } a \ b$ 
  proof (cases x·i = a·i)
    case True
    with ab[of i]  $\langle e > 0 \rangle$  x i show ?thesis
    by (rule_tac x=(min (b·i - a·i) e) / 2 in exI)
      (auto simp add: mem_box field_simps inner_simps inner_Basis)
    next
    case False
    moreover have a · i < x · i
    using False i mem_box(2) x by force
    moreover {
      have a · i * 2 + min (x · i - a · i) e  $\leq$  a·i * 2 + x·i - a·i
      by auto
      also have ... = a·i + x·i
      by auto
      also have ...  $\leq$  2 * (x·i)
      using  $\langle a \cdot i < x \cdot i \rangle$  by auto
      finally have a · i * 2 + min (x · i - a · i) e  $\leq$  x · i * 2
      by auto
    }
    moreover have min (x · i - a · i) e  $\geq$  0
    by (simp add:  $\langle 0 < e \rangle \langle a \cdot i < x \cdot i \rangle$  less_eq_real_def)
  then have x · i * 2  $\leq$  b · i * 2 + min (x · i - a · i) e
  using i mem_box(2) x by force
  ultimately show ?thesis
  using ab[of i]  $\langle e > 0 \rangle$  x i
  by (rule_tac x=- (min (x·i - a·i) e) / 2 in exI)

```



```

      (auto simp add: mem_box field_simps inner_simps inner_Basis)
    qed
  qed (use assms in auto)

lemma frechet_derivative_unique_within_open_interval:
  fixes f::'a::euclidean_space  $\Rightarrow$  'b::real_normed_vector
  assumes x:  $x \in \text{box } a \ b$ 
    and f: (f has_derivative f') (at x within box a b) (f has_derivative f'') (at x
  within box a b)
  shows  $f' = f''$ 
proof -
  have at x within box a b = at x
    by (metis at_within_interior interior_open open_box)
  with f show  $f' = f''$ 
    by (simp add: has_derivative_unique)
qed

```

```

lemma frechet_derivative_at:
  (f has_derivative f') (at x)  $\implies f' = \text{frechet\_derivative } f \text{ (at } x)$ 
  using differentiable_def frechet_derivative_works has_derivative_unique by blast

```

```

lemma frechet_derivative_compose:
  frechet_derivative (f o g) (at x) = frechet_derivative (f) (at (g x)) o frechet_derivative
  g (at x)
  if g differentiable at x f differentiable at (g x)
  by (metis diff_chain_at frechet_derivative_at frechet_derivative_works that)

```

```

lemma frechet_derivative_within_cbox:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::real_normed_vector
  assumes  $\bigwedge i. i \in \text{Basis} \implies a \cdot i < b \cdot i$ 
    and  $x \in \text{cbox } a \ b$ 
    and (f has_derivative f') (at x within cbox a b)
  shows frechet_derivative f (at x within cbox a b) = f'
  using assms
  by (metis Derivative.differentiableI frechet_derivative_unique_within_closed_interval
  frechet_derivative_works)

```

```

lemma frechet_derivative_transform_within_open:
  frechet_derivative f (at x) = frechet_derivative g (at x)
  if f differentiable at x open X  $x \in X \bigwedge x. x \in X \implies f x = g x$ 
  by (meson frechet_derivative_at frechet_derivative_works has_derivative_transform_within_open
  that)

```

4.10.9 Derivatives of local minima and maxima are zero

```

lemma has_derivative_local_min:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  real
  assumes deriv: (f has_derivative f') (at x)
  assumes min: eventually ( $\lambda y. f x \leq f y$ ) (at x)

```

```

shows  $f' = (\lambda h. 0)$ 
proof
  fix  $h :: 'a$ 
  interpret  $f'$ : bounded_linear  $f'$ 
  using deriv by (rule has_derivative_bounded_linear)
  show  $f' h = 0$ 
  proof (cases  $h = 0$ )
    case False
    from min obtain  $d$  where  $d1: 0 < d$  and  $d2: \forall y \in \text{ball } x \ d. f x \leq f y$ 
    unfolding eventually_at by (force simp: dist_commute)
    have FDERIV  $(\lambda r. x + r *_{\mathbb{R}} h) \ 0 :> (\lambda r. r *_{\mathbb{R}} h)$ 
    by (intro derivative_eq_intros) auto
    then have FDERIV  $(\lambda r. f (x + r *_{\mathbb{R}} h)) \ 0 :> (\lambda k. f' (k *_{\mathbb{R}} h))$ 
    by (rule has_derivative_compose, simp add: deriv)
    then have DERIV  $(\lambda r. f (x + r *_{\mathbb{R}} h)) \ 0 :> f' h$ 
    unfolding has_field_derivative_def by (simp add:  $f'.\text{scaleR mult\_commute\_abs}$ )
    moreover have  $0 < d / \text{norm } h$  using  $d1$  and  $\langle h \neq 0 \rangle$  by simp
    moreover have  $\forall y. |0 - y| < d / \text{norm } h \longrightarrow f (x + 0 *_{\mathbb{R}} h) \leq f (x + y *_{\mathbb{R}} h)$ 
    using  $\langle h \neq 0 \rangle$  by (auto simp add:  $d2 \text{ dist\_norm pos\_less\_divide\_eq}$ )
    ultimately show  $f' h = 0$ 
    by (rule DERIV\_local\_min)
  qed simp
qed

lemma has_derivative_local_max:
  fixes  $f :: 'a::\text{real\_normed\_vector} \Rightarrow \text{real}$ 
  assumes  $(f \text{ has\_derivative } f') (at \ x)$ 
  assumes eventually  $(\lambda y. f y \leq f x) (at \ x)$ 
  shows  $f' = (\lambda h. 0)$ 
  using has_derivative_local_min [of  $\lambda x. - f x \ \lambda h. - f' h \ x$ ]
  using assms unfolding fun_eq_iff by simp

lemma differential_zero_maxmin:
  fixes  $f :: 'a::\text{real\_normed\_vector} \Rightarrow \text{real}$ 
  assumes  $x \in S$ 
  and open  $S$ 
  and deriv:  $(f \text{ has\_derivative } f') (at \ x)$ 
  and mono:  $(\forall y \in S. f y \leq f x) \vee (\forall y \in S. f x \leq f y)$ 
  shows  $f' = (\lambda v. 0)$ 
  using mono
proof
  assume  $\forall y \in S. f y \leq f x$ 
  with  $\langle x \in S \rangle$  and  $\langle \text{open } S \rangle$  have eventually  $(\lambda y. f y \leq f x) (at \ x)$ 
  unfolding eventually_at_topological by auto
  with deriv show ?thesis
  by (rule has_derivative_local_max)
next
  assume  $\forall y \in S. f x \leq f y$ 

```

```

with  $\langle x \in S \rangle$  and  $\langle \text{open } S \rangle$  have eventually  $(\lambda y. f\ x \leq f\ y)$   $(\text{at } x)$ 
  unfolding eventually_at_topological by auto
with deriv show ?thesis
  by  $(\text{rule } \text{has\_derivative\_local\_min})$ 
qed

lemma differential_zero_maxmin_component:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes  $k: k \in \text{Basis}$ 
  and  $\text{ball}: 0 < e \ (\forall y \in \text{ball } x\ e. (f\ y) \cdot k \leq (f\ x) \cdot k) \vee (\forall y \in \text{ball } x\ e. (f\ x) \cdot k \leq (f\ y) \cdot k)$ 
  and diff:  $f$  differentiable  $(\text{at } x)$ 
  shows  $(\sum_{j \in \text{Basis}. (\text{frechet\_derivative } f\ (\text{at } x)\ j \cdot k) *_{\mathbb{R}} j) = (0::'a)$  (is ?D k = 0)
proof –
  let  $?f' = \text{frechet\_derivative } f\ (\text{at } x)$ 
  have  $x \in \text{ball } x\ e$  using  $\langle 0 < e \rangle$  by simp
  moreover have open  $(\text{ball } x\ e)$  by simp
  moreover have  $((\lambda x. f\ x \cdot k) \text{ has\_derivative } (\lambda h. ?f' h \cdot k))\ (\text{at } x)$ 
    using bounded_linear_inner_left diff[unfolded frechet_derivative_works]
    by  $(\text{rule } \text{bounded\_linear.has\_derivative})$ 
  ultimately have  $(\lambda h. \text{frechet\_derivative } f\ (\text{at } x)\ h \cdot k) = (\lambda v. 0)$ 
    using ball(2) by  $(\text{rule } \text{differential\_zero\_maxmin})$ 
  then show ?thesis
    unfolding fun_eq_iff by simp
qed

```

4.10.10 One-dimensional mean value theorem

```

lemma mvt_simple:
  fixes  $f :: \text{real} \Rightarrow \text{real}$ 
  assumes  $a < b$ 
  and derf:  $\bigwedge x. [a \leq x; x \leq b] \Longrightarrow (f \text{ has\_derivative } f' x)\ (\text{at } x \text{ within } \{a..b\})$ 
  shows  $\exists x \in \{a <..<b\}. f\ b - f\ a = f' x\ (b - a)$ 
proof  $(\text{rule } \text{mvt})$ 
  have  $f$  differentiable_on  $\{a..b\}$ 
    using derf unfolding differentiable_on_def differentiable_def by force
  then show continuous_on  $\{a..b\}$   $f$ 
    by  $(\text{rule } \text{differentiable\_imp\_continuous\_on})$ 
  show  $(f \text{ has\_derivative } f' x)\ (\text{at } x)$  if  $a < x < b$  for  $x$ 
    by  $(\text{metis } \text{at\_within\_Icc\_at\_derf } \text{leI } \text{order.asym } \text{that})$ 
qed  $(\text{use } \text{assms } \text{in } \text{auto})$ 

```

```

lemma mvt_very_simple:
  fixes  $f :: \text{real} \Rightarrow \text{real}$ 
  assumes  $a \leq b$ 
  and derf:  $\bigwedge x. [a \leq x; x \leq b] \Longrightarrow (f \text{ has\_derivative } f' x)\ (\text{at } x \text{ within } \{a..b\})$ 
  shows  $\exists x \in \{a..b\}. f\ b - f\ a = f' x\ (b - a)$ 
proof  $(\text{cases } a = b)$ 

```

```

interpret bounded_linear f' b
  using assms(2) assms(1) by auto
case True
then show ?thesis
  by force
next
case False
then show ?thesis
  using mvt_simple[OF _ derf]
  by (metis ‹a ≤ b› atLeastAtMost_iff dual_order.order_iff_strict greaterThanLessThan_iff)
qed

```

A nice generalization (see Havin's proof of 5.19 from Rudin's book).

```

lemma mvt_general:
  fixes f :: real ⇒ 'a::real_inner
  assumes a < b
    and contf: continuous_on {a..b} f
    and derf: ⋀x. ‹a < x; x < b› ⇒ (f has_derivative f' x) (at x)
  shows ∃x∈{a<..2
      by (simp add: power2_eq_square)
    also have ... = (f b - f a) · (f b - f a)
      unfolding power2_norm_eq_inner ..
    also have ... = (f b - f a) · f' x (b - a)
      using x(2) by (simp only: inner_diff_right)
    also have ... ≤ norm (f b - f a) * norm (f' x (b - a))
      by (rule norm_cauchy_schwarz)
    finally show ?thesis
      using False x(1)
      by (auto simp add: mult_left_cancel)
  next
    case True
    then show ?thesis
      using ‹a < b› by (rule_tac x=(a + b) / 2 in bexI) auto
  qed
qed

```

4.10.11 More general bound theorems

proposition *differentiable_bound_general*:

fixes $f :: \text{real} \Rightarrow 'a::\text{real\_normed\_vector}$

assumes $a < b$

and $f\_cont: \text{continuous\_on } \{a..b\} f$

and $\phi\_cont: \text{continuous\_on } \{a..b\} \phi$

and $f': \bigwedge x. a < x \implies x < b \implies (f \text{ has\_vector\_derivative } f' x) \text{ (at } x)$

and $\phi': \bigwedge x. a < x \implies x < b \implies (\phi \text{ has\_vector\_derivative } \phi' x) \text{ (at } x)$

and $bnd: \bigwedge x. a < x \implies x < b \implies \text{norm } (f' x) \leq \phi' x$

shows $\text{norm } (f b - f a) \leq \phi b - \phi a$

proof –

```

{
  fix x assume x: a < x x < b
  have 0 ≤ norm (f' x) by simp
  also have ... ≤ ϕ' x using x by (auto intro!: bnd)
  finally have 0 ≤ ϕ' x .
} note phi'_nonneg = this
note f_tendsto = assms(2)[simplified continuous_on_def, rule_format]
note phi_tendsto = assms(3)[simplified continuous_on_def, rule_format]
{
  fix e::real assume e > 0
  define e2 where e2 = e / 2
  with ⟨e > 0⟩ have e2 > 0 by simp
  let ?le = λx1. norm (f x1 - f a) ≤ ϕ x1 - ϕ a + e * (x1 - a) + e
  define A where A = {x2. a ≤ x2 ∧ x2 ≤ b ∧ (∀x1∈{a..

```

```

ultimately
show ?thesis
  by (rule tendsto_le)
qed simp
} note le_cont = this
have a ∈ A
  using assms by (auto simp: A_def)
hence [simp]: A ≠ {} by auto
have A_ivl:  $\bigwedge x1\ x2. x2 \in A \implies x1 \in \{a \dots x2\} \implies x1 \in A$ 
  by (simp add: A_def)
have [simp]: bdd_above A by (auto simp: A_def)
define y where y = Sup A
have y ≤ b
  unfolding y_def
  by (simp add: cSup_le_iff) (simp add: A_def)
have leI:  $\bigwedge x\ x1. a \leq x1 \implies x \in A \implies x1 < x \implies ?le\ x1$ 
  by (auto simp: A_def intro!: le_cont)
have y_all_le:  $\forall x1 \in \{a \dots y\}. ?le\ x1$ 
  by (auto simp: y_def less_cSup_iff leI)
have a ≤ y
  by (metis ⟨a ∈ A⟩ ⟨bdd_above A⟩ cSup_upper y_def)
have y ∈ A
  using y_all_le ⟨a ≤ y⟩ ⟨y ≤ b⟩
  by (auto simp: A_def)
hence A = {a .. y}
  using A_subset by (auto simp: subset_iff y_def cSup_upper intro: A_ivl)
from le_cont[OF ⟨a ≤ y⟩ ⟨y ≤ b⟩ y_all_le] have le_y: ?le y .
have y = b
proof (cases a = y)
case True
  with ⟨a < b⟩ have y < b by simp
  with ⟨a = y⟩ f_cont phi_cont ⟨e2 > 0⟩
  have 1:  $\forall_F\ x\ \text{in}\ \text{at}\ y\ \text{within}\ \{y..b\}. \text{dist}\ (f\ x)\ (f\ y) < e2$ 
  and 2:  $\forall_F\ x\ \text{in}\ \text{at}\ y\ \text{within}\ \{y..b\}. \text{dist}\ (\varphi\ x)\ (\varphi\ y) < e2$ 
  by (auto simp: continuous_on_def tendsto_iff)
  have 3: eventually (λx. y < x) (at y within {y..b})
  by (auto simp: eventually_at_filter)
  have 4: eventually (λx::real. x < b) (at y within {y..b})
  using _ ⟨y < b⟩
  by (rule order_tendstoD) (auto intro!: tendsto_eq_intros)
  from 1 2 3 4
  have eventually_le: eventually (λx. ?le x) (at y within {y .. b})
proof eventually_elim
case (elim x1)
  have norm (f x1 - f a) = norm (f x1 - f y)
  by (simp add: ⟨a = y⟩)
  also have norm (f x1 - f y) ≤ e2
  using elim ⟨a = y⟩ by (auto simp: dist_norm intro!: less_imp_le)
  also have ... ≤ e2 + (φ x1 - φ a + e2 + e * (x1 - a))

```

```

    using ‹0 < e› elim
  by (intro add_increasing2[OF add_nonneg_nonneg order.refl])
    (auto simp: ‹a = y› dist_norm intro!: mult_nonneg_nonneg)
  also have ... =  $\varphi\ x1 - \varphi\ a + e * (x1 - a) + e$ 
    by (simp add: e2_def)
  finally show ?le x1 .
qed
from this[unfolded eventually_at_topological] ‹?le y›
obtain S where S: open S y  $\in$  S  $\wedge$  x. x  $\in$  S  $\implies$  x  $\in$  {y..b}  $\implies$  ?le x
  by metis
from ‹open S› obtain d where d:  $\wedge$ x. dist x y < d  $\implies$  x  $\in$  S d > 0
  by (force simp: dist_commute open_dist ball_def dest!: bspec[OF _ ‹y  $\in$ 
S›])
define d' where d' = min b (y + (d/2))
have d'  $\in$  A
  unfolding A_def
proof safe
  show a  $\leq$  d' using ‹a = y› ‹0 < d› ‹y < b› by (simp add: d'_def)
  show d'  $\leq$  b by (simp add: d'_def)
  fix x1
  assume x1  $\in$  {a.. $d'$ }
  hence x1  $\in$  S x1  $\in$  {y..b}
    by (auto simp: ‹a = y› d'_def dist_real_def intro!: d )
  thus ?le x1
    by (rule S)
qed
hence d'  $\leq$  y
  unfolding y_def
  by (rule cSup_upper) simp
then show y = b using ‹d > 0› ‹y < b›
  by (simp add: d'_def)
next
case False
with ‹a  $\leq$  y› have a < y by simp
show y = b
proof (rule ccontr)
  assume y  $\neq$  b
  hence y < b using ‹y  $\leq$  b› by simp
  let ?F = at y within {y.. $b$ }
  from f' phi'
  have (f has_vector_derivative f' y) ?F
    and ( $\varphi$  has_vector_derivative  $\varphi'$  y) ?F
    using ‹a < y› ‹y < b›
  by (auto simp add: at_within_open[of _ {a<.. $b$ }] has_vector_derivative_def
    intro!: has_derivative_subset[where s={a<.. $b$ } and t={y.. $b$ }] )
  hence  $\forall_F\ x1\ in\ ?F.$  norm (f x1 - f y - (x1 - y) *R f' y)  $\leq$  e2 * |x1 - y|
     $\forall_F\ x1\ in\ ?F.$  norm ( $\varphi$  x1 -  $\varphi$  y - (x1 - y) *R  $\varphi'$  y)  $\leq$  e2 * |x1 - y|
    using ‹e2 > 0›
  by (auto simp: has_derivative_within_alt2 has_vector_derivative_def)

```

```

moreover
have  $\forall_F x1 \text{ in } ?F. y \leq x1 \ \forall_F x1 \text{ in } ?F. x1 < b$ 
  by (auto simp: eventually_at_filter)
ultimately
have  $\forall_F x1 \text{ in } ?F. \text{norm } (f x1 - f y) \leq (\varphi x1 - \varphi y) + e * |x1 - y|$ 
  (is  $\forall_F x1 \text{ in } ?F. ?le' x1$ )
proof eventually_elim
  case (elim x1)
  from norm_triangle_ineq2[THEN order_trans, OF elim(1)]
  have  $\text{norm } (f x1 - f y) \leq \text{norm } (f' y) * |x1 - y| + e2 * |x1 - y|$ 
    by (simp add: ac_simps)
  also have  $\text{norm } (f' y) \leq \varphi' y$  using bnd  $\langle a < y \rangle \langle y < b \rangle$  by simp
  also have  $\varphi' y * |x1 - y| \leq \varphi x1 - \varphi y + e2 * |x1 - y|$ 
    using elim by (simp add: ac_simps)
  finally
  have  $\text{norm } (f x1 - f y) \leq \varphi x1 - \varphi y + e2 * |x1 - y| + e2 * |x1 - y|$ 
    by (auto simp: mult_right_mono)
  thus ?case by (simp add: e2_def)
qed
moreover have ?le' y by simp
ultimately obtain S
where S: open S y  $\in S \wedge x. x \in S \implies x \in \{y..<b\} \implies ?le' x$ 
  unfolding eventually_at_topological
  by metis
from  $\langle \text{open } S \rangle$  obtain d where d:  $\wedge x. \text{dist } x y < d \implies x \in S \ d > 0$ 
  by (force simp: dist_commute open_dist ball_def dest!: bspec[OF _  $\langle y \in S \rangle$ ])
define d' where d' = min ((y + b)/2) (y + (d/2))
have d'  $\in A$ 
  unfolding A_def
proof safe
  show a  $\leq d'$  using  $\langle a < y \rangle \langle 0 < d \rangle \langle y < b \rangle$  by (simp add: d'_def)
  show d'  $\leq b$  using  $\langle y < b \rangle$  by (simp add: d'_def min_def)
  fix x1
  assume x1: x1  $\in \{a..<d'\}$ 
  show ?le x1
  proof (cases x1 < y)
  case True
  then show ?thesis
    using  $\langle y \in A \rangle \text{local.leI } x1$  by auto
  next
  case False
  hence x1': x1  $\in S \ x1 \in \{y..<b\}$  using x1
    by (auto simp: d'_def dist_real_def intro!: d)
  have  $\text{norm } (f x1 - f a) \leq \text{norm } (f x1 - f y) + \text{norm } (f y - f a)$ 
    by (rule order_trans[OF _ norm_triangle_ineq]) simp
  also note S(3)[OF x1']
  also note le_y
  finally show ?le x1

```



```

      using False by (auto simp: algebra_simps)
    qed
  qed
  hence  $d' \leq y$ 
    unfolding  $y\_def$  by (rule  $cSup\_upper$ ) simp
  thus False using  $\langle d > 0 \rangle \langle y < b \rangle$ 
    by (simp add:  $d'\_def$   $min\_def$   $split: if\_split\_asm$ )
  qed
  qed
  with  $le\_y$  have  $norm (f b - f a) \leq \varphi b - \varphi a + e * (b - a + 1)$ 
    by (simp add: algebra_simps)
} note * = this
show ?thesis
proof (rule  $field\_le\_epsilon$ )
  fix  $e::real$  assume  $e > 0$ 
  then show  $norm (f b - f a) \leq \varphi b - \varphi a + e$ 
    using  $*[of e / (b - a + 1)] \langle a < b \rangle$  by simp
  qed
qed

```

lemma *differentiable_bound*:

```

  fixes  $f :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$ 
  assumes  $convex S$ 
    and  $derf: \bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } S)$ 
    and  $B: \bigwedge x. x \in S \implies onorm (f' x) \leq B$ 
    and  $x: x \in S$ 
    and  $y: y \in S$ 
  shows  $norm (f x - f y) \leq B * norm (x - y)$ 
proof -
  let ?p =  $\lambda u. x + u *_R (y - x)$ 
  let ? $\varphi$  =  $\lambda h. h * B * norm (x - y)$ 
  have *:  $x + u *_R (y - x) \in S$  if  $u \in \{0..1\}$  for  $u$ 
  proof -
    have  $u *_R y = u *_R (y - x) + u *_R x$ 
    by (simp add:  $scale\_right\_diff\_distrib$ )
    then show  $x + u *_R (y - x) \in S$ 
    using that  $\langle convex S \rangle x y$  by (simp add:  $convex\_alt$ )
      (metis  $pth\_b(2)$   $pth\_c(1)$   $scaleR\_collapse$ )
  qed
  have  $\bigwedge z. z \in (\lambda u. x + u *_R (y - x))^{-1} \{0..1\} \implies$ 
     $(f \text{ has\_derivative } f' z) \text{ (at } z \text{ within } (\lambda u. x + u *_R (y - x))^{-1} \{0..1\})$ 
  by (auto intro: * has_derivative_subset [OF derf])
  then have  $continuous\_on$  ( $?p^{-1} \{0..1\}$ )  $f$ 
    unfolding  $continuous\_on\_eq\_continuous\_within$ 
    by (meson has_derivative_continuous)
  with * have 1:  $continuous\_on \{0 .. 1\} (f \circ ?p)$ 
    by (intro continuous_intros)+
  {
    fix  $u::real$  assume  $u: u \in \{0 <..< 1\}$ 

```

```

    let ?u = ?p u
    interpret linear (f' ?u)
    using u by (auto intro!: has_derivative_linear derf *)
    have (f ∘ ?p has_derivative (f' ?u) ∘ (λu. 0 + u *R (y - x))) (at u within box
0 1)
    by (intro derivative_intros has_derivative_subset [OF derf]) (use u * in auto)
    hence ((f ∘ ?p) has_vector_derivative f' ?u (y - x)) (at u)
    by (simp add: at_within_open[OF u open_greaterThanLessThan] scaleR
has_vector_derivative_def o_def)
  } note 2 = this
  have 3: continuous_on {0..1} ?φ
  by (rule continuous_intros)+
  have 4: (?φ has_vector_derivative B * norm (x - y)) (at u) for u
  by (auto simp: has_vector_derivative_def intro!: derivative_eq_intros)
  {
    fix u::real assume u: u ∈ {0 <..< 1}
    let ?u = ?p u
    interpret bounded_linear (f' ?u)
    using u by (auto intro!: has_derivative_bounded_linear derf *)
    have norm (f' ?u (y - x)) ≤ onorm (f' ?u) * norm (y - x)
    by (rule onorm) (rule bounded_linear)
    also have onorm (f' ?u) ≤ B
    using u by (auto intro!: assms(3)[rule_format] *)
    finally have norm ((f' ?u) (y - x)) ≤ B * norm (x - y)
    by (simp add: mult_right_mono norm_minus_commute)
  } note 5 = this
  have norm (f x - f y) = norm ((f ∘ (λu. x + u *R (y - x))) 1 - (f ∘ (λu. x
+ u *R (y - x))) 0)
  by (auto simp add: norm_minus_commute)
  also
  from differentiable_bound_general[OF zero_less_one 1, OF 3 2 4 5]
  have norm ((f ∘ ?p) 1 - (f ∘ ?p) 0) ≤ B * norm (x - y)
  by simp
  finally show ?thesis .
qed

```

```

lemma field_differentiable_bound:
  fixes S :: 'a::real_normed_field set
  assumes cvs: convex S
    and df:  $\bigwedge z. z \in S \implies (f \text{ has\_field\_derivative } f' z) \text{ (at } z \text{ within } S)$ 
    and dn:  $\bigwedge z. z \in S \implies \text{norm } (f' z) \leq B$ 
    and x ∈ S y ∈ S
  shows norm(f x - f y) ≤ B * norm(x - y)
  apply (rule differentiable_bound [OF cvs])
  apply (erule df [unfolded has_field_derivative_def])
  apply (rule onorm_le, simp_all add: norm_mult mult_right_mono assms)
  done

```

lemma

```

differentiable_bound_segment:
fixes f::'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
assumes  $\bigwedge t. t \in \{0..1\} \implies x0 + t *_R a \in G$ 
assumes f':  $\bigwedge x. x \in G \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } G)$ 
assumes B:  $\bigwedge x. x \in \{0..1\} \implies \text{onorm } (f' (x0 + x *_R a)) \leq B$ 
shows  $\text{norm } (f (x0 + a) - f x0) \leq \text{norm } a * B$ 
proof -
  let ?G =  $(\lambda x. x0 + x *_R a) \text{ ` } \{0..1\}$ 
  have ?G =  $(+) x0 \text{ ` } (\lambda x. x *_R a) \text{ ` } \{0..1\}$  by auto
  also have convex ...
    by (intro convex_translation convex_scaled convex_real_interval)
  finally have convex ?G .
  moreover have  $?G \subseteq G$   $x0 \in ?G$   $x0 + a \in ?G$  using assms by (auto intro:
image_eqI[where x=1])
  ultimately show ?thesis
    using has_derivative_subset[OF f'  $\langle ?G \subseteq G \rangle$ ] B
      differentiable_bound[of  $(\lambda x. x0 + x *_R a) \text{ ` } \{0..1\}$ ] f f' B x0 + a x0]
    by (force simp: ac_simps)
qed

```

```

lemma differentiable_bound_linearization:
fixes f::'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
assumes S:  $\bigwedge t. t \in \{0..1\} \implies a + t *_R (b - a) \in S$ 
assumes f'[derivative_intros]:  $\bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } S)$ 
assumes B:  $\bigwedge x. x \in S \implies \text{onorm } (f' x - f' x0) \leq B$ 
assumes x0  $\in S$ 
shows  $\text{norm } (f b - f a - f' x0 (b - a)) \leq \text{norm } (b - a) * B$ 
proof -
  define g where [abs_def]:  $g x = f x - f' x0 x$  for x
  have g:  $\bigwedge x. x \in S \implies (g \text{ has\_derivative } (\lambda i. f' x i - f' x0 i)) \text{ (at } x \text{ within } S)$ 
    unfolding g_def using assms
    by (auto intro!: derivative_eq_intros
      bounded_linear.has_derivative[OF has_derivative_bounded_linear, OF f'])
  from B have  $\forall x \in \{0..1\}. \text{onorm } (\lambda i. f' (a + x *_R (b - a)) i - f' x0 i) \leq B$ 
    using assms by (auto simp: fun_diff_def)
  with differentiable_bound_segment[OF S g]  $\langle x0 \in S \rangle$ 
  show ?thesis
    by (simp add: g_def field_simps linear_diff[OF has_derivative_linear[OF f']])
qed

```

```

lemma vector_differentiable_bound_linearization:
fixes f::real  $\Rightarrow$  'b::real_normed_vector
assumes f':  $\bigwedge x. x \in S \implies (f \text{ has\_vector\_derivative } f' x) \text{ (at } x \text{ within } S)$ 
assumes closed_segment a b  $\subseteq S$ 
assumes B:  $\bigwedge x. x \in S \implies \text{norm } (f' x - f' x0) \leq B$ 
assumes x0  $\in S$ 
shows  $\text{norm } (f b - f a - (b - a) *_R f' x0) \leq \text{norm } (b - a) * B$ 
using assms

```

```

by (intro differentiable_bound_linearization[of a b S f  $\lambda x h. h *_R f' x x0 B$ ])
  (force simp: closed_segment_real_eq has_vector_derivative_def
    scaleR_diff_right[symmetric] mult.commute[of B]
    intro!: onorm_le mult_left_mono)+

```

In particular.

```

lemma has_derivative_zero_constant:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes convex s
    and  $\bigwedge x. x \in s \implies (f \text{ has\_derivative } (\lambda h. 0)) \text{ (at } x \text{ within } s)$ 
  shows  $\exists c. \forall x \in s. f x = c$ 
proof -
  { fix x y assume  $x \in s \ y \in s$ 
    then have  $\text{norm } (f x - f y) \leq 0 * \text{norm } (x - y)$ 
      using assms by (intro differentiable_bound[of s]) (auto simp: onorm_zero)
    then have  $f x = f y$ 
      by simp }
  then show ?thesis
    by metis
qed

```

```

lemma has_field_derivative_zero_constant:
  assumes convex s  $\bigwedge x. x \in s \implies (f \text{ has\_field\_derivative } 0) \text{ (at } x \text{ within } s)$ 
  shows  $\exists c. \forall x \in s. f(x) = (c :: 'a :: \text{real\_normed\_field})$ 
proof (rule has_derivative_zero_constant)
  have A:  $(*) \ 0 = (\lambda_. 0 :: 'a) \text{ by (intro ext) simp}$ 
  fix x assume  $x \in s$  thus  $(f \text{ has\_derivative } (\lambda h. 0)) \text{ (at } x \text{ within } s)$ 
    using assms(2)[of x] by (simp add: has_field_derivative_def A)
qed fact

```

```

lemma
  has_vector_derivative_zero_constant:
  assumes convex s
    assumes  $\bigwedge x. x \in s \implies (f \text{ has\_vector\_derivative } 0) \text{ (at } x \text{ within } s)$ 
  obtains c where  $\bigwedge x. x \in s \implies f x = c$ 
  using has_derivative_zero_constant[of s f] assms
  by (auto simp: has_vector_derivative_def)

```

```

lemma has_derivative_zero_unique:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes convex s
    and  $\bigwedge x. x \in s \implies (f \text{ has\_derivative } (\lambda h. 0)) \text{ (at } x \text{ within } s)$ 
    and  $x \in s \ y \in s$ 
  shows  $f x = f y$ 
  using has_derivative_zero_constant[OF assms(1,2)] assms(3-) by force

```

```

lemma has_derivative_zero_unique_connected:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes open s connected s

```

```

assumes f:  $\bigwedge x. x \in s \implies (f \text{ has\_derivative } (\lambda x. 0)) \text{ (at } x)$ 
assumes x  $\in s$  y  $\in s$ 
shows f x = f y
proof (rule connected_local_const[where f=f, OF  $\langle \text{connected } s \rangle \langle x \in s \rangle \langle y \in s \rangle$ ])
show  $\forall a \in s. \text{eventually } (\lambda b. f a = f b) \text{ (at } a \text{ within } s)$ 
proof
fix a assume a  $\in s$ 
with  $\langle \text{open } s \rangle$  obtain e where  $0 < e$  ball a e  $\subseteq s$ 
by (rule openE)
then have  $\exists c. \forall x \in \text{ball } a \text{ e}. f x = c$ 
by (intro has_derivative_zero_constant)
(auto simp: at_within_open[OF open_ball] f)
with  $\langle 0 < e \rangle$  have  $\forall x \in \text{ball } a \text{ e}. f a = f x$ 
by auto
then show eventually  $(\lambda b. f a = f b) \text{ (at } a \text{ within } s)$ 
using  $\langle 0 < e \rangle$  unfolding eventually_at_topological
by (intro exI[of _ ball a e]) auto
qed
qed

```

4.10.12 Differentiability of inverse function (most basic form)

```

lemma has_derivative_inverse_basic:
fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
assumes derf: (f has_derivative f') (at (g y))
and ling': bounded_linear g'
and g'  $\circ$  f' = id
and contg: continuous (at y) g
and open T
and y  $\in T$ 
and fg:  $\bigwedge z. z \in T \implies f (g z) = z$ 
shows (g has_derivative g') (at y)
proof -
interpret f': bounded_linear f'
using assms unfolding has_derivative_def by auto
interpret g': bounded_linear g'
using assms by auto
obtain C where C:  $0 < C \bigwedge x. \text{norm } (g' x) \leq \text{norm } x * C$ 
using bounded_linear.pos_bounded[OF assms(2)] by blast
have lem1:  $\forall e > 0. \exists d > 0. \forall z. \text{norm } (z - y) < d \implies \text{norm } (g z - g y - g'(z - y)) \leq e * \text{norm } (g z - g y)$ 
proof (intro allI impI)
fix e :: real
assume e > 0
with C(1) have *: e / C > 0 by auto
obtain d0 where 0 < d0 and d0:
 $\bigwedge u. \text{norm } (u - g y) < d0 \implies \text{norm } (f u - f (g y) - f' (u - g y)) \leq e /$ 
C * norm (u - g y)
using derf * unfolding has_derivative_at_alt by blast

```

```

    obtain d1 where 0 < d1 and d1:  $\bigwedge x. \llbracket 0 < \text{dist } x \ y; \text{dist } x \ y < d1 \rrbracket \implies \text{dist } (g \ x) \ (g \ y) < d0$ 
    using contg  $\langle 0 < d0 \rangle$  unfolding continuous_at Lim_at by blast
    obtain d2 where 0 < d2 and d2:  $\bigwedge u. \text{dist } u \ y < d2 \implies u \in T$ 
    using  $\langle \text{open } T \rangle \langle y \in T \rangle$  unfolding open_dist by blast
    obtain d where d: 0 < d d < d1 d < d2
    using field_lbound_gt_zero[OF  $\langle 0 < d1 \rangle \langle 0 < d2 \rangle$ ] by blast
    show  $\exists d > 0. \forall z. \text{norm } (z - y) < d \longrightarrow \text{norm } (g \ z - g \ y - g' (z - y)) \leq e * \text{norm } (g \ z - g \ y)$ 
    proof (intro exI allI impI conjI)
      fix z
      assume as:  $\text{norm } (z - y) < d$ 
      then have z ∈ T
      using d2 d unfolding dist_norm by auto
      have  $\text{norm } (g \ z - g \ y - g' (z - y)) \leq \text{norm } (g' (f (g \ z) - y - f' (g \ z - g \ y)))$ 
      unfolding g'.diff f'.diff
      unfolding assms(3)[unfolded o_def id_def, THEN fun_cong] fg[OF  $\langle z \in T \rangle$ ]
      by (simp add: norm_minus_commute)
      also have  $\dots \leq \text{norm } (f (g \ z) - y - f' (g \ z - g \ y)) * C$ 
      by (rule C(2))
      also have  $\dots \leq (e / C) * \text{norm } (g \ z - g \ y) * C$ 
      proof -
        have  $\text{norm } (g \ z - g \ y) < d0$ 
        by (metis as cancel_comm_monoid_add_class.diff_cancel d(2)  $\langle 0 < d0 \rangle$ )
      d1 diff_gt_0_iff_gt diff_strict_mono dist_norm dist_self zero_less_dist_iff)
      then show ?thesis
      by (metis C(1)  $\langle y \in T \rangle$  d0 fg mult_le_cancel_iff1)
    qed
    also have  $\dots \leq e * \text{norm } (g \ z - g \ y)$ 
    using C by (auto simp add: field_simps)
    finally show  $\text{norm } (g \ z - g \ y - g' (z - y)) \leq e * \text{norm } (g \ z - g \ y)$ 
    by simp
  qed (use d in auto)
qed
have *:  $(0::\text{real}) < 1 / 2$ 
by auto
obtain d where 0 < d and d:
 $\bigwedge z. \text{norm } (z - y) < d \implies \text{norm } (g \ z - g \ y - g' (z - y)) \leq 1/2 * \text{norm } (g \ z - g \ y)$ 
using lem1 * by blast
define B where B = C * 2
have B > 0
unfolding B_def using C by auto
have lem2:  $\text{norm } (g \ z - g \ y) \leq B * \text{norm } (z - y)$  if z:  $\text{norm } (z - y) < d$  for z
proof -
  have  $\text{norm } (g \ z - g \ y) \leq \text{norm } (g' (z - y)) + \text{norm } ((g \ z - g \ y) - g' (z - y))$ 
  by (rule norm_triangle_sub)
  also have  $\dots \leq \text{norm } (g' (z - y)) + 1 / 2 * \text{norm } (g \ z - g \ y)$ 

```

```

    by (rule add_left_mono) (use d z in auto)
  also have ... ≤ norm (z - y) * C + 1 / 2 * norm (g z - g y)
    by (rule add_right_mono) (use C in auto)
  finally show norm (g z - g y) ≤ B * norm (z - y)
    unfolding B_def
    by (auto simp add: field_simps)
qed
show ?thesis
  unfolding has_derivative_at_alt
proof (intro conjI assms allI impI)
  fix e :: real
  assume e > 0
  then have *: e / B > 0 by (metis ‹B > 0› divide_pos_pos)
  obtain d' where 0 < d' and d':
    (∧ z. norm (z - y) < d' ⇒ norm (g z - g y - g' (z - y)) ≤ e / B * norm
(g z - g y))
    using lem1 * by blast
  obtain k where k: 0 < k k < d k < d'
    using field_lbound_gt_zero[OF ‹0 < d› ‹0 < d'›] by blast
  show ∃ d > 0. ∀ ya. norm (ya - y) < d ⟶ norm (g ya - g y - g' (ya - y))
≤ e * norm (ya - y)
  proof (intro exI allI impI conjI)
    fix z
    assume as: norm (z - y) < k
    then have norm (g z - g y - g' (z - y)) ≤ e / B * norm (g z - g y)
      using d' k by auto
    also have ... ≤ e * norm (z - y)
      unfolding times_divide_eq_left pos_divide_le_eq[OF ‹B > 0›]
      using lem2[of z] k as ‹e > 0›
      by (auto simp add: field_simps)
    finally show norm (g z - g y - g' (z - y)) ≤ e * norm (z - y)
      by simp
  qed (use k in auto)
qed
qed

```

Inverse function theorem for complex derivatives

lemma *has_field_derivative_inverse_basic*:

shows $DERIV f (g y) :> f' \implies$

$f' \neq 0 \implies$

$continuous (at y) g \implies$

$open t \implies$

$y \in t \implies$

$(\bigwedge z. z \in t \implies f (g z) = z)$

$\implies DERIV g y :> inverse (f')$

unfolding *has_field_derivative_def*

apply (rule *has_derivative_inverse_basic*)

apply (auto simp: *bounded_linear_mult_right*)

done

Simply rewrite that based on the domain point x .

```

lemma has_derivative_inverse_basic_x:
  fixes  $f :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$ 
  assumes  $(f \text{ has\_derivative } f') \text{ (at } x)$ 
    and bounded_linear  $g'$ 
    and  $g' \circ f' = id$ 
    and continuous  $(\text{at } (f\ x))\ g$ 
    and  $g\ (f\ x) = x$ 
    and open  $T$ 
    and  $f\ x \in T$ 
    and  $\bigwedge y. y \in T \implies f\ (g\ y) = y$ 
  shows  $(g \text{ has\_derivative } g') \text{ (at } (f\ x))$ 
  by (rule has_derivative_inverse_basic) (use assms in auto)

```

This is the version in Dieudonné', assuming continuity of f and g .

```

lemma has_derivative_inverse_dieudonne:
  fixes  $f :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$ 
  assumes open  $S$ 
    and open  $(f\ 'S)$ 
    and continuous_on  $S\ f$ 
    and continuous_on  $(f\ 'S)\ g$ 
    and  $\bigwedge x. x \in S \implies g\ (f\ x) = x$ 
    and  $x \in S$ 
    and  $(f \text{ has\_derivative } f') \text{ (at } x)$ 
    and bounded_linear  $g'$ 
    and  $g' \circ f' = id$ 
  shows  $(g \text{ has\_derivative } g') \text{ (at } (f\ x))$ 
  apply (rule has_derivative_inverse_basic_x[OF assms(7-9)  $\_\_\_$  assms(2)])
  using assms(3-6)
  unfolding continuous_on_eq_continuous_at[OF assms(1)] continuous_on_eq_continuous_at[OF
assms(2)]
  apply auto
  done

```

Here's the simplest way of not assuming much about g .

```

proposition has_derivative_inverse:
  fixes  $f :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$ 
  assumes compact  $S$ 
    and  $x \in S$ 
    and  $fx: f\ x \in interior\ (f\ 'S)$ 
    and continuous_on  $S\ f$ 
    and  $gf: \bigwedge y. y \in S \implies g\ (f\ y) = y$ 
    and  $(f \text{ has\_derivative } f') \text{ (at } x)$ 
    and bounded_linear  $g'$ 
    and  $g' \circ f' = id$ 
  shows  $(g \text{ has\_derivative } g') \text{ (at } (f\ x))$ 
proof -
  have *:  $\bigwedge y. y \in interior\ (f\ 'S) \implies f\ (g\ y) = y$ 
    by (metis gf_image_iff interior_subset subsetCE)

```



```

show ?thesis
  apply (rule has_derivative_inverse_basic_x[OF assms(6-8), where  $T =$ 
interior (f ' S)])
  apply (rule continuous_on_interior[OF _ fx])
  apply (rule continuous_on_inv)
  apply (simp_all add: assms *)
  done
qed

```

Invertible derivative continuous at a point implies local injectivity. It's only for this we need continuity of the derivative, except of course if we want the fact that the inverse derivative is also continuous. So if we know for some other reason that the inverse function exists, it's OK.

proposition has_derivative_locally_injective:

```

fixes f :: 'n::euclidean_space  $\Rightarrow$  'm::euclidean_space
assumes a  $\in$  S
  and open S
  and bling: bounded_linear g'
  and g'  $\circ$  f' a = id
  and derf:  $\bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x) \text{ (at } x)$ 
  and  $\bigwedge e. e > 0 \implies \exists d > 0. \forall x. \text{dist } a x < d \longrightarrow \text{onorm } (\lambda v. f' x v - f' a v)$ 
< e
obtains r where r > 0 ball a r  $\subseteq$  S inj_on f (ball a r)
proof -
  interpret bounded_linear g'
  using assms by auto
  note f'g' = assms(4)[unfolded id_def o_def, THEN cong]
  have g' (f' a ( $\sum$  Basis)) = ( $\sum$  Basis) ( $\sum$  Basis)  $\neq$  (0::'n)
  using f'g' by auto
  then have *: 0 < onorm g'
  unfolding onorm_pos_lt[OF assms(3)]
  by fastforce
  define k where k = 1 / onorm g' / 2
  have *: k > 0
  unfolding k_def using * by auto
  obtain d1 where d1:
    0 < d1
     $\bigwedge x. \text{dist } a x < d1 \implies \text{onorm } (\lambda v. f' x v - f' a v) < k$ 
  using assms(6) * by blast
  from ⟨open S⟩ obtain d2 where d2 > 0 ball a d2  $\subseteq$  S
  using ⟨a  $\in$  S⟩ ..
  obtain d2 where d2: 0 < d2 ball a d2  $\subseteq$  S
  using ⟨0 < d2⟩ ⟨ball a d2  $\subseteq$  S⟩ by blast
  obtain d where d: 0 < d d < d1 d < d2
  using field_lbound_gt_zero[OF d1(1) d2(1)] by blast
  show ?thesis
proof
  show 0 < d by (fact d)
  show ball a d  $\subseteq$  S

```

```

    using ⟨d < d2⟩ ⟨ball a d2 ⊆ S⟩ by auto
  show inj_on f (ball a d)
  unfolding inj_on_def
  proof (intro strip)
    fix x y
    assume as: x ∈ ball a d y ∈ ball a d f x = f y
    define ph where [abs_def]: ph w = w - g' (f w - f x) for w
    have ph': ph = g' ∘ (λw. f' a w - (f w - f x))
      unfolding ph_def o_def by (simp add: diff f' g')
    have norm (ph x - ph y) ≤ (1 / 2) * norm (x - y)
  proof (rule differentiable_bound[OF convex_ball _ _ as(1-2)])
    fix u
    assume u: u ∈ ball a d
    then have u ∈ S
      using d d2 by auto
    have *: (λv. v - g' (f' u v)) = g' ∘ (λw. f' a w - f' u w)
      unfolding o_def and diff
      using f' g' by auto
    have blin: bounded_linear (f' a)
      using ⟨a ∈ S⟩ derf by blast
    show (ph has_derivative (λv. v - g' (f' u v))) (at u within ball a d)
      unfolding ph' * comp_def
      by (rule ⟨u ∈ S⟩ derivative_eq_intros has_derivative_at_withinI [OF
derf] bounded_linear.has_derivative [OF blin] bounded_linear.has_derivative [OF
bling] |simp])+
    have **: bounded_linear (λx. f' u x - f' a x) bounded_linear (λx. f' a x
- f' u x)
      using ⟨u ∈ S⟩ blin bounded_linear_sub derf by auto
    then have onorm (λv. v - g' (f' u v)) ≤ onorm g' * onorm (λw. f' a w -
f' u w)
      by (simp add: * bounded_linear_axioms onorm_compose)
    also have ... ≤ onorm g' * k
      apply (rule mult_left_mono)
      using d1(2)[of u]
      using onorm_neg[where f=λx. f' u x - f' a x] d u onorm_pos_le[OF
bling] apply (auto simp: algebra_simps)
      done
    also have ... ≤ 1 / 2
      unfolding k_def by auto
    finally show onorm (λv. v - g' (f' u v)) ≤ 1 / 2 .
  qed
  moreover have norm (ph y - ph x) = norm (y - x)
    by (simp add: as(3) ph_def)
  ultimately show x = y
    unfolding norm_minus_commute by auto
  qed
qed
qed

```

4.10.13 Uniformly convergent sequence of derivatives

lemma *has_derivative_sequence_lipschitz_lemma:*

```

fixes  $f :: \text{nat} \Rightarrow 'a::\text{real\_normed\_vector} \Rightarrow 'b::\text{real\_normed\_vector}$ 
assumes  $\text{convex } S$ 
and  $\text{derf}: \bigwedge n x. x \in S \implies ((f\ n) \text{ has\_derivative } (f'\ n\ x)) \text{ (at } x \text{ within } S)$ 
and  $\text{nle}: \bigwedge n x h. \llbracket n \geq N; x \in S \rrbracket \implies \text{norm } (f'\ n\ x\ h - g'\ x\ h) \leq e * \text{norm } h$ 
and  $0 \leq e$ 
shows  $\forall m \geq N. \forall n \geq N. \forall x \in S. \forall y \in S. \text{norm } ((f\ m\ x - f\ n\ x) - (f\ m\ y - f\ n\ y))$ 
 $\leq 2 * e * \text{norm } (x - y)$ 
proof clarify
fix  $m\ n\ x\ y$ 
assume  $\text{as}: N \leq m \leq n, x \in S, y \in S$ 
show  $\text{norm } ((f\ m\ x - f\ n\ x) - (f\ m\ y - f\ n\ y)) \leq 2 * e * \text{norm } (x - y)$ 
proof (rule differentiable_bound [where  $f' = \lambda x h. f'\ m\ x\ h - f'\ n\ x\ h$ ,  $OF \langle \text{convex } S \rangle$  as  $(3-4)$ ])
fix  $x$ 
assume  $x \in S$ 
show  $((\lambda a. f\ m\ a - f\ n\ a) \text{ has\_derivative } (\lambda h. f'\ m\ x\ h - f'\ n\ x\ h)) \text{ (at } x \text{ within } S)$ 
by (rule derivative_intros derf  $\langle x \in S \rangle$ ) +
show  $\text{onorm } (\lambda h. f'\ m\ x\ h - f'\ n\ x\ h) \leq 2 * e$ 
proof (rule onorm_bound)
fix  $h$ 
have  $\text{norm } (f'\ m\ x\ h - f'\ n\ x\ h) \leq \text{norm } (f'\ m\ x\ h - g'\ x\ h) + \text{norm } (f'\ n\ x\ h - g'\ x\ h)$ 
using norm_triangle_ineq [of  $f'\ m\ x\ h - g'\ x\ h - f'\ n\ x\ h + g'\ x\ h$ ]
by (auto simp add: algebra_simps norm_minus_commute)
also have  $\dots \leq e * \text{norm } h + e * \text{norm } h$ 
using nle [ $OF \langle N \leq m \rangle \langle x \in S \rangle$ , of  $h$ ] nle [ $OF \langle N \leq n \rangle \langle x \in S \rangle$ , of  $h$ ]
by (auto simp add: field_simps)
finally show  $\text{norm } (f'\ m\ x\ h - f'\ n\ x\ h) \leq 2 * e * \text{norm } h$ 
by auto
qed (simp add:  $\langle 0 \leq e \rangle$ )
qed
qed

```

lemma *has_derivative_sequence_Lipschitz:*

```

fixes  $f :: \text{nat} \Rightarrow 'a::\text{real\_normed\_vector} \Rightarrow 'b::\text{real\_normed\_vector}$ 
assumes  $\text{convex } S$ 
and  $\bigwedge n x. x \in S \implies ((f\ n) \text{ has\_derivative } (f'\ n\ x)) \text{ (at } x \text{ within } S)$ 
and  $\text{nle}: \bigwedge e. e > 0 \implies \forall_F n \text{ in sequentially. } \forall x \in S. \forall h. \text{norm } (f'\ n\ x\ h - g'\ x\ h) \leq e * \text{norm } h$ 
and  $e > 0$ 
shows  $\exists N. \forall m \geq N. \forall n \geq N. \forall x \in S. \forall y \in S.$ 
 $\text{norm } ((f\ m\ x - f\ n\ x) - (f\ m\ y - f\ n\ y)) \leq e * \text{norm } (x - y)$ 
proof -
have  $2 * (e/2) = e$ 
using  $\langle e > 0 \rangle$  by auto
obtain  $N$  where  $\forall n \geq N. \forall x \in S. \forall h. \text{norm } (f'\ n\ x\ h - g'\ x\ h) \leq (e/2) * \text{norm } h$ 

```

```

h
  using nle <e > 0>
  unfolding eventually_sequentially
  by (metis less_divide_eq_numeral1(1) mult_zero_left)
  then show  $\exists N. \forall m \geq N. \forall n \geq N. \forall x \in S. \forall y \in S. \text{norm } (f\ m\ x - f\ n\ x - (f\ m\ y - f\ n\ y)) \leq e * \text{norm } (x - y)$ 
  apply (rule_tac x=N in exI)
  apply (rule has_derivative_sequence_lipschitz_lemma[where e=e/2, unfolded
*])
  using assms <e > 0>
  apply auto
  done
qed

```

```

proposition has_derivative_sequence:
  fixes f :: nat  $\Rightarrow$  'a::real_normed_vector  $\Rightarrow$  'b::banach
  assumes convex S
    and derf:  $\bigwedge n. x \in S \implies ((f\ n) \text{ has\_derivative } (f'\ n\ x)) \text{ (at } x \text{ within } S)$ 
    and nle:  $\bigwedge e. e > 0 \implies \forall_F n \text{ in sequentially. } \forall x \in S. \forall h. \text{norm } (f'\ n\ x\ h - g' x\ h) \leq e * \text{norm } h$ 
    and x0  $\in S$ 
    and lim:  $((\lambda n. f\ n\ x0) \longrightarrow l) \text{ sequentially}$ 
  shows  $\exists g. \forall x \in S. (\lambda n. f\ n\ x) \longrightarrow g\ x \wedge (g \text{ has\_derivative } g'(x)) \text{ (at } x \text{ within } S)$ 
proof -
  have lem1:  $\bigwedge e. e > 0 \implies \exists N. \forall m \geq N. \forall n \geq N. \forall x \in S. \forall y \in S. \text{norm } ((f\ m\ x - f\ n\ x) - (f\ m\ y - f\ n\ y)) \leq e * \text{norm } (x - y)$ 
  using assms(1,2,3) by (rule has_derivative_sequence_Lipschitz)
  have  $\exists g. \forall x \in S. ((\lambda n. f\ n\ x) \longrightarrow g\ x) \text{ sequentially}$ 
  proof (intro ballI bchoice)
    fix x
    assume x  $\in S$ 
    show  $\exists y. (\lambda n. f\ n\ x) \longrightarrow y$ 
    unfolding convergent_eq_Cauchy
    proof (cases x = x0)
      case True
        then show Cauchy  $(\lambda n. f\ n\ x)$ 
        using LIMSEQ_imp_Cauchy[OF lim] by auto
      next
        case False
        show Cauchy  $(\lambda n. f\ n\ x)$ 
        unfolding Cauchy_def
        proof (intro allI impI)
          fix e :: real
          assume e > 0
          hence  $e / 2 > 0 \wedge e / 2 / \text{norm } (x - x0) > 0$  using False by auto
          obtain M where M:  $\forall m \geq M. \forall n \geq M. \text{dist } (f\ m\ x0) (f\ n\ x0) < e / 2$ 
            using LIMSEQ_imp_Cauchy[OF lim] * unfolding Cauchy_def by blast
          obtain N where N:

```

```

     $\forall m \geq N. \forall n \geq N.$ 
     $\forall u \in S. \forall y \in S. \text{norm } (f \ m \ u - f \ n \ u - (f \ m \ y - f \ n \ y)) \leq$ 
     $e / 2 / \text{norm } (x - x0) * \text{norm } (u - y)$ 
using lem1 *(2) by blast
show  $\exists M. \forall m \geq M. \forall n \geq M. \text{dist } (f \ m \ x) (f \ n \ x) < e$ 
proof (intro exI allI impI)
  fix m n
  assume as:  $\max M \ N \leq m \ \max M \ N \leq n$ 
  have  $\text{dist } (f \ m \ x) (f \ n \ x) \leq \text{norm } (f \ m \ x0 - f \ n \ x0) + \text{norm } (f \ m \ x - f \ n$ 
 $x - (f \ m \ x0 - f \ n \ x0))$ 
    unfolding dist_norm
    by (rule norm_triangle_sub)
  also have  $\dots \leq \text{norm } (f \ m \ x0 - f \ n \ x0) + e / 2$ 
    using N  $\langle x \in S \rangle \langle x0 \in S \rangle$  as False by fastforce
  also have  $\dots < e / 2 + e / 2$ 
    by (rule add_strict_right_mono) (use as M in  $\langle \text{auto simp: dist\_norm} \rangle$ )
  finally show  $\text{dist } (f \ m \ x) (f \ n \ x) < e$ 
    by auto
qed
qed
qed
qed
then obtain g where  $g: \forall x \in S. (\lambda n. f \ n \ x) \longrightarrow g \ x ..$ 
have lem2:  $\exists N. \forall n \geq N. \forall x \in S. \forall y \in S. \text{norm } ((f \ n \ x - f \ n \ y) - (g \ x - g \ y)) \leq$ 
 $e * \text{norm } (x - y)$  if  $e > 0$  for e
proof -
  obtain N where
     $N: \forall m \geq N. \forall n \geq N. \forall x \in S. \forall y \in S. \text{norm } (f \ m \ x - f \ n \ x - (f \ m \ y - f \ n \ y))$ 
 $\leq e * \text{norm } (x - y)$ 
    using lem1  $\langle e > 0 \rangle$  by blast
  show  $\exists N. \forall n \geq N. \forall x \in S. \forall y \in S. \text{norm } (f \ n \ x - f \ n \ y - (g \ x - g \ y)) \leq e * \text{norm } (x - y)$ 
proof (intro exI ballI allI impI)
    fix n x y
    assume as:  $N \leq n \ x \in S \ y \in S$ 
    have  $((\lambda m. \text{norm } (f \ n \ x - f \ n \ y - (f \ m \ x - f \ m \ y))) \longrightarrow \text{norm } (f \ n \ x - f$ 
 $n \ y - (g \ x - g \ y)))$  sequentially
      by (intro tendsto_intros g [rule_format] as)
    moreover have eventually  $(\lambda m. \text{norm } (f \ n \ x - f \ n \ y - (f \ m \ x - f \ m \ y)) \leq$ 
 $e * \text{norm } (x - y))$  sequentially
      unfolding eventually_sequentially
    proof (intro exI allI impI)
      fix m
      assume  $N \leq m$ 
      then show  $\text{norm } (f \ n \ x - f \ n \ y - (f \ m \ x - f \ m \ y)) \leq e * \text{norm } (x - y)$ 
        using N as by (auto simp add: algebra_simps)
    qed
  ultimately show  $\text{norm } (f \ n \ x - f \ n \ y - (g \ x - g \ y)) \leq e * \text{norm } (x - y)$ 
    by (simp add: tendsto_upperbound)

```

```

    qed
  qed
  have  $\forall x \in S. ((\lambda n. f\ n\ x) \longrightarrow g\ x) \text{ sequentially} \wedge (g \text{ has\_derivative } g'\ x) \text{ (at } x \text{ within } S)$ 
    unfolding has_derivative_within_alt2
  proof (intro ballI conjI allI impI)
    fix x
    assume  $x \in S$ 
    then show  $(\lambda n. f\ n\ x) \longrightarrow g\ x$ 
      by (simp add: g)
    have  $\text{tog': } (\lambda n. f'\ n\ x\ u) \longrightarrow g'\ x\ u \text{ for } u$ 
      unfolding filterlim_def le_nhds_metric_le eventually_filtermap dist_norm
    proof (intro allI impI)
      fix e :: real
      assume  $e > 0$ 
      show eventually  $(\lambda n. \text{norm } (f'\ n\ x\ u - g'\ x\ u) \leq e) \text{ sequentially}$ 
      proof (cases  $u = 0$ )
        case True
          have eventually  $(\lambda n. \text{norm } (f'\ n\ x\ u - g'\ x\ u) \leq e * \text{norm } u) \text{ sequentially}$ 
            using nle  $\langle 0 < e \rangle \langle x \in S \rangle$  by (fast elim: eventually_mono)
          then show ?thesis
            using  $\langle u = 0 \rangle \langle 0 < e \rangle$  by (auto elim: eventually_mono)
        case False
          with  $\langle 0 < e \rangle$  have  $0 < e / \text{norm } u$  by simp
          then have eventually  $(\lambda n. \text{norm } (f'\ n\ x\ u - g'\ x\ u) \leq e / \text{norm } u * \text{norm } u) \text{ sequentially}$ 
            using nle  $\langle x \in S \rangle$  by (fast elim: eventually_mono)
          then show ?thesis
            using  $\langle u \neq 0 \rangle$  by simp
      qed
    qed
  qed
  show bounded_linear (g' x)
  proof
    fix x' y z :: 'a
    fix c :: real
    note lin = assms(2)[rule_format, OF  $\langle x \in S \rangle$ , THEN has_derivative_bounded_linear]
    show  $g'\ x\ (c *_{\mathbb{R}} x') = c *_{\mathbb{R}} g'\ x\ x'$ 
      apply (rule tendsto_unique[OF trivial_limit_sequentially tog'])
      unfolding lin[THEN bounded_linear.linear, THEN linear_cmul]
      apply (intro tendsto_intros tog')
    done
    show  $g'\ x\ (y + z) = g'\ x\ y + g'\ x\ z$ 
      apply (rule tendsto_unique[OF trivial_limit_sequentially tog'])
      unfolding lin[THEN bounded_linear.linear, THEN linear_add]
      apply (rule tendsto_add)
      apply (rule tog')+
    done
    obtain N where  $N: \forall h. \text{norm } (f'\ N\ x\ h - g'\ x\ h) \leq 1 * \text{norm } h$ 

```

```

    using nle ⟨x ∈ S⟩ unfolding eventually_sequentially by (fast intro:
zero_less_one)
  have bounded_linear (f' N x)
    using derf ⟨x ∈ S⟩ by fast
  from bounded_linear.bounded [OF this]
  obtain K where K: ∀ h. norm (f' N x h) ≤ norm h * K ..
  {
    fix h
    have norm (g' x h) = norm (f' N x h - (f' N x h - g' x h))
      by simp
    also have ... ≤ norm (f' N x h) + norm (f' N x h - g' x h)
      by (rule norm_triangle_ineq4)
    also have ... ≤ norm h * K + 1 * norm h
      using N K by (fast intro: add_mono)
    finally have norm (g' x h) ≤ norm h * (K + 1)
      by (simp add: ring_distrib)
  }
  then show ∃ K. ∀ h. norm (g' x h) ≤ norm h * K by fast
qed
show eventually (λy. norm (g y - g x - g' x (y - x)) ≤ e * norm (y - x))
(at x within S)
  if e > 0 for e
proof -
  have *: e / 3 > 0
    using that by auto
  obtain N1 where N1: ∀ n ≥ N1. ∀ x ∈ S. ∀ h. norm (f' n x h - g' x h) ≤ e /
3 * norm h
    using nle * unfolding eventually_sequentially by blast
  obtain N2 where
    N2[rule_format]: ∀ n ≥ N2. ∀ x ∈ S. ∀ y ∈ S. norm (f n x - f n y - (g x - g
y)) ≤ e / 3 * norm (x - y)
    using lem2 * by blast
  let ?N = max N1 N2
  have eventually (λy. norm (f ?N y - f ?N x - f' ?N x (y - x)) ≤ e / 3 *
norm (y - x)) (at x within S)
    using derf[unfolded has_derivative_within_alt2] and ⟨x ∈ S⟩ and * by
fast
  moreover have eventually (λy. y ∈ S) (at x within S)
    unfolding eventually_at by (fast intro: zero_less_one)
  ultimately show ∀ y in at x within S. norm (g y - g x - g' x (y - x)) ≤
e * norm (y - x)
  proof (rule eventually_elim2)
    fix y
    assume y ∈ S
    assume norm (f ?N y - f ?N x - f' ?N x (y - x)) ≤ e / 3 * norm (y - x)
    moreover have norm (g y - g x - (f ?N y - f ?N x)) ≤ e / 3 * norm (y
- x)
      using N2[OF _ ⟨y ∈ S⟩ ⟨x ∈ S⟩]
      by (simp add: norm_minus_commute)
  end

```

```

ultimately have norm (g y - g x - f' ?N x (y - x)) ≤ 2 * e / 3 * norm
(y - x)
  using norm_triangle_le[of g y - g x - (f ?N y - f ?N x) f ?N y - f ?N
x - f' ?N x (y - x) 2 * e / 3 * norm (y - x)]
  by (auto simp add: algebra_simps)
moreover
have norm (f' ?N x (y - x) - g' x (y - x)) ≤ e / 3 * norm (y - x)
  using N1 ⟨x ∈ S⟩ by auto
ultimately show norm (g y - g x - g' x (y - x)) ≤ e * norm (y - x)
  using norm_triangle_le[of g y - g x - f' (max N1 N2) x (y - x) f' (max
N1 N2) x (y - x) - g' x (y - x)]
  by (auto simp add: algebra_simps)
qed
qed
qed
then show ?thesis by fast
qed

```

Can choose to line up antiderivatives if we want.

```

lemma has_antiderivative_sequence:
  fixes f :: nat ⇒ 'a::real_normed_vector ⇒ 'b::banach
  assumes convex S
  and der:  $\bigwedge n x. x \in S \implies ((f\ n) \text{ has\_derivative } (f'\ n\ x)) \text{ (at } x \text{ within } S)$ 
  and no:  $\bigwedge e. e > 0 \implies \forall_F n \text{ in sequentially.}$ 
 $\forall x \in S. \forall h. \text{norm } (f'\ n\ x\ h - g'\ x\ h) \leq e * \text{norm } h$ 
  shows  $\exists g. \forall x \in S. (g \text{ has\_derivative } g'\ x) \text{ (at } x \text{ within } S)$ 
proof (cases S = {})
case False
then obtain a where a ∈ S
  by auto
have *:  $\bigwedge P\ Q. \exists g. \forall x \in S. P\ g\ x \wedge Q\ g\ x \implies \exists g. \forall x \in S. Q\ g\ x$ 
  by auto
show ?thesis
  apply (rule *)
  apply (rule has_antiderivative_sequence [OF ⟨convex S⟩ _ no, of  $\lambda n x. f\ n\ x +$ 
(f 0 a - f n a)])
  apply (metis assms(2) has_antiderivative_add_const)
  using ⟨a ∈ S⟩
  apply auto
done
qed auto

```

```

lemma has_antiderivative_limit:
  fixes g' :: 'a::real_normed_vector ⇒ 'a ⇒ 'b::banach
  assumes convex S
  and  $\bigwedge e. e > 0 \implies \exists f\ f'. \forall x \in S.$ 
 $(f \text{ has\_derivative } (f'\ x)) \text{ (at } x \text{ within } S) \wedge (\forall h. \text{norm } (f'\ x\ h - g'\ x\ h) \leq$ 
e * norm h)
  shows  $\exists g. \forall x \in S. (g \text{ has\_derivative } g'\ x) \text{ (at } x \text{ within } S)$ 

```



```

proof –
  have *:  $\forall n. \exists f f'. \forall x \in S.$ 
    ( $f$  has_derivative ( $f' x$ )) (at  $x$  within  $S$ )  $\wedge$ 
    ( $\forall h. \text{norm}(f' x h - g' x h) \leq \text{inverse}(\text{real}(Suc\ n)) * \text{norm}\ h$ )
  by (simp add: assms(2))
obtain  $f$  where
  *:  $\bigwedge x. \exists f'. \forall xa \in S. (f\ x\ \text{has\_derivative}\ f'\ xa)\ (at\ xa\ \text{within}\ S) \wedge$ 
    ( $\forall h. \text{norm}(f' xa h - g' xa h) \leq \text{inverse}(\text{real}(Suc\ x)) * \text{norm}\ h$ )
  using * by metis
obtain  $f'$  where
   $f'$ :  $\bigwedge x. \forall z \in S. (f\ x\ \text{has\_derivative}\ f'\ x\ z)\ (at\ z\ \text{within}\ S) \wedge$ 
    ( $\forall h. \text{norm}(f' x z h - g' z h) \leq \text{inverse}(\text{real}(Suc\ x)) * \text{norm}\ h$ )
  using * by metis
show ?thesis
proof (rule has_antiderivative_sequence[OF  $\langle \text{convex}\ S \rangle$ , of  $f f'$ ])
  fix  $e :: \text{real}$ 
  assume  $e > 0$ 
  obtain  $N$  where  $N$ :  $\text{inverse}(\text{real}(Suc\ N)) < e$ 
    using reals_Archimedean[OF  $\langle e > 0 \rangle$ ] ..
  show  $\forall_F n$  in sequentially.  $\forall x \in S. \forall h. \text{norm}(f' n x h - g' x h) \leq e * \text{norm}\ h$ 
    unfolding eventually_sequentially
  proof (intro exI allI ballI impI)
    fix  $n\ x\ h$ 
    assume  $n$ :  $N \leq n$  and  $x$ :  $x \in S$ 
    have *:  $\text{inverse}(\text{real}(Suc\ n)) \leq e$ 
      apply (rule order_trans[OF _  $N$  [THEN less_imp_le]])
      using  $n$  apply (auto simp add: field_simps)
      done
    show  $\text{norm}(f' n x h - g' x h) \leq e * \text{norm}\ h$ 
      by (meson * mult_right_mono norm_ge_zero order.trans  $x\ f'$ )
    qed
  qed (use  $f'$  in auto)
qed

```

4.10.14 Differentiation of a series

```

proposition has_derivative_series:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \text{real\_normed\_vector} \Rightarrow 'b :: \text{banach}$ 
  assumes convex  $S$ 
    and  $\bigwedge n\ x. x \in S \implies ((f\ n)\ \text{has\_derivative}\ (f'\ n\ x))\ (at\ x\ \text{within}\ S)$ 
    and  $\bigwedge e. e > 0 \implies \forall_F n$  in sequentially.  $\forall x \in S. \forall h. \text{norm}(\text{sum}(\lambda i. f'\ i\ x\ h) \{..<n\} - g' x h) \leq e * \text{norm}\ h$ 
    and  $x \in S$ 
    and  $(\lambda n. f\ n\ x)\ \text{sums}\ l$ 
  shows  $\exists g. \forall x \in S. (\lambda n. f\ n\ x)\ \text{sums}\ (g\ x) \wedge (g\ \text{has\_derivative}\ g'\ x)\ (at\ x\ \text{within}\ S)$ 
  unfolding sums_def
  apply (rule has_derivative_sequence[OF assms(1) _ assms(3)])
  apply (metis assms(2) has_derivative_sum)

```

```

using assms(4-5)
unfolding sums_def
apply auto
done

```

```

lemma has_field_derivative_series:
  fixes f :: nat  $\Rightarrow$  ('a :: {real_normed_field,banach})  $\Rightarrow$  'a
  assumes convex S
  assumes  $\bigwedge n x. x \in S \implies (f\ n\ \text{has\_field\_derivative}\ f'\ n\ x)$  (at x within S)
  assumes uniform_limit S ( $\lambda n x. \sum i < n. f'\ i\ x$ ) g' sequentially
  assumes  $x0 \in S$  summable ( $\lambda n. f\ n\ x0$ )
  shows  $\exists g. \forall x \in S. (\lambda n. f\ n\ x)\ \text{sums}\ g\ x \wedge (g\ \text{has\_field\_derivative}\ g'\ x)$  (at x
within S)
unfolding has_field_derivative_def
proof (rule has_derivative_series)
  show  $\forall_F n$  in sequentially.
     $\forall x \in S. \forall h. \text{norm} ((\sum i < n. f'\ i\ x * h) - g'\ x * h) \leq e * \text{norm}\ h$  if  $e > 0$ 
  for e
    unfolding eventually_sequentially
  proof -
    from that assms(3) obtain N where  $N: \bigwedge n x. n \geq N \implies x \in S \implies \text{norm}$ 
     $((\sum i < n. f'\ i\ x) - g'\ x) < e$ 
    unfolding uniform_limit_iff eventually_at_top_linorder dist_norm by blast
    {
      fix n :: nat and x h :: 'a assume nx:  $n \geq N \wedge x \in S$ 
      have  $\text{norm} ((\sum i < n. f'\ i\ x * h) - g'\ x * h) = \text{norm} ((\sum i < n. f'\ i\ x) - g'$ 
      x) *  $\text{norm}\ h$ 
        by (simp add: norm_mult [symmetric] ring_distribs sum_distrib_right)
      also from N[OF nx] have  $\text{norm} ((\sum i < n. f'\ i\ x) - g'\ x) \leq e$  by simp
      hence  $\text{norm} ((\sum i < n. f'\ i\ x) - g'\ x) * \text{norm}\ h \leq e * \text{norm}\ h$ 
        by (intro mult_right_mono) simp_all
      finally have  $\text{norm} ((\sum i < n. f'\ i\ x * h) - g'\ x * h) \leq e * \text{norm}\ h$  .
    }
    thus  $\exists N. \forall n \geq N. \forall x \in S. \forall h. \text{norm} ((\sum i < n. f'\ i\ x * h) - g'\ x * h) \leq e * \text{norm}\ h$  by blast
  qed
qed (use assms in <auto simp: has_field_derivative_def>)

```

```

lemma has_field_derivative_series':
  fixes f :: nat  $\Rightarrow$  ('a :: {real_normed_field,banach})  $\Rightarrow$  'a
  assumes convex S
  assumes  $\bigwedge n x. x \in S \implies (f\ n\ \text{has\_field\_derivative}\ f'\ n\ x)$  (at x within S)
  assumes uniformly_convergent_on S ( $\lambda n x. \sum i < n. f'\ i\ x$ )
  assumes  $x0 \in S$  summable ( $\lambda n. f\ n\ x0$ )  $x \in \text{interior}\ S$ 
  shows summable ( $\lambda n. f\ n\ x$ )  $((\lambda x. \sum n. f\ n\ x)\ \text{has\_field\_derivative}\ (\sum n. f'\ n\ x))$  (at x)
proof -
  from <x  $\in \text{interior}\ S$ > have  $x \in S$  using interior_subset by blast
  define g' where [abs_def]:  $g'\ x = (\sum i. f'\ i\ x)$  for x

```

```

from assms(3) have uniform_limit S ( $\lambda n x. \sum i < n. f' i x$ )  $g'$  sequentially
by (simp add: uniformly_convergent_uniform_limit_iff suminf_eq_lim  $g'$ _def)
from has_field_derivative_series[OF assms(1,2) this assms(4,5)] obtain g
where g:
   $\bigwedge x. x \in S \implies (\lambda n. f n x) \text{ sums } g x$ 
   $\bigwedge x. x \in S \implies (g \text{ has\_field\_derivative } g' x) \text{ (at } x \text{ within } S)$  by blast
from g(1)[OF  $\langle x \in S \rangle$ ] show summable ( $\lambda n. f n x$ ) by (simp add: sums_iff)
from g(2)[OF  $\langle x \in S \rangle$ ]  $\langle x \in \text{interior } S \rangle$  have ( $g \text{ has\_field\_derivative } g' x$ ) (at
x)
  by (simp add: at_within_interior[of x S])
also have ( $g \text{ has\_field\_derivative } g' x$ ) (at x)  $\longleftrightarrow$ 
  ( $(\lambda x. \sum n. f n x) \text{ has\_field\_derivative } g' x$ ) (at x)
  using eventually_nhds_in_nhd[OF  $\langle x \in \text{interior } S \rangle$ ] interior_subset[of S] g(1)
  by (intro DERIV_cong_ev) (auto elim!: eventually_mono simp: sums_iff)
finally show ( $(\lambda x. \sum n. f n x) \text{ has\_field\_derivative } g' x$ ) (at x) .
qed

```

lemma differentiable_series:

```

fixes f :: nat  $\Rightarrow$  ('a :: {real_normed_field,banach})  $\Rightarrow$  'a
assumes convex S open S
assumes  $\bigwedge n x. x \in S \implies (f n \text{ has\_field\_derivative } f' n x) \text{ (at } x)$ 
assumes uniformly_convergent_on S ( $\lambda n x. \sum i < n. f' i x$ )
assumes  $x0 \in S$  summable ( $\lambda n. f n x0$ ) and  $x: x \in S$ 
shows summable ( $\lambda n. f n x$ ) and ( $\lambda x. \sum n. f n x$ ) differentiable (at x)
proof -
  from assms(4) obtain  $g'$  where A: uniform_limit S ( $\lambda n x. \sum i < n. f' i x$ )  $g'$ 
  sequentially
  unfolding uniformly_convergent_on_def by blast
  from x and  $\langle \text{open } S \rangle$  have S: at x within S = at x by (rule at_within_open)
  have  $\exists g. \forall x \in S. (\lambda n. f n x) \text{ sums } g x \wedge (g \text{ has\_field\_derivative } g' x) \text{ (at } x \text{ within } S)$ 
  by (intro has_field_derivative_series[of S f f' g' x0] assms A has_field_derivative_at_within)
  then obtain g where g:  $\bigwedge x. x \in S \implies (\lambda n. f n x) \text{ sums } g x$ 
   $\bigwedge x. x \in S \implies (g \text{ has\_field\_derivative } g' x) \text{ (at } x \text{ within } S)$  by blast
  from g[OF x] show summable ( $\lambda n. f n x$ ) by (auto simp: summable_def)
  from g(2)[OF x] have  $g': (g \text{ has\_derivative } (*)) (g' x) \text{ (at } x)$ 
  by (simp add: has_field_derivative_def S)
  have ( $(\lambda x. \sum n. f n x) \text{ has\_derivative } (*) (g' x) \text{ (at } x)$ )
  by (rule has_derivative_transform_within_open[OF g'  $\langle \text{open } S \rangle$  x])
  (insert g, auto simp: sums_iff)
  thus ( $\lambda x. \sum n. f n x$ ) differentiable (at x) unfolding differentiable_def
  by (auto simp: summable_def differentiable_def has_field_derivative_def)
qed

```

lemma differentiable_series':

```

fixes f :: nat  $\Rightarrow$  ('a :: {real_normed_field,banach})  $\Rightarrow$  'a
assumes convex S open S
assumes  $\bigwedge n x. x \in S \implies (f n \text{ has\_field\_derivative } f' n x) \text{ (at } x)$ 
assumes uniformly_convergent_on S ( $\lambda n x. \sum i < n. f' i x$ )

```

```

assumes  $x0 \in S$  summable  $(\lambda n. f\ n\ x0)$ 
shows  $(\lambda x. \sum n. f\ n\ x)$  differentiable  $(at\ x0)$ 
using differentiable_series[OF assms, of  $x0$ ]  $\langle x0 \in S \rangle$  by blast+

```

4.10.15 Derivative as a vector

Considering derivative $real \Rightarrow 'b$ as a vector.

definition *vector_derivative* $f\ net = (SOME\ f'. (f\ has\_vector\_derivative\ f')\ net)$

lemma *vector_derivative_unique_within*:

```

assumes not_bot:  $at\ x\ within\ S \neq bot$ 
and  $f'$ :  $(f\ has\_vector\_derivative\ f')$   $(at\ x\ within\ S)$ 
and  $f''$ :  $(f\ has\_vector\_derivative\ f'')$   $(at\ x\ within\ S)$ 
shows  $f' = f''$ 

```

proof –

```

have  $(\lambda x. x *_{\mathbb{R}} f') = (\lambda x. x *_{\mathbb{R}} f'')$ 
proof (rule frechet_derivative_unique_within, simp_all)
show  $\exists d. d \neq 0 \wedge |d| < e \wedge x + d \in S$  if  $0 < e$  for  $e$ 
proof –
from that
obtain  $x'$  where  $x' \in S$   $x' \neq x$   $|x' - x| < e$ 
using islimpt_approachable_real[of  $x\ S$ ] not_bot
by (auto simp add: trivial_limit_within)
then show ?thesis
using eq_iff_diff_eq_0 by fastforce
qed
qed (use  $f'\ f''$  in  $\langle auto\ simp: has\_vector\_derivative\_def \rangle$ )
then show ?thesis
unfolding fun_eq_iff by (metis scaleR_one)
qed

```

lemma *vector_derivative_unique_at*:

```

 $(f\ has\_vector\_derivative\ f')\ (at\ x) \implies (f\ has\_vector\_derivative\ f'')\ (at\ x) \implies$ 
 $f' = f''$ 
by (rule vector_derivative_unique_within) auto

```

lemma *differentiableI_vector*: $(f\ has\_vector\_derivative\ y)\ F \implies f\ differentiable\ F$

by (*auto* *simp*: *differentiable_def* *has_vector_derivative_def*)

proposition *vector_derivative_works*:

```

 $f\ differentiable\ net \longleftrightarrow (f\ has\_vector\_derivative\ (vector\_derivative\ f\ net))\ net$ 
(is ?l  $=$  ?r)

```

proof

```

assume ?l
obtain  $f'$  where  $f'$ :  $(f\ has\_derivative\ f')$  net
using  $\langle ?l \rangle$  unfolding differentiable_def ..
then interpret bounded_linear  $f'$ 
by auto

```

```

show ?r
  unfolding vector_derivative_def has_vector_derivative_def
  by (rule someI[of _ f' 1]) (simp add: scaleR[symmetric] f')
qed (auto simp: vector_derivative_def has_vector_derivative_def differentiable_def)

lemma vector_derivative_within:
  assumes not_bot: at x within S  $\neq$  bot and y: (f has_vector_derivative y) (at x
  within S)
  shows vector_derivative f (at x within S) = y
  using y
  by (intro vector_derivative_unique_within[OF not_bot vector_derivative_works[THEN
  iffD1] y])
  (auto simp: differentiable_def has_vector_derivative_def)

lemma deriv_of_real [simp]:
  at x within A  $\neq$  bot  $\implies$  vector_derivative of_real (at x within A) = 1
  by (auto intro!: vector_derivative_within derivative_eq_intros)

lemma frechet_derivative_eq_vector_derivative:
  assumes f differentiable (at x)
  shows (frechet_derivative f (at x)) = ( $\lambda r$ . r *R vector_derivative f (at x))
  using assms
  by (auto simp: differentiable_iff_scaleR vector_derivative_def has_vector_derivative_def
  intro: someI frechet_derivative_at [symmetric])

lemma has_real_derivative:
  fixes f :: real  $\Rightarrow$  real
  assumes (f has_derivative f') F
  obtains c where (f has_real_derivative c) F
proof -
  obtain c where f' = ( $\lambda x$ . x * c)
  by (metis assms has_derivative_bounded_linear real_bounded_linear)
  then show ?thesis
  by (metis assms that has_field_derivative_def mult_commute_abs)
qed

lemma has_real_derivative_iff:
  fixes f :: real  $\Rightarrow$  real
  shows ( $\exists c$ . (f has_real_derivative c) F) = ( $\exists D$ . (f has_derivative D) F)
  by (metis has_field_derivative_def has_real_derivative)

lemma has_vector_derivative_cong_ev:
  assumes *: eventually ( $\lambda x$ . x  $\in$  S  $\longrightarrow$  f x = g x) (nhds x) f x = g x
  shows (f has_vector_derivative f') (at x within S) = (g has_vector_derivative
  f') (at x within S)
  unfolding has_vector_derivative_def has_derivative_def
  using *
  apply (cases at x within S  $\neq$  bot)
  apply (intro refl conj_cong filterlim_cong)

```

```

apply (auto simp: Lim_ident_at eventually_at_filter elim: eventually_mono)
done

```

```

lemma vector_derivative_cong_eq:
  assumes eventually ( $\lambda x. x \in A \longrightarrow f\ x = g\ x$ ) (nhds x)  $x = y$   $A = B$   $x \in A$ 
  shows vector_derivative f (at x within A) = vector_derivative g (at y within
B)
proof -
  have  $f\ x = g\ x$ 
  using assms eventually_nhds_x_imp_x by blast
  hence ( $\lambda D. (f\ \text{has\_vector\_derivative}\ D)\ (\text{at}\ x\ \text{within}\ A)$ ) =
    ( $\lambda D. (g\ \text{has\_vector\_derivative}\ D)\ (\text{at}\ x\ \text{within}\ A)$ ) using assms
  by (intro ext has_vector_derivative_cong_ev refl assms) simp_all
  thus ?thesis by (simp add: vector_derivative_def assms)
qed

```

```

lemma islimpt_closure_open:
  fixes s :: 'a::perfect_space set
  assumes open s and t:  $t = \text{closure}\ s$   $x \in t$ 
  shows x islimpt t
proof cases
  assume  $x \in s$ 
  { fix T assume  $x \in T$  open T
    then have open (s  $\cap$  T)
    using <open s> by auto
    then have  $s \cap T \neq \{x\}$ 
    using not_open_singleton[of x] by auto
    with < $x \in T$ > < $x \in s$ > have  $\exists y \in t. y \in T \wedge y \neq x$ 
    using closure_subset[of s] by (auto simp: t) }
  then show ?thesis
    by (auto intro!: islimptI)
next
  assume  $x \notin s$  with t show ?thesis
    unfolding t closure_def by (auto intro: islimpt_subset)
qed

```

```

lemma vector_derivative_unique_within_closed_interval:
  assumes ab:  $a < b$   $x \in \text{cbox}\ a\ b$ 
  assumes D: (f has_vector_derivative f') (at x within cbox a b) (f has_vector_derivative
f'') (at x within cbox a b)
  shows  $f' = f''$ 
  using ab
  by (intro vector_derivative_unique_within[OF _ D])
    (auto simp: trivial_limit_within intro!: islimpt_closure_open[where s={a
<.. $b$ }])

```

```

lemma vector_derivative_at:
  (f has_vector_derivative f') (at x)  $\implies$  vector_derivative f (at x) = f'
  by (intro vector_derivative_within at_neq_bot)

```

lemma *has_vector_derivative_id_at* [simp]: $\text{vector\_derivative } (\lambda x. x) \text{ (at } a) = 1$
by (simp add: *vector_derivative_at*)

lemma *vector_derivative_minus_at* [simp]:
 f differentiable at a
 $\implies \text{vector\_derivative } (\lambda x. - f x) \text{ (at } a) = - \text{vector\_derivative } f \text{ (at } a)$
by (simp add: *vector_derivative_at* *has_vector_derivative_minus* *vector_derivative_works* [symmetric])

lemma *vector_derivative_add_at* [simp]:
 f differentiable at a ; g differentiable at a
 $\implies \text{vector\_derivative } (\lambda x. f x + g x) \text{ (at } a) = \text{vector\_derivative } f \text{ (at } a) + \text{vector\_derivative } g \text{ (at } a)$
by (simp add: *vector_derivative_at* *has_vector_derivative_add* *vector_derivative_works* [symmetric])

lemma *vector_derivative_diff_at* [simp, derivative_intros]:
 f differentiable at a ; g differentiable at a
 $\implies \text{vector\_derivative } (\lambda x. f x - g x) \text{ (at } a) = \text{vector\_derivative } f \text{ (at } a) - \text{vector\_derivative } g \text{ (at } a)$
by (simp add: *vector_derivative_at* *has_vector_derivative_diff* *vector_derivative_works* [symmetric])

lemma *vector_derivative_mult_at* [simp]:
fixes $f g :: \text{real} \Rightarrow 'a :: \text{real\_normed\_algebra}$
shows f differentiable at a ; g differentiable at a
 $\implies \text{vector\_derivative } (\lambda x. f x * g x) \text{ (at } a) = f a * \text{vector\_derivative } g \text{ (at } a) + \text{vector\_derivative } f \text{ (at } a) * g a$
by (simp add: *vector_derivative_at* *has_vector_derivative_mult* *vector_derivative_works* [symmetric])

lemma *vector_derivative_scaleR_at* [simp]:
 f differentiable at a ; g differentiable at a
 $\implies \text{vector\_derivative } (\lambda x. f x *_{\mathbb{R}} g x) \text{ (at } a) = f a *_{\mathbb{R}} \text{vector\_derivative } g \text{ (at } a) + \text{vector\_derivative } f \text{ (at } a) *_{\mathbb{R}} g a$
apply (rule *vector_derivative_at*)
apply (rule *has_vector_derivative_scaleR*)
apply (auto simp: *vector_derivative_works* *has_vector_derivative_def* *has_field_derivative_def* *mult_commute_abs*)
done

lemma *vector_derivative_within_cbox*:
assumes $ab: a < b \ x \in \text{cbox } a \ b$
assumes $f: (f \text{ has\_vector\_derivative } f') \text{ (at } x \text{ within } \text{cbox } a \ b)$
shows $\text{vector\_derivative } f \text{ (at } x \text{ within } \text{cbox } a \ b) = f'$
by (intro *vector_derivative_unique_within_closed_interval* [OF ab f]
 $\text{vector\_derivative_works[THEN iffD1]} \text{ differentiableI\_vector}$)
fact

lemma *vector_derivative_within_closed_interval*:

fixes $f::\text{real} \Rightarrow 'a::\text{euclidean\_space}$
assumes $a < b$ **and** $x \in \{a..b\}$
assumes $(f \text{ has\_vector\_derivative } f') \text{ (at } x \text{ within } \{a..b\})$
shows $\text{vector\_derivative } f \text{ (at } x \text{ within } \{a..b\}) = f'$
using *assms vector_derivative_within_cbox*
by *fastforce*

lemma *has_vector_derivative_within_subset*:

$(f \text{ has\_vector\_derivative } f') \text{ (at } x \text{ within } S) \Longrightarrow T \subseteq S \Longrightarrow (f \text{ has\_vector\_derivative } f') \text{ (at } x \text{ within } T)$
by $(\text{auto simp: has\_vector\_derivative\_def intro: has\_derivative\_subset})$

lemma *has_vector_derivative_at_within*:

$(f \text{ has\_vector\_derivative } f') \text{ (at } x) \Longrightarrow (f \text{ has\_vector\_derivative } f') \text{ (at } x \text{ within } S)$
unfolding *has_vector_derivative_def*
by $(\text{rule has\_derivative\_at\_withinI})$

lemma *has_vector_derivative_weaken*:

fixes $x \ D$ **and** $f \ g \ S \ T$
assumes $f: (f \text{ has\_vector\_derivative } D) \text{ (at } x \text{ within } T)$
and $x \in S \ S \subseteq T$
and $\bigwedge x. x \in S \Longrightarrow f \ x = g \ x$
shows $(g \text{ has\_vector\_derivative } D) \text{ (at } x \text{ within } S)$
proof –
have $(f \text{ has\_vector\_derivative } D) \text{ (at } x \text{ within } S) \longleftrightarrow (g \text{ has\_vector\_derivative } D) \text{ (at } x \text{ within } S)$
unfolding *has_vector_derivative_def has_derivative_iff_norm*
using *assms by (intro conj_cong Lim_cong_within refl) auto*
then show *?thesis*
using *has_vector_derivative_within_subset[OF f ‹S ⊆ T›] by simp*
qed

lemma *has_vector_derivative_transform_within*:

assumes $(f \text{ has\_vector\_derivative } f') \text{ (at } x \text{ within } S)$
and $0 < d$
and $x \in S$
and $\bigwedge x'. \llbracket x' \in S; \text{dist } x' \ x < d \rrbracket \Longrightarrow f \ x' = g \ x'$
shows $(g \text{ has\_vector\_derivative } f') \text{ (at } x \text{ within } S)$
using *assms*
unfolding *has_vector_derivative_def*
by $(\text{rule has\_derivative\_transform\_within})$

lemma *has_vector_derivative_transform_within_open*:

assumes $(f \text{ has\_vector\_derivative } f') \text{ (at } x)$
and *open S*
and $x \in S$

and $\bigwedge y. y \in S \implies f y = g y$
shows $(g \text{ has\_vector\_derivative } f') \text{ (at } x)$
using *assms*
unfolding *has_vector_derivative_def*
by $(\text{rule has\_derivative\_transform\_within\_open})$

lemma *has_vector_derivative_transform*:
assumes $x \in S \bigwedge x. x \in S \implies g x = f x$
assumes $f': (f \text{ has\_vector\_derivative } f') \text{ (at } x \text{ within } S)$
shows $(g \text{ has\_vector\_derivative } f') \text{ (at } x \text{ within } S)$
using *assms*
unfolding *has_vector_derivative_def*
by $(\text{rule has\_derivative\_transform})$

lemma *vector_diff_chain_at*:
assumes $(f \text{ has\_vector\_derivative } f') \text{ (at } x)$
and $(g \text{ has\_vector\_derivative } g') \text{ (at } (f x))$
shows $((g \circ f) \text{ has\_vector\_derivative } (f' *_R g')) \text{ (at } x)$
using *assms has_vector_derivative_at_within has_vector_derivative_def vector_derivative_diff_chain_within* **by** *blast*

lemma *vector_diff_chain_within*:
assumes $(f \text{ has\_vector\_derivative } f') \text{ (at } x \text{ within } s)$
and $(g \text{ has\_vector\_derivative } g') \text{ (at } (f x) \text{ within } f' \text{ ' } s)$
shows $((g \circ f) \text{ has\_vector\_derivative } (f' *_R g')) \text{ (at } x \text{ within } s)$
using *assms has_vector_derivative_def vector_derivative_diff_chain_within* **by** *blast*

lemma *vector_derivative_const_at* [*simp*]: $\text{vector\_derivative } (\lambda x. c) \text{ (at } a) = 0$
by $(\text{simp add: vector\_derivative\_at})$

lemma *vector_derivative_at_within_ivl*:
 $(f \text{ has\_vector\_derivative } f') \text{ (at } x) \implies$
 $a \leq x \implies x \leq b \implies a < b \implies \text{vector\_derivative } f \text{ (at } x \text{ within } \{a..b\}) = f'$
using *has_vector_derivative_at_within vector_derivative_within_cbox* **by** *fastforce*

lemma *vector_derivative_chain_at*:
assumes $f \text{ differentiable at } x \text{ (} g \text{ differentiable at } (f x))$
shows $\text{vector\_derivative } (g \circ f) \text{ (at } x) =$
 $\text{vector\_derivative } f \text{ (at } x) *_R \text{vector\_derivative } g \text{ (at } (f x))$
by $(\text{metis vector\_diff\_chain\_at vector\_derivative\_at vector\_derivative\_works assms})$

lemma *field_vector_diff_chain_at*:
assumes $Df: (f \text{ has\_vector\_derivative } f') \text{ (at } x)$
and $Dg: (g \text{ has\_field\_derivative } g') \text{ (at } (f x))$
shows $((g \circ f) \text{ has\_vector\_derivative } (f' * g')) \text{ (at } x)$
using *diff_chain_at[OF Df(unfolded has_vector_derivative_def)*
 $Dg \text{ [unfolded has\_field\_derivative\_def]}$

by (*auto simp: o_def mult.commute has_vector_derivative_def*)

lemma *vector_derivative_chain_within*:

assumes *at x within S* $\neq$ *bot* *f differentiable (at x within S)*
(g has_derivative g') *(at (f x) within f ' S)*
shows *vector_derivative (g $\circ$ f) (at x within S) =*
g' (vector_derivative f (at x within S))
apply (*rule vector_derivative_within [OF <at x within S $\neq$ bot>]*)
apply (*rule vector_derivative_diff_chain_within*)
using *assms(2-3) vector_derivative_works*
by *auto*

4.10.16 Field differentiability

definition *field_differentiable* :: [*'a* $\Rightarrow$ *'a::real_normed_field*, *'a filter*] $\Rightarrow$ *bool*
(infixr (field'_differentiable) 50)
where *f field_differentiable F* $\equiv \exists f'. (f \text{ has\_field\_derivative } f') F$

lemma *field_differentiable_imp_differentiable*:

f field_differentiable F $\Longrightarrow$ *f differentiable F*
unfolding *field_differentiable_def differentiable_def*
using *has_field_derivative_imp_has_derivative* **by** *auto*

lemma *field_differentiable_imp_continuous_at*:

f field_differentiable (at x within S) $\Longrightarrow$ *continuous (at x within S) f*
by (*metis DERIV_continuous field_differentiable_def*)

lemma *field_differentiable_within_subset*:

$\llbracket f \text{ field\_differentiable (at } x \text{ within } S); T \subseteq S \rrbracket \Longrightarrow f \text{ field\_differentiable (at } x \text{ within } T)$
by (*metis DERIV_subset field_differentiable_def*)

lemma *field_differentiable_at_within*:

$\llbracket f \text{ field\_differentiable (at } x) \rrbracket$
 $\Longrightarrow f \text{ field\_differentiable (at } x \text{ within } S)$
unfolding *field_differentiable_def*
by (*metis DERIV_subset top_greatest*)

lemma *field_differentiable_linear* [*simp,derivative_intros*]: $((*) c) \text{ field\_differentiable } F$

unfolding *field_differentiable_def has_field_derivative_def mult_commute_abs*
by (*force intro: has_derivative_mult_right*)

lemma *field_differentiable_const* [*simp,derivative_intros*]: $(\lambda z. c) \text{ field\_differentiable } F$

unfolding *field_differentiable_def has_field_derivative_def*
using *DERIV_const has_field_derivative_imp_has_derivative* **by** *blast*

lemma *field_differentiable_ident* [*simp,derivative_intros*]: $(\lambda z. z) \text{ field\_differentiable}$

F

unfolding *field_differentiable_def* *has_field_derivative_def*
using *DERIV_ident* *has_field_derivative_def* **by** *blast*

lemma *field_differentiable_id* [*simp, derivative_intros*]: *id* *field_differentiable* F
unfolding *id_def* **by** (*rule field_differentiable_ident*)

lemma *field_differentiable_minus* [*derivative_intros*]:
 f *field_differentiable* $F \implies (\lambda z. - (f z))$ *field_differentiable* F
unfolding *field_differentiable_def*
by (*metis field_differentiable_minus*)

lemma *field_differentiable_add* [*derivative_intros*]:
assumes f *field_differentiable* F g *field_differentiable* F
shows $(\lambda z. f z + g z)$ *field_differentiable* F
using *assms* **unfolding** *field_differentiable_def*
by (*metis field_differentiable_add*)

lemma *field_differentiable_add_const* [*simp, derivative_intros*]:
 $(+)$ c *field_differentiable* F
by (*simp add: field_differentiable_add*)

lemma *field_differentiable_sum* [*derivative_intros*]:
 $(\bigwedge i. i \in I \implies (f i)$ *field_differentiable* $F) \implies (\lambda z. \sum_{i \in I} f i z)$ *field_differentiable* F
by (*induct I rule: infinite_finite_induct*)
(auto intro: field_differentiable_add field_differentiable_const)

lemma *field_differentiable_diff* [*derivative_intros*]:
assumes f *field_differentiable* F g *field_differentiable* F
shows $(\lambda z. f z - g z)$ *field_differentiable* F
using *assms* **unfolding** *field_differentiable_def*
by (*metis field_differentiable_diff*)

lemma *field_differentiable_inverse* [*derivative_intros*]:
assumes f *field_differentiable* (at a within S) $f a \neq 0$
shows $(\lambda z. \text{inverse } (f z))$ *field_differentiable* (at a within S)
using *assms* **unfolding** *field_differentiable_def*
by (*metis DERIV_inverse_fun*)

lemma *field_differentiable_mult* [*derivative_intros*]:
assumes f *field_differentiable* (at a within S)
 g *field_differentiable* (at a within S)
shows $(\lambda z. f z * g z)$ *field_differentiable* (at a within S)
using *assms* **unfolding** *field_differentiable_def*
by (*metis DERIV_mult [of f _ a S g]*)

lemma *field_differentiable_divide* [*derivative_intros*]:
assumes f *field_differentiable* (at a within S)

```

      g field_differentiable (at a within S)
      g a ≠ 0
    shows (λz. f z / g z) field_differentiable (at a within S)
  using assms unfolding field_differentiable_def
  by (metis DERIV_divide [of f _ a S g])

```

```

lemma field_differentiable_power [derivative_intros]:
  assumes f field_differentiable (at a within S)
  shows (λz. f z ^ n) field_differentiable (at a within S)
  using assms unfolding field_differentiable_def
  by (metis DERIV_power)

```

```

lemma field_differentiable_transform_within:
  0 < d ⟹
  x ∈ S ⟹
  (λx'. x' ∈ S ⟹ dist x' x < d ⟹ f x' = g x') ⟹
  f field_differentiable (at x within S)
  ⟹ g field_differentiable (at x within S)
  unfolding field_differentiable_def has_field_derivative_def
  by (blast intro: has_derivative_transform_within)

```

```

lemma field_differentiable_compose_within:
  assumes f field_differentiable (at a within S)
  g field_differentiable (at (f a) within f'S)
  shows (g o f) field_differentiable (at a within S)
  using assms unfolding field_differentiable_def
  by (metis DERIV_image_chain)

```

```

lemma field_differentiable_compose:
  f field_differentiable at z ⟹ g field_differentiable at (f z)
  ⟹ (g o f) field_differentiable at z
  by (metis field_differentiable_at_within field_differentiable_compose_within)

```

```

lemma field_differentiable_within_open:
  [a ∈ S; open S] ⟹ f field_differentiable at a within S ⟷
  f field_differentiable at a
  unfolding field_differentiable_def
  by (metis at_within_open)

```

```

lemma exp_scaleR_has_vector_derivative_right:
  ((λt. exp (t *R A)) has_vector_derivative exp (t *R A) * A) (at t within T)
  unfolding has_vector_derivative_def
proof (rule has_derivativeI)
  let ?F = at t within (T ∩ {t - 1 <..R (A * A
  ^ i)
  have ∀F n in sequentially.

```

```

   $\forall x \in T \cap \{t - 1 < \dots < t + 1\}. \text{norm } (?e \ n \ x) \leq \text{norm } (A \wedge (n + 1) /_R \text{fact } (n + 1))$ 
  apply (auto simp: algebra_split_simps intro!: eventuallyI)
  apply (rule mult_left_mono)
  apply (auto simp add: field_simps power_abs intro!: divide_right_mono power_le_one)
done
  then have uniform_limit  $(T \cap \{t - 1 < \dots < t + 1\}) (\lambda n \ x. \sum_{i < n}. ?e \ i \ x) (\lambda x. \sum_{i. ?e \ i \ x})$  sequentially
  by (rule Weierstrass_m_test_ev) (intro summable_ignore_initial_segment summable_norm_exp)
  moreover
  have  $\forall_F \ x$  in sequentially.  $x > 0$ 
  by (metis eventually_gt_at_top)
  then have
   $\forall_F \ n$  in sequentially.  $((\lambda x. \sum_{i < n}. ?e \ i \ x) \longrightarrow A) \ ?F$ 
  by eventually_elim
  (auto intro!: tendsto_eq_intros
    simp: power_0_left if_distrib if_distribR
    cong: if_cong)
  ultimately
  have [tendsto_intros]:  $((\lambda x. \sum_{i. ?e \ i \ x}) \longrightarrow A) \ ?F$ 
  by (auto intro!: swap_uniform_limit[where  $f = \lambda n \ x. \sum_{i < n}. ?e \ i \ x$  and  $F = \text{sequentially}$ ])
  have [tendsto_intros]:  $((\lambda x. \text{if } x = t \text{ then } 0 \text{ else } 1) \longrightarrow 1) \ ?F$ 
  by (rule tendsto_eventually) (simp add: eventually_at_filter)
  have  $((\lambda y. ((y - t) / \text{abs } (y - t)) *_R ((\sum n. ?e \ n \ y) - A)) \longrightarrow 0)$  (at  $t$  within  $T$ )
  unfolding *
  by (rule tendsto_norm_zero_cancel) (auto intro!: tendsto_eq_intros)

  moreover have  $\forall_F \ x$  in at  $t$  within  $T$ .  $x \neq t$ 
  by (simp add: eventually_at_filter)
  then have  $\forall_F \ x$  in at  $t$  within  $T$ .  $((x - t) / |x - t|) *_R ((\sum n. ?e \ n \ x) - A) = (\text{exp } ((x - t) *_R A) - 1 - (x - t) *_R A) /_R \text{norm } (x - t)$ 
  proof eventually_elim
  case (elim  $x$ )
  have  $(\text{exp } ((x - t) *_R A) - 1 - (x - t) *_R A) /_R \text{norm } (x - t) = ((\sum n. (x - t) *_R ?e \ n \ x) - (x - t) *_R A) /_R \text{norm } (x - t)$ 
  unfolding exp_first_term
  by (simp add: ac_simps)
  also
  have summable  $(\lambda n. ?e \ n \ x)$ 
  proof -
  from elim have  $?e \ n \ x = (((x - t) *_R A) \wedge (n + 1)) /_R \text{fact } (n + 1) /_R (x - t)$  for  $n$ 
  by simp
  then show ?thesis
  by (auto simp only:

```

```

      intro!: summable_scaleR_right summable_ignore_initial_segment summable_exp_generic)
    qed
  then have  $(\sum n. (x - t) *_R ?e\ n\ x) = (x - t) *_R (\sum n. ?e\ n\ x)$ 
    by (rule suminf_scaleR_right[symmetric])
  also have  $(\dots - (x - t) *_R A) /_R \text{norm } (x - t) = (x - t) *_R ((\sum n. ?e\ n\ x) - A) /_R \text{norm } (x - t)$ 
    by (simp add: algebra_simps)
  finally show ?case
    by simp (simp add: field_simps)
  qed

  ultimately have  $((\lambda y. (\exp ((y - t) *_R A) - 1 - (y - t) *_R A) /_R \text{norm } (y - t)) \longrightarrow 0)$  (at  $t$  within  $T$ )
    by (rule Lim_transform_eventually)
  from tendsto_mult_right_zero[OF this, where  $c = \exp (t *_R A)$ ]
  show  $((\lambda y. (\exp (y *_R A) - \exp (t *_R A) - (y - t) *_R (\exp (t *_R A) * A)) /_R \text{norm } (y - t)) \longrightarrow 0)$ 
    (at  $t$  within  $T$ )
    by (rule Lim_transform_eventually)
    (auto simp: field_split_simps exp_add_commuting[symmetric])
  qed (rule bounded_linear_scaleR_left)

lemma exp_times_scaleR_commute:  $\exp (t *_R A) * A = A * \exp (t *_R A)$ 
  using exp_times_arg_commute[symmetric, of  $t *_R A$ ]
  by (auto simp: algebra_simps)

lemma exp_scaleR_has_vector_derivative_left:  $((\lambda t. \exp (t *_R A)) \text{ has\_vector\_derivative } A * \exp (t *_R A))$  (at  $t$ )
  using exp_scaleR_has_vector_derivative_right[of  $A\ t$ ]
  by (simp add: exp_times_scaleR_commute)

lemma field_differentiable_series:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{real\_normed\_field}, \text{banach}\} \Rightarrow 'a$ 
  assumes convex  $S$  open  $S$ 
  assumes  $\bigwedge n\ x. x \in S \implies (f\ n \text{ has\_field\_derivative } f'\ n\ x)$  (at  $x$ )
  assumes uniformly_convergent_on  $S$   $(\lambda n\ x. \sum i < n. f'\ i\ x)$ 
  assumes  $x0 \in S$  summable  $(\lambda n. f\ n\ x0)$  and  $x: x \in S$ 
  shows  $(\lambda x. \sum n. f\ n\ x) \text{ field\_differentiable (at } x)$ 
proof -
  from assms(4) obtain  $g'$  where  $A: \text{uniform\_limit } S\ (\lambda n\ x. \sum i < n. f'\ i\ x)\ g'$ 
    sequentially
  unfolding uniformly_convergent_on_def by blast
  from  $x$  and  $\langle \text{open } S \rangle$  have  $S: \text{at } x \text{ within } S = \text{at } x$  by (rule at_within_open)
  have  $\exists g. \forall x \in S. (\lambda n. f\ n\ x) \text{ sums } g\ x \wedge (g \text{ has\_field\_derivative } g'\ x)$  (at  $x$  within  $S$ )
  by (intro has_field_derivative_series[of  $S\ f\ f'\ g'\ x0$ ] assms  $A \text{ has\_field\_derivative\_at\_within}$ )
  then obtain  $g$  where  $g: \bigwedge x. x \in S \implies (\lambda n. f\ n\ x) \text{ sums } g\ x$ 
  by (intro exE)
  by (intro has_field_derivative_g' x) (at  $x$  within  $S$ ) by blast
  from  $g(2)$ [OF  $x$ ] have  $g': (g \text{ has\_derivative } (*) (g'\ x))$  (at  $x$ )

```

```

  by (simp add: has_field_derivative_def S)
  have (( $\lambda x. \sum n. f\ n\ x$ ) has_derivative (*)) (g' x) (at x)
  by (rule has_derivative_transform_within_open[OF g' ‹open S› x])
    (insert g, auto simp: sums_iff)
  thus ( $\lambda x. \sum n. f\ n\ x$ ) field_differentiable (at x) unfolding differentiable_def
  by (auto simp: summable_def field_differentiable_def has_field_derivative_def)
qed

```

Caratheodory characterization

```

lemma field_differentiable_caratheodory_at:
  f field_differentiable (at z)  $\longleftrightarrow$ 
    ( $\exists g. (\forall w. f(w) - f(z) = g(w) * (w - z)) \wedge \text{continuous (at z) } g$ )
  using CARAT_DERIV [of f]
  by (simp add: field_differentiable_def has_field_derivative_def)

```

```

lemma field_differentiable_caratheodory_within:
  f field_differentiable (at z within s)  $\longleftrightarrow$ 
    ( $\exists g. (\forall w. f(w) - f(z) = g(w) * (w - z)) \wedge \text{continuous (at z within s) } g$ )
  using DERIV_caratheodory_within [of f]
  by (simp add: field_differentiable_def has_field_derivative_def)

```

4.10.17 Field derivative

definition $\text{deriv} :: ('a \Rightarrow 'a::\text{real\_normed\_field}) \Rightarrow 'a \Rightarrow 'a$ **where**
 $\text{deriv } f\ x \equiv \text{SOME } D. \text{DERIV } f\ x :> D$

```

lemma DERIV_imp_deriv: DERIV f x :> f'  $\implies$  deriv f x = f'
  unfolding deriv_def by (metis some_equality DERIV_unique)

```

```

lemma DERIV_deriv_iff_has_field_derivative:
  DERIV f x :> deriv f x  $\longleftrightarrow$  ( $\exists f'. (f \text{ has\_field\_derivative } f') (at\ x)$ )
  by (auto simp: has_field_derivative_def DERIV_imp_deriv)

```

```

lemma DERIV_deriv_iff_real_differentiable:
  fixes x :: real
  shows DERIV f x :> deriv f x  $\longleftrightarrow$  f differentiable at x
  unfolding differentiable_def by (metis DERIV_imp_deriv has_real_derivative_iff)

```

```

lemma DERIV_deriv_iff_field_differentiable:
  DERIV f x :> deriv f x  $\longleftrightarrow$  f field_differentiable at x
  unfolding field_differentiable_def by (metis DERIV_imp_deriv)

```

```

lemma vector_derivative_of_real_left:
  assumes f differentiable at x
  shows vector_derivative ( $\lambda x. \text{of\_real } (f\ x)$ ) (at x) = of_real (deriv f x)
  by (metis DERIV_deriv_iff_real_differentiable assms has_vector_derivative_of_real
    vector_derivative_at)

```

```

lemma vector_derivative_of_real_right:

```

assumes f *field_differentiable* at (*of_real* x)
shows $\text{vector\_derivative } (\lambda x. f \text{ (of\_real } x)) \text{ (at } x) = \text{deriv } f \text{ (of\_real } x)$
by (*metis DERIV_deriv_iff_field_differentiable* *assms* *has_vector_derivative_real_field* *vector_derivative_at*)

lemma *deriv_cong_ev*:

assumes *eventually* $(\lambda x. f \ x = g \ x) \text{ (nhds } x) \ x = y$
shows $\text{deriv } f \ x = \text{deriv } g \ y$
proof –
have $(\lambda D. (f \text{ has\_field\_derivative } D) \text{ (at } x)) = (\lambda D. (g \text{ has\_field\_derivative } D) \text{ (at } y))$
by (*intro ext DERIV_cong_ev refl assms*)
thus *?thesis* **by** (*simp add: deriv_def assms*)
qed

lemma *higher_deriv_cong_ev*:

assumes *eventually* $(\lambda x. f \ x = g \ x) \text{ (nhds } x) \ x = y$
shows $(\text{deriv } \sim^n) f \ x = (\text{deriv } \sim^n) g \ y$
proof –
from *assms*(1) **have** *eventually* $(\lambda x. (\text{deriv } \sim^n) f \ x = (\text{deriv } \sim^n) g \ x) \text{ (nhds } x)$
proof (*induction n arbitrary: f g*)
case (*Suc n*)
from *Suc.prem*s **have** *eventually* $(\lambda y. \text{eventually } (\lambda z. f \ z = g \ z) \text{ (nhds } y)) \text{ (nhds } x)$
by (*simp add: eventually_eventually*)
hence *eventually* $(\lambda x. \text{deriv } f \ x = \text{deriv } g \ x) \text{ (nhds } x)$
by *eventually_elim* (*rule deriv_cong_ev, simp_all*)
thus *?case* **by** (*auto intro!: deriv_cong_ev Suc simp: funpow_Suc_right simp* *del: funpow.simps*)
qed auto
with $\langle x = y \rangle$ *eventually_nhds_x_imp_x* **show** *?thesis* **by** *blast*
qed

lemma *real_derivative_chain*:

fixes $x :: \text{real}$
shows f *differentiable* at $x \implies g$ *differentiable* at $(f \ x)$
 $\implies \text{deriv } (g \circ f) \ x = \text{deriv } g \ (f \ x) * \text{deriv } f \ x$
by (*metis DERIV_deriv_iff_real_differentiable DERIV_chain DERIV_imp_deriv*)
lemma *field_derivative_eq_vector_derivative*:
 $(\text{deriv } f \ x) = \text{vector\_derivative } f \text{ (at } x)$
by (*simp add: mult.commute deriv_def vector_derivative_def has_vector_derivative_def* *has_field_derivative_def*)

proposition *field_differentiable_derivI*:

f *field_differentiable* at $x \implies (f \text{ has\_field\_derivative } \text{deriv } f \ x) \text{ (at } x)$
by (*simp add: field_differentiable_def DERIV_deriv_iff_has_field_derivative*)

lemma *vector_derivative_chain_at_general*:


```

assumes  $f$  differentiable at  $x$   $g$  field_differentiable at  $(f\ x)$ 
shows  $\text{vector\_derivative } (g \circ f) \text{ (at } x) = \text{vector\_derivative } f \text{ (at } x) * \text{deriv } g \text{ (f$ 
 $x)$ 
apply (rule vector_derivative_at [OF field_vector_diff_chain_at])
using assms vector_derivative_works by (auto simp: field_differentiable_derivI)

```

```

lemma deriv_chain:
 $f$  field_differentiable at  $x \implies g$  field_differentiable at  $(f\ x)$ 
 $\implies \text{deriv } (g \circ f) \ x = \text{deriv } g \text{ (f } x) * \text{deriv } f \ x$ 
by (metis DERIV_deriv_iff_field_differentiable DERIV_chain DERIV_imp_deriv)

```

```

lemma deriv_linear [simp]:  $\text{deriv } (\lambda w. c * w) = (\lambda z. c)$ 
by (metis DERIV_imp_deriv DERIV_cmult_Id)

```

```

lemma deriv_uminus [simp]:  $\text{deriv } (\lambda w. -w) = (\lambda z. -1)$ 
using deriv_linear[of  $-1$ ] by (simp del: deriv_linear)

```

```

lemma deriv_ident [simp]:  $\text{deriv } (\lambda w. w) = (\lambda z. 1)$ 
by (metis DERIV_imp_deriv DERIV_ident)

```

```

lemma deriv_id [simp]:  $\text{deriv id} = (\lambda z. 1)$ 
by (simp add: id_def)

```

```

lemma deriv_const [simp]:  $\text{deriv } (\lambda w. c) = (\lambda z. 0)$ 
by (metis DERIV_imp_deriv DERIV_const)

```

```

lemma deriv_add [simp]:
 $\llbracket f \text{ field\_differentiable at } z; g \text{ field\_differentiable at } z \rrbracket$ 
 $\implies \text{deriv } (\lambda w. f\ w + g\ w) \ z = \text{deriv } f \ z + \text{deriv } g \ z$ 
unfolding DERIV_deriv_iff_field_differentiable[symmetric]
by (auto intro!: DERIV_imp_deriv derivative_intros)

```

```

lemma deriv_diff [simp]:
 $\llbracket f \text{ field\_differentiable at } z; g \text{ field\_differentiable at } z \rrbracket$ 
 $\implies \text{deriv } (\lambda w. f\ w - g\ w) \ z = \text{deriv } f \ z - \text{deriv } g \ z$ 
unfolding DERIV_deriv_iff_field_differentiable[symmetric]
by (auto intro!: DERIV_imp_deriv derivative_intros)

```

```

lemma deriv_mult [simp]:
 $\llbracket f \text{ field\_differentiable at } z; g \text{ field\_differentiable at } z \rrbracket$ 
 $\implies \text{deriv } (\lambda w. f\ w * g\ w) \ z = f\ z * \text{deriv } g \ z + \text{deriv } f \ z * g\ z$ 
unfolding DERIV_deriv_iff_field_differentiable[symmetric]
by (auto intro!: DERIV_imp_deriv derivative_eq_intros)

```

```

lemma deriv_cmult:
 $f$  field_differentiable at  $z \implies \text{deriv } (\lambda w. c * f\ w) \ z = c * \text{deriv } f \ z$ 
by simp

```

```

lemma deriv_cmult_right:

```

f field_differentiable at $z \implies \text{deriv } (\lambda w. f w * c) z = \text{deriv } f z * c$
by simp

lemma deriv_inverse [simp]:

$\llbracket f \text{ field\_differentiable at } z; f z \neq 0 \rrbracket$

$\implies \text{deriv } (\lambda w. \text{inverse } (f w)) z = - \text{deriv } f z / f z ^ 2$

unfolding DERIV_deriv_iff_field_differentiable[symmetric]

by (safe intro!: DERIV_imp_deriv derivative_eq_intros) (auto simp: field_split_simps power2_eq_square)

lemma deriv_divide [simp]:

$\llbracket f \text{ field\_differentiable at } z; g \text{ field\_differentiable at } z; g z \neq 0 \rrbracket$

$\implies \text{deriv } (\lambda w. f w / g w) z = (\text{deriv } f z * g z - f z * \text{deriv } g z) / g z ^ 2$

by (simp add: field_class.field_divide_inverse field_differentiable_inverse)

(simp add: field_split_simps power2_eq_square)

lemma deriv_cdivide_right:

$f \text{ field\_differentiable at } z \implies \text{deriv } (\lambda w. f w / c) z = \text{deriv } f z / c$

by (simp add: field_class.field_divide_inverse)

lemma deriv_compose_linear:

$f \text{ field\_differentiable at } (c * z) \implies \text{deriv } (\lambda w. f (c * w)) z = c * \text{deriv } f (c * z)$

apply (rule DERIV_imp_deriv)

unfolding DERIV_deriv_iff_field_differentiable [symmetric]

by (metis (full_types) DERIV_chain2 DERIV_cmult_Id mult.commute)

lemma nonzero_deriv_nonconstant:

assumes df: DERIV $f \xi \text{ :> } df$ **and** S: open $S \ \xi \in S$ **and** $df \neq 0$

shows $\neg f \text{ constant\_on } S$

unfolding constant_on_def

by (metis $\langle df \neq 0 \rangle$ has_field_derivative_transform_within_open [OF df S] DERIV_const DERIV_unique)

4.10.18 Relation between convexity and derivative

proposition convex_on_imp_above_tangent:

assumes convex: convex_on $A f$ **and** connected: connected A

assumes $c: c \in \text{interior } A$ **and** $x: x \in A$

assumes deriv: $(f \text{ has\_field\_derivative } f') \text{ (at } c \text{ within } A)$

shows $f x - f c \geq f' * (x - c)$

proof (cases $x c$ rule: linorder_cases)

assume xc: $x > c$

let $?A' = \text{interior } A \cap \{c<..\}$

from c **have** $c \in \text{interior } A \cap \text{closure } \{c<..\}$ **by** auto

also have $\dots \subseteq \text{closure } (\text{interior } A \cap \{c<..\})$ **by** (intro open_Int_closure_subset)

auto

finally have at c within $?A' \neq \text{bot}$ **by** (subst at_within_eq_bot_iff) auto

moreover from deriv **have** $((\lambda y. (f y - f c) / (y - c)) \longrightarrow f') \text{ (at } c \text{ within } A)$

```

?A')
  unfolding has_field_derivative_iff using interior_subset[of A] by (blast intro:
tendsto_mono at_le)
  moreover from eventually_at_right_real[OF xc]
  have eventually  $(\lambda y. (f y - f c) / (y - c) \leq (f x - f c) / (x - c))$  (at_right c)
  proof eventually_elim
    fix y assume y:  $y \in \{c <..<x\}$ 
    with convex_connected x c have  $f y \leq (f x - f c) / (x - c) * (y - c) + f c$ 
    using interior_subset[of A]
    by (intro convex_onD_Icc' convex_on_subset[OF convex] connected_contains_Icc)
  auto
  hence  $f y - f c \leq (f x - f c) / (x - c) * (y - c)$  by simp
  thus  $(f y - f c) / (y - c) \leq (f x - f c) / (x - c)$  using y xc by (simp add:
field_split_simps)
  qed
  hence eventually  $(\lambda y. (f y - f c) / (y - c) \leq (f x - f c) / (x - c))$  (at c within
?A')
  by (blast intro: filter_leD at_le)
  ultimately have  $f' \leq (f x - f c) / (x - c)$  by (simp add: tendsto_upperbound)
  thus ?thesis using xc by (simp add: field_simps)
next
  assume xc:  $x < c$ 
  let ?A' = interior A  $\cap \{..<c\}$ 
  from c have  $c \in \text{interior } A \cap \text{closure } \{..<c\}$  by auto
  also have  $\dots \subseteq \text{closure } (\text{interior } A \cap \{..<c\})$  by (intro open_Int_closure_subset)
  auto
  finally have at c within ?A'  $\neq \text{bot}$  by (subst at_within_eq_bot_iff) auto
  moreover from deriv have  $((\lambda y. (f y - f c) / (y - c)) \longrightarrow f')$  (at c within
?A')
  unfolding has_field_derivative_iff using interior_subset[of A] by (blast intro:
tendsto_mono at_le)
  moreover from eventually_at_left_real[OF xc]
  have eventually  $(\lambda y. (f y - f c) / (y - c) \geq (f x - f c) / (x - c))$  (at_left c)
  proof eventually_elim
    fix y assume y:  $y \in \{x <..<c\}$ 
    with convex_connected x c have  $f y \leq (f x - f c) / (c - x) * (c - y) + f c$ 
    using interior_subset[of A]
    by (intro convex_onD_Icc'' convex_on_subset[OF convex] connected_contains_Icc)
  auto
  hence  $f y - f c \leq (f x - f c) * ((c - y) / (c - x))$  by simp
  also have  $(c - y) / (c - x) = (y - c) / (x - c)$  using y xc by (simp add:
field_simps)
  finally show  $(f y - f c) / (y - c) \geq (f x - f c) / (x - c)$  using y xc
  by (simp add: field_split_simps)
  qed
  hence eventually  $(\lambda y. (f y - f c) / (y - c) \geq (f x - f c) / (x - c))$  (at c within
?A')
  by (blast intro: filter_leD at_le)
  ultimately have  $f' \geq (f x - f c) / (x - c)$  by (simp add: tendsto_lowerbound)

```

thus ?thesis using xc by (simp add: field_simps)
qed simp_all

4.10.19 Partial derivatives

lemma eventually_at_Pair_within_TimesI1:
fixes x::'a::metric_space
assumes $\forall_F x'$ in at x within X. P x'
assumes P x
shows $\forall_F (x', y')$ in at (x, y) within $X \times Y$. P x'
proof –
from assms[unfolded eventually_at_topological]
obtain S where S: open S x $\in$ S $\wedge$ x'. x' $\in$ X $\implies$ x' $\in$ S $\implies$ P x'
by metis
show $\forall_F (x', y')$ in at (x, y) within $X \times Y$. P x'
unfolding eventually_at_topological
by (auto intro!: exI[where x=S $\times$ UNIV] S open_Times)
qed

lemma eventually_at_Pair_within_TimesI2:
fixes x::'a::metric_space
assumes $\forall_F y'$ in at y within Y. P y' P y
shows $\forall_F (x', y')$ in at (x, y) within $X \times Y$. P y'
proof –
from assms[unfolded eventually_at_topological]
obtain S where S: open S y $\in$ S $\wedge$ y'. y' $\in$ Y $\implies$ y' $\in$ S $\implies$ P y'
by metis
show $\forall_F (x', y')$ in at (x, y) within $X \times Y$. P y'
unfolding eventually_at_topological
by (auto intro!: exI[where x=UNIV $\times$ S] S open_Times)
qed

proposition has_derivativepartialsI:
fixes f::'a::real_normed_vector $\Rightarrow$ 'b::real_normed_vector $\Rightarrow$ 'c::real_normed_vector
assumes fx: ((λ x. f x y) has_derivative fx) (at x within X)
assumes fy: ($\wedge$ x y. x $\in$ X $\implies$ y $\in$ Y $\implies$ ((λ y. f x y) has_derivative blinfun_apply (fy x y)) (at y within Y)
assumes fy_cont[unfolded continuous_within]: continuous (at (x, y) within $X \times Y$) (λ (x, y). fy x y)
assumes y $\in$ Y convex Y
shows ((λ (x, y). f x y) has_derivative (λ (tx, ty). fx tx + fy x y ty)) (at (x, y) within $X \times Y$)
proof (safe intro!: has_derivativeI tendstoI, goal_cases)
case (2 e')
interpret fx: bounded_linear fx using fx by (rule has_derivative_bounded_linear)
define e where e = e' / 9
have e > 0 using $\langle e' > 0 \rangle$ by (simp add: e_def)

from fy_cont[THEN tendstoD, OF $\langle e > 0 \rangle$]

```

have  $\forall_F (x', y')$  in at  $(x, y)$  within  $X \times Y$ .  $\text{dist } (fy \ x' \ y') \ (fy \ x \ y) < e$ 
  by (auto simp: split_beta')
from this[unfolded eventually_at] obtain  $d'$  where
   $d' > 0$ 
   $\wedge x' \ y'. \ x' \in X \implies y' \in Y \implies (x', y') \neq (x, y) \implies \text{dist } (x', y') \ (x, y) < d'$ 
 $\implies$ 
   $\text{dist } (fy \ x' \ y') \ (fy \ x \ y) < e$ 
  by auto
then
  have  $d': x' \in X \implies y' \in Y \implies \text{dist } (x', y') \ (x, y) < d' \implies \text{dist } (fy \ x' \ y') \ (fy \ x \ y) < e$ 
    for  $x' \ y'$ 
    using  $\langle 0 < e \rangle$ 
    by (cases  $(x', y') = (x, y)$ ) auto
  define  $d$  where  $d = d' / \text{sqrt } 2$ 
  have  $d > 0$  using  $\langle 0 < d' \rangle$  by (simp add: d_def)
  have  $d: x' \in X \implies y' \in Y \implies \text{dist } x' \ x < d \implies \text{dist } y' \ y < d \implies \text{dist } (fy \ x' \ y') \ (fy \ x \ y) < e$ 
    for  $x' \ y'$ 
    by (auto simp: dist_prod_def d_def intro!: d' real_sqrt_sum_squares_less)

let  $?S = \text{ball } y \ d \cap Y$ 
have convex  $?S$ 
  by (auto intro!: convex_Int  $\langle \text{convex } Y \rangle$ )
{
  fix  $x'::'a$  and  $y'::'b$ 
  assume  $x': x' \in X$  and  $y': y' \in Y$ 
  assume  $dx': \text{dist } x' \ x < d$  and  $dy': \text{dist } y' \ y < d$ 
  have  $\text{norm } (fy \ x' \ y' - fy \ x' \ y) \leq \text{dist } (fy \ x' \ y') \ (fy \ x \ y) + \text{dist } (fy \ x' \ y) \ (fy \ x \ y)$ 
    by norm
  also have  $\text{dist } (fy \ x' \ y') \ (fy \ x \ y) < e$ 
    by (rule d; fact)
  also have  $\text{dist } (fy \ x' \ y) \ (fy \ x \ y) < e$ 
    by (auto intro!: d simp: dist_prod_def  $x' \ \langle d > 0 \rangle \ \langle y \in Y \rangle \ dx')$ 
  finally
  have  $\text{norm } (fy \ x' \ y' - fy \ x' \ y) < e + e$ 
    by arith
  then have  $\text{onorm } (\text{blinfun\_apply } (fy \ x' \ y') - \text{blinfun\_apply } (fy \ x' \ y)) < e + e$ 
    by (auto simp: norm_blinfun.rep_eq blinfun.diff_left[abs_def] fun_diff_def)
} note onorm = this

have  $\text{ev\_mem}: \forall_F (x', y')$  in at  $(x, y)$  within  $X \times Y$ .  $(x', y') \in X \times Y$ 
  using  $\langle y \in Y \rangle$ 
  by (auto simp: eventually_at intro!: zero_less_one)
moreover
  have  $\text{ev\_dist}: \forall_F xy$  in at  $(x, y)$  within  $X \times Y$ .  $\text{dist } xy \ (x, y) < d$  if  $d > 0$  for  $d$ 
    using eventually_at_ball[OF that]
    by (rule eventually_elim2) (auto simp: dist_commute intro!: eventually_True)

```

```

note ev_dist[OF  $\langle 0 < d \rangle$ ]
ultimately
have  $\forall_F (x', y')$  in at  $(x, y)$  within  $X \times Y$ .
  norm  $(f x' y' - f x' y - (fy x' y) (y' - y)) \leq \text{norm } (y' - y) * (e + e)$ 
proof (eventually_elim, safe)
  fix  $x' y'$ 
  assume  $x' \in X$  and  $y': y' \in Y$ 
  assume dist: dist  $(x', y')$   $(x, y) < d$ 
  then have dx: dist  $x' x < d$  and dy: dist  $y' y < d$ 
    unfolding dist_prod_def fst_conv snd_conv atomize_conj
  by (metis le_less_trans real_sqrt_sum_squares_ge1 real_sqrt_sum_squares_ge2)
  {
    fix t::real
    assume  $t \in \{0 .. 1\}$ 
    then have  $y + t *_R (y' - y) \in \text{closed\_segment } y y'$ 
      by (auto simp: closed_segment_def algebra_simps intro!: exI[where x=t])
    also
    have  $\dots \subseteq \text{ball } y d \cap Y$ 
      using  $\langle y \in Y \rangle \langle 0 < d \rangle \text{ dy } y'$ 
      by (intro  $\langle \text{convex } ?S \rangle [\text{unfolded convex\_contains\_segment, rule\_format, of } y y']$ )
      (auto simp: dist_commute)
    finally have  $y + t *_R (y' - y) \in ?S$  .
  } note seg = this

  have  $\bigwedge x. x \in \text{ball } y d \cap Y \implies \text{onorm } (\text{blinfun\_apply } (fy x' x) - \text{blinfun\_apply } (fy x' y)) \leq e + e$ 
    by (safe intro!: onorm_less_imp_le  $\langle x' \in X \rangle \text{ dx}$ ) (auto simp: dist_commute  $\langle 0 < d \rangle \langle y \in Y \rangle$ )
    with seg_has_derivative_subset[OF assms(2)[OF  $\langle x' \in X \rangle$ ]]
    show norm  $(f x' y' - f x' y - (fy x' y) (y' - y)) \leq \text{norm } (y' - y) * (e + e)$ 
      by (rule differentiable_bound_linearization[where S=?S])
      (auto intro!:  $\langle 0 < d \rangle \langle y \in Y \rangle$ )
qed
moreover
let  $?le = \lambda x'. \text{norm } (f x' y - f x y - (fx) (x' - x)) \leq \text{norm } (x' - x) * e$ 
from fx[unfolded has_derivative_within, THEN conjunct2, THEN tendstoD, OF  $\langle 0 < e \rangle$ ]
have  $\forall_F x'$  in at  $x$  within  $X$ .  $?le x'$ 
  by eventually_elim (simp,
    simp add: dist_norm field_split_simps split: if_split_asm)
then have  $\forall_F (x', y')$  in at  $(x, y)$  within  $X \times Y$ .  $?le x'$ 
  by (rule eventually_at_Pair_within_TimesI1)
    (simp add: blinfun.bilinear_simps)
moreover have  $\forall_F (x', y')$  in at  $(x, y)$  within  $X \times Y$ . norm  $((x', y') - (x, y)) \neq 0$ 
  unfolding norm_eq_zero right_minus_eq
  by (auto simp: eventually_at intro!: zero_less_one)
moreover

```

```

from  $fy\_cont[THEN tendstoD, OF \langle 0 < e \rangle]$ 
have  $\forall_F x' \text{ in at } x \text{ within } X. \text{ norm } (fy\ x'\ y - fy\ x\ y) < e$ 
  unfolding eventually_at
  using  $\langle y \in Y \rangle$ 
  by (auto simp: dist_prod_def dist_norm)
then have  $\forall_F (x', y') \text{ in at } (x, y) \text{ within } X \times Y. \text{ norm } (fy\ x'\ y - fy\ x\ y) < e$ 
  by (rule eventually_at_Pair_within_TimesI1)
  (simp add: blinfun.bilinear_simps \langle 0 < e \rangle)
ultimately
have  $\forall_F (x', y') \text{ in at } (x, y) \text{ within } X \times Y.$ 
   $\text{ norm } ((f\ x'\ y' - f\ x\ y - (f\ x\ (x' - x) + f\ y\ x\ y\ (y' - y))) /_R$ 
   $\text{ norm } ((x', y') - (x, y)))$ 
   $< e'$ 
  apply eventually_elim
proof safe
  fix  $x'\ y'$ 
  have  $\text{ norm } (f\ x'\ y' - f\ x\ y - (f\ x\ (x' - x) + f\ y\ x\ y\ (y' - y))) \leq$ 
     $\text{ norm } (f\ x'\ y' - f\ x'\ y - f\ y\ x'\ y\ (y' - y)) +$ 
     $\text{ norm } (f\ y\ x\ y\ (y' - y) - f\ y\ x'\ y\ (y' - y)) +$ 
     $\text{ norm } (f\ x'\ y - f\ x\ y - f\ x\ (x' - x))$ 
  by norm
  also
  assume  $nz: \text{ norm } ((x', y') - (x, y)) \neq 0$ 
  and  $nfy: \text{ norm } (fy\ x'\ y - fy\ x\ y) < e$ 
  assume  $\text{ norm } (f\ x'\ y' - f\ x'\ y - \text{blinfun\_apply } (fy\ x'\ y)\ (y' - y)) \leq \text{ norm } (y' - y) * (e + e)$ 
  also assume  $\text{ norm } (f\ x'\ y - f\ x\ y - (f\ x)\ (x' - x)) \leq \text{ norm } (x' - x) * e$ 
  also
  have  $\text{ norm } ((fy\ x\ y)\ (y' - y) - (fy\ x'\ y)\ (y' - y)) \leq \text{ norm } ((fy\ x\ y) - (fy\ x'\ y)) * \text{ norm } (y' - y)$ 
  by (auto simp: blinfun.bilinear_simps[symmetric] intro!: norm_blinfun)
  also have  $\dots \leq (e + e) * \text{ norm } (y' - y)$ 
  using  $\langle e > 0 \rangle$  nfy
  by (auto simp: norm_minus_commute intro!: mult_right_mono)
  also have  $\text{ norm } (x' - x) * e \leq \text{ norm } (x' - x) * (e + e)$ 
  using  $\langle 0 < e \rangle$  by simp
  also have  $\text{ norm } (y' - y) * (e + e) + (e + e) * \text{ norm } (y' - y) + \text{ norm } (x' - x) * (e + e) \leq$ 
     $(\text{ norm } (y' - y) + \text{ norm } (x' - x)) * (4 * e)$ 
  using  $\langle e > 0 \rangle$ 
  by (simp add: algebra_simps)
  also have  $\dots \leq 2 * \text{ norm } ((x', y') - (x, y)) * (4 * e)$ 
  using  $\langle 0 < e \rangle$  real_sqrt_sum_squares_ge1 [of norm (x' - x) norm (y' - y)]
    real_sqrt_sum_squares_ge2 [of norm (y' - y) norm (x' - x)]
  by (auto intro!: mult_right_mono simp: norm_prod_def)
  simp del: real_sqrt_sum_squares_ge1 real_sqrt_sum_squares_ge2)
  also have  $\dots \leq \text{ norm } ((x', y') - (x, y)) * (8 * e)$ 
  by simp
  also have  $\dots < \text{ norm } ((x', y') - (x, y)) * e'$ 

```

```

    using ⟨0 < e'⟩ nz
    by (auto simp: e_def)
    finally show norm ((f x' y' - f x y - (f x (x' - x) + f y x y (y' - y))) /R norm
      ((x', y') - (x, y))) < e'
    by (simp add: dist_norm) (auto simp add: field_split_simps)
  qed
  then show ?case
    by eventually_elim (auto simp: dist_norm field_simps)
next
  from has_derivative_bounded_linear[OF fx]
  obtain fxb where fx = blinfun_apply fxb
    by (metis bounded_linear_Blinfun_apply)
  then show bounded_linear (λ(tx, ty). fx tx + blinfun_apply (fy x y) ty)
    by (auto intro!: bounded_linear_intros simp: split_beta')
qed

```

4.10.20 Differentiable case distinction

```

lemma has_derivative_within>If_eq:
  ((λx. if P x then f x else g x) has_derivative f') (at x within S) =
    (bounded_linear f' ∧
      ((λy. (if P y then (f y - ((if P x then f x else g x) + f' (y - x))) /R norm (y
        - x)
        else (g y - ((if P x then f x else g x) + f' (y - x))) /R norm (y - x)))
        ⟶ 0) (at x within S))
  (is _ = _ ∧ (?if ⟶ 0) _)
proof -
  have (λy. (1 / norm (y - x)) *R
    ((if P y then f y else g y) -
      ((if P x then f x else g x) + f' (y - x)))) = ?if
    by (auto simp: inverse_eq_divide)
  thus ?thesis by (auto simp: has_derivative_within)
qed

```

```

lemma has_derivative_within>If_within_closures:
  assumes f': x ∈ S ∪ (closure S ∩ closure T) ⟹
    (f has_derivative f' x) (at x within S ∪ (closure S ∩ closure T))
  assumes g': x ∈ T ∪ (closure S ∩ closure T) ⟹
    (g has_derivative g' x) (at x within T ∪ (closure S ∩ closure T))
  assumes connect: x ∈ closure S ⟹ x ∈ closure T ⟹ f x = g x
  assumes connect': x ∈ closure S ⟹ x ∈ closure T ⟹ f' x = g' x
  assumes x_in: x ∈ S ∪ T
  shows ((λx. if x ∈ S then f x else g x) has_derivative
    (if x ∈ S then f' x else g' x)) (at x within (S ∪ T))
proof -
  from f' x_in interpret f': bounded_linear if x ∈ S then f' x else (λx. 0)
    by (auto simp add: has_derivative_within)
  from g' interpret g': bounded_linear if x ∈ T then g' x else (λx. 0)
    by (auto simp add: has_derivative_within)

```



```

have bl: bounded_linear (if x ∈ S then f' x else g' x)
  using f'.scaleR f'.bounded f'.add g'.scaleR g'.bounded g'.add x_in
  by (unfold_locales; force)
show ?thesis
  using f' g' closure_subset[of T] closure_subset[of S]
  unfolding has_derivative_within>If_eq
  by (intro conjI bl tendsto>If_within_closures x_in)
  (auto simp: has_derivative_within inverse_eq_divide connect connect' subsetD)
qed

```

lemma *has_vector_derivative>If_within_closures*:

```

assumes x_in: x ∈ S ∪ T
assumes u = S ∪ T
assumes f': x ∈ S ∪ (closure S ∩ closure T) ⇒
  (f has_vector_derivative f' x) (at x within S ∪ (closure S ∩ closure T))
assumes g': x ∈ T ∪ (closure S ∩ closure T) ⇒
  (g has_vector_derivative g' x) (at x within T ∪ (closure S ∩ closure T))
assumes connect: x ∈ closure S ⇒ x ∈ closure T ⇒ f x = g x
assumes connect': x ∈ closure S ⇒ x ∈ closure T ⇒ f' x = g' x
shows ((λx. if x ∈ S then f x else g x) has_vector_derivative
  (if x ∈ S then f' x else g' x)) (at x within u)
unfolding has_vector_derivative_def assms
using x_in
  apply (intro has_derivative>If_within_closures[where ?f' = λx a. a *R f' x
and ?g' = λx a. a *R g' x,
  THEN has_derivative_eq_rhs])
  subgoal by (rule f'[unfolded has_vector_derivative_def]; assumption)
  subgoal by (rule g'[unfolded has_vector_derivative_def]; assumption)
  by (auto simp: assms)

```

4.10.21 The Inverse Function Theorem

lemma *linear_injective_contraction*:

```

assumes linear f c < 1 and le: ∧x. norm (f x - x) ≤ c * norm x
shows inj f
unfolding linear_injective_0[OF ⟨linear f⟩]

```

proof *safe*

fix x

assume f x = 0

with le [of x] have norm x ≤ c * norm x

by simp

then show x = 0

using ⟨c < 1⟩ by (simp add: mult_le_cancel_right1)

qed

From an online proof by J. Michael Boardman, Department of Mathematics,
Johns Hopkins University

lemma *inverse_function_theorem_scaled*:

```

fixes  $f::'a::euclidean\_space \Rightarrow 'a$ 
and  $f'::'a \Rightarrow ('a \Rightarrow_L 'a)$ 
assumes  $open\ U$ 
and  $derf: \bigwedge x. x \in U \Rightarrow (f\ has\_derivative\ blinfun\_apply\ (f'\ x))\ (at\ x)$ 
and  $contf: continuous\_on\ U\ f'$ 
and  $0 \in U$  and  $[simp]: f\ 0 = 0$ 
and  $id: f'\ 0 = id\_blinfun$ 
obtains  $U'\ V\ g\ g'$  where  $open\ U'\ U' \subseteq U\ 0 \in U'\ open\ V\ 0 \in V\ homeomorphism\ U'\ V\ f\ g$ 
 $\bigwedge y. y \in V \Rightarrow (g\ has\_derivative\ (g'\ y))\ (at\ y)$ 
 $\bigwedge y. y \in V \Rightarrow g'\ y = inv\ (blinfun\_apply\ (f'(g\ y)))$ 
 $\bigwedge y. y \in V \Rightarrow bij\ (blinfun\_apply\ (f'(g\ y)))$ 
proof –
obtain  $d1$  where  $cball\ 0\ d1 \subseteq U\ d1 > 0$ 
using  $\langle open\ U \rangle \langle 0 \in U \rangle open\_contains\_cball$  by  $blast$ 
obtain  $d2$  where  $d2: \bigwedge x. [x \in U; dist\ x\ 0 \leq d2] \Rightarrow dist\ (f'\ x)\ (f'\ 0) < 1/2$ 
 $0 < d2$ 
using  $continuous\_onE\ [OF\ contf, of\ 0\ 1/2]$  by  $(metis\ \langle 0 \in U \rangle half\_gt\_zero\_iff\ zero\_less\_one)$ 
obtain  $\delta$  where  $le: \bigwedge x. norm\ x \leq \delta \Rightarrow dist\ (f'\ x)\ id\_blinfun \leq 1/2$  and  $0 < \delta$ 
and  $subU: cball\ 0\ \delta \subseteq U$ 
proof
show  $min\ d1\ d2 > 0$ 
by  $(simp\ add: \langle 0 < d1 \rangle \langle 0 < d2 \rangle)$ 
show  $cball\ 0\ (min\ d1\ d2) \subseteq U$ 
using  $\langle cball\ 0\ d1 \subseteq U \rangle$  by  $auto$ 
show  $dist\ (f'\ x)\ id\_blinfun \leq 1/2$  if  $norm\ x \leq min\ d1\ d2$  for  $x$ 
using  $\langle cball\ 0\ d1 \subseteq U \rangle\ d2$  that  $id$  by  $fastforce$ 
qed
let  $?D = cball\ 0\ \delta$ 
define  $V::'a\ set$  where  $V \equiv ball\ 0\ (\delta/2)$ 
have  $4: norm\ (f\ (x + h) - f\ x - h) \leq 1/2 * norm\ h$ 
if  $x \in ?D\ x+h \in ?D$  for  $x\ h$ 
proof –
let  $?w = \lambda x. f\ x - x$ 
have  $B: \bigwedge x. x \in ?D \Rightarrow onorm\ (blinfun\_apply\ (f'\ x - id\_blinfun)) \leq 1/2$ 
by  $(metis\ dist\_norm\ le\ mem\_cball\_0\ norm\_blinfun.rep\_eq)$ 
have  $\bigwedge x. x \in ?D \Rightarrow (?w\ has\_derivative\ (blinfun\_apply\ (f'\ x - id\_blinfun)))$ 
 $(at\ x)$ 
by  $(rule\ derivative\_eq\_intros\ derf\ subsetD\ [OF\ subU] \mid force\ simp: blinfun.diff\_left)+$ 
then have  $Dw: \bigwedge x. x \in ?D \Rightarrow (?w\ has\_derivative\ (blinfun\_apply\ (f'\ x - id\_blinfun)))\ (at\ x\ within\ ?D)$ 
using  $has\_derivative\_at\_withinI$  by  $blast$ 
have  $norm\ (?w\ (x+h) - ?w\ x) \leq (1/2) * norm\ h$ 
using  $differentiable\_bound\ [OF\ convex\_cball\ Dw\ B]$  that by  $fastforce$ 
then show  $?thesis$ 
by  $(auto\ simp: algebra\_simps)$ 

```

```

qed
have for_g:  $\exists!x. \text{norm } x < \delta \wedge f x = y$  if  $y: \text{norm } y < \delta/2$  for  $y$ 
proof -
  let ?u =  $\lambda x. x + (y - f x)$ 
  have *:  $\text{norm } (?u x) < \delta$  if  $x \in ?D$  for  $x$ 
  proof -
    have fxx:  $\text{norm } (f x - x) \leq \delta/2$ 
    using 4 [of 0 x]  $\langle 0 < \delta \rangle \langle f 0 = 0 \rangle$  that by auto
    have  $\text{norm } (?u x) \leq \text{norm } y + \text{norm } (f x - x)$ 
    by (metis add.commute add_diff_eq norm_minus_commute norm_triangle_ineq)
    also have  $\dots < \delta/2 + \delta/2$ 
    using fxx y by auto
    finally show ?thesis
    by simp
  qed
qed
have  $\exists!x \in ?D. ?u x = x$ 
proof (rule banach_fix)
  show  $\text{cball } 0 \ \delta \neq \{\}$ 
  using  $\langle 0 < \delta \rangle$  by auto
  show  $(\lambda x. x + (y - f x)) \text{ ` } \text{cball } 0 \ \delta \subseteq \text{cball } 0 \ \delta$ 
  using * by force
  have  $\text{dist } (x + (y - f x)) (xh + (y - f xh)) * 2 \leq \text{dist } x \ xh$ 
  if  $\text{norm } x \leq \delta$  and  $\text{norm } xh \leq \delta$  for  $x \ xh$ 
  using that 4 [of x xh-x] by (auto simp: dist_norm norm_minus_commute
algebra_simps)
  then show  $\forall x \in \text{cball } 0 \ \delta. \forall ya \in \text{cball } 0 \ \delta. \text{dist } (x + (y - f x)) (ya + (y - f
ya)) \leq (1/2) * \text{dist } x \ ya$ 
  by auto
  qed (auto simp: complete_eq_closed)
  then show ?thesis
  by (metis * add_cancel_right_right eq_iff_diff_eq_0 le_less mem_cball_0)
qed
define g where  $g \equiv \lambda y. \text{THE } x. \text{norm } x < \delta \wedge f x = y$ 
have  $g: \text{norm } (g y) < \delta \wedge f (g y) = y$  if  $\text{norm } y < \delta/2$  for  $y$ 
  unfolding g_def using that theI' [OF for_g] by meson
then have fg[simp]:  $f (g y) = y$  if  $y \in V$  for  $y$ 
  using that by (auto simp: V_def)
have 5:  $\text{norm } (g y' - g y) \leq 2 * \text{norm } (y' - y)$  if  $y \in V \ y' \in V$  for  $y \ y'$ 
proof -
  have no:  $\text{norm } (g y) \leq \delta \ \text{norm } (g y') \leq \delta$  and [simp]:  $f (g y) = y$ 
  using that g unfolding V_def by force+
  have  $\text{norm } (g y' - g y) \leq \text{norm } (g y' - g y - (y' - y)) + \text{norm } (y' - y)$ 
  by (simp add: add_commute norm_triangle_sub)
  also have  $\dots \leq (1/2) * \text{norm } (g y' - g y) + \text{norm } (y' - y)$ 
  using 4 [of g y g y' - g y] that no by (simp add: g_norm_minus_commute
V_def)
  finally show ?thesis
  by auto
qed

```

```

have contg: continuous_on V g
proof
  fix y::'a and e::real
  assume 0 < e and y: y ∈ V
  show ∃ d>0. ∀ x'∈V. dist x' y < d ⟶ dist (g x') (g y) ≤ e
  proof (intro exI conjI ballI impI)
    show 0 < e/2
    by (simp add: ‹0 < e›)
  qed (use 5 y in ‹force simp: dist_norm›)
qed
show thesis
proof
  define U' where U' ≡ (f -' V) ∩ ball 0 δ
  have contf: continuous_on U f
  using derf has_derivative_at_withinI by (fast intro: has_derivative_continuous_on)
  then have continuous_on (ball 0 δ) f
  by (meson ball_subset_cball continuous_on_subset subU)
  then show open U'
  by (simp add: U'_def V_def Int_commute continuous_open_preimage)
  show 0 ∈ U' U' ⊆ U open V 0 ∈ V
  using ‹0 < δ› subU by (auto simp: U'_def V_def)
  show hom: homeomorphism U' V f g
  proof
    show continuous_on U' f
    using ‹U' ⊆ U› contf continuous_on_subset by blast
    show continuous_on V g
    using contg by blast
    show f -' U' ⊆ V
    using U'_def by blast
    show g -' V ⊆ U'
    by (simp add: U'_def V_def g_image_subset_iff)
    show g (f x) = x if x ∈ U' for x
    by (metis that fg Int_iff U'_def V_def for_g g mem_ball_0 vimage_eq)
    show f (g y) = y if y ∈ V for y
    using that by (simp add: g V_def)
  qed
  show bij: bij (blnfun_apply (f'(g y))) if y ∈ V for y
  proof -
    have inj: inj (blnfun_apply (f' (g y)))
    proof (rule linear_injective_contraction)
      show linear (blnfun_apply (f' (g y)))
      using blnfun.bounded_linear_right bounded_linear_def by blast
    next
      fix x
      have norm (blnfun_apply (f' (g y)) x - x) = norm (blnfun_apply (f' (g
y) - id_blnfun) x)
      by (simp add: blnfun.diff_left)
      also have ... ≤ norm (f' (g y) - id_blnfun) * norm x
      by (rule norm_blnfun)
    qed
  
```

```

also have ... ≤ (1/2) * norm x
proof (rule mult_right_mono)
  show norm (f' (g y) - id_blinfun) ≤ 1/2
    using that g [of y] le by (auto simp: V_def dist_norm)
qed auto
finally show norm (blinfun_apply (f' (g y)) x - x) ≤ (1/2) * norm x .
qed auto
moreover
have surj (blinfun_apply (f' (g y)))
  using blinfun.bounded_linear_right bounded_linear_def
  by (blast intro!: linear_inj_imp_surj [OF _ inj])
ultimately show ?thesis
  using bijI by blast
qed
define g' where g' ≡ λy. inv (blinfun_apply (f' (g y)))
show (g has_derivative g' y) (at y) if y ∈ V for y
proof -
  have gy: g y ∈ U
    using g subU that unfolding V_def by fastforce
  obtain e where e: ∧h. f (g y + h) = y + blinfun_apply (f' (g y)) h + e h
    and e0: (λh. norm (e h) / norm h) -> 0
    using iffD1 [OF has_derivative_iff_Ex derf [OF gy]] ⟨y ∈ V⟩ by auto
  have [simp]: e 0 = 0
    using e [of 0] that by simp
  let ?INV = inv (blinfun_apply (f' (g y)))
  have inj: inj (blinfun_apply (f' (g y)))
    using bij bij_betw_def that by blast
  have (g has_derivative g' y) (at y within V)
    unfolding has_derivative_at_within_iff_Ex [OF ⟨y ∈ V⟩ ⟨open V⟩]
  proof
    show blinv: bounded_linear (g' y)
      unfolding g'_def using derf gy inj inj_linear_imp_inv_bounded_linear
by blast
    define eg where eg ≡ λk. - ?INV (e (g (y+k) - g y))
    have g (y+k) = g y + g' y k + eg k if y + k ∈ V for k
    proof -
      have ?INV k = ?INV (blinfun_apply (f' (g y)) (g (y+k) - g y) + e (g
(y+k) - g y))
      using e [of g(y+k) - g y] that by simp
      then have g (y+k) = g y + ?INV k - ?INV (e (g (y+k) - g y))
      using inj blinv by (simp add: linear_simps g'_def)
      then show ?thesis
        by (auto simp: eg_def g'_def)
    qed
    moreover have (λk. norm (eg k) / norm k) -> 0
    proof (rule Lim_null_comparison)
      let ?g = λk. 2 * onorm ?INV * norm (e (g (y+k) - g y)) / norm (g
(y+k) - g y)
      show ∀_F k in at 0. norm (norm (eg k) / norm k) ≤ ?g k

```

```

    unfolding eventually_at_topological
  proof (intro exI conjI ballI impI)
    show open ((+)(-y) ' V)
      using ⟨open V⟩ open_translation by blast
    show 0 ∈ (+)(-y) ' V
      by (simp add: that)
    show norm (norm (eg k) / norm k) ≤ 2 * onorm (inv (blinfun_apply
(f' (g y)))) * norm (e (g (y+k) - g y)) / norm (g (y+k) - g y)
      if k ∈ (+)(-y) ' V k ≠ 0 for k
    proof -
      have y+k ∈ V
        using that by auto
      have norm (norm (eg k) / norm k) ≤ onorm ?INV * norm (e (g (y+k)
- g y)) / norm k
        using blinv g'_def onorm by (force simp: eg_def divide_simps)
      also have ... = (norm (g (y+k) - g y) / norm k) * (onorm ?INV *
(norm (e (g (y+k) - g y)) / norm (g (y+k) - g y)))
        by (simp add: divide_simps)
      also have ... ≤ 2 * (onorm ?INV * (norm (e (g (y+k) - g y)) /
norm (g (y+k) - g y)))
        apply (rule mult_right_mono)
      using 5 [of y y+k] ⟨y ∈ V⟩ ⟨y + k ∈ V⟩ onorm_pos_le [OF blinv]
      apply (auto simp: divide_simps zero_le_mult_iff zero_le_divide_iff
g'_def)
    done
    finally show norm (norm (eg k) / norm k) ≤ 2 * onorm ?INV * norm
(e (g (y+k) - g y)) / norm (g (y+k) - g y)
      by simp
    qed
  qed
  have 1: (λh. norm (e h) / norm h) -0→ (norm (e 0) / norm 0)
    using e0 by auto
  have 2: (λk. g (y+k) - g y) -0→ 0
    using contg ⟨open V⟩ ⟨y ∈ V⟩ LIM_offset_zero_iff LIM_zero_iff
at_within_open continuous_on_def by fastforce
  from tendsto_compose [OF 1 2, simplified]
  have (λk. norm (e (g (y+k) - g y)) / norm (g (y+k) - g y)) -0→ 0 .
  from tendsto_mult_left [OF this] show ?g -0→ 0 by auto
  qed
  ultimately show ∃ e. (∀ k. y + k ∈ V ⟶ g (y+k) = g y + g' y k + e k)
    ∧ (λk. norm (e k) / norm k) -0→ 0
    by blast
  qed
  then show ?thesis
    by (metis ⟨open V⟩ at_within_open that)
  qed
  show g' y = inv (blinfun_apply (f' (g y)))
    if y ∈ V for y
    by (simp add: g'_def)

```

qed
qed

We need all this to justify the scaling and translations.

```

theorem inverse_function_theorem:
  fixes  $f::'a::euclidean\_space \Rightarrow 'a$ 
    and  $f'::'a \Rightarrow ('a \Rightarrow_L 'a)$ 
  assumes open U
    and derf:  $\bigwedge x. x \in U \implies (f \text{ has\_derivative } (blinfun\_apply (f' x))) \text{ (at } x)$ 
    and contf: continuous_on U f'
    and  $x0 \in U$ 
    and invf:  $invf \circ_L f' x0 = id\_blinfun$ 
  obtains  $U' V g g'$  where open U' U'  $\subseteq U$  x0  $\in U'$  open V f x0  $\in V$  homeo-
morphism U' V f g
     $\bigwedge y. y \in V \implies (g \text{ has\_derivative } (g' y)) \text{ (at } y)$ 
     $\bigwedge y. y \in V \implies g' y = inv (blinfun\_apply (f'(g y)))$ 
     $\bigwedge y. y \in V \implies bij (blinfun\_apply (f'(g y)))$ 
proof -
  have apply1 [simp]:  $\bigwedge i. blinfun\_apply invf (blinfun\_apply (f' x0) i) = i$ 
    by (metis blinfun_apply_blinfun_compose blinfun_apply_id_blinfun invf)
  have apply2 [simp]:  $\bigwedge i. blinfun\_apply (f' x0) (blinfun\_apply invf i) = i$ 
    by (metis apply1 bij_inv_eq_iff blinfun_bij1 invf)
  have [simp]:  $(range (blinfun\_apply invf)) = UNIV$ 
    using apply1 surjI by blast
  let  $?f = invf \circ (\lambda x. (f \circ (+) x0) x - f x0)$ 
  let  $?f' = \lambda x. invf \circ_L (f' (x + x0))$ 
  obtain  $U' V g g'$  where open U' and U': U'  $\subseteq (+) (-x0) \text{ ' } U \text{ } 0 \in U'$ 
    and open V 0  $\in V$  and hom: homeomorphism U' V ?f g
    and derg:  $\bigwedge y. y \in V \implies (g \text{ has\_derivative } (g' y)) \text{ (at } y)$ 
    and  $g'$ :  $\bigwedge y. y \in V \implies g' y = inv (?f'(g y))$ 
    and bij:  $\bigwedge y. y \in V \implies bij (?f'(g y))$ 
  proof (rule inverse_function_theorem_scaled [of (+) (-x0) ' U ?f ?f'])
    show ope: open ((+) (- x0) ' U)
      using  $\langle open U \rangle$  open_translation by blast
    show  $(?f \text{ has\_derivative } blinfun\_apply (?f' x)) \text{ (at } x)$ 
      if  $x \in (+) (- x0) \text{ ' } U$  for  $x$ 
      using that
      apply clarify
      apply (rule derf derivative_eq_intros | simp add: blinfun_compose.rep_eq) +
      done
    have YY:  $(\lambda x. f' (x + x0)) -u-x0 \rightarrow f' u$ 
      if  $f' -u \rightarrow f' u$   $u \in U$  for  $u$ 
      using that LIM_offset [where k = x0] by (auto simp: algebra_simps)
    then have continuous_on ((+) (- x0) ' U)  $(\lambda x. f' (x + x0))$ 
      using contf  $\langle open U \rangle$  Lim_at_imp_Lim_at_within
      by (fastforce simp: continuous_on_def at_within_open_NO_MATCH ope)
    then show continuous_on ((+) (- x0) ' U) ?f'
      by (intro continuous_intros) simp
  qed (auto simp: invf  $\langle x0 \in U \rangle$ )

```

```

show thesis
proof
  let ?U' = (+)x0 ' U'
  let ?V = ((+)(f x0) ∘ f' x0) ' V
  let ?g = (+)x0 ∘ g ∘ invf ∘ (+)(- f x0)
  let ?g' = λy. inv (blinfun_apply (f' (?g y)))
  show oU': open ?U'
    by (simp add: ⟨open U'⟩ open_translation)
  show subU: ?U' ⊆ U
    using ComplI ⟨U' ⊆ (+) (- x0) ' U⟩ by auto
  show x0 ∈ ?U'
    by (simp add: ⟨0 ∈ U'⟩)
  show open ?V
    using blinfun_bij2 [OF invf]
  by (metis ⟨open V⟩ bij_is_surj blinfun.bounded_linear_right bounded_linear_def
image_comp open_surjective_linear_image open_translation)
  show f x0 ∈ ?V
    using ⟨0 ∈ V⟩ image_iff by fastforce
  show homeomorphism ?U' ?V f ?g
  proof
    show continuous_on ?U' f
    by (meson subU continuous_on_eq continuous_at derf has_derivative_continuous
oU' subsetD)
    have ?f ' U' ⊆ V
      using hom homeomorphism_image1 by blast
    then show f ' ?U' ⊆ ?V
      unfolding image_subset_iff
    by (clarsimp simp: image_def) (metis apply2 add commute diff_add_cancel)
    show ?g ' ?V ⊆ ?U'
      using hom invf by (auto simp: image_def homeomorphism_def)
    show ?g (f x) = x
      if x ∈ ?U' for x
      using that hom homeomorphism_apply1 by fastforce
    have continuous_on V g
      using hom homeomorphism_def by blast
    then show continuous_on ?V ?g
      by (intro continuous_intros) (auto elim!: continuous_on_subset)
    have fg: ?f (g x) = x if x ∈ V for x
      using hom homeomorphism_apply2 that by blast
    show f (?g y) = y
      if y ∈ ?V for y
      using that fg by (simp add: image_iff) (metis apply2 add commute
diff_add_cancel)
    qed
    show (?g has_derivative ?g' y) (at y) bij (blinfun_apply (f' (?g y)))
      if y ∈ ?V for y
  proof -
    have 1: bij (blinfun_apply invf)
      using blinfun_bij1 invf by blast

```



```

    then have 2: bij (blinfun_apply (f' (x0 + g x))) if  $x \in V$  for  $x$ 
      by (metis add.commute bij bij_betw_comp_iff2 blinfun_compose.rep_eq
that top_greatest)
    then show bij (blinfun_apply (f' (?g y)))
      using that by auto
    have  $g' x \circ \text{blinfun\_apply } \text{invf} = \text{inv } (\text{blinfun\_apply } (f' (x0 + g x)))$ 
      if  $x \in V$  for  $x$ 
      using that
      by (simp add:  $g' \circ \text{inv\_distrib } \text{blinfun\_compose.rep\_eq } 1 \ 2 \ \text{add.commute}$ 
bij_is_inj flip: o_assoc)
    then show (?g has_derivative ?g' y) (at y)
      using that invf
      by clarsimp (rule derg_derivative_eq_intros | simp flip: id_def)+
  qed
qed auto
qed

```

4.10.22 Piecewise differentiable functions

definition *piecewise_differentiable_on*
 (infixr *piecewise'_differentiable'_on* 50)
 where $f \text{ piecewise\_differentiable\_on } i \equiv$
 $\text{continuous\_on } i \ f \ \wedge$
 $(\exists S. \text{finite } S \wedge (\forall x \in i - S. f \text{ differentiable } (\text{at } x \text{ within } i)))$

lemma *piecewise_differentiable_on_imp_continuous_on*:
 $f \text{ piecewise\_differentiable\_on } S \implies \text{continuous\_on } S \ f$
 by (simp add: *piecewise_differentiable_on_def*)

lemma *piecewise_differentiable_on_subset*:
 $f \text{ piecewise\_differentiable\_on } S \implies T \leq S \implies f \text{ piecewise\_differentiable\_on } T$
 using *continuous_on_subset*
 unfolding *piecewise_differentiable_on_def*
 apply safe
 apply (blast elim: *continuous_on_subset*)
 by (meson Diff_iff differentiable_within_subset subsetCE)

lemma *differentiable_on_imp_piecewise_differentiable*:
 fixes $a:: 'a::\{\text{linorder\_topology, real\_normed\_vector}\}$
 shows $f \text{ differentiable\_on } \{a..b\} \implies f \text{ piecewise\_differentiable\_on } \{a..b\}$
 apply (simp add: *piecewise_differentiable_on_def differentiable_imp_continuous_on*)
 apply (rule_tac $x=\{a,b\}$ in *exI*, simp add: *differentiable_on_def*)
 done

lemma *differentiable_imp_piecewise_differentiable*:
 $(\lambda x. x \in S \implies f \text{ differentiable } (\text{at } x \text{ within } S))$
 $\implies f \text{ piecewise\_differentiable\_on } S$
 by (auto simp: *piecewise_differentiable_on_def differentiable_imp_continuous_on*)

differentiable_on_def
intro: differentiable_within_subset)

lemma *piecewise_differentiable_const [iff]:* $(\lambda x. z)$ *piecewise_differentiable_on* S
by (*simp add: differentiable_imp_piecewise_differentiable*)

lemma *piecewise_differentiable_compose:*
 $\llbracket f \text{ piecewise\_differentiable\_on } S; g \text{ piecewise\_differentiable\_on } (f \text{ ' } S);$
 $\wedge x. \text{finite } (S \cap f^{-1}\{x\}) \rrbracket$
 $\implies (g \circ f) \text{ piecewise\_differentiable\_on } S$
apply (*simp add: piecewise_differentiable_on_def, safe*)
apply (*blast intro: continuous_on_compose2*)
apply (*rename_tac A B*)
apply (*rule_tac x=A $\cup$ ($\bigcup_{x \in B}. S \cap f^{-1}\{x\}$) in exI*)
apply (*blast intro!: differentiable_chain_within*)
done

lemma *piecewise_differentiable_affine:*
fixes $m::\text{real}$
assumes $f \text{ piecewise\_differentiable\_on } ((\lambda x. m *_{\mathbb{R}} x + c) \text{ ' } S)$
shows $(f \circ (\lambda x. m *_{\mathbb{R}} x + c)) \text{ piecewise\_differentiable\_on } S$
proof (*cases m = 0*)
case *True*
then show *?thesis*
unfolding *o_def*
by (*force intro: differentiable_imp_piecewise_differentiable differentiable_const*)
next
case *False*
show *?thesis*
apply (*rule piecewise_differentiable_compose [OF differentiable_imp_piecewise_differentiable]*)
apply (*rule assms derivative_intros | simp add: False vimage_def real_vector_affinity_eq*)
done
qed

lemma *piecewise_differentiable_cases:*
fixes $c::\text{real}$
assumes $f \text{ piecewise\_differentiable\_on } \{a..c\}$
 $g \text{ piecewise\_differentiable\_on } \{c..b\}$
 $a \leq c \leq b$
shows $(\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x) \text{ piecewise\_differentiable\_on } \{a..b\}$
proof –
obtain $S T$ **where** *st: finite S finite T*
and *fd:* $\wedge x. x \in \{a..c\} - S \implies f \text{ differentiable at } x \text{ within } \{a..c\}$
and *gd:* $\wedge x. x \in \{c..b\} - T \implies g \text{ differentiable at } x \text{ within } \{c..b\}$
using *assms*
by (*auto simp: piecewise_differentiable_on_def*)
have *finabc:* $\text{finite } (\{a,b,c\} \cup (S \cup T))$
by (*metis <finite S> <finite T> finite_Un finite_insert finite.emptyI*)
have *continuous_on* $\{a..c\}$ f *continuous_on* $\{c..b\}$ g

```

    using assms piecewise_differentiable_on_def by auto
  then have continuous_on {a..b} ( $\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x$ )
    using continuous_on_cases [OF closed_real_atLeastAtMost [of a c],
      OF closed_real_atLeastAtMost [of c b],
      of f g  $\lambda x. x \leq c$ ] assms
    by (force simp: ivl_disj_un_two_touch)
  moreover
  { fix x
    assume x:  $x \in \{a..b\} - (\{a,b,c\} \cup (S \cup T))$ 
    have ( $\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x$ ) differentiable at x within {a..b} (is ?diff_fg)
    proof (cases x c rule: le_cases)
      case le show ?diff_fg
      proof (rule differentiable_transform_within [where d = dist x c])
        have f differentiable at x
          using x le fd [of x] at_within_interior [of x {a..c}] by simp
        then show f differentiable at x within {a..b}
          by (simp add: differentiable_at_withinI)
      qed (use x le st dist_real_def in auto)
    next
      case ge show ?diff_fg
      proof (rule differentiable_transform_within [where d = dist x c])
        have g differentiable at x
          using x ge gd [of x] at_within_interior [of x {c..b}] by simp
        then show g differentiable at x within {a..b}
          by (simp add: differentiable_at_withinI)
      qed (use x ge st dist_real_def in auto)
    qed
  }
  then have  $\exists S. \text{finite } S \wedge$ 
    ( $\forall x \in \{a..b\} - S. (\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x)$  differentiable at x
    within {a..b})
    by (meson finabc)
  ultimately show ?thesis
    by (simp add: piecewise_differentiable_on_def)
  qed

```

lemma *piecewise_differentiable_neg:*

f piecewise_differentiable_on $S \implies (\lambda x. -(f x))$ piecewise_differentiable_on S
 by (auto simp: piecewise_differentiable_on_def continuous_on_minus)

lemma *piecewise_differentiable_add:*

assumes f piecewise_differentiable_on i
 g piecewise_differentiable_on i
 shows $(\lambda x. f x + g x)$ piecewise_differentiable_on i

proof —

obtain $S T$ where st: finite S finite T
 $\forall x \in i - S. f$ differentiable at x within i
 $\forall x \in i - T. g$ differentiable at x within i
 using assms by (auto simp: piecewise_differentiable_on_def)

```

then have finite (S ∪ T) ∧ (∀ x ∈ i - (S ∪ T). (λx. f x + g x) differentiable at
x within i)
by auto
moreover have continuous_on i f continuous_on i g
using assms piecewise_differentiable_on_def by auto
ultimately show ?thesis
by (auto simp: piecewise_differentiable_on_def continuous_on_add)
qed

```

```

lemma piecewise_differentiable_diff:
  ‖f piecewise_differentiable_on S; g piecewise_differentiable_on S‖
  ⇒ (λx. f x - g x) piecewise_differentiable_on S
unfolding diff_conv_add_uminus
by (metis piecewise_differentiable_add piecewise_differentiable_neg)

```

4.10.23 The concept of continuously differentiable

John Harrison writes as follows:

“The usual assumption in complex analysis texts is that a path γ should be piecewise continuously differentiable, which ensures that the path integral exists at least for any continuous f , since all piecewise continuous functions are integrable. However, our notion of validity is weaker, just piecewise differentiability... [namely] continuity plus differentiability except on a finite set... [Our] underlying theory of integration is the Kurzweil-Henstock theory. In contrast to the Riemann or Lebesgue theory (but in common with a simple notion based on antiderivatives), this can integrate all derivatives.”

"Formalizing basic complex analysis." From Insight to Proof: Festschrift in Honour of Andrzej Trybulec. Studies in Logic, Grammar and Rhetoric 10.23 (2007): 151-165.

And indeed he does not assume that his derivatives are continuous, but the penalty is unreasonably difficult proofs concerning winding numbers. We need a self-contained and straightforward theorem asserting that all derivatives can be integrated before we can adopt Harrison's choice.

definition $C1\_differentiable\_on :: (real \Rightarrow 'a::real\_normed\_vector) \Rightarrow real\ set \Rightarrow bool$

(**infix** $C1'\_differentiable'\_on$ 50)

where

$f\ C1\_differentiable\_on\ S \longleftrightarrow$

$(\exists D. (\forall x \in S. (f\ has\_vector\_derivative\ (D\ x))\ (at\ x)) \wedge continuous\_on\ S\ D)$

lemma $C1\_differentiable\_on\_eq$:

$f\ C1\_differentiable\_on\ S \longleftrightarrow$

$(\forall x \in S. f\ differentiable\ at\ x) \wedge continuous\_on\ S\ (\lambda x. vector\_derivative\ f\ (at\ x))$

(**is** ?lhs = ?rhs)

proof

```

  assume ?lhs
  then show ?rhs
    unfolding C1_differentiable_on_def
    by (metis (no_types, lifting) continuous_on_eq differentiableI_vector vector_derivative_at)
next
  assume ?rhs
  then show ?lhs
    using C1_differentiable_on_def vector_derivative_works by fastforce
qed

```

```

lemma C1_differentiable_on_subset:
  f C1_differentiable_on T  $\implies$   $S \subseteq T \implies$  f C1_differentiable_on S
  unfolding C1_differentiable_on_def continuous_on_eq continuous_within
  by (blast intro: continuous_within_subset)

```

```

lemma C1_differentiable_compose:
  assumes fg: f C1_differentiable_on S g C1_differentiable_on (f ` S) and fin:
 $\bigwedge x. \text{finite } (S \cap f^{-1}\{x\})$ 
  shows (g  $\circ$  f) C1_differentiable_on S
proof -
  have  $\bigwedge x. x \in S \implies$  g  $\circ$  f differentiable at x
  by (meson C1_differentiable_on_eq assms differentiable_chain_at imageI)
  moreover have continuous_on S ( $\lambda x. \text{vector\_derivative } (g \circ f) \text{ (at } x)$ )
  proof (rule continuous_on_eq [of _  $\lambda x. \text{vector\_derivative } f \text{ (at } x) *_{\mathbb{R}} \text{vector\_derivative } g \text{ (at } (f x))$ ])
    show continuous_on S ( $\lambda x. \text{vector\_derivative } f \text{ (at } x) *_{\mathbb{R}} \text{vector\_derivative } g \text{ (at } (f x))$ )
    using fg
    apply (clarsimp simp add: C1_differentiable_on_eq)
    apply (rule Limits.continuous_on_scaleR, assumption)
    by (metis (mono_tags, lifting) continuous_at_imp_continuous_on continuous_on_compose continuous_on_cong differentiable_imp_continuous_within o_def)
    show  $\bigwedge x. x \in S \implies \text{vector\_derivative } f \text{ (at } x) *_{\mathbb{R}} \text{vector\_derivative } g \text{ (at } (f x)) = \text{vector\_derivative } (g \circ f) \text{ (at } x)$ 
    by (metis (mono_tags, opaque_lifting) C1_differentiable_on_eq fg imageI vector_derivative_chain_at)
  qed
  ultimately show ?thesis
  by (simp add: C1_differentiable_on_eq)
qed

```

```

lemma C1_diff_imp_diff: f C1_differentiable_on S  $\implies$  f differentiable_on S
  by (simp add: C1_differentiable_on_eq differentiable_at_imp_differentiable_on)

```

```

lemma C1_differentiable_on_ident [simp, derivative_intros]: ( $\lambda x. x$ ) C1_differentiable_on S
  by (auto simp: C1_differentiable_on_eq)

```

lemma *C1_differentiable_on_const* [*simp*, *derivative_intros*]: $(\lambda z. a) \text{ C1\_differentiable\_on } S$

by (*auto simp: C1_differentiable_on_eq*)

lemma *C1_differentiable_on_add* [*simp*, *derivative_intros*]:

$f \text{ C1\_differentiable\_on } S \implies g \text{ C1\_differentiable\_on } S \implies (\lambda x. f\ x + g\ x) \text{ C1\_differentiable\_on } S$

unfolding *C1_differentiable_on_eq* **by** (*auto intro: continuous_intros*)

lemma *C1_differentiable_on_minus* [*simp*, *derivative_intros*]:

$f \text{ C1\_differentiable\_on } S \implies (\lambda x. - f\ x) \text{ C1\_differentiable\_on } S$

unfolding *C1_differentiable_on_eq* **by** (*auto intro: continuous_intros*)

lemma *C1_differentiable_on_diff* [*simp*, *derivative_intros*]:

$f \text{ C1\_differentiable\_on } S \implies g \text{ C1\_differentiable\_on } S \implies (\lambda x. f\ x - g\ x) \text{ C1\_differentiable\_on } S$

unfolding *C1_differentiable_on_eq* **by** (*auto intro: continuous_intros*)

lemma *C1_differentiable_on_mult* [*simp*, *derivative_intros*]:

fixes $f\ g :: \text{real} \Rightarrow 'a :: \text{real\_normed\_algebra}$

shows $f \text{ C1\_differentiable\_on } S \implies g \text{ C1\_differentiable\_on } S \implies (\lambda x. f\ x * g\ x) \text{ C1\_differentiable\_on } S$

unfolding *C1_differentiable_on_eq*

by (*auto simp: continuous_on_add continuous_on_mult continuous_at_imp_continuous_on differentiable_imp_continuous_within*)

lemma *C1_differentiable_on_scaleR* [*simp*, *derivative_intros*]:

$f \text{ C1\_differentiable\_on } S \implies g \text{ C1\_differentiable\_on } S \implies (\lambda x. f\ x *_R g\ x) \text{ C1\_differentiable\_on } S$

unfolding *C1_differentiable_on_eq*

by (*rule continuous_intros | simp add: continuous_at_imp_continuous_on differentiable_imp_continuous_within*)⁺

lemma *C1_differentiable_on_of_real* [*derivative_intros*]: $\text{of\_real } C1\_differentiable\_on\ S$

unfolding *C1_differentiable_on_def*

by (*smt (verit, del_insts) DERIV_ident UNIV_I continuous_on_const has_vector_derivative_of_real has_vector_derivative_transform*)

definition *piecewise_C1_differentiable_on*

(**infixr** *piecewise'_C1'_differentiable'_on* 50)

where $f \text{ piecewise\_C1\_differentiable\_on } i \equiv$

$\text{continuous\_on } i\ f \wedge$

$(\exists S. \text{finite } S \wedge (f \text{ C1\_differentiable\_on } (i - S)))$

lemma *C1_differentiable_imp_piecewise*:

$f \text{ C1\_differentiable\_on } S \implies f \text{ piecewise\_C1\_differentiable\_on } S$

by (auto simp: piecewise_C1_differentiable_on_def C1_differentiable_on_eq continuous_at_imp_continuous_on differentiable_imp_continuous_within)

lemma piecewise_C1_imp_differentiable:

f piecewise_C1_differentiable_on $i \implies f$ piecewise_differentiable_on i

by (auto simp: piecewise_C1_differentiable_on_def piecewise_differentiable_on_def C1_differentiable_on_def differentiable_def has_vector_derivative_def intro: has_derivative_at_withinI)

lemma piecewise_C1_differentiable_compose [derivative_intros]:

assumes fg : f piecewise_C1_differentiable_on S g piecewise_C1_differentiable_on $(f^{-1} S)$ and fin : $\bigwedge x. \text{finite } (S \cap f^{-1} \{x\})$

shows $(g \circ f)$ piecewise_C1_differentiable_on S

proof –

have continuous_on S $(\lambda x. g (f x))$

by (metis continuous_on_compose2 fg order_refl piecewise_C1_differentiable_on_def)

moreover have $\exists T. \text{finite } T \wedge g \circ f$ C1_differentiable_on $S - T$

proof –

obtain F where finite F and F : f C1_differentiable_on $S - F$ and f : f piecewise_C1_differentiable_on S

using fg by (auto simp: piecewise_C1_differentiable_on_def)

obtain G where finite G and G : g C1_differentiable_on $f^{-1} S - G$ and g : g piecewise_C1_differentiable_on $f^{-1} S$

using fg by (auto simp: piecewise_C1_differentiable_on_def)

show ?thesis

proof (intro exI conjI)

show finite $(F \cup (\bigcup_{x \in G. S \cap f^{-1} \{x\}}))$

using fin by (auto simp only: Int_Union finite_F finite_G finite_UN finite_imageI)

show $g \circ f$ C1_differentiable_on $S - (F \cup (\bigcup_{x \in G. S \cap f^{-1} \{x\}}))$

apply (rule C1_differentiable_compose)

apply (blast intro: C1_differentiable_on_subset [OF F])

apply (blast intro: C1_differentiable_on_subset [OF G])

by (simp add: C1_differentiable_on_subset G Diff_Int_distrib2 fin)

qed

qed

ultimately show ?thesis

by (simp add: piecewise_C1_differentiable_on_def)

qed

lemma piecewise_C1_differentiable_on_subset:

f piecewise_C1_differentiable_on $S \implies T \leq S \implies f$ piecewise_C1_differentiable_on T

by (auto simp: piecewise_C1_differentiable_on_def elim!: continuous_on_subset C1_differentiable_on_subset)

lemma C1_differentiable_imp_continuous_on:

f C1_differentiable_on $S \implies \text{continuous\_on } S$

unfolding C1_differentiable_on_eq continuous_on_eq continuous_within

```

using differentiable_at_withinI differentiable_imp_continuous_within by blast

lemma C1_differentiable_on_empty [iff, derivative_intros]: f C1_differentiable_on
{}
  unfolding C1_differentiable_on_def
  by auto

lemma piecewise_C1_differentiable_affine:
  fixes m::real
  assumes f piecewise_C1_differentiable_on (( $\lambda x. m * x + c$ ) ‘ S)
  shows (f o ( $\lambda x. m * x + c$ )) piecewise_C1_differentiable_on S
proof (cases m = 0)
  case True
  then show ?thesis
    unfolding o_def by (auto simp: piecewise_C1_differentiable_on_def)
next
  case False
  have *:  $\bigwedge x. \text{finite } (S \cap \{y. m * y + c = x\})$ 
    using False not_finite_existsD by fastforce
  show ?thesis
    apply (rule piecewise_C1_differentiable_compose [OF C1_differentiable_imp_piecewise])
    apply (rule * assms derivative_intros | simp add: False vimage_def) +
    done
qed

lemma piecewise_C1_differentiable_cases [derivative_intros]:
  fixes c::real
  assumes f piecewise_C1_differentiable_on {a..c}
  g piecewise_C1_differentiable_on {c..b}
  a  $\leq$  c c  $\leq$  b f c = g c
  shows ( $\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x$ ) piecewise_C1_differentiable_on {a..b}
proof -
  obtain S T where st: f C1_differentiable_on ({a..c} - S)
    g C1_differentiable_on ({c..b} - T)
    finite S finite T
  using assms
  by (force simp: piecewise_C1_differentiable_on_def)
  then have f_diff: f differentiable_on {a..<c} - S
    and g_diff: g differentiable_on {c<..b} - T
  by (simp_all add: C1_differentiable_on_eq differentiable_at_withinI differentiable_on_def)
  have continuous_on {a..c} f continuous_on {c..b} g
    using assms piecewise_C1_differentiable_on_def by auto
  then have cab: continuous_on {a..b} ( $\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x$ )
    using continuous_on_cases [OF closed_real_atLeastAtMost [of a c],
      OF closed_real_atLeastAtMost [of c b],
      of f g  $\lambda x. x \leq c$ ] assms
  by (force simp: ivl_disj_un_two_touch)
  { fix x

```



```

assume  $x: x \in \{a..b\} - \text{insert } c (S \cup T)$ 
have  $(\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x) \text{ differentiable at } x$  (is ?diff_fg)
proof (cases  $x \ c$  rule: le_cases)
  case le show ?diff_fg
    apply (rule differentiable_transform_within [where  $f=f$  and  $d = \text{dist } x \ c$ ])
    using  $x \ \text{dist\_real\_def } le \ st$  by (auto simp: C1_differentiable_on_eq)
  next
    case ge show ?diff_fg
      apply (rule differentiable_transform_within [where  $f=g$  and  $d = \text{dist } x$ 
 $c$ ])
      using  $\text{dist\_nz } x \ \text{dist\_real\_def } ge \ st \ x$  by (auto simp: C1_differentiable_on_eq)
    qed
  }
then have  $(\forall x \in \{a..b\} - \text{insert } c (S \cup T). (\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x) \text{ differentiable at } x)$ 
by auto
moreover
{ assume  $fcon: \text{continuous\_on } (\{a <..< c\} - S) (\lambda x. \text{vector\_derivative } f \ (at \ x))$ 
  and  $gcon: \text{continuous\_on } (\{c <..< b\} - T) (\lambda x. \text{vector\_derivative } g \ (at \ x))$ 
have  $\text{open } (\{a <..< c\} - S) \ \text{open } (\{c <..< b\} - T)$ 
using  $st$  by (simp_all add: open_Diff finite_imp_closed)
moreover have  $\text{continuous\_on } (\{a <..< c\} - S) (\lambda x. \text{vector\_derivative } (\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x) \ (at \ x))$ 
proof -
  have  $((\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x) \text{ has\_vector\_derivative } \text{vector\_derivative } f \ (at \ x))$ 
     $(at \ x)$ 
    if  $a < x \ x < c \ x \notin S$  for  $x$ 
  proof -
    have  $f: f \text{ differentiable at } x$ 
    by (meson C1_differentiable_on_eq Diff_iff atLeastAtMost_iff less_eq_real_def
 $st(1)$  that)
    show ?thesis
    using that
    apply (rule_tac  $f=f$  and  $d=\text{dist } x \ c$  in has_vector_derivative_transform_within)
    apply (auto simp: dist_norm vector_derivative_works [symmetric]  $f$ )
    done
  qed
  then show ?thesis
  by (metis (no_types, lifting) continuous_on_eq [OF  $fcon$ ] DiffE greaterThanLessThan_iff
 $\text{vector\_derivative\_at}$ )
  qed
moreover have  $\text{continuous\_on } (\{c <..< b\} - T) (\lambda x. \text{vector\_derivative } (\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x) \ (at \ x))$ 
proof -
  have  $((\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x) \text{ has\_vector\_derivative } \text{vector\_derivative } g \ (at \ x))$ 
     $(at \ x)$ 
    if  $c < x \ x < b \ x \notin T$  for  $x$ 
  proof -
    have  $g: g \text{ differentiable at } x$ 

```

```

    by (metis C1_differentiable_on_eq DiffD1 DiffI atLeastAtMost_diff_ends
greaterThanLessThan_iff st(2) that)
    show ?thesis
    using that
    apply (rule_tac f=g and d=dist x c in has_vector_derivative_transform_within)
    apply (auto simp: dist_norm vector_derivative_works [symmetric] g)
    done
  qed
  then show ?thesis
  by (metis (no_types, lifting) continuous_on_eq [OF gcon] DiffE greaterThanLessThan_iff
vector_derivative_at)
  qed
  ultimately have continuous_on ( $\{a <..<b\} - \text{insert } c (S \cup T)$ )
    ( $\lambda x. \text{vector\_derivative } (\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x) (at x)$ )
  by (rule continuous_on_subset [OF continuous_on_open_Un], auto)
} note * = this
have continuous_on ( $\{a <..<b\} - \text{insert } c (S \cup T)$ ) ( $\lambda x. \text{vector\_derivative } (\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x) (at x)$ )
using st
by (auto simp: C1_differentiable_on_eq elim!: continuous_on_subset intro: *)
ultimately have  $\exists S. \text{finite } S \wedge ((\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x) \text{ C1\_differentiable\_on } \{a..b\} - S)$ 
  apply (rule_tac x= $\{a,b,c\} \cup S \cup T$  in exI)
  using st by (auto simp: C1_differentiable_on_eq elim!: continuous_on_subset)
with cab show ?thesis
by (simp add: piecewise_C1_differentiable_on_def)
qed

lemma piecewise_C1_differentiable_const [derivative_intros]:
  ( $\lambda x. c$ ) piecewise_C1_differentiable_on S
  by (simp add: C1_differentiable_imp_piecewise)

lemma piecewise_C1_differentiable_scaleR [derivative_intros]:
   $\llbracket f \text{ piecewise\_C1\_differentiable\_on } S \rrbracket$ 
 $\implies (\lambda x. c *_{\mathbb{R}} f x) \text{ piecewise\_C1\_differentiable\_on } S$ 
  by (force simp add: piecewise_C1_differentiable_on_def continuous_on_scaleR)

lemma piecewise_C1_differentiable_neg [derivative_intros]:
   $f \text{ piecewise\_C1\_differentiable\_on } S \implies (\lambda x. -(f x)) \text{ piecewise\_C1\_differentiable\_on } S$ 
  unfolding piecewise_C1_differentiable_on_def
  by (auto intro!: continuous_on_minus C1_differentiable_on_minus)

lemma piecewise_C1_differentiable_add [derivative_intros]:
  assumes  $f \text{ piecewise\_C1\_differentiable\_on } i$ 
     $g \text{ piecewise\_C1\_differentiable\_on } i$ 
  shows  $(\lambda x. f x + g x) \text{ piecewise\_C1\_differentiable\_on } i$ 
proof -
  obtain S t where st:  $\text{finite } S \text{ finite } t$ 

```

```

      f C1_differentiable_on (i-S)
      g C1_differentiable_on (i-t)
    using assms by (auto simp: piecewise_C1_differentiable_on_def)
  then have finite (S ∪ t) ∧ (λx. f x + g x) C1_differentiable_on i - (S ∪ t)
    by (auto intro: C1_differentiable_on_add elim!: C1_differentiable_on_subset)
  moreover have continuous_on i f continuous_on i g
    using assms piecewise_C1_differentiable_on_def by auto
  ultimately show ?thesis
    by (auto simp: piecewise_C1_differentiable_on_def continuous_on_add)
qed

```

```

lemma piecewise_C1_differentiable_diff [derivative_intros]:
  ‖f piecewise_C1_differentiable_on S; g piecewise_C1_differentiable_on S‖
  ⇒ (λx. f x - g x) piecewise_C1_differentiable_on S
  unfolding diff_conv_add_uminus
  by (metis piecewise_C1_differentiable_add piecewise_C1_differentiable_neg)

```

```

lemma piecewise_C1_differentiable_cmult_right [derivative_intros]:
  fixes c::complex
  shows f piecewise_C1_differentiable_on S
    ⇒ (λx. f x * c) piecewise_C1_differentiable_on S
  by (force simp: piecewise_C1_differentiable_on_def continuous_on_mult_right)

```

```

lemma piecewise_C1_differentiable_cmult_left [derivative_intros]:
  fixes c::complex
  shows f piecewise_C1_differentiable_on S
    ⇒ (λx. c * f x) piecewise_C1_differentiable_on S
  using piecewise_C1_differentiable_cmult_right [of f S c] by (simp add: mult.commute)

```

```

lemma piecewise_C1_differentiable_on_of_real [derivative_intros]:
  of_real piecewise_C1_differentiable_on S
  by (simp add: C1_differentiable_imp_piecewise C1_differentiable_on_of_real)

```

end

4.11 Finite Cartesian Products of Euclidean Spaces

```

theory Cartesian_Euclidean_Space
imports Derivative
begin

```

```

lemma subspace_special_hyperplane: subspace {x. x $ k = 0}
  by (simp add: subspace_def)

```

```

lemma sum_mult_product:
  sum h {.. $A * B :: nat$ } = (∑ i ∈ {.. $A$ }. ∑ j ∈ {.. $B$ }. h (j + i * B))
  unfolding sum_nat_group [of h B A, unfolded atLeast0LessThan, symmetric]
proof (rule sum.cong, simp, rule sum.reindex_cong)
  fix i

```

```

show inj_on ( $\lambda j. j + i * B$ )  $\{..<B\}$  by (auto intro!: inj_onI)
show  $\{i * B..<i * B + B\} = (\lambda j. j + i * B) \text{ ' } \{..<B\}$ 
proof safe
  fix j assume  $j \in \{i * B..<i * B + B\}$ 
  then show  $j \in (\lambda j. j + i * B) \text{ ' } \{..<B\}$ 
    by (auto intro!: image_eqI[of _ _  $j - i * B$ ])
qed simp
qed simp

lemma interval_cbox_cart:  $\{a::\text{real}^n..b\} = \text{cbox } a \ b$ 
  by (auto simp add: less_eq_vec_def mem_box Basis_vec_def inner_axis)

lemma differentiable_vec:
  fixes  $S :: \text{'a::euclidean\_space set}$ 
  shows vec differentiable_on S
  by (simp add: linear_linear bounded_linear_imp_differentiable_on)

lemma continuous_vec [continuous_intros]:
  fixes  $x :: \text{'a::euclidean\_space}$ 
  shows isCont vec x
  apply (clarsimp simp add: continuous_def LIM_def dist_vec_def L2_set_def)
  apply (rule_tac  $x=r / \text{sqrt } (\text{real CARD('b)})$  in exI)
  by (simp add: mult.commute pos_less_divide_eq real_sqrt_mult)

lemma box_vec_eq_empty [simp]:
  shows  $\text{cbox } (\text{vec } a) (\text{vec } b) = \{\} \longleftrightarrow \text{cbox } a \ b = \{\}$ 
     $\text{box } (\text{vec } a) (\text{vec } b) = \{\} \longleftrightarrow \text{box } a \ b = \{\}$ 
  by (auto simp: Basis_vec_def mem_box box_eq_empty inner_axis)



### 4.11.1 Closures and interiors of halfspaces



lemma interior_halfspace_component_le [simp]:
   $\text{interior } \{x. x\$k \leq a\} = \{x :: (\text{real}^n). x\$k < a\}$  (is ?LE)
  and interior_halfspace_component_ge [simp]:
   $\text{interior } \{x. x\$k \geq a\} = \{x :: (\text{real}^n). x\$k > a\}$  (is ?GE)
proof -
  have axis k  $(1::\text{real}) \neq 0$ 
  by (simp add: axis_def vec_eq_iff)
  moreover have axis k  $(1::\text{real}) \cdot x = x\$k$  for x
  by (simp add: cart_eq_inner_axis inner_commute)
  ultimately show ?LE ?GE
  using interior_halfspace_le [of axis k  $(1::\text{real})$  a]
    interior_halfspace_ge [of axis k  $(1::\text{real})$  a] by auto
qed

lemma closure_halfspace_component_lt [simp]:
   $\text{closure } \{x. x\$k < a\} = \{x :: (\text{real}^n). x\$k \leq a\}$  (is ?LE)
  and closure_halfspace_component_gt [simp]:
   $\text{closure } \{x. x\$k > a\} = \{x :: (\text{real}^n). x\$k \geq a\}$  (is ?GE)

```

```

proof –
  have axis k (1::real)  $\neq 0$ 
    by (simp add: axis_def vec_eq_iff)
  moreover have axis k (1::real)  $\cdot x = x\$k$  for x
    by (simp add: cart_eq_inner_axis inner_commute)
  ultimately show ?LE ?GE
    using closure_halfspace_lt [of axis k (1::real) a]
      closure_halfspace_gt [of axis k (1::real) a] by auto
qed

```

lemma *interior_standard_hyperplane*:

$$\text{interior } \{x :: (\text{real}^n). x\$k = a\} = \{\}$$

```

proof –
  have axis k (1::real)  $\neq 0$ 
    by (simp add: axis_def vec_eq_iff)
  moreover have axis k (1::real)  $\cdot x = x\$k$  for x
    by (simp add: cart_eq_inner_axis inner_commute)
  ultimately show ?thesis
    using interior_hyperplane [of axis k (1::real) a]
    by force
qed

```

lemma *matrix_vector_mul_bounded_linear*[*intro*, *simp*]: *bounded_linear* (($\ast v$) *A*)

for *A* :: 'a::{euclidean_space, real_algebra_1}ⁿ^m

using *matrix_vector_mul_linear*[of *A*]

by (simp add: *linear_conv_bounded_linear* *linear_matrix_vector_mul_eq*)

lemma

fixes *A* :: 'a::{euclidean_space, real_algebra_1}ⁿ^m

shows *matrix_vector_mult_linear_continuous_at* [*continuous_intros*]: *isCont* (($\ast v$) *A*) *z*

and *matrix_vector_mult_linear_continuous_on* [*continuous_intros*]: *continuous_on* *S* (($\ast v$) *A*)

by (simp_all add: *linear_continuous_at* *linear_continuous_on*)

4.11.2 Bounds on components etc. relative to operator norm

lemma *norm_column_le_onorm*:

fixes *A* :: $\text{real}^n{}^m$

shows *norm*(*column* *i* *A*) $\leq$ *onorm*(($\ast v$) *A*)

proof –

have *norm* (χ *j*. *A* \$ *j* \$ *i*) $\leq$ *norm* (*A* $\ast v$ *axis* *i* 1)

by (simp add: *matrix_mult_dot* *cart_eq_inner_axis*)

also have ... $\leq$ *onorm* (($\ast v$) *A*)

using *onorm* [*OF* *matrix_vector_mul_bounded_linear*, of *A* *axis* *i* 1] **by** *auto*

finally have *norm* (χ *j*. *A* \$ *j* \$ *i*) $\leq$ *onorm* (($\ast v$) *A*) .

then show ?thesis

unfolding *column_def* .

qed

lemma *matrix_component_le_onorm*:

fixes $A :: \text{real}^n{}^m$

shows $|A \$ i \$ j| \leq \text{onorm}((\ast v) A)$

proof –

have $|A \$ i \$ j| \leq \text{norm } (\chi \ n. (A \$ n \$ j))$

by (*metis (full_types, lifting) component_le_norm_cart vec_lambda_beta*)

also have $\dots \leq \text{onorm } ((\ast v) A)$

by (*metis (no_types) column_def norm_column_le_onorm*)

finally show *?thesis* .

qed

lemma *component_le_onorm*:

fixes $f :: \text{real}^m \Rightarrow \text{real}^n$

shows $\text{linear } f \implies |\text{matrix } f \$ i \$ j| \leq \text{onorm } f$

by (*metis matrix_component_le_onorm matrix_vector_mul(2)*)

lemma *onorm_le_matrix_component_sum*:

fixes $A :: \text{real}^n{}^m$

shows $\text{onorm}((\ast v) A) \leq (\sum i \in \text{UNIV}. \sum j \in \text{UNIV}. |A \$ i \$ j|)$

proof (*rule onorm_le*)

fix x

have $\text{norm } (A \ast v x) \leq (\sum i \in \text{UNIV}. |(A \ast v x) \$ i|)$

by (*rule norm_le_l1_cart*)

also have $\dots \leq (\sum i \in \text{UNIV}. \sum j \in \text{UNIV}. |A \$ i \$ j| \ast \text{norm } x)$

proof (*rule sum_mono*)

fix i

have $|(A \ast v x) \$ i| \leq |\sum j \in \text{UNIV}. A \$ i \$ j \ast x \$ j|$

by (*simp add: matrix_vector_mult_def*)

also have $\dots \leq (\sum j \in \text{UNIV}. |A \$ i \$ j \ast x \$ j|)$

by (*rule sum_abs*)

also have $\dots \leq (\sum j \in \text{UNIV}. |A \$ i \$ j| \ast \text{norm } x)$

by (*rule sum_mono*) (*simp add: abs_mult component_le_norm_cart mult_left_mono*)

finally show $|(A \ast v x) \$ i| \leq (\sum j \in \text{UNIV}. |A \$ i \$ j| \ast \text{norm } x)$.

qed

finally show $\text{norm } (A \ast v x) \leq (\sum i \in \text{UNIV}. \sum j \in \text{UNIV}. |A \$ i \$ j|) \ast \text{norm } x$

by (*simp add: sum_distrib_right*)

qed

lemma *onorm_le_matrix_component*:

fixes $A :: \text{real}^n{}^m$

assumes $\bigwedge i j. \text{abs}(A \$ i \$ j) \leq B$

shows $\text{onorm}((\ast v) A) \leq \text{real } (\text{CARD}('m)) \ast \text{real } (\text{CARD}('n)) \ast B$

proof (*rule onorm_le*)

fix $x :: \text{real}^n$:

have $\text{norm } (A \ast v x) \leq (\sum i \in \text{UNIV}. |(A \ast v x) \$ i|)$

by (*rule norm_le_l1_cart*)

also have $\dots \leq (\sum i :: 'm \in \text{UNIV}. \text{real } (\text{CARD}('n)) \ast B \ast \text{norm } x)$

proof (*rule sum_mono*)

```

fix i
have |(A *v x) $ i| ≤ norm(A $ i) * norm x
  by (simp add: matrix_mult_dot Cauchy_Schwarz_ineq2)
also have ... ≤ (∑ j∈UNIV. |A $ i $ j|) * norm x
  by (simp add: mult_right_mono norm_le_l1_cart)
also have ... ≤ real (CARD('n)) * B * norm x
  by (simp add: assms sum_bounded_above mult_right_mono)
finally show |(A *v x) $ i| ≤ real (CARD('n)) * B * norm x .
qed
also have ... ≤ CARD('m) * real (CARD('n)) * B * norm x
  by simp
finally show norm (A *v x) ≤ CARD('m) * real (CARD('n)) * B * norm x .
qed

```

```

lemma rational_approximation:
  assumes e > 0
  obtains r::real where r ∈ ℚ |r - x| < e
  using Rats_dense_in_real [of x - e/2 x + e/2] assms by auto

```

```

proposition matrix_rational_approximation:
  fixes A :: real^'n^'m
  assumes e > 0
  obtains B where ∧ i j. B $ i $ j ∈ ℚ onorm(λx. (A - B) *v x) < e
proof -
  have ∀ i j. ∃ q ∈ ℚ. |q - A $ i $ j| < e / (2 * CARD('m) * CARD('n))
    using assms by (force intro: rational_approximation [of e / (2 * CARD('m)
* CARD('n))])
  then obtain B where B: ∧ i j. B $ i $ j ∈ ℚ and Bclo: ∧ i j. |B $ i $ j - A $ i $ j|
< e / (2 * CARD('m) * CARD('n))
  by (auto simp: lambda_skolem Bex_def)
  show ?thesis
proof
  have onorm ((*v) (A - B)) ≤ real CARD('m) * real CARD('n) *
(e / (2 * real CARD('m) * real CARD('n)))
  apply (rule onorm_le_matrix_component)
  using Bclo by (simp add: abs_minus_commute less_imp_le)
  also have ... < e
  using ‹0 < e› by (simp add: field_split_simps)
  finally show onorm ((*v) (A - B)) < e .
qed (use B in auto)
qed

```

```

lemma vector_sub_project_orthogonal_cart: (b::real^'n) • (x - ((b • x) / (b • b))
*s b) = 0
  unfolding inner_simps scalar_mult_eq_scaleR by auto

```

```

lemma infnorm_cart: infnorm (x::real^'n) = Sup {|x $ i| | i. i ∈ UNIV}
  by (simp add: infnorm_def inner_axis Basis_vec_def) (metis (lifting) inner_axis

```

real_inner_1_right)

lemma *component_le_infnorm_cart*: $|x\$i| \leq \text{infnorm } (x::\text{real}^n)$
using *Basis_le_infnorm*[of axis *i* 1 *x*]
by (*simp add: Basis_vec_def axis_eq_axis inner_axis*)

lemma *continuous_component*[*continuous_intros*]: $\text{continuous } F f \implies \text{continuous } F (\lambda x. f x \$ i)$
unfolding *continuous_def* **by** (*rule tendsto_vec_nth*)

lemma *continuous_on_component*[*continuous_intros*]: $\text{continuous\_on } s f \implies \text{continuous\_on } s (\lambda x. f x \$ i)$
unfolding *continuous_on_def* **by** (*fast intro: tendsto_vec_nth*)

lemma *continuous_on_vec_lambda*[*continuous_intros*]:
 $(\bigwedge i. \text{continuous\_on } S (f i)) \implies \text{continuous\_on } S (\lambda x. \chi i. f i x)$
unfolding *continuous_on_def* **by** (*auto intro: tendsto_vec_lambda*)

lemma *closed_positive_orthant*: $\text{closed } \{x::\text{real}^n. \forall i. 0 \leq x\$i\}$
by (*simp add: Collect_all_eq closed_INT closed_Collect_le continuous_on_component*)

lemma *bounded_component_cart*: $\text{bounded } s \implies \text{bounded } ((\lambda x. x \$ i) 's)$
unfolding *bounded_def*
apply *clarify*
apply (*rule_tac x=x \$ i in exI*)
apply (*rule_tac x=e in exI*)
apply *clarify*
apply (*rule order_trans [OF dist_vec_nth_le], simp*)
done

lemma *compact_lemma_cart*:
fixes *f* :: $\text{nat} \Rightarrow 'a::\text{heine\_borel}^n$
assumes *f*: *bounded* (*range f*)
shows $\exists l r. \text{strict\_mono } r \wedge$
 $(\forall e>0. \text{eventually } (\lambda n. \forall i \in d. \text{dist } (f (r n) \$ i) (l \$ i) < e) \text{ sequentially})$
 $(\text{is } ?th \ d)$

proof –
have $\forall d' \subseteq d. ?th \ d'$
by (*rule compact_lemma_general*[**where** *unproj=vec_lambda*])
 $(\text{auto intro! : } f \text{ bounded\_component\_cart})$
then show *?th d* **by** *simp*
qed

instance *vec* :: $(\text{heine\_borel}, \text{finite}) \text{ heine\_borel}$

proof
fix *f* :: $\text{nat} \Rightarrow 'a^b$
assume *f*: *bounded* (*range f*)
then obtain *l r* **where** *r*: *strict_mono r*
and *l*: $\forall e>0. \text{eventually } (\lambda n. \forall i \in \text{UNIV}. \text{dist } (f (r n) \$ i) (l \$ i) < e)$


```

sequentially
  using compact_lemma_cart [OF f] by blast
  let ?d = UNIV::'b set
  { fix e::real assume e>0
    hence 0 < e / (real_of_nat (card ?d))
      using zero_less_card_finite divide_pos_pos[of e, of real_of_nat (card ?d)]
  by auto
  with l have eventually ( $\lambda n. \forall i. \text{dist } (f \text{ (r n) } \$ i) (l \$ i) < e / (\text{real\_of\_nat } (\text{card } ?d))$ ) sequentially
    by simp
  moreover
  { fix n
    assume n:  $\forall i. \text{dist } (f \text{ (r n) } \$ i) (l \$ i) < e / (\text{real\_of\_nat } (\text{card } ?d))$ 
    have  $\text{dist } (f \text{ (r n)}) l \leq (\sum i \in ?d. \text{dist } (f \text{ (r n) } \$ i) (l \$ i))$ 
      unfolding dist_vec_def using zero_le_dist by (rule L2_set_le_sum)
    also have  $\dots < (\sum i \in ?d. e / (\text{real\_of\_nat } (\text{card } ?d)))$ 
      by (rule sum_strict_mono) (simp_all add: n)
    finally have  $\text{dist } (f \text{ (r n)}) l < e$  by simp
  }
  ultimately have eventually ( $\lambda n. \text{dist } (f \text{ (r n)}) l < e$ ) sequentially
    by (rule eventually_mono)
  }
  hence  $((f \circ r) \longrightarrow l)$  sequentially unfolding o_def tendsto_iff by simp
  with r show  $\exists l r. \text{strict\_mono } r \wedge ((f \circ r) \longrightarrow l)$  sequentially by auto
qed

```

lemma interval_cart:

```

fixes a :: real^'n
shows box a b = {x::real^'n.  $\forall i. a\$i < x\$i \wedge x\$i < b\$i$ }
  and cbox a b = {x::real^'n.  $\forall i. a\$i \leq x\$i \wedge x\$i \leq b\$i$ }
  by (auto simp add: set_eq_iff less_vec_def less_eq_vec_def mem_box Basis_vec_def inner_axis)

```

lemma mem_box_cart:

```

fixes a :: real^'n
shows  $x \in \text{box } a b \longleftrightarrow (\forall i. a\$i < x\$i \wedge x\$i < b\$i)$ 
  and  $x \in \text{cbox } a b \longleftrightarrow (\forall i. a\$i \leq x\$i \wedge x\$i \leq b\$i)$ 
  using interval_cart[of a b] by (auto simp add: set_eq_iff less_vec_def less_eq_vec_def)

```

lemma interval_eq_empty_cart:

```

fixes a :: real^'n
shows  $(\text{box } a b = \{\}) \longleftrightarrow (\exists i. b\$i \leq a\$i)$  (is ?th1)
  and  $(\text{cbox } a b = \{\}) \longleftrightarrow (\exists i. b\$i < a\$i)$  (is ?th2)

```

proof -

```

{ fix i x assume as:b\$i ≤ a\$i and x:x∈box a b
  hence a $ i < x $ i ∧ x $ i < b $ i unfolding mem_box_cart by auto
  hence a\$i < b\$i by auto
  hence False using as by auto }

```

moreover

```

{ assume as:  $\forall i. \neg (b\$i \leq a\$i)$ 
  let ?x =  $(1/2) *_R (a + b)$ 
  { fix i
    have  $a\$i < b\$i$  using as[THEN spec[where  $x=i$ ]] by auto
    hence  $a\$i < ((1/2) *_R (a+b)) \$ i ((1/2) *_R (a+b)) \$ i < b\$i$ 
      unfolding vector_smult_component and vector_add_component
      by auto }
    hence  $\text{box } a \ b \neq \{\}$  using mem_box_cart(1)[of ?x a b] by auto }
ultimately show ?th1 by blast

```

```

{ fix i x assume as:  $b\$i < a\$i$  and  $x: x \in \text{cbox } a \ b$ 
  hence  $a \$ i \leq x \$ i \wedge x \$ i \leq b \$ i$  unfolding mem_box_cart by auto
  hence  $a\$i \leq b\$i$  by auto
  hence False using as by auto }

```

moreover

```

{ assume as:  $\forall i. \neg (b\$i < a\$i)$ 
  let ?x =  $(1/2) *_R (a + b)$ 
  { fix i
    have  $a\$i \leq b\$i$  using as[THEN spec[where  $x=i$ ]] by auto
    hence  $a\$i \leq ((1/2) *_R (a+b)) \$ i ((1/2) *_R (a+b)) \$ i \leq b\$i$ 
      unfolding vector_smult_component and vector_add_component
      by auto }
    hence  $\text{cbox } a \ b \neq \{\}$  using mem_box_cart(2)[of ?x a b] by auto }
ultimately show ?th2 by blast

```

qed

lemma interval_ne_empty_cart:

```

fixes a :: realn
shows  $\text{cbox } a \ b \neq \{\} \longleftrightarrow (\forall i. a\$i \leq b\$i)$ 
  and  $\text{box } a \ b \neq \{\} \longleftrightarrow (\forall i. a\$i < b\$i)$ 
unfolding interval_eq_empty_cart[of a b] by (auto simp add: not_less not_le)

```

lemma subset_interval_imp_cart:

```

fixes a :: realn
shows  $(\forall i. a\$i \leq c\$i \wedge d\$i \leq b\$i) \implies \text{cbox } c \ d \subseteq \text{cbox } a \ b$ 
  and  $(\forall i. a\$i < c\$i \wedge d\$i < b\$i) \implies \text{cbox } c \ d \subseteq \text{box } a \ b$ 
  and  $(\forall i. a\$i \leq c\$i \wedge d\$i \leq b\$i) \implies \text{box } c \ d \subseteq \text{cbox } a \ b$ 
  and  $(\forall i. a\$i \leq c\$i \wedge d\$i \leq b\$i) \implies \text{box } c \ d \subseteq \text{box } a \ b$ 
unfolding subset_eq[unfolded Ball_def] unfolding mem_box_cart
by (auto intro: order_trans less_le_trans le_less_trans less_imp_le)

```

lemma interval_sing:

```

fixes a :: 'a::linordern
shows  $\{a .. a\} = \{a\} \wedge \{a < .. < a\} = \{\}$ 
apply (auto simp add: set_eq_iff less_vec_def less_eq_vec_def vec_eq_iff)
done

```

lemma subset_interval_cart:

```

fixes  $a :: \text{real}^n$ 
shows  $\text{cbox } c \ d \subseteq \text{cbox } a \ b \longleftrightarrow (\forall i. c\$i \leq d\$i) \longrightarrow (\forall i. a\$i \leq c\$i \wedge d\$i \leq b\$i)$  (is ?th1)
and  $\text{cbox } c \ d \subseteq \text{box } a \ b \longleftrightarrow (\forall i. c\$i \leq d\$i) \longrightarrow (\forall i. a\$i < c\$i \wedge d\$i < b\$i)$  (is ?th2)
and  $\text{box } c \ d \subseteq \text{cbox } a \ b \longleftrightarrow (\forall i. c\$i < d\$i) \longrightarrow (\forall i. a\$i \leq c\$i \wedge d\$i \leq b\$i)$  (is ?th3)
and  $\text{box } c \ d \subseteq \text{box } a \ b \longleftrightarrow (\forall i. c\$i < d\$i) \longrightarrow (\forall i. a\$i \leq c\$i \wedge d\$i < b\$i)$  (is ?th4)
using  $\text{subset\_box}[of \ c \ d \ a \ b]$  by ( $\text{simp\_all add: Basis\_vec\_def inner\_axis}$ )

```

lemma *disjoint_interval_cart*:

```

fixes  $a :: \text{real}^n$ 
shows  $\text{cbox } a \ b \cap \text{cbox } c \ d = \{\} \longleftrightarrow (\exists i. (b\$i < a\$i \vee d\$i < c\$i \vee b\$i < c\$i \vee d\$i < a\$i))$  (is ?th1)
and  $\text{cbox } a \ b \cap \text{box } c \ d = \{\} \longleftrightarrow (\exists i. (b\$i < a\$i \vee d\$i \leq c\$i \vee b\$i \leq c\$i \vee d\$i \leq a\$i))$  (is ?th2)
and  $\text{box } a \ b \cap \text{cbox } c \ d = \{\} \longleftrightarrow (\exists i. (b\$i \leq a\$i \vee d\$i < c\$i \vee b\$i \leq c\$i \vee d\$i \leq a\$i))$  (is ?th3)
and  $\text{box } a \ b \cap \text{box } c \ d = \{\} \longleftrightarrow (\exists i. (b\$i \leq a\$i \vee d\$i \leq c\$i \vee b\$i \leq c\$i \vee d\$i \leq a\$i))$  (is ?th4)
using  $\text{disjoint\_interval}[of \ a \ b \ c \ d]$  by ( $\text{simp\_all add: Basis\_vec\_def inner\_axis}$ )

```

lemma *Int_interval_cart*:

```

fixes  $a :: \text{real}^n$ 
shows  $\text{cbox } a \ b \cap \text{cbox } c \ d = \{(\chi \ i. \max(a\$i) (c\$i)) .. (\chi \ i. \min(b\$i) (d\$i))\}$ 
unfolding Int_interval
by ( $\text{auto simp: mem\_box less\_eq\_vec\_def}$ )
    ( $\text{auto simp: Basis\_vec\_def inner\_axis}$ )

```

lemma *closed_interval_left_cart*:

```

fixes  $b :: \text{real}^n$ 
shows  $\text{closed } \{x :: \text{real}^n. \forall i. x\$i \leq b\$i\}$ 
by ( $\text{simp add: Collect\_all\_eq closed\_INT closed\_Collect\_le continuous\_on\_component}$ )

```

lemma *closed_interval_right_cart*:

```

fixes  $a :: \text{real}^n$ 
shows  $\text{closed } \{x :: \text{real}^n. \forall i. a\$i \leq x\$i\}$ 
by ( $\text{simp add: Collect\_all\_eq closed\_INT closed\_Collect\_le continuous\_on\_component}$ )

```

lemma *is_interval_cart*:

```

 $\text{is\_interval } (s :: (\text{real}^n) \text{ set}) \longleftrightarrow$ 
   $(\forall a \in s. \forall b \in s. \forall x. (\forall i. ((a\$i \leq x\$i \wedge x\$i \leq b\$i) \vee (b\$i \leq x\$i \wedge x\$i \leq a\$i))))$ 
 $\longrightarrow x \in s$ 
by ( $\text{simp add: is\_interval\_def Ball\_def Basis\_vec\_def inner\_axis imp\_ex}$ )

```

lemma *closed_halfspace_component_le_cart*: $\text{closed } \{x :: \text{real}^n. x\$i \leq a\}$

by ($\text{simp add: closed\_Collect\_le continuous\_on\_component}$)

lemma *closed_halfspace_component_ge_cart*: *closed* $\{x::\text{real}^n. x\$i \geq a\}$
by (*simp add: closed_Collect_le continuous_on_component*)

lemma *open_halfspace_component_lt_cart*: *open* $\{x::\text{real}^n. x\$i < a\}$
by (*simp add: open_Collect_less continuous_on_component*)

lemma *open_halfspace_component_gt_cart*: *open* $\{x::\text{real}^n. x\$i > a\}$
by (*simp add: open_Collect_less continuous_on_component*)

lemma *Lim_component_le_cart*:
fixes $f :: 'a \Rightarrow \text{real}^n$
assumes $(f \longrightarrow l) \text{ net} \neg (\text{trivial\_limit } \text{net}) \text{ eventually } (\lambda x. f\ x\ \$i \leq b) \text{ net}$
shows $l\$i \leq b$
by (*rule tendsto_le[OF assms(2) tendsto_const tendsto_vec_nth, OF assms(1, 3)]*)

lemma *Lim_component_ge_cart*:
fixes $f :: 'a \Rightarrow \text{real}^n$
assumes $(f \longrightarrow l) \text{ net} \neg (\text{trivial\_limit } \text{net}) \text{ eventually } (\lambda x. b \leq (f\ x)\ \$i) \text{ net}$
shows $b \leq l\$i$
by (*rule tendsto_le[OF assms(2) tendsto_vec_nth tendsto_const, OF assms(1, 3)]*)

lemma *Lim_component_eq_cart*:
fixes $f :: 'a \Rightarrow \text{real}^n$
assumes $\text{net}: (f \longrightarrow l) \text{ net} \neg \text{trivial\_limit } \text{net}$ **and** $\text{ev}: \text{eventually } (\lambda x. f(x)\ \$i = b) \text{ net}$
shows $l\$i = b$
using *ev[unfolded order_eq_iff eventually_conj_iff]* **and**
Lim_component_ge_cart[*OF net, of b i*] **and**
Lim_component_le_cart[*OF net, of i b*] **by auto**

lemma *connected_ivt_component_cart*:
fixes $x :: \text{real}^n$
shows $\text{connected } s \implies x \in s \implies y \in s \implies x\$k \leq a \implies a \leq y\$k \implies (\exists z \in s. z\$k = a)$
using *connected_ivt_hyperplane*[*of s x y axis k 1 a*]
by (*auto simp add: inner_axis inner_commute*)

lemma *subspace_substandard_cart*: *vec.subspace* $\{x. (\forall i. P\ i \longrightarrow x\$i = 0)\}$
unfolding *vec.subspace_def* **by auto**

lemma *closed_substandard_cart*:
closed $\{x::'a::\text{real\_normed\_vector}^n. \forall i. P\ i \longrightarrow x\$i = 0\}$
proof –
{ **fix** $i::'n$
have *closed* $\{x::'a^{\wedge} n. P\ i \longrightarrow x\$i = 0\}$
by (*cases P i*) (*simp_all add: closed_Collect_eq continuous_on_component*)
}

```

thus ?thesis
  unfolding Collect_all_eq by (simp add: closed_INT)
qed

```

4.11.3 Convex Euclidean Space

```

lemma Cart_1:(1::realn) =  $\sum$  Basis
  using const_vector_cart[of 1] by (simp add: one_vec_def)

```

```

declare vector_add_ldistrib[simp] vector_ssub_ldistrib[simp] vector_smult_assoc[simp]
vector_smult_rneg[simp]
declare vector_sadd_rdistib[simp] vector_sub_rdistib[simp]

```

```

lemmas vector_component_simps = vector_minus_component vector_smult_component
vector_add_component less_eq_vec_def vec_lambda_beta vector_uminus_component

```

```

lemma convex_box_cart:
  assumes  $\bigwedge i. \text{convex } \{x. P\ i\ x\}$ 
  shows  $\text{convex } \{x. \forall i. P\ i\ (x\$i)\}$ 
  using assms unfolding convex_def by auto

```

4.11.4 Derivative

```

definition jacobian f net = matrix(frechet_derivative f net)

```

```

proposition jacobian_works:
  (f::(reala)  $\Rightarrow$  (realb)) differentiable net  $\longleftrightarrow$ 
  (f has_derivative ( $\lambda h. (jacobian\ f\ net) * v\ h$ )) net (is ?lhs = ?rhs)

```

```

proof
  assume ?lhs then show ?rhs
    by (simp add: frechet_derivative_works has_derivative_linear jacobian_def)
  next
    assume ?rhs then show ?lhs
      by (rule differentiableI)
qed

```

Component of the differential must be zero if it exists at a local maximum or minimum for that corresponding component

```

proposition differential_zero_maxmin_cart:
  fixes f::reala  $\Rightarrow$  realb
  assumes  $0 < e \wedge (\forall y \in \text{ball } x\ e. (f\ y)\$k \leq (f\ x)\$k) \vee (\forall y \in \text{ball } x\ e. (f\ x)\$k \leq (f\ y)\$k)$ 
  shows  $\text{jacobian } f\ (\text{at } x)\ \$k = 0$ 
  using differential_zero_maxmin_component[of axis k 1 e x f] assms
  vector_cart[of  $\lambda j. \text{frechet\_derivative } f\ (\text{at } x)\ j\ \$k$ ]
  by (simp add: Basis_vec_def axis_eq_axis_inner_axis jacobian_def matrix_def)

```

4.11.5 Routine results connecting the types $(real, 1)$ *vec* and *real*

```

lemma vec_cbox_1_eq [simp]:
  shows vec ‘ cbox u v = cbox (vec u) (vec v :: real1)
  by (force simp: Basis_vec_def cart_eq_inner_axis [symmetric] mem_box)

lemma vec_nth_cbox_1_eq [simp]:
  fixes u v :: 'a::euclidean_space1'
  shows ( $\lambda x. x \$ 1$ ) ‘ cbox u v = cbox (u$1) (v$1)
  by (auto simp: Basis_vec_def cart_eq_inner_axis [symmetric] mem_box image_iff Bex_def inner_axis) (metis vec_component)

lemma vec_nth_1_iff_cbox [simp]:
  fixes a b :: 'a::euclidean_space'
  shows ( $\lambda x::'^a$ 1.  $x \$ 1$ ) ‘  $S = \text{cbox } a \ b \longleftrightarrow S = \text{cbox } (\text{vec } a) (\text{vec } b)$ 
  (is ?lhs = ?rhs)
proof
  assume L: ?lhs show ?rhs
  proof (intro equalityI subsetI)
    fix x
    assume  $x \in S$ 
    then have  $x \$ 1 \in (\lambda v. v \$ (1::1))$  ‘ cbox (vec a) (vec b)
    using L by auto
    then show  $x \in \text{cbox } (\text{vec } a) (\text{vec } b)$ 
    by (metis (no_types, lifting) imageE vector_one_nth)
  next
    fix x :: 'a1'
    assume  $x \in \text{cbox } (\text{vec } a) (\text{vec } b)$ 
    then show  $x \in S$ 
    by (metis (no_types, lifting) L imageE imageI vec_component vec_nth_cbox_1_eq vector_one_nth)
  qed
qed simp

lemma vec_nth_real_1_iff_cbox [simp]:
  fixes a b :: real
  shows ( $\lambda x::\text{real}$ 1.  $x \$ 1$ ) ‘  $S = \{a..b\} \longleftrightarrow S = \text{cbox } (\text{vec } a) (\text{vec } b)$ 
  using vec_nth_1_iff_cbox[of S a b]
  by simp

lemma interval_split_cart:
   $\{a..b::\text{real}$ n $\} \cap \{x. x \$ k \leq c\} = \{a.. (\chi \ i. \text{if } i = k \text{ then } \min (b \$ k) \ c \text{ else } b \$ i)\}$ 
 $\text{cbox } a \ b \cap \{x. x \$ k \geq c\} = \{(\chi \ i. \text{if } i = k \text{ then } \max (a \$ k) \ c \text{ else } a \$ i) .. b\}$ 
  apply (rule_tac [!] set_eqI)
  unfolding Int_iff mem_box_cart mem_Collect_eq interval_cbox_cart
  unfolding vec_lambda_beta
  by auto

lemmas cartesian_euclidean_space_uniform_limit_intros[uniform_limit_intros]

```

```

=
  bounded_linear.uniform_limit[OF blinfun.bounded_linear_right]
  bounded_linear.uniform_limit[OF bounded_linear_vec_nth]
end

```


Chapter 5

Unsorted

```
theory Starlike
imports
  Convex_Euclidean_Space
  Line_Segment
begin

lemma affine_hull_closed_segment [simp]:
  affine_hull (closed_segment a b) = affine_hull {a,b}
by (simp add: segment_convex_hull)

lemma affine_hull_open_segment [simp]:
  fixes a :: 'a::euclidean_space
  shows affine_hull (open_segment a b) = (if a = b then {} else affine_hull {a,b})
by (metis affine_hull_convex_hull affine_hull_empty closure_open_segment closure_same_affine_hull segment_convex_hull)

lemma rel_interior_closure_convex_segment:
  fixes S :: _::euclidean_space set
  assumes convex S a ∈ rel_interior S b ∈ closure S
  shows open_segment a b ⊆ rel_interior S
proof
  fix x
  have [simp]: (1 - u) *R a + u *R b = b - (1 - u) *R (b - a) for u
  by (simp add: algebra_simps)
  assume x ∈ open_segment a b
  then show x ∈ rel_interior S
  unfolding closed_segment_def open_segment_def using assms
  by (auto intro: rel_interior_closure_convex_shrink)
qed

lemma convex_hull_insert_segments:
  convex_hull (insert a S) =
  (if S = {} then {a} else ⋃ x ∈ convex_hull S. closed_segment a x)
by (force simp add: convex_hull_insert_alt in_segment)
```

```

lemma Int_convex_hull_insert_rel_exterior:
  fixes  $z :: 'a::euclidean\_space$ 
  assumes  $\text{convex } C \ T \subseteq C$  and  $z: z \in \text{rel\_interior } C$  and  $\text{dis: disjoint } S \ (\text{rel\_interior } C)$ 
  shows  $S \cap (\text{convex hull } (\text{insert } z \ T)) = S \cap (\text{convex hull } T)$  (is ?lhs = ?rhs)
proof
  have  $T = \{\}$   $\implies z \notin S$ 
  using  $\text{dis } z$  by (auto simp add: disjoint_def)
  then show ?lhs  $\subseteq$  ?rhs
  proof (clarsimp simp add: convex_hull_insert_segments)
    fix  $x \ y$ 
    assume  $x \in S$  and  $y: y \in \text{convex hull } T$  and  $x \in \text{closed\_segment } z \ y$ 
    have  $y \in \text{closure } C$ 
    by (metis y <convex C> <T ⊆ C> closure_subset contra_subsetD convex_hull_eq hull_mono)
    moreover have  $x \notin \text{rel\_interior } C$ 
    by (meson <x ∈ S> dis disjoint_iff)
    moreover have  $x \in \text{open\_segment } z \ y \cup \{z, y\}$ 
    using  $\langle x \in \text{closed\_segment } z \ y \rangle \text{closed\_segment\_eq\_open}$  by blast
    ultimately show  $x \in \text{convex hull } T$ 
    using  $\text{rel\_interior\_closure\_convex\_segment } [OF \ \langle \text{convex } C \rangle \ z]$ 
    using  $y \ z$  by blast
  qed
  show ?rhs  $\subseteq$  ?lhs
  by (meson hull_mono inf_mono subset_insertI subset_refl)
qed

```

5.0.1 Shrinking towards the interior of a convex set

```

lemma mem_interior_convex_shrink:
  fixes  $S :: 'a::euclidean\_space \text{ set}$ 
  assumes  $\text{convex } S$ 
  and  $c \in \text{interior } S$ 
  and  $x \in S$ 
  and  $0 < e$ 
  and  $e \leq 1$ 
  shows  $x - e *_R (x - c) \in \text{interior } S$ 
proof –
  obtain  $d$  where  $d > 0$  and  $d: \text{ball } c \ d \subseteq S$ 
  using assms(2) unfolding mem_interior by auto
  show ?thesis
  unfolding mem_interior
proof (intro exI subsetI conjI)
  fix  $y$ 
  assume  $y \in \text{ball } (x - e *_R (x - c)) \ (e * d)$ 
  then have as:  $\text{dist } (x - e *_R (x - c)) \ y < e * d$ 
  by simp
  have  $*$ :  $y = (1 - (1 - e)) *_R ((1 / e) *_R y - ((1 - e) / e) *_R x) + (1 - e)$ 

```

```

*_R x
  using ⟨e > 0⟩ by (auto simp add: scaleR_left_diff_distrib scaleR_right_diff_distrib)
  have c - ((1 / e) *_R y - ((1 - e) / e) *_R x) = (1 / e) *_R (e *_R c - y + (1
- e) *_R x)
    using ⟨e > 0⟩
    by (auto simp add: euclidean_eq_iff[where 'a='a] field_simps inner_simps)
  then have dist c ((1 / e) *_R y - ((1 - e) / e) *_R x) = |1/e| * norm (e *_R c
- y + (1 - e) *_R x)
    by (simp add: dist_norm)
  also have ... = |1/e| * norm (x - e *_R (x - c) - y)
    by (auto intro!: arg_cong[where f=norm] simp add: algebra_simps)
  also have ... < d
    using as[unfolded dist_norm] and ⟨e > 0⟩
    by (auto simp add: pos_divide_less_eq[OF ⟨e > 0⟩] mult.commute)
  finally have (1 - (1 - e)) *_R ((1 / e) *_R y - ((1 - e) / e) *_R x) + (1 -
e) *_R x ∈ S
    using assms(3-5) d
    by (intro convexD_alt [OF ⟨convex S⟩]) (auto intro: convexD_alt [OF ⟨convex
S⟩])
  with ⟨e > 0⟩ show y ∈ S
    by (auto simp add: scaleR_left_diff_distrib scaleR_right_diff_distrib)
qed (use ⟨e>0⟩ ⟨d>0⟩ in auto)
qed

```

lemma *mem_interior_closure_convex_shrink*:

fixes $S :: 'a::euclidean\_space\ set$

assumes *convex S*

and $c \in \text{interior } S$

and $x \in \text{closure } S$

and $0 < e$

and $e \leq 1$

shows $x - e *_R (x - c) \in \text{interior } S$

proof -

obtain d **where** $d > 0$ **and** $d: \text{ball } c\ d \subseteq S$

using *assms(2) unfolding mem_interior by auto*

have $\exists y \in S. \text{norm } (y - x) * (1 - e) < e * d$

proof (*cases* $x \in S$)

case *True*

then show *?thesis*

using $\langle e > 0 \rangle \langle d > 0 \rangle$ **by force**

next

case *False*

then have $x: x \text{ islimpt } S$

using *assms(3)[unfolded closure_def] by auto*

show *?thesis*

proof (*cases* $e = 1$)

case *True*

obtain y **where** $y \in S\ y \neq x\ \text{dist } y\ x < 1$

using $x[\text{unfolded islimpt\_approachable, THEN spec[where } x=1]]$ **by auto**

```

    then show ?thesis
      using True ⟨0 < d⟩ by auto
  next
    case False
    then have 0 < e * d / (1 - e) and *: 1 - e > 0
      using ⟨e ≤ 1⟩ ⟨e > 0⟩ ⟨d > 0⟩ by auto
    then obtain y where y ∈ S y ≠ x dist y x < e * d / (1 - e)
      using islimpt_approachable x by blast
    then have norm (y - x) * (1 - e) < e * d
      by (metis * dist_norm mult_imp_div_pos_le not_less)
    then show ?thesis
      using ⟨y ∈ S⟩ by blast
  qed
qed
then obtain y where y ∈ S and y: norm (y - x) * (1 - e) < e * d
  by auto
define z where z = c + ((1 - e) / e) *R (x - y)
have *: x - e *R (x - c) = y - e *R (y - z)
  unfolding z_def using ⟨e > 0⟩
  by (auto simp add: scaleR_right_diff_distrib scaleR_right_distrib scaleR_left_diff_distrib)
have (1 - e) * norm (x - y) / e < d
  using y ⟨0 < e⟩ by (simp add: field_simps norm_minus_commute)
then have z ∈ interior (ball c d)
  using ⟨0 < e⟩ ⟨e ≤ 1⟩ by (simp add: interior_open[OF open_ball] z_def
dist_norm)
then have z ∈ interior S
  using d interiorI interior_ball by blast
then show ?thesis
  unfolding * using mem_interior_convex_shrink ⟨y ∈ S⟩ assms by blast
qed

lemma in_interior_closure_convex_segment:
  fixes S :: 'a::euclidean_space set
  assumes convex S and a: a ∈ interior S and b: b ∈ closure S
  shows open_segment a b ⊆ interior S
proof (clarsimp simp: in_segment)
  fix u::real
  assume u: 0 < u < 1
  have (1 - u) *R a + u *R b = b - (1 - u) *R (b - a)
    by (simp add: algebra_simps)
  also have ... ∈ interior S using mem_interior_closure_convex_shrink [OF
assms] u
    by simp
  finally show (1 - u) *R a + u *R b ∈ interior S .
qed

lemma convex_closure_interior:
  fixes S :: 'a::euclidean_space set
  assumes convex S and int: interior S ≠ {}

```

```

shows closure(interior S) = closure S
proof -
  obtain a where a: a ∈ interior S
  using int by auto
  have closure S ⊆ closure(interior S)
  proof
    fix x
    assume x: x ∈ closure S
    show x ∈ closure (interior S)
    proof (cases x=a)
      case True
      then show ?thesis
        using ⟨a ∈ interior S⟩ closure_subset by blast
    next
      case False
      show ?thesis
      proof (clarsimp simp add: closure_def islimpt_approachable)
        fix e::real
        assume xnotS: x ∉ interior S and 0 < e
        show ∃ x' ∈ interior S. x' ≠ x ∧ dist x' x < e
        proof (intro bexI conjI)
          show x - min (e/2 / norm (x - a)) 1 *R (x - a) ≠ x
            using False ⟨0 < e⟩ by (auto simp: algebra_simps min_def)
          show dist (x - min (e/2 / norm (x - a)) 1 *R (x - a)) x < e
            using ⟨0 < e⟩ by (auto simp: dist_norm min_def)
          show x - min (e/2 / norm (x - a)) 1 *R (x - a) ∈ interior S
            using ⟨0 < e⟩ False
            by (auto simp add: min_def a intro: mem_interior_closure_convex_shrink
[OF ⟨convex S⟩ a x])
          qed
        qed
      qed
    then show ?thesis
      by (simp add: closure_mono interior_subset subset_antisym)
  qed

lemma closure_convex_Int_superset:
  fixes S :: 'a::euclidean_space set
  assumes convex S interior S ≠ {} interior S ⊆ closure T
  shows closure(S ∩ T) = closure S
proof -
  have closure S ⊆ closure(interior S)
    by (simp add: convex_closure_interior assms)
  also have ... ⊆ closure (S ∩ T)
    using interior_subset [of S] assms
    by (metis (no_types, lifting) Int_assoc Int_lower2 closure_mono closure_open_Int_superset
inf.orderE open_interior)
  finally show ?thesis

```

by (simp add: closure_mono dual_order.antisym)
qed

5.0.2 Some obvious but surprisingly hard simplex lemmas

lemma simplex:
assumes *finite S*
and $0 \notin S$
shows $\text{convex hull } (\text{insert } 0 \ S) = \{y. \exists u. (\forall x \in S. 0 \leq u \ x) \wedge \text{sum } u \ S \leq 1 \wedge \text{sum } (\lambda x. u \ x \ *_R \ x) \ S = y\}$
proof (simp add: convex_hull_finite_set_eq_iff assms, safe)
fix *x* **and** *u* :: '*a* $\Rightarrow$ real'
assume $0 \leq u \ 0 \ \forall x \in S. 0 \leq u \ x \ u \ 0 + \text{sum } u \ S = 1$
then show $\exists v. (\forall x \in S. 0 \leq v \ x) \wedge \text{sum } v \ S \leq 1 \wedge (\sum x \in S. v \ x \ *_R \ x) = (\sum x \in S. u \ x \ *_R \ x)$
by force
next
fix *x* **and** *u* :: '*a* $\Rightarrow$ real'
assume $\forall x \in S. 0 \leq u \ x \ \text{sum } u \ S \leq 1$
then show $\exists v. 0 \leq v \ 0 \wedge (\forall x \in S. 0 \leq v \ x) \wedge v \ 0 + \text{sum } v \ S = 1 \wedge (\sum x \in S. v \ x \ *_R \ x) = (\sum x \in S. u \ x \ *_R \ x)$
by (rule_tac *x*= $\lambda x. \text{if } x = 0 \text{ then } 1 - \text{sum } u \ S \text{ else } u \ x$ in exI) (auto simp: sum_delta_notmem assms if_smult)
qed

lemma substd_simplex:
assumes *d*: $d \subseteq \text{Basis}$
shows $\text{convex hull } (\text{insert } 0 \ d) = \{x. (\forall i \in \text{Basis}. 0 \leq x \cdot i) \wedge (\sum i \in d. x \cdot i) \leq 1 \wedge (\forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0)\}$
(is $\text{convex hull } (\text{insert } 0 \ ?p) = ?s$)
proof –
let *?D* = *d*
have $0 \notin ?p$
using *assms* **by** (auto simp: image_def)
from *d* **have** *finite d*
by (blast intro: finite_subset finite_Basis)
show *?thesis*
unfolding simplex[OF '*finite d*' ' $0 \notin ?p$ ']
proof (intro set_eqI; safe)
fix *u* :: '*a* $\Rightarrow$ real'
assume *as*: $\forall x \in ?D. 0 \leq u \ x \ \text{sum } u \ ?D \leq 1$
let *?x* = $(\sum x \in ?D. u \ x \ *_R \ x)$
have *ind*: $\forall i \in \text{Basis}. i \in d \longrightarrow u \ i = ?x \cdot i$
and *notind*: $(\forall i \in \text{Basis}. i \notin d \longrightarrow ?x \cdot i = 0)$
using substdbasis_expansion_unique[OF *assms*] **by** blast+
then have **: $\text{sum } u \ ?D = \text{sum } ((\cdot) \ ?x) \ ?D$
using *assms* **by** (auto intro!: sum.cong)
show $0 \leq ?x \cdot i$ **if** *i* $\in \text{Basis}$ **for** *i*
using *as*(1) *ind* *notind* **that** **by** fastforce

```

  show  $\text{sum } ((\cdot) \text{ ?}x) \text{ ?}D \leq 1$ 
    using ** as(2) by linarith
  show  $\text{?}x \cdot i = 0$  if  $i \in \text{Basis}$   $i \notin d$  for  $i$ 
    using notind that by blast
next
  fix  $x$ 
  assume  $\forall i \in \text{Basis}. 0 \leq x \cdot i \text{ sum } ((\cdot) x) \text{ ?}D \leq 1$  ( $\forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0$ )
  with  $d$  show  $\exists u. (\forall x \in \text{?}D. 0 \leq u x) \wedge \text{sum } u \text{ ?}D \leq 1 \wedge (\sum x \in \text{?}D. u x *_R x) = x$ 
    unfolding substdbasis_expansion_unique[OF assms]
    by (rule_tac  $x = \text{inner } x$  in exI) auto
qed
qed

```

lemma *std_simplex*:

```

  convex hull (insert 0 Basis) =
    { $x :: 'a :: \text{euclidean\_space}. (\forall i \in \text{Basis}. 0 \leq x \cdot i) \wedge \text{sum } (\lambda i. x \cdot i) \text{ Basis} \leq 1$ }
  using substd_simplex[of Basis] by auto

```

lemma *interior_std_simplex*:

```

  interior (convex hull (insert 0 Basis)) =
    { $x :: 'a :: \text{euclidean\_space}. (\forall i \in \text{Basis}. 0 < x \cdot i) \wedge \text{sum } (\lambda i. x \cdot i) \text{ Basis} < 1$ }
  unfolding set_eq_iff mem_interior std_simplex
proof (intro allI iffI CollectI; clarify)
  fix  $x :: 'a$ 
  fix  $e$ 
  assume  $e > 0$  and as:  $\text{ball } x e \subseteq \{x. (\forall i \in \text{Basis}. 0 \leq x \cdot i) \wedge \text{sum } ((\cdot) x) \text{ Basis} \leq 1\}$ 
  show  $(\forall i \in \text{Basis}. 0 < x \cdot i) \wedge \text{sum } ((\cdot) x) \text{ Basis} < 1$ 
  proof safe
    fix  $i :: 'a$ 
    assume  $i: i \in \text{Basis}$ 
    then show  $0 < x \cdot i$ 
      using as[THEN subsetD[where  $c = x - (e/2) *_R i$ ]] and  $\langle e > 0 \rangle$ 
      by (force simp add: inner_simps)
  next
    have **:  $\text{dist } x (x + (e/2) *_R (\text{SOME } i. i \in \text{Basis})) < e$  using  $\langle e > 0 \rangle$ 
      unfolding dist_norm
      by (auto intro!: mult_strict_left_mono simp: SOME_Basis)
    have  $\bigwedge i. i \in \text{Basis} \implies (x + (e/2) *_R (\text{SOME } i. i \in \text{Basis})) \cdot i = x \cdot i + (\text{if } i = (\text{SOME } i. i \in \text{Basis}) \text{ then } e/2 \text{ else } 0)$ 
      by (auto simp: SOME_Basis inner_Basis inner_simps)
    then have *:  $\text{sum } ((\cdot) (x + (e/2) *_R (\text{SOME } i. i \in \text{Basis}))) \text{ Basis} = \text{sum } (\lambda i. x \cdot i + (\text{if } (\text{SOME } i. i \in \text{Basis}) = i \text{ then } e/2 \text{ else } 0)) \text{ Basis}$ 
      by (auto simp: intro!: sum.cong)
    have  $\text{sum } ((\cdot) x) \text{ Basis} < \text{sum } ((\cdot) (x + (e/2) *_R (\text{SOME } i. i \in \text{Basis}))) \text{ Basis}$ 
      using  $\langle e > 0 \rangle$  DIM_positive by (auto simp: SOME_Basis sum.distrib *)
    also have  $\dots \leq 1$ 

```

```

    using ** as by force
    finally show  $\text{sum } ((\cdot) x) \text{ Basis} < 1$  by auto
  qed
next
  fix x :: 'a
  assume as:  $\forall i \in \text{Basis}. 0 < x \cdot i$ 
  sum ((\cdot) x) Basis < 1
  obtain a :: 'b where a  $\in \text{UNIV}$  using UNIV_witness ..
  let ?d =  $(1 - \text{sum } ((\cdot) x) \text{ Basis}) / \text{real } (\text{DIM}('a))$ 
  show  $\exists e > 0. \text{ball } x e \subseteq \{x. (\forall i \in \text{Basis}. 0 \leq x \cdot i) \wedge \text{sum } ((\cdot) x) \text{ Basis} \leq 1\}$ 
  proof (rule_tac x=min (Min ((\cdot) x) 'Basis)) D for D in exI, intro conjI
  subsetI CollectI
    fix y
    assume y:  $y \in \text{ball } x (\min (\text{Min } ((\cdot) x) 'Basis)) ?d$ 
    have  $\text{sum } ((\cdot) y) \text{ Basis} \leq \text{sum } (\lambda i. x \cdot i + ?d) \text{ Basis}$ 
    proof (rule sum_mono)
      fix i :: 'a
      assume i:  $i \in \text{Basis}$ 
      have  $|y \cdot i - x \cdot i| \leq \text{norm } (y - x)$ 
      by (metis Basis_le_norm i inner_commute inner_diff_right)
      also have  $\dots < ?d$ 
      using y by (simp add: dist_norm norm_minus_commute)
      finally have  $|y \cdot i - x \cdot i| < ?d$  .
      then show  $y \cdot i \leq x \cdot i + ?d$  by auto
    qed
    also have  $\dots \leq 1$ 
    unfolding sum.distrib sum_constant
    by (auto simp add: Suc_le_eq)
  finally show  $\text{sum } ((\cdot) y) \text{ Basis} \leq 1$  .
  show  $(\forall i \in \text{Basis}. 0 \leq y \cdot i)$ 
  proof safe
    fix i :: 'a
    assume i:  $i \in \text{Basis}$ 
    have  $\text{norm } (x - y) < \text{Min } ((\cdot) x) 'Basis$ 
    using y by (auto simp: dist_norm less_eq_real_def)
    also have  $\dots \leq x \cdot i$ 
    using i by auto
    finally have  $\text{norm } (x - y) < x \cdot i$  .
    then show  $0 \leq y \cdot i$ 
    using Basis_le_norm[OF i, of x - y] and as(1)[rule_format, OF i]
    by (auto simp: inner_simps)
  qed
next
  have  $\text{Min } ((\cdot) x) 'Basis > 0$ 
  using as by simp
  moreover have  $?d > 0$ 
  using as by (auto simp: Suc_le_eq)
  ultimately show  $0 < \min (\text{Min } ((\cdot) x) 'Basis) ((1 - \text{sum } ((\cdot) x) \text{ Basis}) / \text{real } \text{DIM}('a))$ 
  by linarith

```



```

qed
qed

lemma interior_std_simplex_nonempty:
  obtains a :: 'a::euclidean_space where
    a ∈ interior(convex hull (insert 0 Basis))
proof -
  let ?D = Basis :: 'a set
  let ?a = sum (λb::'a. inverse (2 * real DIM('a)) *R b) Basis
  {
    fix i :: 'a
    assume i: i ∈ Basis
    have ?a • i = inverse (2 * real DIM('a))
      by (rule trans[of _ sum (λj. if i = j then inverse (2 * real DIM('a)) else 0)
        ?D])
    (simp_all add: sum.If_cases i) }
  note ** = this
  show ?thesis
proof
  show ?a ∈ interior(convex hull (insert 0 Basis))
    unfolding interior_std_simplex mem_Collect_eq
  proof safe
    fix i :: 'a
    assume i: i ∈ Basis
    show 0 < ?a • i
      unfolding **[OF i] by (auto simp add: Suc_le_eq)
  next
    have sum ((•) ?a) ?D = sum (λi. inverse (2 * real DIM('a))) ?D
      by (auto intro: sum.cong)
    also have ... < 1
      unfolding sum_constant divide_inverse[symmetric]
      by (auto simp add: field_simps)
    finally show sum ((•) ?a) ?D < 1 by auto
  qed
qed
qed

lemma rel_interior_substd_simplex:
  assumes D: D ⊆ Basis
  shows rel_interior (convex hull (insert 0 D)) =
    {x::'a::euclidean_space. (∀ i∈D. 0 < x•i) ∧ (∑ i∈D. x•i) < 1 ∧ (∀ i∈Basis.
i ∉ D ⟶ x•i = 0)}
  (is _ = ?s)
proof -
  have finite D
    using D finite_Basis finite_subset by blast
  show ?thesis
proof (cases D = {})
  case True

```

```

then show ?thesis
  using rel_interior_sing using euclidean_eq_iff[of _ 0] by auto
next
case False
have h0: affine_hull (convex_hull (insert 0 D)) =
  {x::'a::euclidean_space. (∀ i∈Basis. i ∉ D ⟶ x·i = 0)}
  using affine_hull_convex_hull affine_hull_substd_basis assms by auto
have aux: ⋀ x::'a. ∀ i∈Basis. (∀ i∈D. 0 ≤ x·i) ∧ (∀ i∈Basis. i ∉ D ⟶ x·i =
0) ⟶ 0 ≤ x·i
  by auto
{
  fix x :: 'a::euclidean_space
  assume x: x ∈ rel_interior (convex_hull (insert 0 D))
  then obtain e where e > 0 and
    ball x e ∩ {xa. (∀ i∈Basis. i ∉ D ⟶ xa·i = 0)} ⊆ convex_hull (insert 0 D)
    using mem_rel_interior_ball[of x convex_hull (insert 0 D)] h0 by auto
  then have as: ⋀ y. [dist x y < e ∧ (∀ i∈Basis. i ∉ D ⟶ y·i = 0)] ⟹
    (∀ i∈D. 0 ≤ y·i) ∧ sum ((·) y) D ≤ 1
    using assms by (force simp: substd_simplex)
  have x0: (∀ i∈Basis. i ∉ D ⟶ x·i = 0)
    using x rel_interior_subset substd_simplex[OF assms] by auto
  have (∀ i∈D. 0 < x·i) ∧ sum ((·) x) D < 1 ∧ (∀ i∈Basis. i ∉ D ⟶ x·i =
0)
    proof (intro conjI ballI)
      fix i :: 'a
      assume i ∈ D
      then have ∀ j∈D. 0 ≤ (x - (e/2) *R i) · j
        using D ⟨e > 0⟩ x0
        by (intro as[THEN conjunct1]) (force simp: dist_norm inner_simps
inner_Basis)
      then show 0 < x · i
        using ⟨e > 0⟩ ⟨i ∈ D⟩ D by (force simp: inner_simps inner_Basis)
    next
      obtain a where a: a ∈ D
        using ⟨D ≠ {}⟩ by auto
      then have **: dist x (x + (e/2) *R a) < e
        using ⟨e > 0⟩ norm_Basis[of a] D by (auto simp: dist_norm)
      have ⋀ i. i ∈ Basis ⟹ (x + (e/2) *R a) · i = x·i + (if i = a then e/2
else 0)
        using a D by (auto simp: inner_simps inner_Basis)
      then have *: sum ((·) (x + (e/2) *R a)) D = sum (λi. x·i + (if a = i then
e/2 else 0)) D
        using D by (intro sum.cong) auto
      have a ∈ Basis
        using a ∈ D D by auto
      then have h1: (∀ i∈Basis. i ∉ D ⟶ (x + (e/2) *R a) · i = 0)
        using x0 D ⟨a ∈ D⟩ by (auto simp add: inner_add_left inner_Basis)
      have sum ((·) x) D < sum ((·) (x + (e/2) *R a)) D
        using ⟨e > 0⟩ ⟨a ∈ D⟩ ⟨finite D⟩ by (auto simp add: * sum.distrib)

```

```

    also have ... ≤ 1
      using ** h1 as [rule_format, of x + (e/2) *R a]
      by auto
    finally show sum ((·) x) D < 1 ∧ i. i ∈ Basis ⇒ i ∉ D → x·i = 0
      using x0 by auto
  qed
}
moreover
{
  fix x :: 'a::euclidean_space
  assume as: x ∈ ?s
  have ∀ i. 0 < x·i ∨ 0 = x·i → 0 ≤ x·i
    by auto
  moreover have ∀ i. i ∈ D ∨ i ∉ D by auto
  ultimately
  have ∀ i. (∀ i ∈ D. 0 < x·i) ∧ (∀ i. i ∉ D → x·i = 0) → 0 ≤ x·i
    by metis
  then have h2: x ∈ convex hull (insert 0 D)
    using as assms by (force simp add: substd_simplex)
  obtain a where a: a ∈ D
    using ⟨D ≠ {}⟩ by auto
  define d where d ≡ (1 - sum ((·) x) D) / real (card D)
  have ∃ e > 0. ball x e ∩ {x. ∀ i ∈ Basis. i ∉ D → x · i = 0} ⊆ convex hull
    insert 0 D
    unfolding substd_simplex[OF assms]
  proof (intro exI; safe)
    have 0 < card D using ⟨D ≠ {}⟩ ⟨finite D⟩
      by (simp add: card_gt_0_iff)
    have Min (((·) x) ' D) > 0
      using as ⟨D ≠ {}⟩ ⟨finite D⟩ by (simp)
    moreover have d > 0
      using as ⟨0 < card D⟩ by (auto simp: d_def)
    ultimately show min (Min (((·) x) ' D)) d > 0
      by auto
    fix y :: 'a
    assume y2: ∀ i ∈ Basis. i ∉ D → y·i = 0
    assume y ∈ ball x (min (Min ((·) x ' D)) d)
    then have y: dist x y < min (Min ((·) x ' D)) d
      by auto
    have sum ((·) y) D ≤ sum (λ i. x·i + d) D
    proof (rule sum_mono)
      fix i
      assume i ∈ D
      with D have i: i ∈ Basis
        by auto
      have |y·i - x·i| ≤ norm (y - x)
        by (metis i inner_commute inner_diff_right norm_bound_Basis_le
          order_refl)
      also have ... < d

```

```

    by (metis dist_norm min_less_iff_conj norm_minus_commute y)
  finally have  $|y \cdot i - x \cdot i| < d$  .
  then show  $y \cdot i \leq x \cdot i + d$  by auto
qed
also have  $\dots \leq 1$ 
  unfolding sum.distrib sum_constant d_def using  $\langle 0 < \text{card } D \rangle$ 
  by auto
finally show  $\text{sum } ((\cdot) \ y) \ D \leq 1$  .

fix i :: 'a
assume i:  $i \in \text{Basis}$ 
then show  $0 \leq y \cdot i$ 
proof (cases  $i \in D$ )
  case True
  have  $\text{norm } (x - y) < x \cdot i$ 
    using y Min_gr_iff[of  $(\cdot)$  x  $\langle D \text{ norm } (x - y) \rangle \langle 0 < \text{card } D \rangle \langle i \in D \rangle$ ]
    by (simp add: dist_norm card_gt_0_iff)
  then show  $0 \leq y \cdot i$ 
    using Basis_le_norm[OF i, of  $x - y$ ] and as(1)[rule_format]
    by (auto simp: inner_simps)
  qed (use y2 in auto)
qed
then have  $x \in \text{rel\_interior } (\text{convex hull } (\text{insert } 0 \ D))$ 
  using h0 h2 rel_interior_ball by force
}
ultimately have
 $\bigwedge x. x \in \text{rel\_interior } (\text{convex hull } \text{insert } 0 \ D) \longleftrightarrow$ 
 $x \in \{x. (\forall i \in D. 0 < x \cdot i) \wedge \text{sum } ((\cdot) \ x) \ D < 1 \wedge (\forall i \in \text{Basis}. i \notin D \longrightarrow x$ 
 $\cdot i = 0)\}$ 
  by blast
then show ?thesis by (rule set_eqI)
qed
qed

lemma rel_interior_substd_simplex_nonempty:
  assumes  $D \neq \{\}$ 
  and  $D \subseteq \text{Basis}$ 
  obtains  $a :: 'a::\text{euclidean\_space}$ 
  where  $a \in \text{rel\_interior } (\text{convex hull } (\text{insert } 0 \ D))$ 
proof -
  let ?a =  $\text{sum } (\lambda b::'a::\text{euclidean\_space}. \text{inverse } (2 * \text{real } (\text{card } D)) *_{\mathbb{R}} b) \ D$ 
  have finite D
    using assms finite_Basis infinite_super by blast
  then have d1:  $0 < \text{real } (\text{card } D)$ 
    using  $\langle D \neq \{\} \rangle$  by auto
  {
    fix i
    assume  $i \in D$ 
    have  $?a \cdot i = \text{sum } (\lambda j. \text{if } i = j \text{ then } \text{inverse } (2 * \text{real } (\text{card } D)) \text{ else } 0) \ D$ 

```

```

    unfolding inner_sum_left
    using ⟨i ∈ D⟩ by (auto simp: inner_Basis subsetD[OF assms(2)] intro:
sum.cong)
    also have ... = inverse (2 * real (card D))
    using ⟨i ∈ D⟩ ⟨finite D⟩ by auto
    finally have ?a · i = inverse (2 * real (card D)) .
  }
  note ** = this
  show ?thesis
  proof
    show ?a ∈ rel_interior (convex hull (insert 0 D))
    unfolding rel_interior_substd_simplex[OF assms(2)]
    proof safe
      fix i
      assume i ∈ D
      have 0 < inverse (2 * real (card D))
      using d1 by auto
      also have ... = ?a · i using **[of i] ⟨i ∈ D⟩
      by auto
      finally show 0 < ?a · i by auto
    next
      have sum ((·) ?a) D = sum (λi. inverse (2 * real (card D))) D
      by (rule sum.cong) (rule refl, rule **)
      also have ... < 1
      unfolding sum_constant_divide_real_def[symmetric]
      by (auto simp add: field_simps)
      finally show sum ((·) ?a) D < 1 by auto
    next
      fix i
      assume i ∈ Basis and i ∉ D
      have ?a ∈ span D
      proof (rule span_sum[of D (λb. b /R (2 * real (card D))) D])
      {
        fix x :: 'a::euclidean_space
        assume x ∈ D
        then have x ∈ span D
        using span_base[of _ D] by auto
        then have x /R (2 * real (card D)) ∈ span D
        using span_mul[of x D (inverse (real (card D)) / 2)] by auto
      }
      then show ∧x. x ∈ D ⇒ x /R (2 * real (card D)) ∈ span D
      by auto
    qed
    then show ?a · i = 0
    using ⟨i ∉ D⟩ unfolding span_substd_basis[OF assms(2)] using ⟨i ∈
Basis⟩ by auto
  qed
  qed
  qed

```

5.0.3 Relative interior of convex set

```

lemma rel_interior_convex_nonempty_aux:
  fixes  $S :: 'n::euclidean\_space$  set
  assumes convex  $S$ 
  and  $0 \in S$ 
  shows  $\text{rel\_interior } S \neq \{\}$ 
proof (cases  $S = \{0\}$ )
  case True
  then show ?thesis using rel_interior_sing by auto
next
  case False
  obtain  $B$  where  $B$ : independent  $B \wedge B \leq S \wedge S \leq \text{span } B \wedge \text{card } B = \text{dim } S$ 
    using basis_exists[of  $S$ ] by metis
  then have  $B \neq \{\}$ 
    using  $B$  assms  $\langle S \neq \{0\} \rangle$  span_empty by auto
  have  $\text{insert } 0 B \leq \text{span } B$ 
    using subspace_span[of  $B$ ] subspace_0[of  $\text{span } B$ ]
    span_superset by auto
  then have  $\text{span } (\text{insert } 0 B) \leq \text{span } B$ 
    using span_span[of  $B$ ] span_mono[of  $\text{insert } 0 B$   $\text{span } B$ ] by blast
  then have  $\text{convex hull insert } 0 B \leq \text{span } B$ 
    using convex_hull_subset_span[of  $\text{insert } 0 B$ ] by auto
  then have  $\text{span } (\text{convex hull insert } 0 B) \leq \text{span } B$ 
    using span_span[of  $B$ ]
    span_mono[of  $\text{convex hull insert } 0 B$   $\text{span } B$ ] by blast
  then have *:  $\text{span } (\text{convex hull insert } 0 B) = \text{span } B$ 
    using span_mono[of  $B$   $\text{convex hull insert } 0 B$ ] hull_subset[of  $\text{insert } 0 B$ ] by
auto
  then have  $\text{span } (\text{convex hull insert } 0 B) = \text{span } S$ 
    using  $B$  span_mono[of  $B$   $S$ ] span_mono[of  $S$   $\text{span } B$ ]
    span_span[of  $B$ ] by auto
  moreover have  $0 \in \text{affine hull } (\text{convex hull insert } 0 B)$ 
    using hull_subset[of  $\text{convex hull insert } 0 B$ ] hull_subset[of  $\text{insert } 0 B$ ] by auto
  ultimately have **:  $\text{affine hull } (\text{convex hull insert } 0 B) = \text{affine hull } S$ 
    using affine_hull_span_0[of  $\text{convex hull insert } 0 B$ ] affine_hull_span_0[of  $S$ ]
    assms hull_subset[of  $S$ ]
    by auto
  obtain  $d$  and  $f :: 'n \Rightarrow 'n$  where
     $fd$ :  $\text{card } d = \text{card } B$  linear  $f$   $f' B = d$ 
     $f' \text{span } B = \{x. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = (0::\text{real})\} \wedge \text{inj\_on } f (\text{span } B)$ 
    and  $d$ :  $d \subseteq \text{Basis}$ 
    using basis_to_substdbasis_subspace_isomorphism[of  $B, OF \_$ ]  $B$  by auto
  then have bounded_linear  $f$ 
    using linear_conv_bounded_linear by auto
  have  $d \neq \{\}$ 
    using  $fd$   $B$   $\langle B \neq \{\} \rangle$  by auto
  have  $\text{insert } 0 d = f' (\text{insert } 0 B)$ 
    using  $fd$  linear_0 by auto
  then have  $(\text{convex hull } (\text{insert } 0 d)) = f' (\text{convex hull } (\text{insert } 0 B))$ 

```

```

    using convex_hull_linear_image[of f (insert 0 d)]
    convex_hull_linear_image[of f (insert 0 B)] <linear f>
  by auto
  moreover have rel_interior (f ` (convex hull insert 0 B)) = f ` rel_interior
    (convex hull insert 0 B)
  proof (rule rel_interior_injective_on_span_linear_image[OF <bounded_linear
    f>])
    show inj_on f (span (convex hull insert 0 B))
      using fd * by auto
  qed
  ultimately have rel_interior (convex hull insert 0 B) ≠ {}
    using rel_interior_substd_simplex_nonempty[OF <d ≠ {}> d] by fastforce
  moreover have convex hull (insert 0 B) ⊆ S
    using B assms hull_mono[of insert 0 B S convex] convex_hull_eq by auto
  ultimately show ?thesis
    using subset_rel_interior[of convex hull insert 0 B S] ** by auto
qed

```

```

lemma rel_interior_eq_empty:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows rel_interior S = {} ⟷ S = {}
proof -
  {
    assume S ≠ {}
    then obtain a where a ∈ S by auto
    then have 0 ∈ (+) (-a) ` S
      using assms exI[of (λx. x ∈ S ∧ - a + x = 0) a] by auto
    then have rel_interior ((+) (-a) ` S) ≠ {}
      using rel_interior_convex_nonempty_aux[of (+) (-a) ` S]
      convex_translation[of S -a] assms
      by auto
    then have rel_interior S ≠ {}
      using rel_interior_translation [of - a] by simp
  }
  then show ?thesis by auto
qed

```

```

lemma interior_simplex_nonempty:
  fixes S :: 'N :: euclidean_space set
  assumes independent S finite S card S = DIM('N)
  obtains a where a ∈ interior (convex hull (insert 0 S))
proof -
  have affine hull (insert 0 S) = UNIV
    by (simp add: hull_inc affine_hull_span_0 dim_eq_full[symmetric]
      assms(1) assms(3) dim_eq_card_independent)
  moreover have rel_interior (convex hull insert 0 S) ≠ {}
    using rel_interior_eq_empty [of convex hull (insert 0 S)] by auto
  ultimately have interior (convex hull insert 0 S) ≠ {}

```

```

    by (simp add: rel_interior_interior)
  with that show ?thesis
    by auto
qed

```

```

lemma convex_rel_interior:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows convex (rel_interior S)
proof -
  {
    fix x y and u :: real
    assume assm:  $x \in \text{rel\_interior } S$   $y \in \text{rel\_interior } S$   $0 \leq u$   $u \leq 1$ 
    then have  $x \in S$ 
      using rel_interior_subset by auto
    have  $x - u *_R (x - y) \in \text{rel\_interior } S$ 
    proof (cases  $0 = u$ )
    case False
    then have  $0 < u$  using assm by auto
    then show ?thesis
      using assm rel_interior_convex_shrink[of S y x u] assms  $\langle x \in S \rangle$  by auto
    next
    case True
    then show ?thesis using assm by auto
  }
  then show ?thesis
    unfolding convex_alt by auto
qed

```

```

lemma convex_closure_rel_interior:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows  $\text{closure } (\text{rel\_interior } S) = \text{closure } S$ 
proof -
  have h1:  $\text{closure } (\text{rel\_interior } S) \leq \text{closure } S$ 
    using closure_mono[of rel_interior S S] rel_interior_subset[of S] by auto
  show ?thesis
  proof (cases  $S = \{\}$ )
  case False
  then obtain a where  $a \in \text{rel\_interior } S$ 
    using rel_interior_eq_empty assms by auto
  { fix x
    assume  $x \in \text{closure } S$ 
    {
      assume  $x = a$ 
      then have  $x \in \text{closure } (\text{rel\_interior } S)$ 

```



```

    using a unfolding closure_def by auto
  }
  moreover
  {
    assume  $x \neq a$ 
    {
      fix e :: real
      assume  $e > 0$ 
      define e1 where  $e1 = \min 1 (e / \text{norm } (x - a))$ 
      then have e1:  $e1 > 0 \wedge e1 \leq 1 \wedge e1 * \text{norm } (x - a) \leq e$ 
        using  $\langle x \neq a \rangle \langle e > 0 \rangle$  le_divide_eq[of e1 e norm (x - a)]
        by simp_all
      then have *:  $x - e1 * (x - a) \in \text{rel\_interior } S$ 
        using rel_interior_closure_convex_shrink[of S a x e1] assms x a e1_def
        by auto
      have  $\exists y. y \in \text{rel\_interior } S \wedge y \neq x \wedge \text{dist } y x \leq e$ 
        using *  $\langle x \neq a \rangle$  e1 by force
    }
    then have x_islimpt rel_interior S
      unfolding islimpt_approachable_le by auto
    then have  $x \in \text{closure}(\text{rel\_interior } S)$ 
      unfolding closure_def by auto
  }
  ultimately have  $x \in \text{closure}(\text{rel\_interior } S)$  by auto
}
then show ?thesis using h1 by auto
qed auto
qed

lemma rel_interior_same_affine_hull:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows affine_hull (rel_interior S) = affine_hull S
  by (metis assms closure_same_affine_hull convex_closure_rel_interior)

lemma rel_interior_aff_dim:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows aff_dim (rel_interior S) = aff_dim S
  by (metis aff_dim_affine_hull2 assms rel_interior_same_affine_hull)

lemma rel_interior_rel_interior:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows rel_interior (rel_interior S) = rel_interior S
proof -
  have openin (top_of_set (affine_hull (rel_interior S))) (rel_interior S)
    using openin_rel_interior[of S] rel_interior_same_affine_hull[of S] assms by
    auto

```

```

    then show ?thesis
      using rel_interior_def by auto
qed

```

```

lemma rel_interior_rel_open:
  fixes  $S :: 'n::euclidean\_space$  set
  assumes convex  $S$ 
  shows rel_open (rel_interior  $S$ )
  unfolding rel_open_def using rel_interior_rel_interior assms by auto

```

```

lemma convex_rel_interior_closure_aux:
  fixes  $x\ y\ z :: 'n::euclidean\_space$ 
  assumes  $0 < a\ 0 < b\ (a + b) *_R z = a *_R x + b *_R y$ 
  obtains  $e$  where  $0 < e\ e < 1\ z = y - e *_R (y - x)$ 
proof -
  define  $e$  where  $e = a / (a + b)$ 
  have  $z = (1 / (a + b)) *_R ((a + b) *_R z)$ 
    using assms by (simp add: eq_vector_fraction_iff)
  also have  $\dots = (1 / (a + b)) *_R (a *_R x + b *_R y)$ 
    using assms scaleR_cancel_left[of  $1/(a+b)$   $(a + b) *_R z\ a *_R x + b *_R y$ ]
    by auto
  also have  $\dots = y - e *_R (y - x)$ 
    using e_def assms
    by (simp add: divide_simps vector_fraction_eq_iff) (simp add: algebra_simps)
  finally have  $z = y - e *_R (y - x)$ 
    by auto
  moreover have  $e > 0\ e < 1$  using e_def assms by auto
  ultimately show ?thesis using that[of  $e$ ] by auto
qed

```

```

lemma convex_rel_interior_closure:
  fixes  $S :: 'n::euclidean\_space$  set
  assumes convex  $S$ 
  shows rel_interior (closure  $S$ ) = rel_interior  $S$ 
proof (cases  $S = \{\}$ )
case True
  then show ?thesis
    using assms rel_interior_eq_empty by auto
next
case False
  have rel_interior (closure  $S$ )  $\supseteq$  rel_interior  $S$ 
    using subset_rel_interior[of  $S$  closure  $S$ ] closure_same_affine_hull closure_subset
    by auto
  moreover
  {
    fix  $z$ 
    assume  $z: z \in \text{rel\_interior (closure } S)$ 
    obtain  $x$  where  $x: x \in \text{rel\_interior } S$ 
      using  $\langle S \neq \{\} \rangle$  assms rel_interior_eq_empty by auto

```

```

have  $z \in \text{rel\_interior } S$ 
proof (cases  $x = z$ )
  case True
    then show ?thesis using  $x$  by auto
  next
  case False
    obtain  $e$  where  $e: e > 0$  cball  $z \in \cap \text{affine hull closure } S \leq \text{closure } S$ 
      using  $\text{rel\_interior\_cball}[of \text{closure } S]$  by auto
    hence *:  $0 < e/\text{norm}(z-x)$  using  $e$  False by auto
    define  $y$  where  $y = z + (e/\text{norm}(z-x)) *_R (z-x)$ 
    have yball:  $y \in \text{cball } z \ e$ 
      using  $y\_def \text{dist\_norm}[of z y] \ e$  by auto
    have  $x \in \text{affine hull closure } S$ 
      using  $x \text{rel\_interior\_subset\_closure hull\_inc}[of x \text{closure } S]$  by blast
    moreover have  $z \in \text{affine hull closure } S$ 
      using  $\text{rel\_interior\_subset hull\_subset}[of \text{closure } S]$  by blast
    ultimately have  $y \in \text{affine hull closure } S$ 
      using  $y\_def \text{affine\_affine\_hull}[of \text{closure } S]$ 
      mem_affine_3_minus [of affine hull closure  $S$   $z \ z \ x \ e/\text{norm}(z-x)$ ] by auto
    then have  $y \in \text{closure } S$  using  $e$  yball by auto
    have  $(1 + (e/\text{norm}(z-x))) *_R z = (e/\text{norm}(z-x)) *_R x + y$ 
      using  $y\_def$  by (simp add: algebra_simps)
    then obtain  $e1$  where  $0 < e1 \ e1 < 1 \ z = y - e1 *_R (y - x)$ 
      using * convex_rel_interior_closure_aux[ $of \ e \ / \ \text{norm } (z - x) \ 1 \ z \ x \ y$ ]
      by (auto simp add: algebra_simps)
    then show ?thesis
      using rel_interior_closure_convex_shrink assms  $x \ \langle y \in \text{closure } S \rangle$ 
      by fastforce
  qed
}
ultimately show ?thesis by auto
qed

```

```

lemma convex_interior_closure:
  fixes  $S :: 'n::\text{euclidean\_space} \text{ set}$ 
  assumes convex  $S$ 
  shows interior (closure  $S$ ) = interior  $S$ 
  using closure_aff_dim[ $of S$ ] interior_rel_interior_gen[ $of S$ ]
    interior_rel_interior_gen[ $of \text{closure } S$ ]
    convex_rel_interior_closure[ $of S$ ] assms
  by auto

```

```

lemma closure_eq_rel_interior_eq:
  fixes  $S1 \ S2 :: 'n::\text{euclidean\_space} \text{ set}$ 
  assumes convex  $S1$ 
    and convex  $S2$ 
  shows closure  $S1 = \text{closure } S2 \longleftrightarrow \text{rel\_interior } S1 = \text{rel\_interior } S2$ 
  by (metis convex_rel_interior_closure convex_closure_rel_interior assms)

```

```

lemma closure_eq_between:
  fixes S1 S2 :: 'n::euclidean_space set
  assumes convex S1
    and convex S2
  shows closure S1 = closure S2  $\longleftrightarrow$  rel_interior S1  $\subseteq$  S2  $\wedge$  S2  $\subseteq$  closure S1
    (is ?A  $\longleftrightarrow$  ?B)
proof
  assume ?A
  then show ?B
    by (metis assms closure_subset convex_rel_interior_closure rel_interior_subset)
next
  assume ?B
  then have closure S1  $\subseteq$  closure S2
    by (metis assms(1) convex_closure_rel_interior_closure_mono)
  moreover from <?B> have closure S1  $\supseteq$  closure S2
    by (metis closed_closure_closure_minimal)
  ultimately show ?A ..
qed

```

```

lemma open_inter_closure_rel_interior:
  fixes S A :: 'n::euclidean_space set
  assumes convex S
    and open A
  shows A  $\cap$  closure S =  $\{\}$   $\longleftrightarrow$  A  $\cap$  rel_interior S =  $\{\}$ 
  by (metis assms convex_closure_rel_interior open_Int_closure_eq_empty)

```

```

lemma rel_interior_open_segment:
  fixes a :: 'a :: euclidean_space
  shows rel_interior(open_segment a b) = open_segment a b
proof (cases a = b)
  case True then show ?thesis by auto
next
  case False then
    have open_segment a b = affine_hull {a, b}  $\cap$  ball ((a + b) /R 2) (norm (b - a) / 2)
      by (simp add: open_segment_as_ball)
    then show ?thesis
      unfolding rel_interior_eq_openin_open
      by (metis Elementary_Metric_Spaces.open_ball False affine_hull_open_segment)
qed

```

```

lemma rel_interior_closed_segment:
  fixes a :: 'a :: euclidean_space
  shows rel_interior(closed_segment a b) =
    (if a = b then {a} else open_segment a b)
proof (cases a = b)
  case True then show ?thesis by auto
next
  case False then show ?thesis

```

```

  by simp
  (metis closure_open_segment convex_open_segment convex_rel_interior_closure
    rel_interior_open_segment)
qed

lemmas rel_interior_segment = rel_interior_closed_segment rel_interior_open_segment

```

5.0.4 The relative frontier of a set

definition $rel\_frontier\ S = closure\ S - rel\_interior\ S$

lemma $rel\_frontier\_empty$ [simp]: $rel\_frontier\ \{\} = \{\}$
 by (simp add: rel_frontier_def)

lemma $rel\_frontier\_eq\_empty$:
 fixes $S :: 'n::euclidean\_space\ set$
 shows $rel\_frontier\ S = \{\} \longleftrightarrow affine\ S$
 unfolding rel_frontier_def
 using rel_interior_subset_closure by (auto simp add: rel_interior_eq_closure
 [symmetric])

lemma $rel\_frontier\_sing$ [simp]:
 fixes $a :: 'n::euclidean\_space$
 shows $rel\_frontier\ \{a\} = \{\}$
 by (simp add: rel_frontier_def)

lemma $rel\_frontier\_affine\_hull$:
 fixes $S :: 'a::euclidean\_space\ set$
 shows $rel\_frontier\ S \subseteq affine\ hull\ S$
 using closure_affine_hull rel_frontier_def by fastforce

lemma $rel\_frontier\_cball$ [simp]:
 fixes $a :: 'n::euclidean\_space$
 shows $rel\_frontier(cball\ a\ r) = (if\ r = 0\ then\ \{\}\ else\ sphere\ a\ r)$
proof (cases rule: linorder_cases [of $r\ 0$])
 case less then show ?thesis
 by (force simp: sphere_def)
next
 case equal then show ?thesis by simp
next
 case greater then show ?thesis
 by simp (metis centre_in_ball empty_iff frontier_cball frontier_def inte-
 rior_cball interior_rel_interior_gen rel_frontier_def)
qed

lemma $rel\_frontier\_translation$:
 fixes $a :: 'a::euclidean\_space$
 shows $rel\_frontier((\lambda x. a + x) \cdot S) = (\lambda x. a + x) \cdot (rel\_frontier\ S)$
 by (simp add: rel_frontier_def translation_diff rel_interior_translation clo-

sure_translation)

lemma *rel_frontier_nonempty_interior*:
 fixes $S :: 'n::\text{euclidean\_space set}$
 shows $\text{interior } S \neq \{\} \implies \text{rel\_frontier } S = \text{frontier } S$
 by (metis *frontier_def interior_rel_interior_gen rel_frontier_def*)

lemma *rel_frontier_frontier*:
 fixes $S :: 'n::\text{euclidean\_space set}$
 shows $\text{affine hull } S = \text{UNIV} \implies \text{rel\_frontier } S = \text{frontier } S$
 by (simp add: *frontier_def rel_frontier_def rel_interior_interior*)

lemma *closest_point_in_rel_frontier*:
 $\llbracket \text{closed } S; S \neq \{\}; x \in \text{affine hull } S - \text{rel\_interior } S \rrbracket$
 $\implies \text{closest\_point } S x \in \text{rel\_frontier } S$
 by (simp add: *closest_point_in_rel_interior closest_point_in_set rel_frontier_def*)

lemma *closed_rel_frontier [iff]*:
 fixes $S :: 'n::\text{euclidean\_space set}$
 shows $\text{closed } (\text{rel\_frontier } S)$
proof –
 have *: $\text{closedin } (\text{top\_of\_set } (\text{affine hull } S)) (\text{closure } S - \text{rel\_interior } S)$
 by (simp add: *closed_subset closedin_diff closure_affine_hull openin_rel_interior*)
 show ?thesis
proof (rule *closedin_closed_trans*[of $\text{affine hull } S$ $\text{rel\_frontier } S$])
 show $\text{closedin } (\text{top\_of\_set } (\text{affine hull } S)) (\text{rel\_frontier } S)$
 by (simp add: * *rel_frontier_def*)
qed simp
qed

lemma *closed_rel_boundary*:
 fixes $S :: 'n::\text{euclidean\_space set}$
 shows $\text{closed } S \implies \text{closed}(S - \text{rel\_interior } S)$
 by (metis *closed_rel_frontier closure_closed rel_frontier_def*)

lemma *compact_rel_boundary*:
 fixes $S :: 'n::\text{euclidean\_space set}$
 shows $\text{compact } S \implies \text{compact}(S - \text{rel\_interior } S)$
 by (metis *bounded_diff closed_rel_boundary closure_eq compact_closure compact_imp_closed*)

lemma *bounded_rel_frontier*:
 fixes $S :: 'n::\text{euclidean\_space set}$
 shows $\text{bounded } S \implies \text{bounded}(\text{rel\_frontier } S)$
 by (simp add: *bounded_closure bounded_diff rel_frontier_def*)

lemma *compact_rel_frontier_bounded*:
 fixes $S :: 'n::\text{euclidean\_space set}$
 shows $\text{bounded } S \implies \text{compact}(\text{rel\_frontier } S)$

using *bounded_rel_frontier closed_rel_frontier compact_eq_bounded_closed* **by** *blast*

lemma *compact_rel_frontier*:
fixes $S :: 'n::\text{euclidean\_space set}$
shows $\text{compact } S \implies \text{compact}(\text{rel\_frontier } S)$
by (*meson compact_eq_bounded_closed compact_rel_frontier_bounded*)

lemma *convex_same_rel_interior_closure*:
fixes $S :: 'n::\text{euclidean\_space set}$
shows $\llbracket \text{convex } S; \text{convex } T \rrbracket$
 $\implies \text{rel\_interior } S = \text{rel\_interior } T \longleftrightarrow \text{closure } S = \text{closure } T$
by (*simp add: closure_eq_rel_interior_eq*)

lemma *convex_same_rel_interior_closure_straddle*:
fixes $S :: 'n::\text{euclidean\_space set}$
shows $\llbracket \text{convex } S; \text{convex } T \rrbracket$
 $\implies \text{rel\_interior } S = \text{rel\_interior } T \longleftrightarrow$
 $\text{rel\_interior } S \subseteq T \wedge T \subseteq \text{closure } S$
by (*simp add: closure_eq_between_convex_same_rel_interior_closure*)

lemma *convex_rel_frontier_aff_dim*:
fixes $S1\ S2 :: 'n::\text{euclidean\_space set}$
assumes *convex S1*
and *convex S2*
and $S2 \neq \{\}$
and $S1 \leq \text{rel\_frontier } S2$
shows $\text{aff\_dim } S1 < \text{aff\_dim } S2$
proof –
have $S1 \subseteq \text{closure } S2$
using *assms unfolding rel_frontier_def* **by** *auto*
then have $*, \text{affine hull } S1 \subseteq \text{affine hull } S2$
using *hull_mono[of S1 closure S2] closure_same_affine_hull[of S2]* **by** *blast*
then have $\text{aff\_dim } S1 \leq \text{aff\_dim } S2$
using $*, \text{aff\_dim\_affine\_hull}[of S1] \text{aff\_dim\_affine\_hull}[of S2]$
 $\text{aff\_dim\_subset}[of \text{affine hull } S1 \text{affine hull } S2]$
by *auto*
moreover
{
assume $\text{eq: aff\_dim } S1 = \text{aff\_dim } S2$
then have $S1 \neq \{\}$
using $\text{aff\_dim\_empty}[of S1] \text{aff\_dim\_empty}[of S2] \langle S2 \neq \{\} \rangle$ **by** *auto*
have $**, \text{affine hull } S1 = \text{affine hull } S2$
by (*simp_all add: * eq $\langle S1 \neq \{\} \rangle$ affine_dim_equal*)
obtain a **where** $a \in \text{rel\_interior } S1$
using $\langle S1 \neq \{\} \rangle \text{rel\_interior\_eq\_empty assms}$ **by** *auto*
obtain T **where** $T: \text{open } T \wedge a \in T \cap S1 \wedge T \cap \text{affine hull } S1 \subseteq S1$
using $\text{mem\_rel\_interior}[of a S1]$ **a** **by** *auto*
then have $a \in T \cap \text{closure } S2$

```

    using a assms unfolding rel_frontier_def by auto
  then obtain b where b:  $b \in T \cap \text{rel\_interior } S2$ 
    using open_inter_closure_rel_interior[of S2 T] assms T by auto
  then have  $b \in \text{affine hull } S1$ 
    using rel_interior_subset hull_subset[of S2] ** by auto
  then have  $b \in S1$ 
    using T b by auto
  then have False
    using b assms unfolding rel_frontier_def by auto
}
ultimately show ?thesis
  using less_le by auto
qed

```

```

lemma convex_rel_interior_if:
  fixes S :: 'n::euclidean_space set
  assumes convex S
    and  $z \in \text{rel\_interior } S$ 
  shows  $\forall x \in \text{affine hull } S. \exists m. m > 1 \wedge (\forall e. e > 1 \wedge e \leq m \longrightarrow (1 - e) *_R x + e *_R z \in S)$ 
proof -
  obtain e1 where e1:  $e1 > 0 \wedge \text{cball } z \ e1 \cap \text{affine hull } S \subseteq S$ 
    using mem_rel_interior_cball[of z S] assms by auto
  {
    fix x
    assume x:  $x \in \text{affine hull } S$ 
    {
      assume  $x \neq z$ 
      define m where  $m = 1 + e1 / \text{norm}(x - z)$ 
      hence  $m > 1$  using e1  $\langle x \neq z \rangle$  by auto
      {
        fix e
        assume e:  $e > 1 \wedge e \leq m$ 
        have  $z \in \text{affine hull } S$ 
          using assms rel_interior_subset hull_subset[of S] by auto
        then have *:  $(1 - e) *_R x + e *_R z \in \text{affine hull } S$ 
          using mem_affine[of affine hull S x z (1 - e) e] affine_affine_hull[of S] x
          by auto
        have  $\text{norm } (z + e *_R x - (x + e *_R z)) = \text{norm } ((e - 1) *_R (x - z))$ 
          by (simp add: algebra_simps)
        also have  $\dots = (e - 1) * \text{norm } (x - z)$ 
          using norm_scaleR e by auto
        also have  $\dots \leq (m - 1) * \text{norm } (x - z)$ 
          using e mult_right_mono[of _ _ norm(x-z)] by auto
        also have  $\dots = (e1 / \text{norm } (x - z)) * \text{norm } (x - z)$ 
          using m_def by auto
        also have  $\dots = e1$ 
          using  $\langle x \neq z \rangle$  e1 by simp
        finally have **:  $\text{norm } (z + e *_R x - (x + e *_R z)) \leq e1$ 

```



```

      by auto
    have  $(1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in \text{cball } z \ e1$ 
      using  $m\_def$  **
      unfolding  $\text{cball\_def}$   $\text{dist\_norm}$ 
      by (auto simp add:  $\text{algebra\_simps}$ )
    then have  $(1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S$ 
      using  $e * e1$  by auto
  }
  then have  $\exists m. m > 1 \wedge (\forall e. e > 1 \wedge e \leq m \longrightarrow (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S)$ 
    using  $\langle m > 1 \rangle$  by auto
  }
  moreover
  {
    assume  $x = z$ 
    define  $m$  where  $m = 1 + e1$ 
    then have  $m > 1$ 
      using  $e1$  by auto
    {
      fix  $e$ 
      assume  $e: e > 1 \wedge e \leq m$ 
      then have  $(1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S$ 
        using  $e1$   $x \langle x = z \rangle$  by (auto simp add:  $\text{algebra\_simps}$ )
      then have  $(1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S$ 
        using  $e$  by auto
    }
    then have  $\exists m. m > 1 \wedge (\forall e. e > 1 \wedge e \leq m \longrightarrow (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S)$ 
      using  $\langle m > 1 \rangle$  by auto
  }
  ultimately have  $\exists m. m > 1 \wedge (\forall e. e > 1 \wedge e \leq m \longrightarrow (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S)$ 
    by blast
  }
  then show ?thesis by auto
qed

```

```

lemma convex_rel_interior_if2:
  fixes  $S :: 'n::\text{euclidean\_space}$  set
  assumes convex  $S$ 
  assumes  $z \in \text{rel\_interior } S$ 
  shows  $\forall x \in \text{affine hull } S. \exists e. e > 1 \wedge (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S$ 
  using convex_rel_interior_if[of  $S$   $z$ ] assms by auto

```

```

lemma convex_rel_interior_only_if:
  fixes  $S :: 'n::\text{euclidean\_space}$  set
  assumes convex  $S$ 
  and  $S \neq \{\}$ 
  assumes  $\forall x \in S. \exists e. e > 1 \wedge (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S$ 

```

```

    shows  $z \in \text{rel\_interior } S$ 
  proof -
    obtain  $x$  where  $x: x \in \text{rel\_interior } S$ 
      using  $\text{rel\_interior\_eq\_empty } \text{assms}$  by auto
    then have  $x \in S$ 
      using  $\text{rel\_interior\_subset}$  by auto
    then obtain  $e$  where  $e: e > 1 \wedge (1 - e) *_R x + e *_R z \in S$ 
      using  $\text{assms}$  by auto
    define  $y$  where  $[\text{abs\_def}]: y = (1 - e) *_R x + e *_R z$ 
    then have  $y \in S$  using  $e$  by auto
    define  $e1$  where  $e1 = 1/e$ 
    then have  $0 < e1 \wedge e1 < 1$  using  $e$  by auto
    then have  $z = y - (1 - e1) *_R (y - x)$ 
      using  $e1\_def\ y\_def$  by (auto simp add: algebra_simps)
    then show ?thesis
      using  $\text{rel\_interior\_convex\_shrink}[of\ S\ x\ y\ 1-e1]\ \langle 0 < e1 \wedge e1 < 1 \rangle\ \langle y \in S \rangle$ 
    x assms
      by auto
  qed

```

```

lemma convex_rel_interior_iff:
  fixes  $S :: 'n::\text{euclidean\_space}$  set
  assumes convex  $S$ 
    and  $S \neq \{\}$ 
  shows  $z \in \text{rel\_interior } S \longleftrightarrow (\forall x \in S. \exists e. e > 1 \wedge (1 - e) *_R x + e *_R z \in S)$ 
  using assms hull_subset[of  $S$  affine]
    convex_rel_interior_if[of  $S\ z$ ] convex_rel_interior_only_if[of  $S\ z$ ]
  by auto

```

```

lemma convex_rel_interior_iff2:
  fixes  $S :: 'n::\text{euclidean\_space}$  set
  assumes convex  $S$ 
    and  $S \neq \{\}$ 
  shows  $z \in \text{rel\_interior } S \longleftrightarrow (\forall x \in \text{affine hull } S. \exists e. e > 1 \wedge (1 - e) *_R x + e *_R z \in S)$ 
  using assms hull_subset[of  $S$ ] convex_rel_interior_if2[of  $S\ z$ ] convex_rel_interior_only_if[of  $S\ z$ ]
  by auto

```

```

lemma convex_interior_iff:
  fixes  $S :: 'n::\text{euclidean\_space}$  set
  assumes convex  $S$ 
  shows  $z \in \text{interior } S \longleftrightarrow (\forall x. \exists e. e > 0 \wedge z + e *_R x \in S)$ 
proof (cases  $\text{aff\_dim } S = \text{int } \text{DIM}('n)$ )
case False
{ assume  $z \in \text{interior } S$ 
  then have False
    using False interior_rel_interior_gen[of  $S$ ] by auto }
moreover

```

```

{ assume r:  $\forall x. \exists e. e > 0 \wedge z + e *_R x \in S$ 
  { fix x
    obtain e1 where e1:  $e1 > 0 \wedge z + e1 *_R (x - z) \in S$ 
    using r by auto
    obtain e2 where e2:  $e2 > 0 \wedge z + e2 *_R (z - x) \in S$ 
    using r by auto
    define x1 where [abs_def]:  $x1 = z + e1 *_R (x - z)$ 
    then have x1:  $x1 \in \text{affine hull } S$ 
    using e1 hull_subset[of S] by auto
    define x2 where [abs_def]:  $x2 = z + e2 *_R (z - x)$ 
    then have x2:  $x2 \in \text{affine hull } S$ 
    using e2 hull_subset[of S] by auto
    have *:  $e1/(e1+e2) + e2/(e1+e2) = 1$ 
    using add_divide_distrib[of e1 e2 e1+e2] e1 e2 by simp
    then have z =  $(e2/(e1+e2)) *_R x1 + (e1/(e1+e2)) *_R x2$ 
    by (simp add: x1_def x2_def algebra_simps) (simp add: * pth_8)
    then have z:  $z \in \text{affine hull } S$ 
    using mem_affine[of affine hull S x1 x2 e2/(e1+e2) e1/(e1+e2)]
      x1 x2 affine_affine_hull[of S] *
    by auto
    have  $x1 - x2 = (e1 + e2) *_R (x - z)$ 
    using x1_def x2_def by (auto simp add: algebra_simps)
    then have x =  $z + (1/(e1+e2)) *_R (x1 - x2)$ 
    using e1 e2 by simp
    then have x  $\in \text{affine hull } S$ 
    using mem_affine_3_minus[of affine hull S z x1 x2 1/(e1+e2)]
      x1 x2 z affine_affine_hull[of S]
    by auto
  }
  then have affine hull S = UNIV
  by auto
  then have aff_dim S = int DIM('n)
  using aff_dim_affine_hull[of S] by (simp)
  then have False
  using False by auto
}
ultimately show ?thesis by auto
next
case True
then have  $S \neq \{\}$ 
using aff_dim_empty[of S] by auto
have *: affine hull S = UNIV
using True affine_hull_UNIV by auto
{
  assume  $z \in \text{interior } S$ 
  then have  $z \in \text{rel\_interior } S$ 
  using True interior_rel_interior_gen[of S] by auto
  then have **:  $\forall x. \exists e. e > 1 \wedge (1 - e) *_R x + e *_R z \in S$ 
  using convex_rel_interior_iff2[of S z] assms ⟨S ≠ {}⟩ * by auto
}

```

```

fix x
obtain e1 where e1: e1 > 1 (1 - e1) *R (z - x) + e1 *R z ∈ S
  using **[rule_format, of z-x] by auto
define e where [abs_def]: e = e1 - 1
then have (1 - e1) *R (z - x) + e1 *R z = z + e *R x
  by (simp add: algebra_simps)
then have e > 0 z + e *R x ∈ S
  using e1 e_def by auto
then have ∃ e. e > 0 ∧ z + e *R x ∈ S
  by auto
}
moreover
{
  assume r: ∀ x. ∃ e. e > 0 ∧ z + e *R x ∈ S
  {
    fix x
    obtain e1 where e1: e1 > 0 z + e1 *R (z - x) ∈ S
      using r[rule_format, of z-x] by auto
    define e where e = e1 + 1
    then have z + e1 *R (z - x) = (1 - e) *R x + e *R z
      by (simp add: algebra_simps)
    then have e > 1 (1 - e) *R x + e *R z ∈ S
      using e1 e_def by auto
    then have ∃ e. e > 1 ∧ (1 - e) *R x + e *R z ∈ S by auto
  }
  then have z ∈ rel_interior S
    using convex_rel_interior_iff2[of S z] assms ⟨S ≠ {}⟩ by auto
  then have z ∈ interior S
    using True interior_rel_interior_gen[of S] by auto
}
ultimately show ?thesis by auto
qed

```

Relative interior and closure under common operations

lemma *rel_interior_inter_aux*: $\bigcap \{\text{rel\_interior } S \mid S. S \in I\} \subseteq \bigcap I$

proof -

```

{
  fix y
  assume y ∈ ⋂ {rel_interior S | S. S ∈ I}
  then have y: ∀ S ∈ I. y ∈ rel_interior S
    by auto
  {
    fix S
    assume S ∈ I
    then have y ∈ S
      using rel_interior_subset y by auto
  }
  then have y ∈ ⋂ I by auto
}

```

```

}
then show ?thesis by auto
qed

lemma convex_closure_rel_interior_inter:
  assumes  $\forall S \in I. \text{convex } (S :: 'n::\text{euclidean\_space set})$ 
  and  $\bigcap \{\text{rel\_interior } S \mid S. S \in I\} \neq \{\}$ 
  shows  $\bigcap \{\text{closure } S \mid S. S \in I\} \leq \text{closure } (\bigcap \{\text{rel\_interior } S \mid S. S \in I\})$ 
proof -
  obtain x where x:  $\forall S \in I. x \in \text{rel\_interior } S$ 
  using assms by auto
  {
    fix y
    assume  $y \in \bigcap \{\text{closure } S \mid S. S \in I\}$ 
    then have y:  $\forall S \in I. y \in \text{closure } S$ 
    by auto
    {
      assume  $y = x$ 
      then have  $y \in \text{closure } (\bigcap \{\text{rel\_interior } S \mid S. S \in I\})$ 
      using x closure_subset[of  $\bigcap \{\text{rel\_interior } S \mid S. S \in I\}$ ] by auto
    }
    moreover
    {
      assume  $y \neq x$ 
      { fix e :: real
        assume e:  $e > 0$ 
        define e1 where  $e1 = \min 1 (e / \text{norm } (y - x))$ 
        then have e1:  $e1 > 0 \wedge e1 \leq 1 \wedge e1 * \text{norm } (y - x) \leq e$ 
        using  $\langle y \neq x \rangle \langle e > 0 \rangle$  le_divide_eq[of e1 e norm (y - x)]
        by simp_all
        define z where  $z = y - e1 * (y - x)$ 
        {
          fix S
          assume  $S \in I$ 
          then have  $z \in \text{rel\_interior } S$ 
          using rel_interior_closure_convex_shrink[of S x y e1] assms x y e1
          z_def
          by auto
        }
        then have *:  $z \in \bigcap \{\text{rel\_interior } S \mid S. S \in I\}$ 
        by auto
        have  $\exists z. z \in \bigcap \{\text{rel\_interior } S \mid S. S \in I\} \wedge z \neq y \wedge \text{dist } z y \leq e$ 
        using  $\langle y \neq x \rangle$  z_def * e1 e dist_norm[of z y]
        by (rule_tac x=z in exI) auto
      }
    }
  then have y islimpt  $\bigcap \{\text{rel\_interior } S \mid S. S \in I\}$ 
  unfolding islimpt_approachable_le by blast
  then have  $y \in \text{closure } (\bigcap \{\text{rel\_interior } S \mid S. S \in I\})$ 
  unfolding closure_def by auto

```

```

    }
    ultimately have  $y \in \text{closure } (\bigcap \{\text{rel\_interior } S \mid S. S \in I\})$ 
      by auto
  }
  then show ?thesis by auto
qed

```

```

lemma convex_closure_inter:
  assumes  $\forall S \in I. \text{convex } (S :: 'n::\text{euclidean\_space set})$ 
  and  $\bigcap \{\text{rel\_interior } S \mid S. S \in I\} \neq \{\}$ 
  shows  $\text{closure } (\bigcap I) = \bigcap \{\text{closure } S \mid S. S \in I\}$ 
proof -
  have  $\bigcap \{\text{closure } S \mid S. S \in I\} \leq \text{closure } (\bigcap \{\text{rel\_interior } S \mid S. S \in I\})$ 
    using convex_closure_rel_interior_inter assms by auto
  moreover
  have  $\text{closure } (\bigcap \{\text{rel\_interior } S \mid S. S \in I\}) \leq \text{closure } (\bigcap I)$ 
    using rel_interior_inter_aux closure_mono[of  $\bigcap \{\text{rel\_interior } S \mid S. S \in I\}$ 
 $\bigcap I$ ]
    by auto
  ultimately show ?thesis
    using closure_Int[of I] by auto
qed

```

```

lemma convex_inter_rel_interior_same_closure:
  assumes  $\forall S \in I. \text{convex } (S :: 'n::\text{euclidean\_space set})$ 
  and  $\bigcap \{\text{rel\_interior } S \mid S. S \in I\} \neq \{\}$ 
  shows  $\text{closure } (\bigcap \{\text{rel\_interior } S \mid S. S \in I\}) = \text{closure } (\bigcap I)$ 
proof -
  have  $\bigcap \{\text{closure } S \mid S. S \in I\} \leq \text{closure } (\bigcap \{\text{rel\_interior } S \mid S. S \in I\})$ 
    using convex_closure_rel_interior_inter assms by auto
  moreover
  have  $\text{closure } (\bigcap \{\text{rel\_interior } S \mid S. S \in I\}) \leq \text{closure } (\bigcap I)$ 
    using rel_interior_inter_aux closure_mono[of  $\bigcap \{\text{rel\_interior } S \mid S. S \in I\}$ 
 $\bigcap I$ ]
    by auto
  ultimately show ?thesis
    using closure_Int[of I] by auto
qed

```

```

lemma convex_rel_interior_inter:
  assumes  $\forall S \in I. \text{convex } (S :: 'n::\text{euclidean\_space set})$ 
  and  $\bigcap \{\text{rel\_interior } S \mid S. S \in I\} \neq \{\}$ 
  shows  $\text{rel\_interior } (\bigcap I) \subseteq \bigcap \{\text{rel\_interior } S \mid S. S \in I\}$ 
proof -
  have convex  $(\bigcap I)$ 
    using assms convex_Inter by auto
  moreover
  have convex  $(\bigcap \{\text{rel\_interior } S \mid S. S \in I\})$ 
    using assms convex_rel_interior by (force intro: convex_Inter)

```

```

ultimately
have rel_interior ( $\bigcap \{ \text{rel\_interior } S \mid S. S \in I \} = \text{rel\_interior } (\bigcap I)$ )
  using convex_inter_rel_interior_same_closure assms
  closure_eq_rel_interior_eq[of  $\bigcap \{ \text{rel\_interior } S \mid S. S \in I \} \bigcap I$ ]
  by blast
then show ?thesis
  using rel_interior_subset[of  $\bigcap \{ \text{rel\_interior } S \mid S. S \in I \}$ ] by auto
qed

lemma convex_rel_interior_finite_inter:
  assumes  $\forall S \in I. \text{convex } (S :: 'n::\text{euclidean\_space set})$ 
  and  $\bigcap \{ \text{rel\_interior } S \mid S. S \in I \} \neq \{ \}$ 
  and finite I
  shows  $\text{rel\_interior } (\bigcap I) = \bigcap \{ \text{rel\_interior } S \mid S. S \in I \}$ 
proof -
  have  $\bigcap I \neq \{ \}$ 
  using assms rel_interior_inter_aux[of I] by auto
  have convex ( $\bigcap I$ )
  using convex_Inter assms by auto
  show ?thesis
  proof (cases  $I = \{ \}$ )
    case True
    then show ?thesis
      using Inter_empty rel_interior_UNIV by auto
  next
    case False
    {
      fix z
      assume  $z: z \in \bigcap \{ \text{rel\_interior } S \mid S. S \in I \}$ 
      {
        fix x
        assume  $x: x \in \bigcap I$ 
        {
          fix S
          assume  $S: S \in I$ 
          then have  $z \in \text{rel\_interior } S \ x \in S$ 
          using z x by auto
          then have  $\exists m. m > 1 \wedge (\forall e. e > 1 \wedge e \leq m \longrightarrow (1 - e) *_R x + e *_R$ 
 $z \in S)$ 
          using convex_rel_interior_if[of S z] S assms hull_subset[of S] by auto
        }
        then obtain mS where
           $mS: \forall S \in I. mS S > 1 \wedge (\forall e. e > 1 \wedge e \leq mS S \longrightarrow (1 - e) *_R x + e$ 
 $*_R z \in S)$  by metis
        define e where  $e = \text{Min } (mS \text{ ' } I)$ 
        then have  $e \in mS \text{ ' } I$  using assms  $\langle I \neq \{ \} \rangle$  by simp
        then have  $e > 1$  using mS by auto
        moreover have  $\forall S \in I. e \leq mS S$ 
        using e_def assms by auto
      }
    }
  }

```

```

      ultimately have  $\exists e > 1. (1 - e) *_{\mathcal{R}} x + e *_{\mathcal{R}} z \in \bigcap I$ 
      using mS by auto
    }
    then have  $z \in \text{rel\_interior } (\bigcap I)$ 
    using convex_rel_interior_iff[of  $\bigcap I$  z]  $\langle \bigcap I \neq \{\} \rangle \langle \text{convex } (\bigcap I) \rangle$  by auto
  }
  then show ?thesis
  using convex_rel_interior_inter[of I] assms by auto
qed
qed

```

```

lemma convex_closure_inter_two:
  fixes  $S\ T :: 'n::\text{euclidean\_space set}$ 
  assumes convex S
  and convex T
  assumes  $\text{rel\_interior } S \cap \text{rel\_interior } T \neq \{\}$ 
  shows  $\text{closure } (S \cap T) = \text{closure } S \cap \text{closure } T$ 
  using convex_closure_inter[of  $\{S, T\}$ ] assms by auto

```

```

lemma convex_rel_interior_inter_two:
  fixes  $S\ T :: 'n::\text{euclidean\_space set}$ 
  assumes convex S
  and convex T
  and  $\text{rel\_interior } S \cap \text{rel\_interior } T \neq \{\}$ 
  shows  $\text{rel\_interior } (S \cap T) = \text{rel\_interior } S \cap \text{rel\_interior } T$ 
  using convex_rel_interior_finite_inter[of  $\{S, T\}$ ] assms by auto

```

```

lemma convex_affine_closure_Int:
  fixes  $S\ T :: 'n::\text{euclidean\_space set}$ 
  assumes convex S
  and affine T
  and  $\text{rel\_interior } S \cap T \neq \{\}$ 
  shows  $\text{closure } (S \cap T) = \text{closure } S \cap T$ 
  by (metis affine_imp_convex assms convex_closure_inter_two rel_interior_affine
  rel_interior_eq_closure)

```

```

lemma connected_component_1_gen:
  fixes  $S :: 'a :: \text{euclidean\_space set}$ 
  assumes  $\text{DIM}('a) = 1$ 
  shows  $\text{connected\_component } S\ a\ b \longleftrightarrow \text{closed\_segment } a\ b \subseteq S$ 
  unfolding connected_component_def
  by (metis (no_types, lifting) assms subsetD subsetI convex_contains_segment convex_segment(1)
  ends_in_segment connected_convex_1_gen)

```

```

lemma connected_component_1:
  fixes  $S :: \text{real set}$ 
  shows  $\text{connected\_component } S\ a\ b \longleftrightarrow \text{closed\_segment } a\ b \subseteq S$ 
  by (simp add: connected_component_1_gen)

```



```

lemma convex_affine_rel_interior_Int:
  fixes  $S\ T :: 'n::euclidean\_space\ set$ 
  assumes convex  $S$ 
    and affine  $T$ 
    and  $rel\_interior\ S \cap T \neq \{\}$ 
  shows  $rel\_interior\ (S \cap T) = rel\_interior\ S \cap T$ 
  by (simp add: affine_imp_convex assms convex_rel_interior_inter_two rel_interior_affine)

lemma convex_affine_rel_frontier_Int:
  fixes  $S\ T :: 'n::euclidean\_space\ set$ 
  assumes convex  $S$ 
    and affine  $T$ 
    and  $interior\ S \cap T \neq \{\}$ 
  shows  $rel\_frontier(S \cap T) = frontier\ S \cap T$ 
using assms
  unfolding rel_frontier_def frontier_def
  using convex_affine_closure_Int convex_affine_rel_interior_Int rel_interior_nonempty_interior
  by fastforce

lemma rel_interior_convex_Int_affine:
  fixes  $S :: 'a::euclidean\_space\ set$ 
  assumes convex  $S$  affine  $T$   $interior\ S \cap T \neq \{\}$ 
  shows  $rel\_interior(S \cap T) = interior\ S \cap T$ 
  by (metis Int_emptyI assms convex_affine_rel_interior_Int empty_iff interior_rel_interior_gen)

lemma subset_rel_interior_convex:
  fixes  $S\ T :: 'n::euclidean\_space\ set$ 
  assumes convex  $S$ 
    and convex  $T$ 
    and  $S \leq closure\ T$ 
    and  $\neg S \subseteq rel\_frontier\ T$ 
  shows  $rel\_interior\ S \subseteq rel\_interior\ T$ 
proof -
  have *:  $S \cap closure\ T = S$ 
    using assms by auto
  have  $\neg rel\_interior\ S \subseteq rel\_frontier\ T$ 
    using closure_mono[of  $rel\_interior\ S\ rel\_frontier\ T$ ] closed_rel_frontier[of  $T$ ]
    closure_closed[of  $S$ ] convex_closure_rel_interior[of  $S$ ] closure_subset[of  $S$ ]
  assms
  by auto
  then have  $rel\_interior\ S \cap rel\_interior\ (closure\ T) \neq \{\}$ 
    using assms rel_frontier_def[of  $T$ ] rel_interior_subset convex_rel_interior_closure[of  $T$ ]
  by auto
  then have  $rel\_interior\ S \cap rel\_interior\ T = rel\_interior\ (S \cap closure\ T)$ 
    using assms convex_closure_convex_rel_interior_inter_two[of  $S\ closure\ T$ ]
    convex_rel_interior_closure[of  $T$ ]

```

```

    by auto
  also have ... = rel_interior S
    using * by auto
  finally show ?thesis
    by auto
qed

lemma rel_interior_convex_linear_image:
  fixes f :: 'm::euclidean_space  $\Rightarrow$  'n::euclidean_space
  assumes linear f
    and convex S
  shows f ' (rel_interior S) = rel_interior (f ' S)
proof (cases S = {})
  case True
  then show ?thesis
    using assms by auto
next
  case False
  interpret linear f by fact
  have *: f ' (rel_interior S)  $\subseteq$  f ' S
    unfolding image_mono using rel_interior_subset by auto
  have f ' S  $\subseteq$  f ' (closure S)
    unfolding image_mono using closure_subset by auto
  also have ... = f ' (closure (rel_interior S))
    using convex_closure_rel_interior assms by auto
  also have ...  $\subseteq$  closure (f ' (rel_interior S))
    using closure_linear_image_subset assms by auto
  finally have closure (f ' S) = closure (f ' rel_interior S)
    using closure_mono[of f ' S closure (f ' rel_interior S)] closure_closure
      closure_mono[of f ' rel_interior S f ' S] *
    by auto
  then have rel_interior (f ' S) = rel_interior (f ' rel_interior S)
    using assms convex_rel_interior
      linear_conv_bounded_linear[of f] convex_linear_image[of _ S]
      convex_linear_image[of _ rel_interior S]
      closure_eq_rel_interior_eq[of f ' S f ' rel_interior S]
    by auto
  then have rel_interior (f ' S)  $\subseteq$  f ' rel_interior S
    using rel_interior_subset by auto
  moreover
  {
    fix z
    assume z  $\in$  f ' rel_interior S
    then obtain z1 where z1: z1  $\in$  rel_interior S f z1 = z by auto
    {
      fix x
      assume x  $\in$  f ' S
      then obtain x1 where x1: x1  $\in$  S f x1 = x by auto
      then obtain e where e: e > 1 (1 - e) *R x1 + e *R z1  $\in$  S
    }
  }

```

```

    using convex_rel_interior_iff[of S z1] ⟨convex S⟩ x1 z1 by auto
  moreover have f ((1 - e) *R x1 + e *R z1) = (1 - e) *R x + e *R z
    using x1 z1 by (simp add: linear_add linear_scale ⟨linear f⟩)
  ultimately have (1 - e) *R x + e *R z ∈ f-1 S
    using imageI[of (1 - e) *R x1 + e *R z1 S f] by auto
  then have ∃ e. e > 1 ∧ (1 - e) *R x + e *R z ∈ f-1 S
    using e by auto
}
then have z ∈ rel_interior (f-1 S)
  using convex_rel_interior_iff[of f-1 S z] ⟨convex S⟩ ⟨linear f⟩
  ⟨S ≠ {}⟩ convex_linear_image[of f S] linear_conv_bounded_linear[of f]
  by auto
}
ultimately show ?thesis by auto
qed

```

```

lemma rel_interior_convex_linear_preimage:
  fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
  assumes linear f
  and convex S
  and f-1 (rel_interior S) ≠ {}
  shows rel_interior (f-1 S) = f-1 (rel_interior S)
proof -
  interpret linear f by fact
  have S ≠ {}
    using assms by auto
  have nonemp: f-1 S ≠ {}
    by (metis assms(3) rel_interior_subset subset_empty vimage_mono)
  then have S ∩ (range f) ≠ {}
    by auto
  have conv: convex (f-1 S)
    using convex_linear_vimage assms by auto
  then have convex (S ∩ range f)
    by (simp add: assms(2) convex_Int convex_linear_image linear_axioms)
  {
    fix z
    assume z ∈ f-1 (rel_interior S)
    then have z: f z ∈ rel_interior S
      by auto
    {
      fix x
      assume x ∈ f-1 S
      then have f x ∈ S by auto
      then obtain e where e: e > 1 (1 - e) *R f x + e *R f z ∈ S
        using convex_rel_interior_iff[of S f z] z assms ⟨S ≠ {}⟩ by auto
      moreover have (1 - e) *R f x + e *R f z = f ((1 - e) *R x + e *R z)
        using ⟨linear f⟩ by (simp add: linear_iff)
      ultimately have ∃ e. e > 1 ∧ (1 - e) *R x + e *R z ∈ f-1 S
        using e by auto
    }
  }

```

```

    }
    then have  $z \in \text{rel\_interior } (f - ' S)$ 
      using  $\text{convex\_rel\_interior\_iff}[of f - ' S z] \text{ conv nonemp}$  by auto
  }
  moreover
  {
    fix  $z$ 
    assume  $z: z \in \text{rel\_interior } (f - ' S)$ 
    {
      fix  $x$ 
      assume  $x \in S \cap \text{range } f$ 
      then obtain  $y$  where  $y: f y = x$   $y \in f - ' S$  by auto
      then obtain  $e$  where  $e: e > 1 \ (1 - e) * _R y + e * _R z \in f - ' S$ 
        using  $\text{convex\_rel\_interior\_iff}[of f - ' S z] z \text{ conv}$  by auto
      moreover have  $(1 - e) * _R x + e * _R f z = f ((1 - e) * _R y + e * _R z)$ 
        using  $\langle \text{linear } f \rangle y$  by (simp add: linear_iff)
      ultimately have  $\exists e. e > 1 \wedge (1 - e) * _R x + e * _R f z \in S \cap \text{range } f$ 
        using  $e$  by auto
    }
    then have  $f z \in \text{rel\_interior } (S \cap \text{range } f)$ 
      using  $\langle \text{convex } (S \cap (\text{range } f)) \rangle \langle S \cap \text{range } f \neq \{\} \rangle$ 
         $\text{convex\_rel\_interior\_iff}[of S \cap (\text{range } f) f z]$ 
      by auto
    moreover have  $\text{affine } (\text{range } f)$ 
      by (simp add: linear_axioms linear_subspace_image subspace_imp_affine)
    ultimately have  $f z \in \text{rel\_interior } S$ 
      using  $\text{convex\_affine\_rel\_interior\_Int}[of S \text{ range } f] \text{ assms}$  by auto
    then have  $z \in f - ' (\text{rel\_interior } S)$ 
      by auto
  }
  ultimately show ?thesis by auto
qed

```

lemma $\text{rel\_interior\_Times}$:

```

  fixes  $S :: 'n::\text{euclidean\_space}$  set
    and  $T :: 'm::\text{euclidean\_space}$  set
  assumes  $\text{convex } S$ 
    and  $\text{convex } T$ 
  shows  $\text{rel\_interior } (S \times T) = \text{rel\_interior } S \times \text{rel\_interior } T$ 
proof (cases  $S = \{\} \vee T = \{\}$ )
case True
  then show ?thesis
    by auto
next
case False
  then have  $S \neq \{\} \ T \neq \{\}$ 
    by auto
  then have  $ri: \text{rel\_interior } S \neq \{\} \ \text{rel\_interior } T \neq \{\}$ 
    using  $\text{rel\_interior\_eq\_empty}$  assms by auto

```

```

then have fst - ' rel_interior S ≠ {}
using fst_vimage_eq_Times[of rel_interior S] by auto
then have rel_interior ((fst :: 'n * 'm ⇒ 'n) - ' S) = fst - ' rel_interior S
using linear_fst ⟨convex S⟩ rel_interior_convex_linear_preimage[of fst S] by
auto
then have s: rel_interior (S × (UNIV :: 'm set)) = rel_interior S × UNIV
by (simp add: fst_vimage_eq_Times)
from ri have snd - ' rel_interior T ≠ {}
using snd_vimage_eq_Times[of rel_interior T] by auto
then have rel_interior ((snd :: 'n * 'm ⇒ 'm) - ' T) = snd - ' rel_interior T
using linear_snd ⟨convex T⟩ rel_interior_convex_linear_preimage[of snd T]
by auto
then have t: rel_interior ((UNIV :: 'n set) × T) = UNIV × rel_interior T
by (simp add: snd_vimage_eq_Times)
from s t have *: rel_interior (S × (UNIV :: 'm set)) ∩ rel_interior ((UNIV ::
'n set) × T) =
  rel_interior S × rel_interior T by auto
have S × T = S × (UNIV :: 'm set) ∩ (UNIV :: 'n set) × T
by auto
then have rel_interior (S × T) = rel_interior ((S × (UNIV :: 'm set)) ∩
((UNIV :: 'n set) × T))
by auto
also have ... = rel_interior (S × (UNIV :: 'm set)) ∩ rel_interior ((UNIV ::
'n set) × T)
using * ri assms convex_Times
by (subst convex_rel_interior_inter_two) auto
finally show ?thesis using * by auto
qed

```

```

lemma rel_interior_scaleR:
fixes S :: 'n::euclidean_space set
assumes c ≠ 0
shows ((*R) c) - ' (rel_interior S) = rel_interior (((*R) c) - ' S)
using rel_interior_injective_linear_image[of ((*R) c) S]
  linear_conv_bounded_linear[of (*R) c] linear_scaleR injective_scaleR[of c]
assms
by auto

```

```

lemma rel_interior_convex_scaleR:
fixes S :: 'n::euclidean_space set
assumes convex S
shows ((*R) c) - ' (rel_interior S) = rel_interior (((*R) c) - ' S)
by (metis assms linear_scaleR rel_interior_convex_linear_image)

```

```

lemma convex_rel_open_scaleR:
fixes S :: 'n::euclidean_space set
assumes convex S
and rel_open S
shows convex ((*R) c) - ' S ∧ rel_open (((*R) c) - ' S)

```

by (metis assms convex_scaling rel_interior_convex_scaleR rel_open_def)

lemma convex_rel_open_finite_inter:

assumes $\forall S \in I. \text{convex } (S :: 'n::\text{euclidean\_space set}) \wedge \text{rel\_open } S$
and finite I

shows $\text{convex } (\bigcap I) \wedge \text{rel\_open } (\bigcap I)$

proof (cases $\bigcap \{\text{rel\_interior } S \mid S. S \in I\} = \{\}$)

case True

then have $\bigcap I = \{\}$

using assms **unfolding** rel_open_def **by** auto

then show ?thesis

unfolding rel_open_def **by** auto

next

case False

then have $\text{rel\_open } (\bigcap I)$

using assms **unfolding** rel_open_def

using convex_rel_interior_finite_inter[of I]

by auto

then show ?thesis

using convex_Inter assms **by** auto

qed

lemma convex_rel_open_linear_image:

fixes $f :: 'm::\text{euclidean\_space} \Rightarrow 'n::\text{euclidean\_space}$

assumes linear f

and convex S

and rel_open S

shows $\text{convex } (f \text{ ` } S) \wedge \text{rel\_open } (f \text{ ` } S)$

by (metis assms convex_linear_image rel_interior_convex_linear_image rel_open_def)

lemma convex_rel_open_linear_preimage:

fixes $f :: 'm::\text{euclidean\_space} \Rightarrow 'n::\text{euclidean\_space}$

assumes linear f

and convex S

and rel_open S

shows $\text{convex } (f \text{ -` } S) \wedge \text{rel\_open } (f \text{ -` } S)$

proof (cases $f \text{ -` } (\text{rel\_interior } S) = \{\}$)

case True

then have $f \text{ -` } S = \{\}$

using assms **unfolding** rel_open_def **by** auto

then show ?thesis

unfolding rel_open_def **by** auto

next

case False

then have $\text{rel\_open } (f \text{ -` } S)$

using assms **unfolding** rel_open_def

using rel_interior_convex_linear_preimage[of f S]

by auto

then show ?thesis

```

    using convex_linear_image assms
    by auto
qed

lemma rel_interior_projection:
  fixes S :: ('m::euclidean_space × 'n::euclidean_space) set
  and f :: 'm::euclidean_space ⇒ 'n::euclidean_space set
  assumes convex S
  and f = (λy. {z. (y, z) ∈ S})
  shows (y, z) ∈ rel_interior S ⟷ (y ∈ rel_interior {y. (f y ≠ {})}) ∧ z ∈
rel_interior (f y))
proof -
  {
    fix y
    assume y ∈ {y. f y ≠ {}}
    then obtain z where (y, z) ∈ S
      using assms by auto
    then have ∃ x. x ∈ S ∧ y = fst x
      by auto
    then obtain x where x ∈ S ∧ y = fst x
      by blast
    then have y ∈ fst `S
      unfolding image_def by auto
  }
  then have fst `S = {y. f y ≠ {}}
    unfolding fst_def using assms by auto
  then have h1: fst `rel_interior S = rel_interior {y. f y ≠ {}}
    using rel_interior_convex_linear_image[of fst S] assms linear_fst by auto
  {
    fix y
    assume y ∈ rel_interior {y. f y ≠ {}}
    then have y ∈ fst `rel_interior S
      using h1 by auto
    then have *: rel_interior S ∩ fst -` {y} ≠ {}
      by auto
    moreover have aff: affine (fst -` {y})
      unfolding affine_alt by (simp add: algebra_simps)
    ultimately have **: rel_interior (S ∩ fst -` {y}) = rel_interior S ∩ fst -`
{y}
      using convex_affine_rel_interior_Int[of S fst -` {y}] assms by auto
    have conv: convex (S ∩ fst -` {y})
      using convex_Int assms aff affine_imp_convex by auto
    {
      fix x
      assume x ∈ f y
      then have (y, x) ∈ S ∩ (fst -` {y})
        using assms by auto
      moreover have x = snd (y, x) by auto
      ultimately have x ∈ snd `(S ∩ fst -` {y})

```

```

      by blast
    }
  then have snd ' (S ∩ fst - ' {y}) = f y
    using assms by auto
  then have ***: rel_interior (f y) = snd ' rel_interior (S ∩ fst - ' {y})
    using rel_interior_convex_linear_image[of snd S ∩ fst - ' {y}] linear_snd
conv
  by auto
  {
    fix z
    assume z ∈ rel_interior (f y)
    then have z ∈ snd ' rel_interior (S ∩ fst - ' {y})
      using *** by auto
    moreover have {y} = fst ' rel_interior (S ∩ fst - ' {y})
      using * ** rel_interior_subset by auto
    ultimately have (y, z) ∈ rel_interior (S ∩ fst - ' {y})
      by force
    then have (y, z) ∈ rel_interior S
      using ** by auto
  }
  moreover
  {
    fix z
    assume (y, z) ∈ rel_interior S
    then have (y, z) ∈ rel_interior (S ∩ fst - ' {y})
      using ** by auto
    then have z ∈ snd ' rel_interior (S ∩ fst - ' {y})
      by (metis Range_iff snd_eq_Range)
    then have z ∈ rel_interior (f y)
      using *** by auto
  }
  ultimately have ∧z. (y, z) ∈ rel_interior S ⟷ z ∈ rel_interior (f y)
    by auto
}
then have h2: ∧y z. y ∈ rel_interior {t. f t ≠ {}} ⟹
  (y, z) ∈ rel_interior S ⟷ z ∈ rel_interior (f y)
  by auto
{
  fix y z
  assume asm: (y, z) ∈ rel_interior S
  then have y ∈ fst ' rel_interior S
    by (metis Domain_iff fst_eq_Domain)
  then have y ∈ rel_interior {t. f t ≠ {}}
    using h1 by auto
  then have y ∈ rel_interior {t. f t ≠ {}} and (z ∈ rel_interior (f y))
    using h2 asm by auto
}
then show ?thesis using h2 by blast
qed

```



```

lemma rel_frontier_Times:
  fixes S :: 'n::euclidean_space set
    and T :: 'm::euclidean_space set
  assumes convex S
    and convex T
  shows rel_frontier S  $\times$  rel_frontier T  $\subseteq$  rel_frontier (S  $\times$  T)
    by (force simp: rel_frontier_def rel_interior_Times assms closure_Times)

```

Relative interior of convex cone

```

lemma cone_rel_interior:
  fixes S :: 'm::euclidean_space set
  assumes cone S
  shows cone ({0}  $\cup$  rel_interior S)
proof (cases S = {})
  case True
  then show ?thesis
    by (simp add: cone_0)
next
  case False
  then have *: 0  $\in$  S  $\wedge$  ( $\forall$  c. c > 0  $\longrightarrow$  (*R) c ' S = S)
    using cone_iff[of S] assms by auto
  then have *: 0  $\in$  ({0}  $\cup$  rel_interior S)
    and  $\forall$  c. c > 0  $\longrightarrow$  (*R) c ' ({0}  $\cup$  rel_interior S) = ({0}  $\cup$  rel_interior S)
    by (auto simp add: rel_interior_scaleR)
  then show ?thesis
    using cone_iff[of {0}  $\cup$  rel_interior S] by auto
qed

```

```

lemma rel_interior_convex_cone_aux:
  fixes S :: 'm::euclidean_space set
  assumes convex S
  shows (c, x)  $\in$  rel_interior (cone hull ({1 :: real}  $\times$  S))  $\longleftrightarrow$ 
    c > 0  $\wedge$  x  $\in$  ((*R) c) ' (rel_interior S)
proof (cases S = {})
  case True
  then show ?thesis
    by (simp add: cone_hull_empty)
next
  case False
  then obtain s where s  $\in$  S by auto
  have conv: convex ({1 :: real}  $\times$  S)
    using convex_Times[of {1 :: real} S] assms convex_singleton[of 1 :: real]
    by auto
  define f where f y = {z. (y, z)  $\in$  cone hull ({1 :: real}  $\times$  S)} for y
  then have *: (c, x)  $\in$  rel_interior (cone hull ({1 :: real}  $\times$  S)) =
    (c  $\in$  rel_interior {y. f y  $\neq$  {}}  $\wedge$  x  $\in$  rel_interior (f c))
    using convex_cone_hull[of {1 :: real}  $\times$  S] conv

```

```

    by (subst rel_interior_projection) auto
  {
    fix y :: real
    assume y ≥ 0
    then have y *R (1,s) ∈ cone hull ({1 :: real} × S)
      using cone_hull_expl[of {(1 :: real)} × S] ‹s ∈ S› by auto
    then have f y ≠ {}
      using f_def by auto
  }
  then have {y. f y ≠ {}} = {0..}
    using f_def cone_hull_expl[of {1 :: real} × S] by auto
  then have **: rel_interior {y. f y ≠ {}} = {0<..}
    using rel_interior_real_semiline by auto
  {
    fix c :: real
    assume c > 0
    then have f c = ((*R) c ‘ S)
      using f_def cone_hull_expl[of {1 :: real} × S] by auto
    then have rel_interior (f c) = ((*R) c ‘ rel_interior S)
      using rel_interior_convex_scaleR[of S c] assms by auto
  }
  then show ?thesis using * ** by auto
qed

lemma rel_interior_convex_cone:
  fixes S :: 'm::euclidean_space set
  assumes convex S
  shows rel_interior (cone hull ({1 :: real} × S)) =
    {(c, c *R x) | c x. c > 0 ∧ x ∈ rel_interior S}
  (is ?lhs = ?rhs)
proof -
  {
    fix z
    assume z ∈ ?lhs
    have *: z = (fst z, snd z)
      by auto
    then have z ∈ ?rhs
      using rel_interior_convex_cone_aux[of S fst z snd z] assms ‹z ∈ ?lhs› by
    fastforce
  }
  moreover
  {
    fix z
    assume z ∈ ?rhs
    then have z ∈ ?lhs
      using rel_interior_convex_cone_aux[of S fst z snd z] assms
      by auto
  }
  ultimately show ?thesis by blast

```

qed

lemma *convex_hull_finite_union*:

assumes *finite I*

assumes $\forall i \in I. \text{convex } (S\ i) \wedge (S\ i) \neq \{\}$

shows $\text{convex\_hull } (\bigcup (S\ 'I)) =$

$\{ \text{sum } (\lambda i. c\ i *_{\mathbb{R}} s\ i)\ I \mid c\ s. (\forall i \in I. c\ i \geq 0) \wedge \text{sum } c\ I = 1 \wedge (\forall i \in I. s\ i \in S\ i) \}$

(is ?lhs = ?rhs)

proof –

have $?lhs \supseteq ?rhs$

proof

fix *x*

assume $x \in ?rhs$

then obtain *c s* **where** $\ast: \text{sum } (\lambda i. c\ i *_{\mathbb{R}} s\ i)\ I = x \text{ sum } c\ I = 1$

$(\forall i \in I. c\ i \geq 0) \wedge (\forall i \in I. s\ i \in S\ i)$ **by** *auto*

then have $\forall i \in I. s\ i \in \text{convex\_hull } (\bigcup (S\ 'I))$

using *hull_subset[of $\bigcup (S\ 'I)$ convex]* **by** *auto*

then show $x \in ?lhs$

unfolding $\ast(1)[\text{symmetric}]$

using $\ast$ *assms convex_convex_hull*

by *(subst convex_sum) auto*

qed

{

fix *i*

assume $i \in I$

with *assms* **have** $\exists p. p \in S\ i$ **by** *auto*

}

then obtain *p* **where** $p: \forall i \in I. p\ i \in S\ i$ **by** *metis*

{

fix *i*

assume $i \in I$

{

fix *x*

assume $x \in S\ i$

define *c* **where** $c\ j = (\text{if } j = i \text{ then } 1::\text{real} \text{ else } 0)$ **for** *j*

then have $\ast: \text{sum } c\ I = 1$

using $\langle \text{finite } I \rangle \langle i \in I \rangle \text{sum.delta}[of\ I\ i\ \lambda j::'a. 1::\text{real}]$

by *auto*

define *s* **where** $s\ j = (\text{if } j = i \text{ then } x \text{ else } p\ j)$ **for** *j*

then have $\forall j. c\ j *_{\mathbb{R}} s\ j = (\text{if } j = i \text{ then } x \text{ else } 0)$

using *c_def* **by** *(auto simp add: algebra_simps)*

then have $x = \text{sum } (\lambda i. c\ i *_{\mathbb{R}} s\ i)\ I$

using *s_def c_def* $\langle \text{finite } I \rangle \langle i \in I \rangle \text{sum.delta}[of\ I\ i\ \lambda j::'a. x]$

by *auto*

moreover have $(\forall i \in I. 0 \leq c\ i) \wedge \text{sum } c\ I = 1 \wedge (\forall i \in I. s\ i \in S\ i)$

using $\ast$ *c_def s_def* *p* $\langle x \in S\ i \rangle$ **by** *auto*

ultimately have $x \in ?rhs$

by *force*

```

    }
    then have ?rhs  $\supseteq$   $S\ i$  by auto
  }
  then have *: ?rhs  $\supseteq \bigcup (S\ 'I)$  by auto

  {
    fix u v :: real
    assume uv:  $u \geq 0 \wedge v \geq 0 \wedge u + v = 1$ 
    fix x y
    assume xy:  $x \in ?rhs \wedge y \in ?rhs$ 
    from xy obtain c s where
      xc:  $x = \text{sum } (\lambda i. c\ i *_{\mathbb{R}} s\ i)\ I \wedge (\forall i \in I. c\ i \geq 0) \wedge \text{sum } c\ I = 1 \wedge (\forall i \in I. s$ 
 $i \in S\ i)$ 
      by auto
    from xy obtain d t where
      yc:  $y = \text{sum } (\lambda i. d\ i *_{\mathbb{R}} t\ i)\ I \wedge (\forall i \in I. d\ i \geq 0) \wedge \text{sum } d\ I = 1 \wedge (\forall i \in I. t$ 
 $i \in S\ i)$ 
      by auto
    define e where  $e\ i = u * c\ i + v * d\ i$  for i
    have ge0:  $\forall i \in I. e\ i \geq 0$ 
      using e_def xc yc uv by simp
    have sum ( $\lambda i. u * c\ i$ )  $I = u * \text{sum } c\ I$ 
      by (simp add: sum_distrib_left)
    moreover have  $\text{sum } (\lambda i. v * d\ i)\ I = v * \text{sum } d\ I$ 
      by (simp add: sum_distrib_left)
    ultimately have sum1:  $\text{sum } e\ I = 1$ 
      using e_def xc yc uv by (simp add: sum.distrib)
    define q where  $q\ i = (\text{if } e\ i = 0 \text{ then } p\ i \text{ else } (u * c\ i / e\ i) *_{\mathbb{R}} s\ i + (v * d\ i$ 
 $/ e\ i) *_{\mathbb{R}} t\ i)$ 
      for i
    {
      fix i
      assume i:  $i \in I$ 
      have q i  $\in S\ i$ 
      proof (cases  $e\ i = 0$ )
        case True
          then show ?thesis using i p q_def by auto
        next
          case False
            then show ?thesis
              using mem_convex_alt[of  $S\ i\ s\ i\ t\ i\ u * (c\ i)\ v * (d\ i)$ ]
                mult_nonneg_nonneg[of  $u\ c\ i$ ] mult_nonneg_nonneg[of  $v\ d\ i$ ]
                  assms q_def e_def i False xc yc uv
                by (auto simp del: mult_nonneg_nonneg)
      qed
    }
  }
  then have qs:  $\forall i \in I. q\ i \in S\ i$  by auto
  {
    fix i

```

```

    assume  $i: i \in I$ 
    have  $(u * c \ i) *_R s \ i + (v * d \ i) *_R t \ i = e \ i *_R q \ i$ 
    proof (cases  $e \ i = 0$ )
      case True
        have  $ge: u * (c \ i) \geq 0 \wedge v * d \ i \geq 0$ 
          using  $xc \ yc \ uv \ i$  by simp
        moreover from  $ge$  have  $u * c \ i \leq 0 \wedge v * d \ i \leq 0$ 
          using True  $e\_def \ i$  by simp
        ultimately have  $u * c \ i = 0 \wedge v * d \ i = 0$  by auto
        with True show ?thesis by auto
      next
      case False
        then have  $(u * (c \ i)/(e \ i)) *_R (s \ i) + (v * (d \ i)/(e \ i)) *_R (t \ i) = q \ i$ 
          using  $q\_def$  by auto
        then have  $e \ i *_R ((u * (c \ i)/(e \ i)) *_R (s \ i) + (v * (d \ i)/(e \ i)) *_R (t \ i))$ 
           $= (e \ i) *_R (q \ i)$  by auto
        with False show ?thesis by (simp add: algebra_simps)
    qed
  }
  then have  $*: \forall i \in I. (u * c \ i) *_R s \ i + (v * d \ i) *_R t \ i = e \ i *_R q \ i$ 
    by auto
  have  $u *_R x + v *_R y = \text{sum } (\lambda i. (u * c \ i) *_R s \ i + (v * d \ i) *_R t \ i) \ I$ 
    using  $xc \ yc$  by (simp add: algebra_simps scaleR_right.sum sum.distrib)
  also have  $\dots = \text{sum } (\lambda i. e \ i *_R q \ i) \ I$ 
    using  $*$  by auto
  finally have  $u *_R x + v *_R y = \text{sum } (\lambda i. (e \ i) *_R (q \ i)) \ I$ 
    by auto
  then have  $u *_R x + v *_R y \in ?rhs$ 
    using  $ge0 \ sum1 \ qs$  by auto
}
then have  $\text{convex } ?rhs$  unfolding  $\text{convex\_def}$  by auto
then show ?thesis
  using  $\langle ?lhs \supseteq ?rhs \rangle * \text{hull\_minimal}[of \bigcup (S \text{ ' } I) \ ?rhs \ \text{convex}]$ 
  by blast
qed

```

lemma $\text{convex\_hull\_union\_two}$:

fixes $S \ T :: 'm::\text{euclidean\_space} \ \text{set}$

assumes $\text{convex } S$

and $S \neq \{\}$

and $\text{convex } T$

and $T \neq \{\}$

shows $\text{convex } \text{hull } (S \cup T) =$

$\{u *_R s + v *_R t \mid u \ v \ s \ t. u \geq 0 \wedge v \geq 0 \wedge u + v = 1 \wedge s \in S \wedge t \in T\}$

(is $?lhs = ?rhs$)

proof

define $I :: \text{nat set}$ where $I = \{1, 2\}$

define s where $s \ i = (\text{if } i = (1::\text{nat}) \text{ then } S \text{ else } T)$ for i

have $\bigcup (s \text{ ' } I) = S \cup T$

```

    using s_def I_def by auto
  then have convex_hull ( $\bigcup (s \text{ ' } I)$ ) = convex_hull ( $S \cup T$ )
    by auto
  moreover have convex_hull  $\bigcup (s \text{ ' } I) =$ 
    { $\sum i \in I. c \ i \ *_R \ sa \ i \mid c \ sa. (\forall i \in I. 0 \leq c \ i) \wedge \text{sum } c \ I = 1 \wedge (\forall i \in I. sa \ i \in s \ i)$ }
    using assms s_def I_def
    by (subst convex_hull_finite_union) auto
  moreover have
    { $\sum i \in I. c \ i \ *_R \ sa \ i \mid c \ sa. (\forall i \in I. 0 \leq c \ i) \wedge \text{sum } c \ I = 1 \wedge (\forall i \in I. sa \ i \in s \ i)$ }  $\leq ?rhs$ 
    using s_def I_def by auto
  ultimately show  $?lhs \subseteq ?rhs$  by auto
  {
    fix x
    assume  $x \in ?rhs$ 
    then obtain  $u \ v \ s \ t$  where  $*$ :  $x = u \ *_R \ s + v \ *_R \ t \wedge u \geq 0 \wedge v \geq 0 \wedge u + v = 1 \wedge s \in S \wedge t \in T$ 
    by auto
    then have  $x \in \text{convex\_hull } \{s, t\}$ 
    using convex_hull_2[of s t] by auto
    then have  $x \in \text{convex\_hull } (S \cup T)$ 
    using * hull_mono[of {s, t}  $S \cup T$ ] by auto
  }
  then show  $?lhs \supseteq ?rhs$  by blast
qed

```

proposition ray_to_rel_frontier:

fixes $a :: 'a::\text{real\_inner}$

assumes bounded S

and $a: a \in \text{rel\_interior } S$

and $\text{aff}: (a + l) \in \text{affine hull } S$

and $l \neq 0$

obtains d where $0 < d \wedge (a + d \ *_R \ l) \in \text{rel\_frontier } S$

$\wedge e. \llbracket 0 \leq e; e < d \rrbracket \implies (a + e \ *_R \ l) \in \text{rel\_interior } S$

proof –

have $\text{aaff}: a \in \text{affine hull } S$

by (meson a hull_subset rel_interior_subset rev_subsetD)

let $?D = \{d. 0 < d \wedge a + d \ *_R \ l \notin \text{rel\_interior } S\}$

obtain B where $B > 0$ and $B: S \subseteq \text{ball } a \ B$

using bounded_subset_ballD [OF bounded S] by blast

have $a + (B / \text{norm } l) \ *_R \ l \notin \text{ball } a \ B$

by (simp add: dist_norm l ≠ 0)

with B have $a + (B / \text{norm } l) \ *_R \ l \notin \text{rel\_interior } S$

using rel_interior_subset subsetCE by blast

with $\langle B > 0 \rangle \ \langle l \neq 0 \rangle$ have $\text{nonMT}: ?D \neq \{\}$

using divide_pos_pos zero_less_norm_iff by fastforce

have $\text{bdd}: \text{bdd\_below } ?D$

by (metis (no_types, lifting) bdd_belowI le_less mem_Collect_eq)

```

have relin_Ex:  $\bigwedge x. x \in \text{rel\_interior } S \implies$ 
 $\exists e > 0. \forall x' \in \text{affine hull } S. \text{dist } x' x < e \implies x' \in \text{rel\_interior } S$ 
using openin_rel_interior [of S] by (simp add: openin_euclidean_subtopology_iff)
define d where d = Inf ?D
obtain  $\varepsilon$  where  $0 < \varepsilon$  and  $\varepsilon: \bigwedge \eta. \llbracket 0 \leq \eta; \eta < \varepsilon \rrbracket \implies (a + \eta *_R l) \in \text{rel\_interior}$ 
S
proof -
  obtain e where e > 0
    and e:  $\bigwedge x'. x' \in \text{affine hull } S \implies \text{dist } x' a < e \implies x' \in \text{rel\_interior } S$ 
    using relin_Ex a by blast
  show thesis
  proof (rule_tac  $\varepsilon = e / \text{norm } l$  in that)
    show  $0 < e / \text{norm } l$  by (simp add:  $\langle 0 < e \rangle \langle l \neq 0 \rangle$ )
  next
    show  $a + \eta *_R l \in \text{rel\_interior } S$  if  $0 \leq \eta$   $\eta < e / \text{norm } l$  for  $\eta$ 
    proof (rule e)
      show  $a + \eta *_R l \in \text{affine hull } S$ 
      by (metis (no_types) add_diff_cancel_left' aff_affine_affine_hull mem_affine_3_minus
aaff)
      show  $\text{dist } (a + \eta *_R l) a < e$ 
        using that by (simp add:  $\langle l \neq 0 \rangle \text{dist\_norm pos\_less\_divide\_eq}$ )
    qed
  qed
  qed
  have inint:  $\bigwedge e. \llbracket 0 \leq e; e < d \rrbracket \implies a + e *_R l \in \text{rel\_interior } S$ 
    unfolding d_def using cInf_lower [OF bdd]
    by (metis (no_types, lifting) a add.right_neutral le_less mem_Collect_eq
not_less real_vector.scale_zero_left)
  have  $\varepsilon \leq d$ 
    unfolding d_def
    using  $\varepsilon \text{dual\_order.strict\_implies\_order le\_less\_linear}$ 
    by (blast intro: cInf_greatest [OF nonMT])
  with  $\langle 0 < \varepsilon \rangle$  have  $0 < d$  by simp
  have  $a + d *_R l \notin \text{rel\_interior } S$ 
  proof
    assume adl:  $a + d *_R l \in \text{rel\_interior } S$ 
    obtain e where e > 0
      and e:  $\bigwedge x'. x' \in \text{affine hull } S \implies \text{dist } x' (a + d *_R l) < e \implies x' \in$ 
rel_interior S
      using relin_Ex adl by blast
    have  $d + e / \text{norm } l \leq \text{Inf } \{d. 0 < d \wedge a + d *_R l \notin \text{rel\_interior } S\}$ 
    proof (rule cInf_greatest [OF nonMT], clarsimp)
      fix x::real
      assume  $0 < x$  and nonrel:  $a + x *_R l \notin \text{rel\_interior } S$ 
      show  $d + e / \text{norm } l \leq x$ 
      proof (cases  $x < d$ )
        case True with inint nonrel  $\langle 0 < x \rangle$ 
          show ?thesis by auto
      next

```

```

    case False
    then have dle:  $x < d + e / \text{norm } l \implies \text{dist } (a + x *_R l) (a + d *_R l) < e$ 
    by (simp add: field_simps  $\langle l \neq 0 \rangle$ )
    have ain:  $a + x *_R l \in \text{affine hull } S$ 
    by (metis add_diff_cancel_left' aff affine_affine_hull mem_affine_3_minus
    aaff)
    show ?thesis
    using e [OF ain] nonrel dle by force
  qed
  qed
  then show False
  using  $\langle 0 < e \rangle \langle l \neq 0 \rangle$  by (simp add: d_def [symmetric] field_simps)
  qed
  moreover have  $a + d *_R l \in \text{closure } S$ 
  proof (clarsimp simp: closure_approachable)
    fix  $\eta :: \text{real}$  assume  $0 < \eta$ 
    have 1:  $a + (d - \min d (\eta / 2 / \text{norm } l)) *_R l \in S$ 
    proof (rule subsetD [OF rel_interior_subset inint])
      show  $d - \min d (\eta / 2 / \text{norm } l) < d$ 
      using  $\langle l \neq 0 \rangle \langle 0 < d \rangle \langle 0 < \eta \rangle$  by auto
    qed auto
    have  $\text{norm } l * \min d (\eta / (\text{norm } l * 2)) \leq \text{norm } l * (\eta / (\text{norm } l * 2))$ 
    by (metis min_def mult_left_mono norm_ge_zero order_refl)
    also have  $\dots < \eta$ 
    using  $\langle l \neq 0 \rangle \langle 0 < \eta \rangle$  by (simp add: field_simps)
    finally have 2:  $\text{norm } l * \min d (\eta / (\text{norm } l * 2)) < \eta$ 
    show  $\exists y \in S. \text{dist } y (a + d *_R l) < \eta$ 
    using 1 2  $\langle 0 < d \rangle \langle 0 < \eta \rangle$ 
    by (rule_tac  $x = a + (d - \min d (\eta / 2 / \text{norm } l)) *_R l$  in bexI) (auto simp:
    algebra_simps)
  qed
  ultimately have infront:  $a + d *_R l \in \text{rel\_frontier } S$ 
  by (simp add: rel_frontier_def)
  show ?thesis
  by (rule that [OF  $\langle 0 < d \rangle$  infront inint])
  qed

corollary ray_to_frontier:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  assumes bounded S
  and  $a \in \text{interior } S$ 
  and  $l \neq 0$ 
  obtains  $d$  where  $0 < d$   $(a + d *_R l) \in \text{frontier } S$ 
   $\bigwedge e. [0 \leq e; e < d] \implies (a + e *_R l) \in \text{interior } S$ 
proof -
  have §:  $\text{interior } S = \text{rel\_interior } S$ 
  using a rel_interior_nonempty_interior by auto
  then have  $a \in \text{rel\_interior } S$ 

```



```

    using a by simp
  moreover have  $a + l \in \text{affine hull } S$ 
    using a affine_hull_nonempty_interior by blast
  ultimately show thesis
    by (metis § ⟨bounded S⟩ ⟨ $l \neq 0$ ⟩ frontier_def ray_to_rel_frontier rel_frontier_def
that)
qed

```

lemma segment_to_rel_frontier_aux:

```

  fixes x :: 'a::euclidean_space
  assumes convex S bounded S and x:  $x \in \text{rel\_interior } S$  and y:  $y \in S$  and xy:
 $x \neq y$ 
  obtains z where  $z \in \text{rel\_frontier } S$   $y \in \text{closed\_segment } x z$ 
    open_segment  $x z \subseteq \text{rel\_interior } S$ 

```

proof –

```

  have  $x + (y - x) \in \text{affine hull } S$ 
    using hull_inc [OF y] by auto
  then obtain d where  $0 < d$  and df:  $(x + d *_R (y - x)) \in \text{rel\_frontier } S$ 
    and di:  $\bigwedge e. [0 \leq e; e < d] \implies (x + e *_R (y - x)) \in \text{rel\_interior } S$ 
    by (rule ray_to_rel_frontier [OF ⟨bounded S⟩ x]) (use xy in auto)
  show ?thesis

```

proof

```

  show  $x + d *_R (y - x) \in \text{rel\_frontier } S$ 
    by (simp add: df)
  next
    have open_segment  $x y \subseteq \text{rel\_interior } S$ 
      using rel_interior_closure_convex_segment [OF ⟨convex S⟩ x] closure_subset
y by blast
    moreover have  $x + d *_R (y - x) \in \text{open\_segment } x y$  if  $d < 1$ 
      using xy ⟨ $0 < d$ ⟩ that by (force simp: in_segment algebra_simps)
    ultimately have  $1 \leq d$ 
      using df rel_frontier_def by fastforce
    moreover have  $x = (1 / d) *_R x + ((d - 1) / d) *_R x$ 
      by (metis ⟨ $0 < d$ ⟩ add commute add_divide_distrib diff_add_cancel di-
vide_self_if_less_irrefl scaleR_add_left scaleR_one)
    ultimately show  $y \in \text{closed\_segment } x (x + d *_R (y - x))$ 
      unfolding in_segment
      by (rule_tac  $x=1/d$  in exI) (auto simp: algebra_simps)
  next
    show open_segment  $x (x + d *_R (y - x)) \subseteq \text{rel\_interior } S$ 
  proof (rule rel_interior_closure_convex_segment [OF ⟨convex S⟩ x])
    show  $x + d *_R (y - x) \in \text{closure } S$ 
      using df rel_frontier_def by auto
  qed
qed
qed
qed

```

lemma segment_to_rel_frontier:

```

fixes  $x :: 'a::euclidean\_space$ 
assumes  $S$ : convex  $S$  bounded  $S$  and  $x$ :  $x \in rel\_interior\ S$ 
and  $y$ :  $y \in S$  and  $xy$ :  $\neg(x = y \wedge S = \{x\})$ 
obtains  $z$  where  $z \in rel\_frontier\ S$   $y \in closed\_segment\ x\ z$ 
open\_segment  $x\ z \subseteq rel\_interior\ S$ 
proof (cases  $x=y$ )
  case True
    with  $xy$  have  $S \neq \{x\}$ 
    by blast
    with True show ?thesis
    by (metis Set.set_insert all_not_in_conv ends_in_segment(1) insert_iff segment_to_rel_frontier_aux[OF S x] that y)
  next
    case False
    then show ?thesis
    using segment_to_rel_frontier_aux [OF S x y] that by blast
qed

proposition rel_frontier_not_sing:
  fixes  $a :: 'a::euclidean\_space$ 
  assumes bounded  $S$ 
  shows  $rel\_frontier\ S \neq \{a\}$ 
proof (cases  $S = \{\}$ )
  case True then show ?thesis by simp
next
  case False
  then obtain  $z$  where  $z \in S$ 
  by blast
  then show ?thesis
  proof (cases  $S = \{z\}$ )
    case True then show ?thesis by simp
  next
    case False
    then obtain  $w$  where  $w \in S$   $w \neq z$ 
    using  $\langle z \in S \rangle$  by blast
    show ?thesis
    proof
      assume  $rel\_frontier\ S = \{a\}$ 
      then consider  $w \notin rel\_frontier\ S \mid z \notin rel\_frontier\ S$ 
      using  $\langle w \neq z \rangle$  by auto
      then show False
    proof cases
      case 1
      then have  $w$ :  $w \in rel\_interior\ S$ 
      using  $\langle w \in S \rangle$  closure_subset rel_frontier_def by fastforce
      have  $w + (w - z) \in affine\ hull\ S$ 
      by (metis  $\langle w \in S \rangle \langle z \in S \rangle$  affine_affine_hull hull_inc mem_affine_3_minus scaleR_one)
      then obtain  $e$  where  $0 < e$   $(w + e *_R (w - z)) \in rel\_frontier\ S$ 

```

```

    using ⟨w ≠ z⟩ ⟨z ∈ S⟩ by (metis assms ray_to_rel_frontier right_minus_eq
w)
    moreover obtain d where 0 < d (w + d *R (z - w)) ∈ rel_frontier S
    using ray_to_rel_frontier [OF ⟨bounded S⟩ w, of 1 *R (z - w)] ⟨w ≠ z⟩
⟨z ∈ S⟩
    by (metis add commute add.right_neutral diff_add_cancel hull_inc
scaleR_one)
    ultimately have d *R (z - w) = e *R (w - z)
    using ⟨rel_frontier S = {a}⟩ by force
    moreover have e ≠ -d
    using ⟨0 < e⟩ ⟨0 < d⟩ by force
    ultimately show False
    by (metis (no_types, lifting) ⟨w ≠ z⟩ eq_iff_diff_eq_0 minus_diff_eq
real_vector.scale_cancel_right real_vector.scale_minus_right scaleR_left.minus)
next
case 2
then have z: z ∈ rel_interior S
    using ⟨z ∈ S⟩ closure_subset rel_frontier_def by fastforce
    have z + (z - w) ∈ affine hull S
    by (metis ⟨z ∈ S⟩ ⟨w ∈ S⟩ affine_affine_hull hull_inc mem_affine_3_minus
scaleR_one)
    then obtain e where 0 < e (z + e *R (z - w)) ∈ rel_frontier S
    using ⟨w ≠ z⟩ ⟨w ∈ S⟩ by (metis assms ray_to_rel_frontier right_minus_eq
z)
    moreover obtain d where 0 < d (z + d *R (w - z)) ∈ rel_frontier S
    using ray_to_rel_frontier [OF ⟨bounded S⟩ z, of 1 *R (w - z)] ⟨w ≠ z⟩
⟨w ∈ S⟩
    by (metis add commute add.right_neutral diff_add_cancel hull_inc
scaleR_one)
    ultimately have d *R (w - z) = e *R (z - w)
    using ⟨rel_frontier S = {a}⟩ by force
    moreover have e ≠ -d
    using ⟨0 < e⟩ ⟨0 < d⟩ by force
    ultimately show False
    by (metis (no_types, lifting) ⟨w ≠ z⟩ eq_iff_diff_eq_0 minus_diff_eq
real_vector.scale_cancel_right real_vector.scale_minus_right scaleR_left.minus)
qed
qed
qed
qed

```

5.0.5 Convexity on direct sums

lemma *closure_sum*:

fixes $S\ T :: 'a::\text{real\_normed\_vector}\ \text{set}$

shows $\text{closure } S + \text{closure } T \subseteq \text{closure } (S + T)$

unfolding *set_plus_image closure_Times [symmetric] split_def*

by (*intro closure_bounded_linear_image_subset bounded_linear_add*
bounded_linear_fst bounded_linear_snd)

```

lemma fst_snd_linear: linear ( $\lambda(x,y). x + y$ )
  unfolding linear_iff by (simp add: algebra_simps)

lemma rel_interior_sum:
  fixes  $S\ T :: 'n::euclidean\_space\ set$ 
  assumes convex S
  and convex T
  shows  $rel\_interior\ (S + T) = rel\_interior\ S + rel\_interior\ T$ 
proof -
  have  $rel\_interior\ S + rel\_interior\ T = (\lambda(x,y). x + y)\ ' (rel\_interior\ S \times rel\_interior\ T)$ 
  by (simp add: set_plus_image)
  also have  $\dots = (\lambda(x,y). x + y)\ ' rel\_interior\ (S \times T)$ 
  using rel_interior_Times assms by auto
  also have  $\dots = rel\_interior\ (S + T)$ 
  using fst_snd_linear convex_Times assms
    rel_interior_convex_linear_image[of ( $\lambda(x,y). x + y$ )  $S \times T$ ]
  by (auto simp add: set_plus_image)
  finally show ?thesis ..
qed

lemma rel_interior_sum_gen:
  fixes  $S :: 'a \Rightarrow 'n::euclidean\_space\ set$ 
  assumes  $\bigwedge i. i \in I \implies convex\ (S\ i)$ 
  shows  $rel\_interior\ (sum\ S\ I) = sum\ (\lambda i. rel\_interior\ (S\ i))\ I$ 
  using rel_interior_sum rel_interior_sing[of 0] assms
  by (subst sum_set_cond_linear[of convex], auto simp add: convex_set_plus)

lemma convex_rel_open_direct_sum:
  fixes  $S\ T :: 'n::euclidean\_space\ set$ 
  assumes convex S
  and rel_open S
  and convex T
  and rel_open T
  shows  $convex\ (S \times T) \wedge rel\_open\ (S \times T)$ 
  by (metis assms convex_Times rel_interior_Times rel_open_def)

lemma convex_rel_open_sum:
  fixes  $S\ T :: 'n::euclidean\_space\ set$ 
  assumes convex S
  and rel_open S
  and convex T
  and rel_open T
  shows  $convex\ (S + T) \wedge rel\_open\ (S + T)$ 
  by (metis assms convex_set_plus rel_interior_sum rel_open_def)

lemma convex_hull_finite_union_cones:
  assumes finite I

```

```

    and  $I \neq \{\}$ 
    assumes  $\bigwedge i. i \in I \implies \text{convex } (S\ i) \wedge \text{cone } (S\ i) \wedge S\ i \neq \{\}$ 
    shows  $\text{convex\_hull } (\bigcup (S\ ` I)) = \text{sum } S\ I$ 
    (is ?lhs = ?rhs)
  proof -
    {
      fix  $x$ 
      assume  $x \in ?lhs$ 
      then obtain  $c\ xs$  where
         $x = \text{sum } (\lambda i. c\ i *_R xs\ i)\ I \wedge (\forall i \in I. c\ i \geq 0) \wedge \text{sum } c\ I = 1 \wedge (\forall i \in I. xs\ i \in S\ i)$ 
        using  $\text{convex\_hull\_finite\_union}[of\ I\ S]$  assms by auto
        define  $s$  where  $s\ i = c\ i *_R xs\ i$  for  $i$ 
        have  $\forall i \in I. s\ i \in S\ i$ 
          using  $s\_def\ x\ \text{assms}$  by ( $\text{simp add: mem\_cone}$ )
        moreover have  $x = \text{sum } s\ I$  using  $x\ s\_def$  by auto
        ultimately have  $x \in ?rhs$ 
          using  $\text{set\_sum\_alt}[of\ I\ S]$  assms by auto
    }
    moreover
    {
      fix  $x$ 
      assume  $x \in ?rhs$ 
      then obtain  $s$  where  $x = \text{sum } s\ I \wedge (\forall i \in I. s\ i \in S\ i)$ 
        using  $\text{set\_sum\_alt}[of\ I\ S]$  assms by auto
      define  $xs$  where  $xs\ i = \text{of\_nat}(\text{card } I) *_R s\ i$  for  $i$ 
      then have  $x = \text{sum } (\lambda i. ((1 :: \text{real}) / \text{of\_nat}(\text{card } I)) *_R xs\ i)\ I$ 
        using  $x\ \text{assms}$  by auto
      moreover have  $\forall i \in I. xs\ i \in S\ i$ 
        using  $x\ s\_def\ \text{assms}$  by ( $\text{simp add: cone\_def}$ )
      moreover have  $\forall i \in I. (1 :: \text{real}) / \text{of\_nat}(\text{card } I) \geq 0$ 
        by auto
      moreover have  $\text{sum } (\lambda i. (1 :: \text{real}) / \text{of\_nat}(\text{card } I))\ I = 1$ 
        using  $\text{assms}$  by auto
      ultimately have  $x \in ?lhs$ 
        using  $\text{assms}$ 
        apply ( $\text{simp add: convex\_hull\_finite\_union}[of\ I\ S]$ )
        by ( $\text{rule\_tac } x = (\lambda i. 1 / (\text{card } I))\ \text{in } exI$ ) auto
    }
    ultimately show ?thesis by auto
  qed

```

lemma $\text{convex\_hull\_union\_cones\_two}$:

fixes $S\ T :: 'm::\text{euclidean\_space}\ \text{set}$

assumes $\text{convex } S$

and $\text{cone } S$

and $S \neq \{\}$

assumes $\text{convex } T$

and $\text{cone } T$

```

    and  $T \neq \{\}$ 
    shows  $\text{convex hull } (S \cup T) = S + T$ 
  proof -
    define  $I :: \text{nat set}$  where  $I = \{1, 2\}$ 
    define  $A$  where  $A\ i = (\text{if } i = (1 :: \text{nat}) \text{ then } S \text{ else } T)$  for  $i$ 
    have  $\bigcup (A \text{ ` } I) = S \cup T$ 
    using  $A\_def\ I\_def$  by auto
    then have  $\text{convex hull } (\bigcup (A \text{ ` } I)) = \text{convex hull } (S \cup T)$ 
    by auto
    moreover have  $\text{convex hull } \bigcup (A \text{ ` } I) = \text{sum } A\ I$ 
    using  $A\_def\ I\_def$ 
    by (metis assms convex_hull_finite_union_cones empty_iff finite.emptyI finite.insertI insertI1)
    moreover have  $\text{sum } A\ I = S + T$ 
    using  $A\_def\ I\_def$  by (force simp add: set_plus_def)
    ultimately show ?thesis by auto
  qed

lemma rel_interior_convex_hull_union:
  fixes  $S :: 'a \Rightarrow 'n :: \text{euclidean\_space set}$ 
  assumes finite I
    and  $\forall i \in I. \text{convex } (S\ i) \wedge S\ i \neq \{\}$ 
  shows  $\text{rel\_interior } (\text{convex hull } (\bigcup (S \text{ ` } I))) =$ 
     $\{ \text{sum } (\lambda i. c\ i *_{\mathbb{R}} s\ i)\ I \mid c\ s. (\forall i \in I. c\ i > 0) \wedge \text{sum } c\ I = 1 \wedge$ 
     $(\forall i \in I. s\ i \in \text{rel\_interior}(S\ i)) \}$ 
  (is ?lhs = ?rhs)
proof (cases  $I = \{\}$ )
  case True
    then show ?thesis
    using convex_hull_empty by auto
  next
    case False
    define  $C0$  where  $C0 = \text{convex hull } (\bigcup (S \text{ ` } I))$ 
    have  $\forall i \in I. C0 \geq S\ i$ 
    unfolding  $C0\_def$  using hull_subset[of  $\bigcup (S \text{ ` } I)$ ] by auto
    define  $K0$  where  $K0 = \text{cone hull } (\{1 :: \text{real}\} \times C0)$ 
    define  $K$  where  $K\ i = \text{cone hull } (\{1 :: \text{real}\} \times S\ i)$  for  $i$ 
    have  $\forall i \in I. K\ i \neq \{\}$ 
    unfolding  $K\_def$  using assms
    by (simp add: cone_hull_empty_iff[symmetric])
    have convK:  $\forall i \in I. \text{convex } (K\ i)$ 
    unfolding  $K\_def$ 
    by (simp add: assms(2) convex_Times convex_cone_hull)
    have  $K0 \supseteq K\ i$  if  $i \in I$  for  $i$ 
    unfolding  $K0\_def\ K\_def$ 
    by (simp add: Sigma_mono  $\langle \forall i \in I. S\ i \subseteq C0 \rangle$  hull_mono that)
    then have  $K0 \supseteq \bigcup (K \text{ ` } I)$  by auto
    moreover have  $\text{convex } K0$ 
    unfolding  $K0\_def$  by (simp add: C0_def convex_Times convex_cone_hull)

```

```

ultimately have  $geq: K0 \supseteq \text{convex hull } (\bigcup (K \text{ ' } I))$ 
  using  $\text{hull\_minimal[of\_ } K0 \text{ convex] by blast}$ 
have  $\forall i \in I. K \ i \supseteq \{1 :: \text{real}\} \times S \ i$ 
  using  $K\_def$  by ( $\text{simp add: hull\_subset}$ )
then have  $\bigcup (K \text{ ' } I) \supseteq \{1 :: \text{real}\} \times \bigcup (S \text{ ' } I)$ 
  by auto
then have  $\text{convex hull } \bigcup (K \text{ ' } I) \supseteq \text{convex hull } (\{1 :: \text{real}\} \times \bigcup (S \text{ ' } I))$ 
  by ( $\text{simp add: hull\_mono}$ )
then have  $\text{convex hull } \bigcup (K \text{ ' } I) \supseteq \{1 :: \text{real}\} \times C0$ 
  unfolding  $C0\_def$ 
  using  $\text{convex\_hull\_Times[of } \{\{1 :: \text{real}\}\} \bigcup (S \text{ ' } I) \text{ convex\_hull\_singleton}$ 
  by auto
moreover have  $\text{cone } (\text{convex hull } (\bigcup (K \text{ ' } I)))$ 
  by ( $\text{simp add: } K\_def \text{ cone\_Union cone\_cone\_hull cone\_convex\_hull}$ )
ultimately have  $\text{convex hull } (\bigcup (K \text{ ' } I)) \supseteq K0$ 
  unfolding  $K0\_def$ 
  using  $\text{hull\_minimal[of\_ convex hull } (\bigcup (K \text{ ' } I)) \text{ cone]}$ 
  by blast
then have  $K0 = \text{convex hull } (\bigcup (K \text{ ' } I))$ 
  using  $geq$  by auto
also have  $\dots = \text{sum } K \ I$ 
  using  $\text{assms False } \langle \forall i \in I. K \ i \neq \{\} \rangle \text{ cone\_hull\_eq convK}$ 
  by ( $\text{intro convex\_hull\_finite\_union\_cones; fastforce simp: } K\_def$ )
finally have  $K0 = \text{sum } K \ I$  by auto
then have  $*, \text{rel\_interior } K0 = \text{sum } (\lambda i. (\text{rel\_interior } (K \ i))) \ I$ 
  using  $\text{rel\_interior\_sum\_gen[of } I \ K] \text{ convK by auto}$ 
{
  fix  $x$ 
  assume  $x \in ?lhs$ 
  then have  $(1 :: \text{real}, x) \in \text{rel\_interior } K0$ 
    using  $K0\_def \ C0\_def \text{rel\_interior\_convex\_cone\_aux[of } C0 \ 1 :: \text{real } x] \text{ convex\_convex\_hull}$ 
    by auto
  then obtain  $k$  where  $k: (1 :: \text{real}, x) = \text{sum } k \ I \wedge (\forall i \in I. k \ i \in \text{rel\_interior } (K \ i))$ 
    using  $\langle \text{finite } I \rangle * \text{set\_sum\_alt[of } I \ \lambda i. \text{rel\_interior } (K \ i)]$  by auto
  {
    fix  $i$ 
    assume  $i \in I$ 
    then have  $\text{convex } (S \ i) \wedge k \ i \in \text{rel\_interior } (\text{cone hull } \{1\} \times S \ i)$ 
      using  $k \ K\_def \text{assms by auto}$ 
    then have  $\exists ci \ si. k \ i = (ci, ci *_R si) \wedge 0 < ci \wedge si \in \text{rel\_interior } (S \ i)$ 
      using  $\text{rel\_interior\_convex\_cone[of } S \ i]$  by auto
  }
}
then obtain  $c \ s$  where  $cs: \forall i \in I. k \ i = (c \ i, c \ i *_R s \ i) \wedge 0 < c \ i \wedge s \ i \in \text{rel\_interior } (S \ i)$ 
  by metis
then have  $x = (\sum i \in I. c \ i *_R s \ i) \wedge \text{sum } c \ I = 1$ 
  using  $k$  by ( $\text{simp add: sum\_prod}$ )

```

```

    then have  $x \in ?rhs$ 
      using  $k\ cs$  by auto
  }
  moreover
  {
    fix  $x$ 
    assume  $x \in ?rhs$ 
    then obtain  $c\ s$  where  $cs: x = \text{sum } (\lambda i. c\ i *_R s\ i)\ I \wedge$ 
       $(\forall i \in I. c\ i > 0) \wedge \text{sum } c\ I = 1 \wedge (\forall i \in I. s\ i \in \text{rel\_interior } (S\ i))$ 
      by auto
    define  $k$  where  $k\ i = (c\ i, c\ i *_R s\ i)$  for  $i$ 
    {
      fix  $i$  assume  $i \in I$ 
      then have  $k\ i \in \text{rel\_interior } (K\ i)$ 
        using  $k\_def\ K\_def\ assms\ cs\ \text{rel\_interior\_convex\_cone}[of\ S\ i]$ 
        by auto
    }
    then have  $(1, x) \in \text{rel\_interior } K0$ 
      using  $*\ \text{set\_sum\_alt}[of\ I\ (\lambda i. \text{rel\_interior } (K\ i))]\ assms\ cs$ 
      by  $(\text{simp add: } k\_def)\ (\text{metis } (mono\_tags, lifting)\ \text{sum\_prod})$ 
    then have  $x \in ?lhs$ 
      using  $K0\_def\ C0\_def\ \text{rel\_interior\_convex\_cone\_aux}[of\ C0\ 1\ x]$ 
      by auto
  }
  ultimately show  $?thesis$  by blast
qed

```

```

lemma convex_le_Inf_differential:
  fixes  $f :: real \Rightarrow real$ 
  assumes convex_on  $I\ f$ 
    and  $x \in \text{interior } I$ 
    and  $y \in I$ 
  shows  $f\ y \geq f\ x + \text{Inf } ((\lambda t. (f\ x - f\ t) / (x - t)) \text{ ` } (\{x <.. \} \cap I)) * (y - x)$ 
    (is  $\_ \geq \_ + \text{Inf } (?F\ x) * (y - x)$ )
proof (cases rule: linorder_cases)
  assume  $x < y$ 
  moreover
  have open  $(\text{interior } I)$  by auto
  from openE[OF this  $\langle x \in \text{interior } I \rangle$ ]
  obtain  $e$  where  $e: 0 < e$   $\text{ball } x\ e \subseteq \text{interior } I$  .
  moreover define  $t$  where  $t = \min (x + e / 2) ((x + y) / 2)$ 
  ultimately have  $x < t < y$   $t \in \text{ball } x\ e$ 
    by  $(\text{auto simp: } \text{dist\_real\_def}\ \text{field\_simps}\ \text{split: } \text{split\_min})$ 
  with  $\langle x \in \text{interior } I \rangle\ e\ \text{interior\_subset}[of\ I]$  have  $t \in I$   $x \in I$  by auto

  define  $K$  where  $K = x - e / 2$ 
  with  $\langle 0 < e \rangle$  have  $K \in \text{ball } x\ e$   $K < x$ 
    by  $(\text{auto simp: } \text{dist\_real\_def})$ 

```



```

then have  $K \in I$ 
  using  $\langle \text{interior } I \subseteq I \rangle e(2)$  by blast

have  $\text{Inf } (?F \ x) \leq (f \ x - f \ y) / (x - y)$ 
proof (intro bdd_belowI cInf_lower2)
  show  $(f \ x - f \ t) / (x - t) \in ?F \ x$ 
    using  $\langle t \in I \rangle \langle x < t \rangle$  by auto
  show  $(f \ x - f \ t) / (x - t) \leq (f \ x - f \ y) / (x - y)$ 
    using  $\langle \text{convex\_on } I f \rangle \langle x \in I \rangle \langle y \in I \rangle \langle x < t \rangle \langle t < y \rangle$ 
    by (rule convex_on_diff)
next
fix  $y$ 
assume  $y \in ?F \ x$ 
  with order_trans[OF convex_on_diff[OF  $\langle \text{convex\_on } I f \rangle \langle K \in I \rangle \_ \langle K < x \rangle$ ]]
  show  $(f \ K - f \ x) / (K - x) \leq y$  by auto
qed
then show  $?thesis$ 
  using  $\langle x < y \rangle$  by (simp add: field_simps)
next
assume  $y < x$ 
moreover
  have open (interior  $I$ ) by auto
  from openE[OF this  $\langle x \in \text{interior } I \rangle$ ]
  obtain  $e$  where  $e: 0 < e$  ball  $x \ e \subseteq \text{interior } I$  .
  moreover define  $t$  where  $t = x + e / 2$ 
  ultimately have  $x < t \ t \in \text{ball } x \ e$ 
    by (auto simp: dist_real_def field_simps)
  with  $\langle x \in \text{interior } I \rangle e$  interior_subset[of  $I$ ] have  $t \in I \ x \in I$  by auto

have  $(f \ x - f \ y) / (x - y) \leq \text{Inf } (?F \ x)$ 
proof (rule cInf_greatest)
  have  $(f \ x - f \ y) / (x - y) = (f \ y - f \ x) / (y - x)$ 
    using  $\langle y < x \rangle$  by (auto simp: field_simps)
  also
  fix  $z$ 
  assume  $z \in ?F \ x$ 
  with order_trans[OF convex_on_diff[OF  $\langle \text{convex\_on } I f \rangle \langle y \in I \rangle \_ \langle y < x \rangle$ ]]
  have  $(f \ y - f \ x) / (y - x) \leq z$ 
    by auto
  finally show  $(f \ x - f \ y) / (x - y) \leq z$  .
next
have  $x + e / 2 \in \text{ball } x \ e$ 
  using  $e$  by (auto simp: dist_real_def)
with  $e$  interior_subset[of  $I$ ] have  $x + e / 2 \in \{x <.. \} \cap I$ 
  by auto
then show  $?F \ x \neq \{\}$ 
  by blast
qed

```

```

    then show ?thesis
      using ⟨y < x⟩ by (simp add: field_simps)
qed simp

```

5.0.6 Explicit formulas for interior and relative interior of convex hull

```

lemma at_within_cbox_finite:
  assumes x ∈ box a b x ∉ S finite S
  shows (at x within cbox a b - S) = at x
proof -
  have interior (cbox a b - S) = box a b - S
    using ⟨finite S⟩ by (simp add: interior_diff finite_imp_closed)
  then show ?thesis
    using at_within_interior assms by fastforce
qed

lemma affine_independent_convex_affine_hull:
  fixes S :: 'a::euclidean_space set
  assumes ¬ affine_dependent S T ⊆ S
  shows convex hull T = affine hull T ∩ convex hull S
proof -
  have fin: finite S finite T using assms aff_independent_finite finite_subset by
  auto
  have convex_hull_T ⊆ affine_hull T
    using convex_hull_subset_affine_hull by blast
  moreover have convex_hull T ⊆ convex_hull S
    using assms hull_mono by blast
  moreover have affine_hull T ∩ convex_hull S ⊆ convex_hull T
  proof -
    have 0: ⋀u. sum u S = 0 ⟹ (∀v∈S. u v = 0) ∨ (∑v∈S. u v *R v) ≠ 0
      using affine_dependent_explicit_finite assms(1) fin(1) by auto
    show ?thesis
    proof (clarsimp simp add: affine_hull_finite fin)
      fix u
      assume S: (∑v∈T. u v *R v) ∈ convex_hull S
      and T1: sum u T = 1
      then obtain v where v: ∀x∈S. 0 ≤ v x sum v S = 1 (∑x∈S. v x *R x) =
        (∑v∈T. u v *R v)
      by (auto simp add: convex_hull_finite fin)
      { fix x
        assume x ∈ T
        then have S: S = (S - T) ∪ T — split into separate cases
          using assms by auto
        have [simp]: (∑x∈T. v x *R x) + (∑x∈S - T. v x *R x) = (∑x∈T. u
          x *R x)
          sum v T + sum v (S - T) = 1
          using v fin S
          by (auto simp: sum.union_disjoint [symmetric] Un_commute)
      }
    }
  }

```

```

    have ( $\sum x \in S. \text{if } x \in T \text{ then } v\ x - u\ x \text{ else } v\ x = 0$ )
      ( $\sum x \in S. (\text{if } x \in T \text{ then } v\ x - u\ x \text{ else } v\ x) *_{\mathcal{R}} x = 0$ )
    using  $v\ \text{fin}\ T1$ 
  by (subst  $S$ , subst  $\text{sum.union\_disjoint}$ , auto simp:  $\text{algebra\_simps sum\_subtractf}$ ) +
} note [simp] = this
have ( $\forall x \in T. 0 \leq u\ x$ )
  using 0 [of  $\lambda x. \text{if } x \in T \text{ then } v\ x - u\ x \text{ else } v\ x$ ]  $\langle T \subseteq S \rangle v(1)$  by fastforce
then show ( $\sum v \in T. u\ v *_{\mathcal{R}} v \in \text{convex hull } T$ )
  using 0 [of  $\lambda x. \text{if } x \in T \text{ then } v\ x - u\ x \text{ else } v\ x$ ]  $\langle T \subseteq S \rangle T1$ 
  by (fastforce simp add:  $\text{convex\_hull\_finite fin}$ )
qed
qed
ultimately show ?thesis
  by blast
qed

lemma affine_independent_span_eq:
  fixes  $S :: 'a::\text{euclidean\_space set}$ 
  assumes  $\neg \text{affine\_dependent } S$   $\text{card } S = \text{Suc } (\text{DIM } ('a))$ 
  shows  $\text{affine hull } S = \text{UNIV}$ 
proof (cases  $S = \{\}$ )
  case True then show ?thesis
    using  $\text{assms}$  by simp
  next
  case False
    then obtain  $a\ T$  where  $T: a \notin T$   $S = \text{insert } a\ T$ 
    by blast
    then have  $\text{fin}: \text{finite } T$  using  $\text{assms}$ 
    by (metis  $\text{finite\_insert aff\_independent\_finite}$ )
    have  $\text{UNIV} \subseteq (+)\ a \text{ ' span } ((\lambda x. x - a) \text{ ' } T)$ 
    proof (intro  $\text{card\_ge\_dim\_independent Fun.vimage\_subsetD}$ )
      show  $\text{independent } ((\lambda x. x - a) \text{ ' } T)$ 
        using  $T$   $\text{affine\_dependent\_iff\_dependent assms}(1)$  by auto
      show  $\text{dim } ((+)\ a - \text{ ' UNIV}) \leq \text{card } ((\lambda x. x - a) \text{ ' } T)$ 
        using  $\text{assms } T\ \text{fin}$  by (auto simp:  $\text{card\_image inj\_on\_def}$ )
    qed (use  $\text{surj\_plus in auto}$ )
    then show ?thesis
      using  $T(2)$   $\text{affine\_hull\_insert\_span\_gen equalityI}$  by fastforce
  qed

lemma affine_independent_span_gt:
  fixes  $S :: 'a::\text{euclidean\_space set}$ 
  assumes  $\text{ind}: \neg \text{affine\_dependent } S$  and  $\text{dim}: \text{DIM } ('a) < \text{card } S$ 
  shows  $\text{affine hull } S = \text{UNIV}$ 
proof (intro  $\text{affine\_independent\_span\_eq [OF ind]}$   $\text{antisym}$ )
  show  $\text{card } S \leq \text{Suc } \text{DIM } ('a)$ 
    using  $\text{aff\_independent\_finite affine\_dependent\_biggerset ind}$  by fastforce
  show  $\text{Suc } \text{DIM } ('a) \leq \text{card } S$ 
    using  $\text{Suc\_leI dim}$  by blast

```

qed

lemma *empty_interior_affine_hull*:
fixes $S :: 'a::\text{euclidean\_space}$ *set*
assumes *finite S and dim: card S $\leq$ DIM ('a)*
shows $\text{interior}(\text{affine hull } S) = \{\}$
using *assms*
proof (*induct S rule: finite_induct*)
case (*insert x S*)
then have $\text{dim}(\text{span}((\lambda y. y - x) ' S)) < \text{DIM}('a)$
by (*auto simp: Suc_le_lessD card_image_le dual_order.trans intro!: dim_le_card'[THEN le_less_trans]*)
then show ?*case*
by (*simp add: empty_interior_lowdim affine_hull_insert_span_gen interior_translation*)
qed *auto*

lemma *empty_interior_convex_hull*:
fixes $S :: 'a::\text{euclidean\_space}$ *set*
assumes *finite S and dim: card S $\leq$ DIM ('a)*
shows $\text{interior}(\text{convex hull } S) = \{\}$
by (*metis Diff_empty Diff_eq_empty_iff convex_hull_subset_affine_hull interior_mono empty_interior_affine_hull [OF assms]*)

lemma *explicit_subset_rel_interior_convex_hull*:
fixes $S :: 'a::\text{euclidean\_space}$ *set*
shows *finite S*
 $\implies \{y. \exists u. (\forall x \in S. 0 < u x \wedge u x < 1) \wedge \text{sum } u S = 1 \wedge \text{sum } (\lambda x. u x *_R x) S = y\}$
 $\subseteq \text{rel\_interior}(\text{convex hull } S)$
by (*force simp add: rel_interior_convex_hull_union [where S= $\lambda x. \{x\}$ and I=S, simplified]*)

lemma *explicit_subset_rel_interior_convex_hull_minimal*:
fixes $S :: 'a::\text{euclidean\_space}$ *set*
shows *finite S*
 $\implies \{y. \exists u. (\forall x \in S. 0 < u x) \wedge \text{sum } u S = 1 \wedge \text{sum } (\lambda x. u x *_R x) S = y\}$
 $\subseteq \text{rel\_interior}(\text{convex hull } S)$
by (*force simp add: rel_interior_convex_hull_union [where S= $\lambda x. \{x\}$ and I=S, simplified]*)

lemma *rel_interior_convex_hull_explicit*:
fixes $S :: 'a::\text{euclidean\_space}$ *set*
assumes $\neg \text{affine\_dependent } S$
shows $\text{rel\_interior}(\text{convex hull } S) =$
 $\{y. \exists u. (\forall x \in S. 0 < u x) \wedge \text{sum } u S = 1 \wedge \text{sum } (\lambda x. u x *_R x) S = y\}$
(is ?lhs = ?rhs)
proof
show ?*rhs* $\leq$?*lhs*

```

  by (simp add: aff_independent_finite explicit_subset_rel_interior_convex_hull_minimal
    assms)
next
  show ?lhs ≤ ?rhs
  proof (cases ∃ a. S = {a})
    case True then show ?lhs ≤ ?rhs
      by force
  next
    case False
    have fs: finite S
    using assms by (simp add: aff_independent_finite)
    { fix a b and d::real
      assume ab: a ∈ S b ∈ S a ≠ b
      then have S: S = (S - {a,b}) ∪ {a,b} — split into separate cases
        by auto
      have (∑ x∈S. if x = a then - d else if x = b then d else 0) = 0
        (∑ x∈S. (if x = a then - d else if x = b then d else 0) *R x) = d *R b
    - d *R a
      using ab fs
      by (subst S, subst sum.union_disjoint, auto)+
    } note [simp] = this
    { fix y
      assume y: y ∈ convex hull S y ∉ ?rhs
      have *: False if
        ua: ∀ x∈S. 0 ≤ u x sum u S = 1 ∧ 0 < u a a ∈ S
        and yT: y = (∑ x∈S. u x *R x) y ∈ T open T
        and sb: T ∩ affine hull S ⊆ {w. ∃ u. (∀ x∈S. 0 ≤ u x) ∧ sum u S = 1 ∧
          (∑ x∈S. u x *R x) = w}
      for u T a
      proof -
        have ua0: u a = 0
        using ua by auto
        obtain b where b: b ∈ S a ≠ b
        using ua False by auto
        obtain e where e: 0 < e ball (∑ x∈S. u x *R x) e ⊆ T
        using yT by (auto elim: openE)
        with b obtain d where d: 0 < d norm(d *R (a-b)) < e
        by (auto intro: that [of e / 2 / norm(a-b)])
        have (∑ x∈S. u x *R x) ∈ affine hull S
        using yT y by (metis affine_hull_convex_hull hull_redundant_eq)
        then have (∑ x∈S. u x *R x) - d *R (a - b) ∈ affine hull S
        using ua b by (auto simp: hull_inc intro: mem_affine_3_minus2)
        then have y - d *R (a - b) ∈ T ∩ affine hull S
        using d e yT by auto
        then obtain v where v: ∀ x∈S. 0 ≤ v x
          sum v S = 1
          (∑ x∈S. v x *R x) = (∑ x∈S. u x *R x) - d *R (a - b)
        using subsetD [OF sb] yT
        by auto
      }
    }
  }

```

```

have aff:  $\bigwedge u. \text{sum } u \ S = 0 \implies (\forall v \in S. u \ v = 0) \vee (\sum v \in S. u \ v *_{\mathbb{R}} v) \neq 0$ 
using assms by (simp add: affine_dependent_explicit_finite fs)
show False
using ua b d v aff [of  $\lambda x. (v \ x - u \ x) - (\text{if } x = a \text{ then } -d \text{ else if } x = b \text{ then } d \text{ else } 0)$ ]
by (auto simp: algebra_simps sum_subtractf sum.distrib)
qed
have  $y \notin \text{rel\_interior}(\text{convex\_hull } S)$ 
using y
apply (simp add: mem_rel_interior)
apply (auto simp: convex_hull_finite [OF fs])
apply (drule_tac  $x=u$  in spec)
apply (auto intro:  $*$ )
done
} with rel_interior_subset show  $?lhs \leq ?rhs$ 
by blast
qed
qed

```

```

lemma interior_convex_hull_explicit_minimal:
fixes  $S :: 'a :: \text{euclidean\_space}$  set
assumes  $\neg \text{affine\_dependent } S$ 
shows
 $\text{interior}(\text{convex\_hull } S) =$ 
 $(\text{if } \text{card}(S) \leq \text{DIM}('a) \text{ then } \{\}$ 
 $\text{else } \{y. \exists u. (\forall x \in S. 0 < u \ x) \wedge \text{sum } u \ S = 1 \wedge (\sum x \in S. u \ x *_{\mathbb{R}} x)$ 
 $= y\})$ 
 $(\text{is } \_ = (\text{if } \_ \text{ then } \_ \text{ else } ?rhs))$ 
proof (clarsimp simp: aff_independent_finite empty_interior_convex_hull assms)
assume  $S: \neg \text{card } S \leq \text{DIM}('a)$ 
have  $\text{interior}(\text{convex\_hull } S) = \text{rel\_interior}(\text{convex\_hull } S)$ 
using assms S by (simp add: affine_independent_span_gt_rel_interior_interior)
then show  $\text{interior}(\text{convex\_hull } S) = ?rhs$ 
by (simp add: assms S rel_interior_convex_hull_explicit)
qed

```

```

lemma interior_convex_hull_explicit:
fixes  $S :: 'a :: \text{euclidean\_space}$  set
assumes  $\neg \text{affine\_dependent } S$ 
shows
 $\text{interior}(\text{convex\_hull } S) =$ 
 $(\text{if } \text{card}(S) \leq \text{DIM}('a) \text{ then } \{\}$ 
 $\text{else } \{y. \exists u. (\forall x \in S. 0 < u \ x \wedge u \ x < 1) \wedge \text{sum } u \ S = 1 \wedge (\sum x \in S.$ 
 $u \ x *_{\mathbb{R}} x) = y\})$ 
proof -
{ fix  $u :: 'a \Rightarrow \text{real}$  and  $a$ 
assume  $\text{card } \text{Basis} < \text{card } S$  and  $u: \bigwedge x. x \in S \implies 0 < u \ x \text{ sum } u \ S = 1$  and
 $a: a \in S$ 
then have  $cs: \text{Suc } 0 < \text{card } S$ 

```

```

    by (metis DIM_positive less_trans_Suc)
  obtain b where b:  $b \in S$   $a \neq b$ 
  proof (cases  $S \leq \{a\}$ )
    case True
    then show thesis
      using cs subset_singletonD by fastforce
  qed blast
  have  $u\ a + u\ b \leq \text{sum } u\ \{a, b\}$ 
    using a b by simp
  also have  $\dots \leq \text{sum } u\ S$ 
    using a b u
  by (intro Groups_Big.sum_mono2) (auto simp: less_imp_le aff_independent_finite
  assms)
  finally have  $u\ a < 1$ 
    using  $\langle b \in S \rangle\ u$  by fastforce
} note [simp] = this
show ?thesis
  using assms by (force simp add: not_le interior_convex_hull_explicit_minimal)
qed

lemma interior_closed_segment_ge2:
  fixes a :: 'a::euclidean_space
  assumes  $2 \leq \text{DIM}('a)$ 
  shows  $\text{interior}(\text{closed\_segment } a\ b) = \{\}$ 
  using assms unfolding segment_convex_hull
  proof -
    have  $\text{card } \{a, b\} \leq \text{DIM}('a)$ 
      using assms
    by (simp add: card_insert_if_linear not_less_eq_eq numeral_2_eq_2)
    then show  $\text{interior } (\text{convex\_hull } \{a, b\}) = \{\}$ 
      by (metis empty_interior_convex_hull finite.insertI finite.emptyI)
  qed

lemma interior_open_segment:
  fixes a :: 'a::euclidean_space
  shows  $\text{interior}(\text{open\_segment } a\ b) =$ 
     $(\text{if } 2 \leq \text{DIM}('a) \text{ then } \{\} \text{ else } \text{open\_segment } a\ b)$ 
  proof (simp add: not_le, intro conjI impI)
    assume  $2 \leq \text{DIM}('a)$ 
    then show  $\text{interior } (\text{open\_segment } a\ b) = \{\}$ 
      using interior_closed_segment_ge2 interior_mono segment_open_subset_closed
    by blast
  next
    assume le2:  $\text{DIM}('a) < 2$ 
    show  $\text{interior } (\text{open\_segment } a\ b) = \text{open\_segment } a\ b$ 
      proof (cases  $a = b$ )
        case True then show ?thesis by auto
      next
        case False

```

```

with le2 have affine_hull (open_segment a b) = UNIV
by (simp add: False affine_independent_span_gt)
then show interior (open_segment a b) = open_segment a b
using rel_interior_interior rel_interior_open_segment by blast
qed
qed

```

```

lemma interior_closed_segment:
  fixes a :: 'a::euclidean_space
  shows interior(closed_segment a b) =
    (if 2 ≤ DIM('a) then {} else open_segment a b)
proof (cases a = b)
  case True then show ?thesis by simp
next
  case False
  then have closure (open_segment a b) = closed_segment a b
  by simp
  then show ?thesis
  by (metis (no_types) convex_interior_closure convex_open_segment inte-
rior_open_segment)
qed

```

lemmas interior_segment = interior_closed_segment interior_open_segment

```

lemma closed_segment_eq [simp]:
  fixes a :: 'a::euclidean_space
  shows closed_segment a b = closed_segment c d ⟷ {a,b} = {c,d}
proof
  assume abcd: closed_segment a b = closed_segment c d
  show {a,b} = {c,d}
  proof (cases a=b ∨ c=d)
    case True with abcd show ?thesis by force
  next
    case False
    then have neq: a ≠ b ∧ c ≠ d by force
    have *: closed_segment c d - {a, b} = rel_interior (closed_segment c d)
    using neq abcd by (metis (no_types) open_segment_def rel_interior_closed_segment)
    have b ∈ {c, d}
    proof -
      have insert_b (closed_segment c d) = closed_segment c d
      using abcd by blast
      then show ?thesis
      by (metis DiffD2 Diff_insert2 False * insertI1 insert_Diff_if open_segment_def
rel_interior_closed_segment)
    qed
    moreover have a ∈ {c, d}
    by (metis Diff_iff False * abcd ends_in_segment(1) insertI1 open_segment_def
rel_interior_closed_segment)
    ultimately show {a, b} = {c, d}
  qed

```



```

    using neq by fastforce
  qed
next
  assume  $\{a,b\} = \{c,d\}$ 
  then show  $\text{closed\_segment } a \ b = \text{closed\_segment } c \ d$ 
    by (simp add: segment_convex_hull)
  qed

lemma closed_open_segment_eq [simp]:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  shows  $\text{closed\_segment } a \ b \neq \text{open\_segment } c \ d$ 
by (metis DiffE closed_segment_neq_empty closure_closed_segment closure_open_segment
ends_in_segment(1) insertI1 open_segment_def)

lemma open_closed_segment_eq [simp]:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  shows  $\text{open\_segment } a \ b \neq \text{closed\_segment } c \ d$ 
using closed_open_segment_eq by blast

lemma open_segment_eq [simp]:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  shows  $\text{open\_segment } a \ b = \text{open\_segment } c \ d \longleftrightarrow a = b \wedge c = d \vee \{a,b\} = \{c,d\}$ 
    (is ?lhs = ?rhs)
proof
  assume abcd: ?lhs
  show ?rhs
  proof (cases  $a=b \vee c=d$ )
    case True with abcd show ?thesis
      using finite_open_segment by fastforce
    next
      case False
      then have a2:  $a \neq b \wedge c \neq d$  by force
      with abcd show ?rhs
        unfolding open_segment_def
        by (metis (no_types) abcd closed_segment_eq closure_open_segment)
  qed
next
  assume ?rhs
  then show ?lhs
    by (metis Diff cancel_convex_hull_singleton insert_absorb2 open_segment_def
segment_convex_hull)
  qed

```

5.0.7 Similar results for closure and (relative or absolute) frontier

```

lemma closure_convex_hull [simp]:
  fixes  $S :: 'a::\text{euclidean\_space}$  set

```

shows $\text{compact } S \implies \text{closure}(\text{convex hull } S) = \text{convex hull } S$
by (*simp add: compact_imp_closed compact_convex_hull*)

lemma *rel_frontier_convex_hull_explicit*:

fixes $S :: 'a::\text{euclidean\_space set}$

assumes $\neg \text{affine\_dependent } S$

shows $\text{rel\_frontier}(\text{convex hull } S) =$

$\{y. \exists u. (\forall x \in S. 0 \leq u x) \wedge (\exists x \in S. u x = 0) \wedge \text{sum } u S = 1 \wedge \text{sum}$
 $(\lambda x. u x *_{\mathbb{R}} x) S = y\}$

proof –

have *fs*: $\text{finite } S$

using *assms* **by** (*simp add: aff_independent_finite*)

have $\bigwedge u y v.$

$\llbracket y \in S; u y = 0; \text{sum } u S = 1; \forall x \in S. 0 < v x;$

$\text{sum } v S = 1; (\sum_{x \in S} v x *_{\mathbb{R}} x) = (\sum_{x \in S} u x *_{\mathbb{R}} x) \rrbracket$

$\implies \exists u. \text{sum } u S = 0 \wedge (\exists v \in S. u v \neq 0) \wedge (\sum_{v \in S} u v *_{\mathbb{R}} v) = 0$

apply (*rule_tac x = \lambda x. u x - v x in exI*)

apply (*force simp: sum_subtractf scaleR_diff_left*)

done

then show *?thesis*

using *fs assms*

apply (*simp add: rel_frontier_def finite_imp_compact rel_interior_convex_hull_explicit*)

apply (*auto simp: convex_hull_finite*)

apply (*metis less_eq_real_def*)

by (*simp add: affine_dependent_explicit_finite*)

qed

lemma *frontier_convex_hull_explicit*:

fixes $S :: 'a::\text{euclidean\_space set}$

assumes $\neg \text{affine\_dependent } S$

shows $\text{frontier}(\text{convex hull } S) =$

$\{y. \exists u. (\forall x \in S. 0 \leq u x) \wedge (\text{DIM } ('a) < \text{card } S \longrightarrow (\exists x \in S. u x = 0))$
 $\wedge$

$\text{sum } u S = 1 \wedge \text{sum } (\lambda x. u x *_{\mathbb{R}} x) S = y\}$

proof –

have *fs*: $\text{finite } S$

using *assms* **by** (*simp add: aff_independent_finite*)

show *?thesis*

proof (*cases DIM ('a) < card S*)

case *True*

with *assms fs* **show** *?thesis*

by (*simp add: rel_frontier_def frontier_def rel_frontier_convex_hull_explicit*
 $[\text{symmetric}]$

interior_convex_hull_explicit_minimal rel_interior_convex_hull_explicit)

next

case *False*

then have $\text{card } S \leq \text{DIM } ('a)$

by *linarith*

then show *?thesis*

```

    using assms fs
  apply (simp add: frontier_def interior_convex_hull_explicit finite_imp_compact)
  apply (simp add: convex_hull_finite)
  done
qed
qed

```

lemma *rel_frontier_convex_hull_cases*:

```

  fixes S :: 'a::euclidean_space set
  assumes ¬ affine_dependent S
  shows rel_frontier(convex hull S) =  $\bigcup \{ \text{convex hull } (S - \{x\}) \mid x. x \in S \}$ 
proof -
  have fs: finite S
  using assms by (simp add: aff_independent_finite)
  { fix u a
  have  $\forall x \in S. 0 \leq u x \implies a \in S \implies u a = 0 \implies \text{sum } u S = 1 \implies$ 
 $\exists x v. x \in S \wedge$ 
 $(\forall x \in S - \{x\}. 0 \leq v x) \wedge$ 
 $\text{sum } v (S - \{x\}) = 1 \wedge (\sum_{x \in S - \{x\}} v x *_R x) = (\sum_{x \in S} u$ 
 $x *_R x)$ 
  apply (rule_tac x=a in exI)
  apply (rule_tac x=u in exI)
  apply (simp add: Groups_Big.sum_diff1 fs)
  done }
  moreover
  { fix a u
  have  $a \in S \implies \forall x \in S - \{a\}. 0 \leq u x \implies \text{sum } u (S - \{a\}) = 1 \implies$ 
 $\exists v. (\forall x \in S. 0 \leq v x) \wedge$ 
 $(\exists x \in S. v x = 0) \wedge \text{sum } v S = 1 \wedge (\sum_{x \in S} v x *_R x) = (\sum_{x \in S - \{a\}} u x *_R x)$ 
  apply (rule_tac x= $\lambda x. \text{if } x = a \text{ then } 0 \text{ else } u x$  in exI)
  apply (auto simp: sum.If_cases Diff_eq if_smult fs)
  done }
  ultimately show ?thesis
  using assms
  apply (simp add: rel_frontier_convex_hull_explicit)
  apply (auto simp add: convex_hull_finite fs Union_SetCompr_eq)
  done
qed

```

lemma *frontier_convex_hull_eq_rel_frontier*:

```

  fixes S :: 'a::euclidean_space set
  assumes ¬ affine_dependent S
  shows frontier(convex hull S) =
    (if card S ≤ DIM ('a) then convex hull S else rel_frontier(convex hull S))
  using assms
  unfolding rel_frontier_def frontier_def
  by (simp add: affine_independent_span_gt rel_interior_interior
    finite_imp_compact empty_interior_convex_hull aff_independent_finite)

```

```

lemma frontier_convex_hull_cases:
  fixes S :: 'a::euclidean_space set
  assumes  $\neg$  affine_dependent S
  shows frontier(convex hull S) =
    (if card S  $\leq$  DIM ('a) then convex hull S else  $\bigcup \{ \text{convex hull } (S - \{x\})$ 
    | x. x  $\in$  S })
by (simp add: assms frontier_convex_hull_eq_rel_frontier rel_frontier_convex_hull_cases)

```

```

lemma in_frontier_convex_hull:
  fixes S :: 'a::euclidean_space set
  assumes finite S card S  $\leq$  Suc (DIM ('a)) x  $\in$  S
  shows x  $\in$  frontier(convex hull S)
proof (cases affine_dependent S)
case True
  with assms obtain y where y  $\in$  S and y: y  $\in$  affine hull (S - {y})
  by (auto simp: affine_dependent_def)
  moreover have x  $\in$  closure (convex hull S)
  by (meson closure_subset hull_inc subset_eq  $\langle x \in S \rangle$ )
  moreover have x  $\notin$  interior (convex hull S)
  using assms
  by (metis Suc_mono affine_hull_convex_hull affine_hull_nonempty_interior
     $\langle y \in S \rangle$  y card.remove empty_iff empty_interior_affine_hull finite_Diff hull_redundant
    insert_Diff interior_UNIV not_less)
  ultimately show ?thesis
  unfolding frontier_def by blast
next
case False
  { assume card S = Suc (card Basis)
    then have cs: Suc 0 < card S
      by (simp)
    with subset_singletonD have  $\exists y \in S. y \neq x$ 
      by (cases S  $\leq$  {x}) fastforce+
    } note [dest!] = this
  show ?thesis using assms
    unfolding frontier_convex_hull_cases [OF False] Union_SetCompr_eq
    by (auto simp: le_Suc_eq hull_inc)
qed

```

```

lemma not_in_interior_convex_hull:
  fixes S :: 'a::euclidean_space set
  assumes finite S card S  $\leq$  Suc (DIM ('a)) x  $\in$  S
  shows x  $\notin$  interior(convex hull S)
using in_frontier_convex_hull [OF assms]
by (metis Diff_iff frontier_def)

```

```

lemma interior_convex_hull_eq_empty:
  fixes S :: 'a::euclidean_space set
  assumes card S = Suc (DIM ('a))

```

```

  shows interior(convex hull S) = {}  $\longleftrightarrow$  affine_dependent S
proof
  show affine_dependent S  $\implies$  interior (convex hull S) = {}
  proof (clarsimp simp: affine_dependent_def)
    fix a b
    assume b  $\in$  S b  $\in$  affine hull (S - {b})
    then have interior(affine hull S) = {} using assms
    by (metis DIM_positive One_nat_def Suc_mono card.remove card.infinite
        empty_interior_affine_hull eq_iff_hull_redundant insert_Diff not_less zero_le_one)
    then show interior (convex hull S) = {}
    using affine_hull_nonempty_interior by fastforce
  qed
next
  show interior (convex hull S) = {}  $\implies$  affine_dependent S
  by (metis affine_hull_convex_hull affine_hull_empty affine_independent_span_eq
      assms convex_convex_hull empty_not_UNIV rel_interior_eq_empty rel_interior_interior)
qed

```

5.0.8 Coplanarity, and collinearity in terms of affine hull

definition *coplanar* where

coplanar S $\equiv \exists u v w. S \subseteq \text{affine hull } \{u, v, w\}$

lemma *collinear_affine_hull*:

collinear S $\longleftrightarrow (\exists u v. S \subseteq \text{affine hull } \{u, v\})$

proof (cases S={})

case True then show ?thesis

by simp

next

case False

then obtain x where x: x $\in$ S by auto

{ fix u

assume *: $\bigwedge x y. [x \in S; y \in S] \implies \exists c. x - y = c *_R u$

have $\bigwedge y c. x - y = c *_R u \implies \exists a b. y = a *_R x + b *_R (x + u) \wedge a + b = 1$

by (rule_tac x=1+c in exI, rule_tac x=-c in exI, simp add: algebra_simps)

then have $\exists u v. S \subseteq \{a *_R u + b *_R v \mid a b. a + b = 1\}$

using * [OF x] by (rule_tac x=x in exI, rule_tac x=x+u in exI, force)

} moreover

{ fix u v x y

assume *: $S \subseteq \{a *_R u + b *_R v \mid a b. a + b = 1\}$

have $\exists c. x - y = c *_R (v - u)$ if $x \in S y \in S$

proof -

obtain a r where a + r = 1 x = a *_R u + r *_R v

using * $\langle x \in S \rangle$ by blast

moreover

obtain b s where b + s = 1 y = b *_R u + s *_R v

using * $\langle y \in S \rangle$ by blast

ultimately have $x - y = (r - s) *_R (v - u)$

by (simp add: algebra_simps) (metis scaleR_left.add)

```

      then show ?thesis
        by blast
    qed
  } ultimately
  show ?thesis
    unfolding collinear_def affine_hull_2
    by blast
qed

lemma collinear_closed_segment [simp]: collinear (closed_segment a b)
  by (metis affine_hull_convex_hull collinear_affine_hull hull_subset segment_convex_hull)

lemma collinear_open_segment [simp]: collinear (open_segment a b)
  unfolding open_segment_def
  by (metis convex_hull_subset_affine_hull segment_convex_hull dual_order.trans
    convex_hull_subset_affine_hull Diff_subset collinear_affine_hull)

lemma collinear_between_cases:
  fixes c :: 'a::euclidean_space
  shows collinear {a,b,c}  $\longleftrightarrow$  between (b,c) a  $\vee$  between (c,a) b  $\vee$  between (a,b) c
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain u v where uv:  $\bigwedge x. x \in \{a, b, c\} \implies \exists c. x = u + c *_R v$ 
    by (auto simp: collinear_alt)
  show ?rhs
    using uv [of a] uv [of b] uv [of c] by (auto simp: between_1)
next
  assume ?rhs
  then show ?lhs
    unfolding between_mem_convex_hull
    by (metis (no_types, opaque_lifting) collinear_closed_segment collinear_subset
    hull_redundant hull_subset insert_commute segment_convex_hull)
qed

lemma subset_continuous_image_segment_1:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes continuous_on (closed_segment a b) f
  shows closed_segment (f a) (f b)  $\subseteq$  image f (closed_segment a b)
  by (metis connected_segment convex_contains_segment ends_in_segment imageI
    is_interval_connected_1 is_interval_convex connected_continuous_image
    [OF assms])

lemma continuous_injective_image_segment_1:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes conf: continuous_on (closed_segment a b) f
    and inj: inj_on f (closed_segment a b)
  shows f ` (closed_segment a b) = closed_segment (f a) (f b)

```

```

proof
  show closed_segment (f a) (f b)  $\subseteq$  f ' closed_segment a b
    by (metis subset_continuous_image_segment_1 contf)
  show f ' closed_segment a b  $\subseteq$  closed_segment (f a) (f b)
  proof (cases a = b)
    case True
    then show ?thesis by auto
  next
    case False
    then have fnot: f a  $\neq$  f b
      using inj_onD injf by fastforce
    moreover
    have f a  $\notin$  open_segment (f c) (f b) if c: c  $\in$  closed_segment a b for c
    proof (clarsimp simp add: open_segment_def)
      assume fa: f a  $\in$  closed_segment (f c) (f b)
      moreover have closed_segment (f c) (f b)  $\subseteq$  f ' closed_segment c b
      by (meson closed_segment_subset contf continuous_on_subset convex_closed_segment
ends_in_segment(2) subset_continuous_image_segment_1 that)
      ultimately have f a  $\in$  f ' closed_segment c b
      by blast
      then have a: a  $\in$  closed_segment c b
      by (meson ends_in_segment inj_on_image_mem_iff injf subset_closed_segment
that)
      have cb: closed_segment c b  $\subseteq$  closed_segment a b
      by (simp add: closed_segment_subset that)
      show f a = f c
      proof (rule between_antisym)
        show between (f c, f b) (f a)
          by (simp add: between_mem_segment fa)
        show between (f a, f b) (f c)
          by (metis a cb between_antisym between_mem_segment between_triv1
subset_iff)
      qed
    qed
    moreover
    have f b  $\notin$  open_segment (f a) (f c) if c: c  $\in$  closed_segment a b for c
    proof (clarsimp simp add: open_segment_def fnot eq_commute)
      assume fb: f b  $\in$  closed_segment (f a) (f c)
      moreover have closed_segment (f a) (f c)  $\subseteq$  f ' closed_segment a c
      by (meson contf continuous_on_subset ends_in_segment(1) subset_closed_segment
subset_continuous_image_segment_1 that)
      ultimately have f b  $\in$  f ' closed_segment a c
      by blast
      then have b: b  $\in$  closed_segment a c
      by (meson ends_in_segment inj_on_image_mem_iff injf subset_closed_segment
that)
      have ca: closed_segment a c  $\subseteq$  closed_segment a b
      by (simp add: closed_segment_subset that)
      show f b = f c

```

```

proof (rule between_antisym)
  show between (f c, f a) (f b)
    by (simp add: between_commute between_mem_segment fb)
  show between (f b, f a) (f c)
    by (metis b between_antisym between_commute between_mem_segment
between_triv2 that)
  qed
qed
ultimately show ?thesis
  by (force simp: closed_segment_eq_real_ivl open_segment_eq_real_ivl split:
if_split_asm)
  qed
qed

```

```

lemma continuous_injective_image_open_segment_1:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes contf: continuous_on (closed_segment a b) f
    and inj: inj_on f (closed_segment a b)
  shows f ` (open_segment a b) = open_segment (f a) (f b)
proof -
  have f ` (open_segment a b) = f ` (closed_segment a b) - {f a, f b}
    by (metis (no_types, opaque_lifting) empty_subsetI ends_in_segment im-
age_insert image_is_empty inj_on_image_set_diff inj insert_subset open_segment_def
segment_open_subset_closed)
  also have ... = open_segment (f a) (f b)
    using continuous_injective_image_segment_1 [OF assms]
    by (simp add: open_segment_def inj_on_image_set_diff [OF inj])
  finally show ?thesis .
qed

```

```

lemma collinear_imp_coplanar:
  collinear s ==> coplanar s
by (metis collinear_affine_hull coplanar_def insert_absorb2)

```

```

lemma collinear_small:
  assumes finite s card s  $\leq$  2
  shows collinear s
proof -
  have card s = 0  $\vee$  card s = 1  $\vee$  card s = 2
    using assms by linarith
  then show ?thesis using assms
    using card_eq_SucD numeral_2_eq_2 by (force simp: card_1_singleton_iff)
qed

```

```

lemma coplanar_small:
  assumes finite s card s  $\leq$  3
  shows coplanar s
proof -
  consider card s  $\leq$  2 | card s = Suc (Suc (Suc 0))

```



```

    using assms by linarith
  then show ?thesis
  proof cases
    case 1
    then show ?thesis
    by (simp add: ⟨finite s⟩ collinear_imp_coplanar collinear_small)
  next
    case 2
    then show ?thesis
    using hull_subset [of {_,_,_}]
    by (fastforce simp: coplanar_def dest!: card_eq_SucD)
  qed
qed

lemma coplanar_empty: coplanar {}
  by (simp add: coplanar_small)

lemma coplanar_sing: coplanar {a}
  by (simp add: coplanar_small)

lemma coplanar_2: coplanar {a,b}
  by (auto simp: card_insert_if coplanar_small)

lemma coplanar_3: coplanar {a,b,c}
  by (auto simp: card_insert_if coplanar_small)

lemma collinear_affine_hull_collinear: collinear (affine hull s)  $\longleftrightarrow$  collinear s
  unfolding collinear_affine_hull
  by (metis affine_affine_hull subset_hull hull_hull hull_mono)

lemma coplanar_affine_hull_coplanar: coplanar (affine hull s)  $\longleftrightarrow$  coplanar s
  unfolding coplanar_def
  by (metis affine_affine_hull subset_hull hull_hull hull_mono)

lemma coplanar_linear_image:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::real_normed_vector
  assumes coplanar S linear f shows coplanar (f ` S)
proof -
  { fix u v w
    assume S  $\subseteq$  affine hull {u, v, w}
    then have f ` S  $\subseteq$  f ` (affine hull {u, v, w})
      by (simp add: image_mono)
    then have f ` S  $\subseteq$  affine hull (f ` {u, v, w})
      by (metis assms(2) linear_conv_bounded_linear affine_hull_linear_image)
  } then
  show ?thesis
  by auto (meson assms(1) coplanar_def)
qed

```

```

lemma coplanar_translation_imp:
  assumes coplanar S shows coplanar (( $\lambda x. a + x$ ) ' S)
proof -
  obtain u v w where  $S \subseteq \text{affine hull } \{u, v, w\}$ 
  by (meson assms coplanar_def)
  then have (+)  $a ' S \subseteq \text{affine hull } \{u + a, v + a, w + a\}$ 
  using affine_hull_translation [of a {u,v,w}] for u v w
  by (force simp: add.commute)
  then show ?thesis
  unfolding coplanar_def by blast
qed

lemma coplanar_translation_eq:  $\text{coplanar}((\lambda x. a + x) ' S) \longleftrightarrow \text{coplanar } S$ 
  by (metis (no_types) coplanar_translation_imp translation_galois)

lemma coplanar_linear_image_eq:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes linear f inj f shows  $\text{coplanar}(f ' S) = \text{coplanar } S$ 
proof
  assume coplanar S
  then show coplanar (f ' S)
  using assms(1) coplanar_linear_image by blast
next
  obtain g where  $g: \text{linear } g \ g \circ f = \text{id}$ 
  using linear_injective_left_inverse [OF assms]
  by blast
  assume coplanar (f ' S)
  then show coplanar S
  by (metis coplanar_linear_image g(1) g(2) id_apply image_comp image_id)
qed

lemma coplanar_subset:  $\llbracket \text{coplanar } t; S \subseteq t \rrbracket \Longrightarrow \text{coplanar } S$ 
  by (meson coplanar_def order_trans)

lemma affine_hull_3_imp_collinear:  $c \in \text{affine hull } \{a, b\} \Longrightarrow \text{collinear } \{a, b, c\}$ 
  by (metis collinear_2 collinear_affine_hull_collinear hull_redundant_insert_commute)

lemma collinear_3_imp_in_affine_hull:
  assumes collinear {a,b,c}  $a \neq b$  shows  $c \in \text{affine hull } \{a, b\}$ 
proof -
  obtain u x y where  $b - a = y *_R u$   $c - a = x *_R u$ 
  using assms unfolding collinear_def by auto
  with  $\langle a \neq b \rangle$  have  $\exists v. c = (1 - x / y) *_R a + v *_R b \wedge 1 - x / y + v = 1$ 
  by (simp add: algebra_simps)
  then show ?thesis
  by (simp add: hull_inc mem_affine)
qed

lemma collinear_3_affine_hull:

```

```

    assumes  $a \neq b$ 
    shows  $\text{collinear } \{a, b, c\} \longleftrightarrow c \in \text{affine hull } \{a, b\}$ 
    using  $\text{affine\_hull\_3\_imp\_collinear}$   $\text{assms collinear\_3\_imp\_in\_affine\_hull}$  by
    blast

lemma  $\text{collinear\_3\_eq\_affine\_dependent}$ :
   $\text{collinear}\{a, b, c\} \longleftrightarrow a = b \vee a = c \vee b = c \vee \text{affine\_dependent } \{a, b, c\}$ 
proof (cases  $a = b \vee a = c \vee b = c$ )
  case True
  then show ?thesis
    by (auto simp: insert_commute)
next
  case False
  then have  $\text{collinear}\{a, b, c\}$  if  $\text{affine\_dependent } \{a, b, c\}$ 
    using that unfolding  $\text{affine\_dependent\_def}$ 
    by (auto simp: insert_Diff_if;metis  $\text{affine\_hull\_3\_imp\_collinear insert\_commute}$ )
  moreover
  have  $\text{affine\_dependent } \{a, b, c\}$  if  $\text{collinear}\{a, b, c\}$ 
    using False that by (auto simp:  $\text{affine\_dependent\_def collinear\_3\_affine\_hull insert\_Diff\_if}$ )
  ultimately
  show ?thesis
    using False by blast
qed

lemma  $\text{affine\_dependent\_imp\_collinear\_3}$ :
   $\text{affine\_dependent } \{a, b, c\} \implies \text{collinear}\{a, b, c\}$ 
  by (simp add:  $\text{collinear\_3\_eq\_affine\_dependent}$ )

lemma  $\text{collinear\_3}$ :  $\text{NO\_MATCH } 0 \ x \implies \text{collinear } \{x, y, z\} \longleftrightarrow \text{collinear } \{0, x - y, z - y\}$ 
  by (auto simp add:  $\text{collinear\_def}$ )

lemma  $\text{collinear\_3\_expand}$ :
   $\text{collinear}\{a, b, c\} \longleftrightarrow a = c \vee (\exists u. b = u *_R a + (1 - u) *_R c)$ 
proof -
  have  $\text{collinear}\{a, b, c\} = \text{collinear}\{a, c, b\}$ 
    by (simp add: insert_commute)
  also have  $\dots = \text{collinear } \{0, a - c, b - c\}$ 
    by (simp add:  $\text{collinear\_3}$ )
  also have  $\dots \longleftrightarrow (a = c \vee b = c \vee (\exists ca. b - c = ca *_R (a - c)))$ 
    by (simp add:  $\text{collinear\_lemma}$ )
  also have  $\dots \longleftrightarrow a = c \vee (\exists u. b = u *_R a + (1 - u) *_R c)$ 
    by (cases  $a = c \vee b = c$ ) (auto simp:  $\text{algebra\_simps}$ )
  finally show ?thesis .
qed

lemma  $\text{collinear\_aff\_dim}$ :  $\text{collinear } S \longleftrightarrow \text{aff\_dim } S \leq 1$ 
proof

```

```

    assume collinear S
    then obtain u and v :: 'a where aff_dim S ≤ aff_dim {u,v}
    by (metis ‹collinear S› aff_dim_affine_hull aff_dim_subset collinear_affine_hull)
    then show aff_dim S ≤ 1
    using order_trans by fastforce
next
    assume aff_dim S ≤ 1
    then have le1: aff_dim (affine hull S) ≤ 1
    by simp
    obtain B where B ⊆ S and B: ¬ affine_dependent B affine_hull S = affine_hull B
    using affine_basis_exists [of S] by auto
    then have finite B card B ≤ 2
    using B le1 by (auto simp: affine_independent_iff_card)
    then have collinear B
    by (rule collinear_small)
    then show collinear S
    by (metis ‹affine_hull S = affine_hull B› collinear_affine_hull_collinear)
qed

```

```

lemma collinear_midpoint: collinear{a, midpoint a b, b}
proof -
  have §: ‹a ≠ midpoint a b; b - midpoint a b ≠ - 1 *R (a - midpoint a b)› ⇒
  b = midpoint a b
  by (simp add: algebra_simps)
  show ?thesis
  by (auto simp: collinear_3 collinear_lemma intro: §)
qed

```

```

lemma midpoint_collinear:
  fixes a b c :: 'a::real_normed_vector
  assumes a ≠ c
  shows b = midpoint a c ⇔ collinear{a,b,c} ∧ dist a b = dist b c
proof -
  have *: a - (u *R a + (1 - u) *R c) = (1 - u) *R (a - c)
    u *R a + (1 - u) *R c - c = u *R (a - c)
    |1 - u| = |u| ⇔ u = 1/2 for u::real
  by (auto simp: algebra_simps)
  have b = midpoint a c ⇒ collinear{a,b,c}
  using collinear_midpoint by blast
  moreover have b = midpoint a c ⇔ dist a b = dist b c if collinear{a,b,c}
  proof -
    consider a = c | u where b = u *R a + (1 - u) *R c
    using ‹collinear {a,b,c}› unfolding collinear_3_expand by blast
    then show ?thesis
  proof cases
    case 2
    with assms have dist a b = dist b c ⇒ b = midpoint a c
    by (simp add: dist_norm * midpoint_def scaleR_add_right del: divide_const_simps)

```

```

    then show ?thesis
      by (auto simp: dist_midpoint)
    qed (use assms in auto)
  qed
  ultimately show ?thesis by blast
qed

lemma between_imp_collinear:
  fixes x :: 'a :: euclidean_space
  assumes between (a,b) x
  shows collinear {a,x,b}
proof (cases x = a  $\vee$  x = b  $\vee$  a = b)
  case True with assms show ?thesis
    by (auto simp: dist_commute)
  next
  case False
  then have False if  $\bigwedge c. b - x \neq c *_R (a - x)$ 
    using that [of  $-(\text{norm}(b - x) / \text{norm}(x - a))$ ] assms
    by (simp add: between_norm vector_add_divide_simps flip: real_vector.scale_minus_right)
  then show ?thesis
    by (auto simp: collinear_3 collinear_lemma)
qed

lemma midpoint_between:
  fixes a b :: 'a::euclidean_space
  shows b = midpoint a c  $\longleftrightarrow$  between (a,c) b  $\wedge$  dist a b = dist b c
proof (cases a = c)
  case False
  show ?thesis
    using False between_imp_collinear between_midpoint(1) midpoint_collinear
  by blast
qed (auto simp: dist_commute)

lemma collinear_triples:
  assumes a  $\neq$  b
  shows collinear(insert a (insert b S))  $\longleftrightarrow$  ( $\forall x \in S. \text{collinear}\{a,b,x\}$ )
    (is ?lhs = ?rhs)
proof safe
  fix x
  assume ?lhs and x  $\in$  S
  then show collinear {a, b, x}
    using collinear_subset by force
  next
  assume ?rhs
  then have  $\forall x \in S. \text{collinear}\{a,x,b\}$ 
    by (simp add: insert_commute)
  then have *:  $\exists u. x = u *_R a + (1 - u) *_R b$  if x  $\in$  insert a (insert b S) for x
    using that assms collinear_3_expand by fastforce+
  have  $\exists c. x - y = c *_R (b - a)$ 

```

```

    if  $x: x \in \text{insert } a (\text{insert } b S)$  and  $y: y \in \text{insert } a (\text{insert } b S)$  for  $x y$ 
  proof -
    obtain  $u v$  where  $x = u *_R a + (1 - u) *_R b$   $y = v *_R a + (1 - v) *_R b$ 
    using  $* x y$  by presburger
    then have  $x - y = (v - u) *_R (b - a)$ 
    by (simp add: scale_left_diff_distrib scale_right_diff_distrib)
    then show ?thesis ..
  qed
  then show ?lhs
    unfolding collinear_def by metis
qed

```

```

lemma collinear_4_3:
  assumes  $a \neq b$ 
  shows  $\text{collinear } \{a, b, c, d\} \longleftrightarrow \text{collinear } \{a, b, c\} \wedge \text{collinear } \{a, b, d\}$ 
  using collinear_triples [OF assms, of  $\{c, d\}$ ] by (force simp:)

```

```

lemma collinear_3_trans:
  assumes  $\text{collinear } \{a, b, c\}$   $\text{collinear } \{b, c, d\}$   $b \neq c$ 
  shows  $\text{collinear } \{a, b, d\}$ 
proof -
  have  $\text{collinear } \{b, c, a, d\}$ 
  by (metis (full_types) assms collinear_4_3 insert_commute)
  then show ?thesis
    by (simp add: collinear_subset)
qed

```

```

lemma affine_hull_2_alt:
  fixes  $a b :: 'a::\text{real\_vector}$ 
  shows  $\text{affine\_hull } \{a, b\} = \text{range } (\lambda u. a + u *_R (b - a))$ 
proof -
  have  $1: u *_R a + v *_R b = a + v *_R (b - a)$  if  $u + v = 1$  for  $u v$ 
  using that
  by (simp add: algebra_simps flip: scaleR_add_left)
  have  $2: a + u *_R (b - a) = (1 - u) *_R a + u *_R b$  for  $u$ 
  by (auto simp: algebra_simps)
  show ?thesis
    by (force simp add: affine_hull_2 dest: 1 intro!: 2)
qed

```

```

lemma interior_convex_hull_3_minimal:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  assumes  $\neg \text{collinear } \{a, b, c\}$  and  $2: \text{DIM}('a) = 2$ 
  shows  $\text{interior}(\text{convex\_hull } \{a, b, c\}) =$ 
     $\{v. \exists x y z. 0 < x \wedge 0 < y \wedge 0 < z \wedge x + y + z = 1 \wedge x *_R a + y *_R b$ 
     $+ z *_R c = v\}$ 
    (is ?lhs = ?rhs)
proof
  have abc:  $a \neq b \wedge a \neq c \wedge b \neq c \rightarrow \text{affine\_dependent } \{a, b, c\}$ 

```

```

    using assms by (auto simp: collinear_3_eq_affine_dependent)
  with 2 show ?lhs  $\subseteq$  ?rhs
    by (fastforce simp add: interior_convex_hull_explicit_minimal)
  show ?rhs  $\subseteq$  ?lhs
    using abc 2
    apply (clarsimp simp add: interior_convex_hull_explicit_minimal)
    subgoal for x y z
      by (rule_tac x= $\lambda r.$  (if  $r=a$  then  $x$  else if  $r=b$  then  $y$  else if  $r=c$  then  $z$  else
0) in exI) auto
    done
qed

```

5.0.9 Basic lemmas about hyperplanes and halfspaces

lemma *halfspace_Int_eq*:

```

  {x.  $a \cdot x \leq b$ }  $\cap$  {x.  $b \leq a \cdot x$ } = {x.  $a \cdot x = b$ }
  {x.  $b \leq a \cdot x$ }  $\cap$  {x.  $a \cdot x \leq b$ } = {x.  $a \cdot x = b$ }
  by auto

```

lemma *hyperplane_eq_Ex*:

```

  assumes  $a \neq 0$  obtains x where  $a \cdot x = b$ 
  by (rule_tac x = (b / (a · a)) *R a in that) (simp add: assms)

```

lemma *hyperplane_eq_empty*:

```

  {x.  $a \cdot x = b$ } = {}  $\longleftrightarrow$   $a = 0 \wedge b \neq 0$ 
  using hyperplane_eq_Ex
  by (metis (mono_tags, lifting) empty_Collect_eq inner_zero_left)

```

lemma *hyperplane_eq_UNIV*:

```

  {x.  $a \cdot x = b$ } = UNIV  $\longleftrightarrow$   $a = 0 \wedge b = 0$ 
  proof -
    have  $a = 0 \wedge b = 0$  if UNIV  $\subseteq$  {x.  $a \cdot x = b$ }
      using subsetD [OF that, where  $c = ((b+1) / (a \cdot a)) *_{\mathbb{R}} a$ ]
      by (simp add: field_split_simps split: if_split_asm)
    then show ?thesis by force
  qed

```

lemma *halfspace_eq_empty_lt*:

```

  {x.  $a \cdot x < b$ } = {}  $\longleftrightarrow$   $a = 0 \wedge b \leq 0$ 
  proof -
    have  $a = 0 \wedge b \leq 0$  if {x.  $a \cdot x < b$ }  $\subseteq$  {}
      using subsetD [OF that, where  $c = ((b-1) / (a \cdot a)) *_{\mathbb{R}} a$ ]
      by (force simp add: field_split_simps split: if_split_asm)
    then show ?thesis by force
  qed

```

lemma *halfspace_eq_empty_gt*:

```

  {x.  $a \cdot x > b$ } = {}  $\longleftrightarrow$   $a = 0 \wedge b \geq 0$ 
  using halfspace_eq_empty_lt [of  $-a -b$ ]

```

by simp

lemma *halfspace_eq_empty_le*:

$\{x. a \cdot x \leq b\} = \{\} \longleftrightarrow a = 0 \wedge b < 0$

proof -

have $a = 0 \wedge b < 0$ if $\{x. a \cdot x \leq b\} \subseteq \{\}$

using subsetD [OF that, where $c = ((b-1) / (a \cdot a)) *_{\mathbb{R}} a$]

by (force simp add: field_split_simps split: if_split_asm)

then show ?thesis by force

qed

lemma *halfspace_eq_empty_ge*:

$\{x. a \cdot x \geq b\} = \{\} \longleftrightarrow a = 0 \wedge b > 0$

using halfspace_eq_empty_le [of $-a -b$] by simp

5.0.10 Use set distance for an easy proof of separation properties

proposition *separation_closures*:

fixes $S :: 'a::euclidean\_space$ set

assumes $S \cap \text{closure } T = \{\}$ $T \cap \text{closure } S = \{\}$

obtains $U \ V$ where $U \cap V = \{\}$ open U open V $S \subseteq U$ $T \subseteq V$

proof (cases $S = \{\} \vee T = \{\}$)

case True with that show ?thesis by auto

next

case False

define f where $f \equiv \lambda x. \text{setdist } \{x\} T - \text{setdist } \{x\} S$

have $\text{contf}: \text{continuous\_on } \text{UNIV } f$

unfolding f_def by (intro continuous_intros continuous_on_setdist)

show ?thesis

proof (rule_tac $U = \{x. f x > 0\}$ and $V = \{x. f x < 0\}$ in that)

show $\{x. 0 < f x\} \cap \{x. f x < 0\} = \{\}$

by auto

show open $\{x. 0 < f x\}$

by (simp add: open_Collect_less contf)

show open $\{x. f x < 0\}$

by (simp add: open_Collect_less contf)

have $\bigwedge x. x \in S \implies \text{setdist } \{x\} T \neq 0 \wedge x. x \in T \implies \text{setdist } \{x\} S \neq 0$

by (meson False assms disjoint_iff setdist_eq_0_sing_1)+

then show $S \subseteq \{x. 0 < f x\}$ $T \subseteq \{x. f x < 0\}$

using less_eq_real_def by (fastforce simp add: f_def setdist_sing_in_set)+

qed

qed

lemma *separation_normal*:

fixes $S :: 'a::euclidean\_space$ set

assumes closed S closed T $S \cap T = \{\}$

obtains $U \ V$ where open U open V $S \subseteq U$ $T \subseteq V$ $U \cap V = \{\}$

using separation_closures [of $S \ T$]

by (metis assms closure_closed disjnt_def inf_commute)

lemma separation_normal_local:

```

  fixes S :: 'a::euclidean_space set
  assumes US: closedin (top_of_set U) S
    and UT: closedin (top_of_set U) T
    and S ∩ T = {}
  obtains S' T' where openin (top_of_set U) S'
    openin (top_of_set U) T'
    S ⊆ S' T ⊆ T' S' ∩ T' = {}
proof (cases S = {} ∨ T = {})
  case True with that show ?thesis
    using UT US by (blast dest: closedin_subset)
next
  case False
  define f where f ≡ λx. setdist {x} T - setdist {x} S
  have conf: continuous_on U f
    unfolding f_def by (intro continuous_intros)
  show ?thesis
  proof (rule_tac S' = (U ∩ f - ' {0<..}) and T' = (U ∩ f - ' {..<0}) in that)
    show (U ∩ f - ' {0<..}) ∩ (U ∩ f - ' {..<0}) = {}
    by auto
    show openin (top_of_set U) (U ∩ f - ' {0<..})
      by (rule continuous_openin_preimage [where T=UNIV]) (simp_all add:
conf)
  next
    show openin (top_of_set U) (U ∩ f - ' {..<0})
      by (rule continuous_openin_preimage [where T=UNIV]) (simp_all add:
conf)
  next
    have S ⊆ U T ⊆ U
      using closedin_imp_subset assms by blast+
    then show S ⊆ U ∩ f - ' {0<..} T ⊆ U ∩ f - ' {..<0}
      using assms False by (force simp add: f_def setdistSing_in_set intro!:
setdist_gt_0_closedin)+
  qed
qed

```

lemma separation_normal_compact:

```

  fixes S :: 'a::euclidean_space set
  assumes compact S closed T S ∩ T = {}
  obtains U V where open U compact(closure U) open V S ⊆ U T ⊆ V U ∩ V
= {}
proof -
  have closed S bounded S
    using assms by (auto simp: compact_eq_bounded_closed)
  then obtain r where r>0 and r: S ⊆ ball 0 r
    by (auto dest!: bounded_subset_ballD)
  have **: closed (T ∪ - ball 0 r) S ∩ (T ∪ - ball 0 r) = {}

```

```

    using assms r by blast+
  then obtain U V where UV: open U open V S ⊆ U T ∪ - ball 0 r ⊆ V U ∩
V = {}
    by (meson ⟨closed S⟩ separation_normal)
  then have compact(closure U)
    by (meson bounded_ball bounded_subset compact_closure compl_le_swap2
disjoint_eq_subset_Compl le_sup_iff)
  with UV show thesis
    using that by auto
qed

```

5.0.11 Connectedness of the intersection of a chain

proposition *connected_chain*:

```

  fixes F :: 'a :: euclidean_space set set
  assumes cc:  $\bigwedge S. S \in \mathcal{F} \implies \text{compact } S \wedge \text{connected } S$ 
    and linear:  $\bigwedge S T. S \in \mathcal{F} \wedge T \in \mathcal{F} \implies S \subseteq T \vee T \subseteq S$ 
  shows connected( $\bigcap \mathcal{F}$ )
proof (cases F = {})
  case True then show ?thesis
    by auto
next
  case False
  then have cf: compact( $\bigcap \mathcal{F}$ )
    by (simp add: cc compact_Inter)
  have False if AB: closed A closed B A ∩ B = {}
    and ABeq: A ∪ B =  $\bigcap \mathcal{F}$  and A ≠ {} B ≠ {} for A B
  proof -
    obtain U V where open U open V A ⊆ U B ⊆ V U ∩ V = {}
      using separation_normal [OF AB] by metis
    obtain K where K ∈ F compact K
      using cc False by blast
    then obtain N where open N and K ⊆ N
      by blast
    let ?C = insert (U ∪ V) ((λS. N - S) ' F)
    obtain D where D ⊆ ?C finite D K ⊆  $\bigcup D$ 
      proof (rule compactE [OF ⟨compact K⟩])
        show K ⊆  $\bigcup (\text{insert } (U \cup V) ((-) N ' \mathcal{F}))$ 
          using ⟨K ⊆ N⟩ ABeq ⟨A ⊆ U⟩ ⟨B ⊆ V⟩ by auto
        show  $\bigwedge B. B \in \text{insert } (U \cup V) ((-) N ' \mathcal{F}) \implies \text{open } B$ 
          by (auto simp: ⟨open U⟩ ⟨open V⟩ open_Un ⟨open N⟩ cc compact_imp_closed
open_Diff)
      qed
    then have finite(D - {U ∪ V})
      by blast
    moreover have D - {U ∪ V} ⊆ (λS. N - S) ' F
      using ⟨D ⊆ ?C⟩ by blast
    ultimately obtain G where G ⊆ F finite G and Deq: D - {U ∪ V} = (λS.
N - S) ' G

```

```

    using finite_subset_image by metis
  obtain J where J ∈  $\mathcal{F}$  and J:  $(\bigcup S \in \mathcal{G}. N - S) \subseteq N - J$ 
  proof (cases  $\mathcal{G} = \{\}$ )
    case True
    with  $\langle \mathcal{F} \neq \{\} \rangle$  that show ?thesis
    by auto
  next
  case False
  have  $\bigwedge S T. \llbracket S \in \mathcal{G}; T \in \mathcal{G} \rrbracket \implies S \subseteq T \vee T \subseteq S$ 
    by (meson  $\langle \mathcal{G} \subseteq \mathcal{F} \rangle$  in_mono local.linear)
  with  $\langle \text{finite } \mathcal{G} \rangle \langle \mathcal{G} \neq \{\} \rangle$ 
  have  $\exists J \in \mathcal{G}. (\bigcup S \in \mathcal{G}. N - S) \subseteq N - J$ 
  proof induction
    case (insert X  $\mathcal{H}$ )
    show ?case
    proof (cases  $\mathcal{H} = \{\}$ )
      case True then show ?thesis by auto
    next
    case False
    then have  $\bigwedge S T. \llbracket S \in \mathcal{H}; T \in \mathcal{H} \rrbracket \implies S \subseteq T \vee T \subseteq S$ 
      by (simp add: insert.premis)
    with insert.IH False obtain J where J ∈  $\mathcal{H}$  and J:  $(\bigcup Y \in \mathcal{H}. N - Y) \subseteq N - J$ 
      by metis
    have  $N - J \subseteq N - X \vee N - X \subseteq N - J$ 
      by (meson Diff_mono  $\langle J \in \mathcal{H} \rangle$  insert.premis(2) insert_iff order_refl)
    then show ?thesis
    proof
      assume  $N - J \subseteq N - X$  with J show ?thesis
      by auto
    next
      assume  $N - X \subseteq N - J$ 
      with J have  $N - X \cup \bigcup ((-) N \setminus \mathcal{H}) \subseteq N - J$ 
      by auto
      with  $\langle J \in \mathcal{H} \rangle$  show ?thesis
      by blast
    qed
  qed
  qed simp
  with  $\langle \mathcal{G} \subseteq \mathcal{F} \rangle$  show ?thesis by (blast intro: that)
  qed
  have  $K \subseteq \bigcup (\text{insert } (U \cup V) (\mathcal{D} - \{U \cup V\}))$ 
    using  $\langle K \subseteq \bigcup \mathcal{D} \rangle$  by auto
  also have  $\dots \subseteq (U \cup V) \cup (N - J)$ 
    by (metis (no_types, opaque_lifting) Deq Un_subset_iff Un_upper2 J Union_insert
    order_trans sup_ge1)
  finally have  $J \cap K \subseteq U \cup V$ 
    by blast
  moreover have connected( $J \cap K$ )

```

```

    by (metis Int_absorb1 ⟨J ∈ F⟩ ⟨K ∈ F⟩ cc inf.orderE local.linear)
  moreover have U ∩ (J ∩ K) ≠ {}
    using ABeq ⟨J ∈ F⟩ ⟨K ∈ F⟩ ⟨A ≠ {}⟩ ⟨A ⊆ U⟩ by blast
  moreover have V ∩ (J ∩ K) ≠ {}
    using ABeq ⟨J ∈ F⟩ ⟨K ∈ F⟩ ⟨B ≠ {}⟩ ⟨B ⊆ V⟩ by blast
  ultimately show False
    using connectedD [of J ∩ K U V] ⟨open U⟩ ⟨open V⟩ ⟨U ∩ V = {}⟩ by
auto
qed
with cf show ?thesis
  by (auto simp: connected_closed_set compact_imp_closed)
qed

```

```

lemma connected_chain_gen:
  fixes F :: 'a :: euclidean_space set set
  assumes X: X ∈ F compact X
    and cc: ∧T. T ∈ F ⟹ closed T ∧ connected T
    and linear: ∧S T. S ∈ F ∧ T ∈ F ⟹ S ⊆ T ∨ T ⊆ S
  shows connected(⋂F)
proof -
  have ⋂F = (⋂T∈F. X ∩ T)
    using X by blast
  moreover have connected (⋂T∈F. X ∩ T)
  proof (rule connected_chain)
    show ∧T. T ∈ (⋂) X ' F ⟹ compact T ∧ connected T
      using cc X by auto (metis inf.absorb2 inf.orderE local.linear)
    show ∧S T. S ∈ (⋂) X ' F ∧ T ∈ (⋂) X ' F ⟹ S ⊆ T ∨ T ⊆ S
      using local.linear by blast
  qed
  ultimately show ?thesis
    by metis
qed

```

```

lemma connected_nest:
  fixes S :: 'a::linorder ⇒ 'b::euclidean_space set
  assumes S: ∧n. compact(S n) ∧n. connected(S n)
    and nest: ∧m n. m ≤ n ⟹ S n ⊆ S m
  shows connected(⋂ (range S))
proof (rule connected_chain)
  show ∧A T. A ∈ range S ∧ T ∈ range S ⟹ A ⊆ T ∨ T ⊆ A
    by (metis image_iff le_cases nest)
qed (use S in blast)

```

```

lemma connected_nest_gen:
  fixes S :: 'a::linorder ⇒ 'b::euclidean_space set
  assumes S: ∧n. closed(S n) ∧n. connected(S n) compact(S n)
    and nest: ∧m n. m ≤ n ⟹ S n ⊆ S m
  shows connected(⋂ (range S))
proof (rule connected_chain_gen [of S k])

```

```

show  $\bigwedge A T. A \in \text{range } S \wedge T \in \text{range } S \implies A \subseteq T \vee T \subseteq A$ 
  by (metis imageE le_cases nest)
qed (use S in auto)

```

5.0.12 Proper maps, including projections out of compact sets

```

lemma finite_indexed_bound:
  assumes A: finite A  $\bigwedge x. x \in A \implies \exists n::'a::\text{linorder}. P x n$ 
  shows  $\exists m. \forall x \in A. \exists k \leq m. P x k$ 
using A
proof (induction A)
  case empty then show ?case by force
next
  case (insert a A)
  then obtain m n where  $\forall x \in A. \exists k \leq m. P x k$  P a n
  by force
  then show ?case
  by (metis dual_order.trans insert_iff le_cases)
qed

proposition proper_map:
  fixes f :: 'a::heine_borel  $\Rightarrow$  'b::heine_borel
  assumes closedin (top_of_set S) K
    and com:  $\bigwedge U. [U \subseteq T; \text{compact } U] \implies \text{compact } (S \cap f^{-1} U)$ 
    and f' S  $\subseteq T$ 
  shows closedin (top_of_set T) (f' K)
proof -
  have K  $\subseteq S$ 
  using assms closedin_imp_subset by metis
  obtain C where closed C and Keq:  $K = S \cap C$ 
  using assms by (auto simp: closedin_closed)
  have *:  $y \in f^{-1} K$  if  $y \in T$  and  $y: y \text{ islimpt } f^{-1} K$  for y
  proof -
    obtain h where  $\forall n. (\exists x \in K. h n = f x) \wedge h n \neq y \text{ inj } h$  and hlim:  $(h \longrightarrow y)$  sequentially
    using  $\langle y \in T \rangle y$  by (force simp: limpt_sequential_inj)
    then obtain X where X:  $\bigwedge n. X n \in K \wedge h n = f (X n) \wedge h n \neq y$ 
    by metis
    then have fX:  $\bigwedge n. f (X n) = h n$ 
    by metis
    define  $\Psi$  where  $\Psi \equiv \lambda n. \{a \in K. f a \in \text{insert } y (\text{range } (\lambda i. f (X (n + i))))\}$ 
    have compact  $(C \cap (S \cap f^{-1} \text{insert } y (\text{range } (\lambda i. f (X (n + i))))))$  for n
    proof (intro closed_Int_compact [OF  $\langle \text{closed } C \rangle \text{com}$ ] compact_sequence_with_limit)
      show  $\text{insert } y (\text{range } (\lambda i. f (X (n + i)))) \subseteq T$ 
      using X  $\langle K \subseteq S \rangle \langle f^{-1} S \subseteq T \rangle \langle y \in T \rangle$  by blast
      show  $(\lambda i. f (X (n + i))) \longrightarrow y$ 
      by (simp add: fX add.commute [of n] LIMSEQ_ignore_initial_segment [OF hlim])
    qed
  qed

```

```

qed
then have comf: compact ( $\Psi$   $n$ ) for  $n$ 
  by (simp add: Keq Int_def  $\Psi$ _def conj_commute)
have ne:  $\bigcap \mathcal{F} \neq \{\}$ 
  if finite  $\mathcal{F}$ 
  and  $\mathcal{F}$ :  $\bigwedge t. t \in \mathcal{F} \implies (\exists n. t = \Psi\ n)$ 
  for  $\mathcal{F}$ 
proof -
  obtain  $m$  where  $m$ :  $\bigwedge t. t \in \mathcal{F} \implies \exists k \leq m. t = \Psi\ k$ 
  by (rule exE [OF finite_indexed_bound [OF  $\langle$ finite  $\mathcal{F}$  $\rangle$   $\mathcal{F}$ ]], force+)
  have  $X\ m \in \bigcap \mathcal{F}$ 
  using  $X$  le_Suc_ex by (fastforce simp:  $\Psi$ _def dest:  $m$ )
  then show ?thesis by blast
qed
have ( $\bigcap n. \Psi\ n$ )  $\neq \{\}$ 
proof (rule compact_fip_Heine_Borel)
  show  $\bigwedge \mathcal{F}'. \llbracket \text{finite } \mathcal{F}'; \mathcal{F}' \subseteq \text{range } \Psi \rrbracket \implies \bigcap \mathcal{F}' \neq \{\}$ 
  by (meson ne rangeE subset_eq)
qed (use comf in blast)
then obtain  $x$  where  $x \in K \bigwedge n. (f\ x = y \vee (\exists u. f\ x = h\ (n + u)))$ 
  by (force simp add:  $\Psi$ _def fX)
then show ?thesis
  unfolding image_iff by (metis  $\langle$ inj  $h$  $\rangle$  le_add1 not_less_eq_eq rangeI
range_ex1_eq)
qed
with asms closedin_subset show ?thesis
  by (force simp: closedin_limpt)
qed

```

```

lemma compact_continuous_image_eq:
  fixes  $f :: 'a::\text{heine\_borel} \Rightarrow 'b::\text{heine\_borel}$ 
  assumes  $f$ :  $\text{inj\_on } f\ S$ 
  shows  $\text{continuous\_on } S\ f \longleftrightarrow (\forall T. \text{compact } T \wedge T \subseteq S \longrightarrow \text{compact}(f\ 'T))$ 
  (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
    by (metis continuous_on_subset compact_continuous_image)
next
  assume RHS: ?rhs
  obtain  $g$  where  $gf$ :  $\bigwedge x. x \in S \implies g\ (f\ x) = x$ 
  by (metis inv_into_f_f)
  then have *:  $(S \cap f\ -' U) = g\ ' U$  if  $U \subseteq f\ ' S$  for  $U$ 
  using that by fastforce
  have  $gfm$ :  $g\ ' f\ ' S \subseteq S$  using  $gf$  by auto
  have **:  $\text{compact } (f\ ' S \cap g\ -' C)$  if  $C$ :  $C \subseteq S$  compact  $C$  for  $C$ 
  proof -
    obtain  $h$  where  $h\ C \in C \wedge h\ C \notin S \vee \text{compact } (f\ ' C)$ 
    by (force simp:  $C$  RHS)
  
```

```

    moreover have  $f \cdot C = (f \cdot S \cap g \cdot C)$ 
      using  $C$  gf by auto
    ultimately show ?thesis
      using  $C$  by auto
  qed
show ?lhs
  using proper_map [OF _ _ gfm] **
  by (simp add: continuous_on_closed * closedin_imp_subset)
qed

```

5.0.13 Trivial fact: convexity equals connectedness for collinear sets

```

lemma convex_connected_collinear:
  fixes  $S :: 'a::euclidean\_space$  set
  assumes collinear  $S$ 
  shows  $\text{convex } S \longleftrightarrow \text{connected } S$ 
proof
  assume convex  $S$ 
  then show connected  $S$ 
    using convex_connected by blast
next
  assume  $S$ : connected  $S$ 
  show convex  $S$ 
  proof (cases  $S = \{\}$ )
    case True
    then show ?thesis by simp
  next
    case False
    then obtain  $a$  where  $a \in S$  by auto
    have collinear (affine hull  $S$ )
      by (simp add: assms collinear_affine_hull_collinear)
    then obtain  $z$  where  $z \neq 0 \wedge x. x \in \text{affine hull } S \implies \exists c. x - a = c *_R z$ 
      by (meson  $\langle a \in S \rangle$  collinear hull_inc)
    then obtain  $f$  where  $f: \bigwedge x. x \in \text{affine hull } S \implies x - a = f x *_R z$ 
      by metis
    then have inj_f: inj_on  $f$  (affine hull  $S$ )
      by (metis diff_add_cancel inj_onI)
    have diff:  $x - y = (f x - f y) *_R z$  if  $x: x \in \text{affine hull } S$  and  $y: y \in \text{affine hull } S$  for  $x y$ 
    proof -
      have  $f x *_R z = x - a$ 
        by (simp add: f hull_inc x)
      moreover have  $f y *_R z = y - a$ 
        by (simp add: f hull_inc y)
      ultimately show ?thesis
        by (simp add: scaleR_left.diff)
    qed
    have cont_f: continuous_on (affine hull  $S$ )  $f$ 

```

```

proof (clarsimp simp: dist_norm continuous_on_iff diff)
  show  $\bigwedge x \in S. 0 < e \implies \exists d > 0. \forall y \in \text{affine hull } S. |f y - f x| * \text{norm } z < d$ 
 $\implies |f y - f x| < e$ 
  by (metis  $\langle z \neq 0 \rangle$  mult_pos_pos mult_less_iff1 zero_less_norm_iff)
qed
then have conn_fS: connected (f ` S)
by (meson S connected_continuous_image continuous_on_subset hull_subset)
show ?thesis
proof (clarsimp simp: convex_contains_segment)
  fix x y z
  assume  $x \in S \ y \in S \ z \in \text{closed\_segment } x \ y$ 
  have False if  $z \notin S$ 
  proof -
    have f ` (closed_segment x y) = closed_segment (f x) (f y)
    proof (rule continuous_injective_image_segment_1)
      show continuous_on (closed_segment x y) f
      by (meson  $\langle x \in S \rangle \langle y \in S \rangle$  convex_affine_hull convex_contains_segment
        hull_inc continuous_on_subset [OF cont_f])
      show inj_on f (closed_segment x y)
      by (meson  $\langle x \in S \rangle \langle y \in S \rangle$  convex_affine_hull convex_contains_segment
        hull_inc inj_on_subset [OF inj_f])
    qed
    then have fz:  $f z \in \text{closed\_segment } (f x) (f y)$ 
    using  $\langle z \in \text{closed\_segment } x y \rangle$  by blast
    have  $z \in \text{affine hull } S$ 
    by (meson  $\langle x \in S \rangle \langle y \in S \rangle \langle z \in \text{closed\_segment } x y \rangle$  convex_affine_hull
      convex_contains_segment hull_inc subset_eq)
    then have fz_notin:  $f z \notin f ` S$ 
    using hull_subset inj_f inj_onD that by fastforce
    moreover have  $\{..<f z\} \cap f ` S \neq \{\}$   $\{f z<..\} \cap f ` S \neq \{\}$ 
    proof -
      consider  $f x \leq f z \wedge f z \leq f y \mid f y \leq f z \wedge f z \leq f x$ 
      using fz
      by (auto simp add: closed_segment_eq_real_ivl split: if_split_asm)
      then have  $\{..<f z\} \cap f ` \{x,y\} \neq \{\} \wedge \{f z<..\} \cap f ` \{x,y\} \neq \{\}$ 
      by cases (use fz_notin  $\langle x \in S \rangle \langle y \in S \rangle$  in  $\langle \text{auto simp: image\_iff} \rangle$ )
      then show  $\{..<f z\} \cap f ` S \neq \{\} \wedge \{f z<..\} \cap f ` S \neq \{\}$ 
      using  $\langle x \in S \rangle \langle y \in S \rangle$  by blast+
    qed
    ultimately show False
    using connectedD [OF conn_fS, of  $\{..<f z\} \{f z<..\}$ ] by force
  qed
then show  $z \in S$  by meson
qed
qed
qed

```

lemma compact_convex_collinear_segment_alt:
fixes $S :: 'a::\text{euclidean\_space}$ **set**


```

    assumes  $S \neq \{\}$  compact  $S$  connected  $S$  collinear  $S$ 
    obtains  $a\ b$  where  $S = \text{closed\_segment } a\ b$ 
  proof -
    obtain  $\xi$  where  $\xi \in S$  using  $\langle S \neq \{\} \rangle$  by auto
    have collinear (affine hull  $S$ )
      by (simp add: assms collinear_affine_hull_collinear)
    then obtain  $z$  where  $z \neq 0 \wedge x. x \in \text{affine hull } S \implies \exists c. x - \xi = c *_R z$ 
      by (meson  $\langle \xi \in S \rangle$  collinear_hull_inc)
    then obtain  $f$  where  $f: \wedge x. x \in \text{affine hull } S \implies x - \xi = f\ x *_R z$ 
      by metis
    let  $?g = \lambda r. r *_R z + \xi$ 
    have  $gf: ?g\ (f\ x) = x$  if  $x \in \text{affine hull } S$  for  $x$ 
      by (metis diff_add_cancel  $f$  that)
    then have  $\text{inj\_}f: \text{inj\_on } f\ (\text{affine hull } S)$ 
      by (metis  $\text{inj\_onI}$ )
    have  $\text{diff}: x - y = (f\ x - f\ y) *_R z$  if  $x: x \in \text{affine hull } S$  and  $y: y \in \text{affine hull } S$  for  $x\ y$ 
    proof -
      have  $f\ x *_R z = x - \xi$ 
        by (simp add:  $f\ \text{hull\_inc } x$ )
      moreover have  $f\ y *_R z = y - \xi$ 
        by (simp add:  $f\ \text{hull\_inc } y$ )
      ultimately show  $?thesis$ 
        by (simp add:  $\text{scaleR\_left.diff}$ )
    qed
    have  $\text{cont\_}f: \text{continuous\_on } (\text{affine hull } S)\ f$ 
    proof (clarsimp simp:  $\text{dist\_norm continuous\_on\_iff diff}$ )
      show  $\wedge x\ e. 0 < e \implies \exists d > 0. \forall y \in \text{affine hull } S. |f\ y - f\ x| * \text{norm } z < d \implies |f\ y - f\ x| < e$ 
        by (metis  $\langle z \neq 0 \rangle$  mult_pos_pos mult_less_iff1 zero_less_norm_iff)
    qed
    then have connected  $(f\ ^\circ S)$ 
      by (meson  $\langle \text{connected } S \rangle$  connected_continuous_image continuous_on_subset hull_subset)
    moreover have compact  $(f\ ^\circ S)$ 
      by (meson  $\langle \text{compact } S \rangle$  compact_continuous_image_eq cont_f hull_subset inj_f)
    ultimately obtain  $x\ y$  where  $f\ ^\circ S = \{x..y\}$ 
      by (meson connected_compact_interval_1)
    then have  $fS\_eq: f\ ^\circ S = \text{closed\_segment } x\ y$ 
      using  $\langle S \neq \{\} \rangle$  closed_segment_eq_real_ivl by auto
    obtain  $a\ b$  where  $a \in S\ f\ a = x\ b \in S\ f\ b = y$ 
      by (metis (full_types) ends_in_segment  $fS\_eq\ \text{imageE}$ )
    have  $f\ ^\circ (\text{closed\_segment } a\ b) = \text{closed\_segment } (f\ a)\ (f\ b)$ 
    proof (rule continuous_injective_image_segment_1)
      show continuous_on  $(\text{closed\_segment } a\ b)\ f$ 
        by (meson  $\langle a \in S \rangle \langle b \in S \rangle$  convex_affine_hull convex_contains_segment hull_inc continuous_on_subset [OF cont_f])
      show  $\text{inj\_on } f\ (\text{closed\_segment } a\ b)$ 

```

```

      by (meson ⟨a ∈ S⟩ ⟨b ∈ S⟩ convex_affine_hull convex_contains_segment
hull_inc inj_on_subset [OF inj_f])
    qed
    then have f ' (closed_segment a b) = f ' S
      by (simp add: ⟨f a = x⟩ ⟨f b = y⟩ fS_eq)
    then have ?g ' f ' (closed_segment a b) = ?g ' f ' S
      by simp
    moreover have (λx. f x *R z + ξ) ' closed_segment a b = closed_segment a b
      unfolding image_def using ⟨a ∈ S⟩ ⟨b ∈ S⟩
    by (safe; metis (mono_tags, lifting) convex_affine_hull convex_contains_segment
gf hull_subset subsetCE)
    ultimately have closed_segment a b = S
      using gf by (simp add: image_comp o_def hull_inc cong: image_cong)
    then show ?thesis
      using that by blast
  qed

```

```

lemma compact_convex_collinear_segment:
  fixes S :: 'a::euclidean_space set
  assumes S ≠ {} compact S convex S collinear S
  obtains a b where S = closed_segment a b
  using assms convex_connected_collinear compact_convex_collinear_segment_alt
  by blast

```

```

lemma proper_map_from_compact:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes contf: continuous_on S f and imf: f ' S ⊆ T and compact S
  closedin (top_of_set T) K
  shows compact (S ∩ f -' K)
  by (rule closedin_compact [OF ⟨compact S⟩] continuous_closedin_preimage_gen
assms)+

```

```

lemma proper_map_fst:
  assumes compact T K ⊆ S compact K
  shows compact (S × T ∩ fst -' K)
  proof -
    have (S × T ∩ fst -' K) = K × T
      using assms by auto
    then show ?thesis by (simp add: assms compact_Times)
  qed

```

```

lemma closed_map_fst:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes compact T closedin (top_of_set (S × T)) c
  shows closedin (top_of_set S) (fst ' c)
  proof -
    have *: fst ' (S × T) ⊆ S
      by auto

```

```

  show ?thesis
    using proper_map [OF _ _] by (simp add: proper_map_fst assms)
qed

```

```

lemma proper_map_snd:
  assumes compact S K  $\subseteq$  T compact K
  shows compact (S  $\times$  T  $\cap$  snd  $^{-1}$  K)
proof -
  have (S  $\times$  T  $\cap$  snd  $^{-1}$  K) = S  $\times$  K
    using assms by auto
  then show ?thesis by (simp add: assms compact_Times)
qed

```

```

lemma closed_map_snd:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes compact S closedin (top_of_set (S  $\times$  T)) c
  shows closedin (top_of_set T) (snd  $^{-1}$  c)
proof -
  have *: snd  $^{-1}$  (S  $\times$  T)  $\subseteq$  T
    by auto
  show ?thesis
    using proper_map [OF _ _] by (simp add: proper_map_snd assms)
qed

```

```

lemma closedin_compact_projection:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes compact S and clo: closedin (top_of_set (S  $\times$  T)) U
  shows closedin (top_of_set T) {y.  $\exists x. x \in S \wedge (x, y) \in U$ }
proof -
  have U  $\subseteq$  S  $\times$  T
    by (metis clo closedin_imp_subset)
  then have {y.  $\exists x. x \in S \wedge (x, y) \in U$ } = snd  $^{-1}$  U
    by force
  moreover have closedin (top_of_set T) (snd  $^{-1}$  U)
    by (rule closed_map_snd [OF assms])
  ultimately show ?thesis
    by simp
qed

```

```

lemma closed_compact_projection:
  fixes S :: 'a::euclidean_space set
  and T :: ('a * 'b::euclidean_space) set
  assumes compact S and clo: closed T
  shows closed {y.  $\exists x. x \in S \wedge (x, y) \in T$ }
proof -
  have *: {y.  $\exists x. x \in S \wedge \text{Pair } x y \in T$ } = {y.  $\exists x. x \in S \wedge \text{Pair } x y \in ((S \times \text{UNIV}) \cap T)$ }
    by auto

```

```

show ?thesis
  unfolding *
  by (intro clo closedin_closed_Int closedin_closed_trans [OF _ closed_UNIV]
closedin_compact_projection [OF ⟨compact S⟩])
qed

```

Representing affine hull as a finite intersection of hyperplanes

```

proposition affine_hull_convex_Int_nonempty_interior:
  fixes S :: 'a::real_normed_vector set
  assumes convex S S ∩ interior T ≠ {}
  shows affine_hull (S ∩ T) = affine_hull S
proof
  show affine_hull (S ∩ T) ⊆ affine_hull S
    by (simp add: hull_mono)
  next
    obtain a where a ∈ S a ∈ T and at: a ∈ interior T
      using assms interior_subset by blast
    then obtain e where e > 0 and e: cball a e ⊆ T
      using mem_interior_cball by blast
    have *: x ∈ (+) a ‘ span ((λx. x - a) ‘ (S ∩ T)) if x ∈ S for x
      proof (cases x = a)
        case True with that span_0 eq_add_iff image_def mem_Collect_eq show
          ?thesis
            by blast
        next
          case False
            define k where k = min (1/2) (e / norm (x-a))
            have k: 0 < k k < 1
              using ⟨e > 0⟩ False by (auto simp: k_def)
            then have xa: (x-a) = inverse k *R k *R (x-a)
              by simp
            have e / norm (x - a) ≥ k
              using k_def by linarith
            then have a + k *R (x - a) ∈ cball a e
              using ⟨0 < k⟩ False
              by (simp add: dist_norm) (simp add: field_simps)
            then have T: a + k *R (x - a) ∈ T
              using e by blast
            have S: a + k *R (x - a) ∈ S
              using k ⟨a ∈ S⟩ convexD [OF ⟨convex S⟩ ⟨a ∈ S⟩ ⟨x ∈ S⟩, of 1-k k]
              by (simp add: algebra_simps)
            have inverse k *R k *R (x-a) ∈ span ((λx. x - a) ‘ (S ∩ T))
              by (intro span_mul [OF span_base] image_eqI [where x = a + k *R (x -
a)]) (auto simp: S T)
            with xa image_iff show ?thesis by fastforce
          qed
      have S ⊆ affine_hull (S ∩ T)
        by (force simp: * ⟨a ∈ S⟩ ⟨a ∈ T⟩ hull_inc affine_hull_span_gen [of a])

```

```

    then show affine_hull S  $\subseteq$  affine_hull (S  $\cap$  T)
      by (simp add: subset_hull)
qed

```

```

corollary affine_hull_convex_Int_open:
  fixes S :: 'a::real_normed_vector set
  assumes convex S open T S  $\cap$  T  $\neq$  {}
  shows affine_hull (S  $\cap$  T) = affine_hull S
  using affine_hull_convex_Int_nonempty_interior assms interior_eq by blast

```

```

corollary affine_hull_affine_Int_nonempty_interior:
  fixes S :: 'a::real_normed_vector set
  assumes affine S S  $\cap$  interior T  $\neq$  {}
  shows affine_hull (S  $\cap$  T) = affine_hull S
  by (simp add: affine_hull_convex_Int_nonempty_interior affine_imp_convex
    assms)

```

```

corollary affine_hull_affine_Int_open:
  fixes S :: 'a::real_normed_vector set
  assumes affine S open T S  $\cap$  T  $\neq$  {}
  shows affine_hull (S  $\cap$  T) = affine_hull S
  by (simp add: affine_hull_convex_Int_open affine_imp_convex assms)

```

```

corollary affine_hull_convex_Int_openin:
  fixes S :: 'a::real_normed_vector set
  assumes convex S openin (top_of_set (affine_hull S)) T S  $\cap$  T  $\neq$  {}
  shows affine_hull (S  $\cap$  T) = affine_hull S
  using assms unfolding openin_open
  by (metis affine_hull_convex_Int_open hull_subset inf.orderE inf_assoc)

```

```

corollary affine_hull_openin:
  fixes S :: 'a::real_normed_vector set
  assumes openin (top_of_set (affine_hull T)) S S  $\neq$  {}
  shows affine_hull S = affine_hull T
  using assms unfolding openin_open
  by (metis affine_affine_hull affine_hull_affine_Int_open hull_hull)

```

```

corollary affine_hull_open:
  fixes S :: 'a::real_normed_vector set
  assumes open S S  $\neq$  {}
  shows affine_hull S = UNIV
  by (metis affine_hull_convex_Int_nonempty_interior assms convex_UNIV hull_UNIV
    inf_top.left_neutral interior_open)

```

```

lemma aff_dim_convex_Int_nonempty_interior:
  fixes S :: 'a::euclidean_space set
  shows  $\llbracket$ convex S; S  $\cap$  interior T  $\neq$  {} $\rrbracket \implies \text{aff\_dim}(S \cap T) = \text{aff\_dim } S$ 
  using aff_dim_affine_hull2 affine_hull_convex_Int_nonempty_interior by blast

```

```

lemma aff_dim_convex_Int_open:
  fixes S :: 'a::euclidean_space set
  shows  $\llbracket \text{convex } S; \text{ open } T; S \cap T \neq \{\} \rrbracket \implies \text{aff\_dim}(S \cap T) = \text{aff\_dim } S$ 
  using aff_dim_convex_Int_nonempty_interior_interior_eq by blast

lemma affine_hull_Diff:
  fixes S :: 'a::real_normed_vector set
  assumes ope:  $\text{openin } (\text{top\_of\_set } (\text{affine hull } S)) \ S$  and finite F  $F \subset S$ 
  shows  $\text{affine hull } (S - F) = \text{affine hull } S$ 
proof -
  have clo:  $\text{closedin } (\text{top\_of\_set } S) \ F$ 
    using assms finite_imp_closedin by auto
  moreover have  $S - F \neq \{\}$ 
    using assms by auto
  ultimately show ?thesis
    by (metis ope closedin_def topspace_euclidean_subtopology affine_hull_openin
    openin_trans)
qed

lemma affine_hull_halfspace_lt:
  fixes a :: 'a::euclidean_space
  shows  $\text{affine hull } \{x. a \cdot x < r\} = (\text{if } a = 0 \wedge r \leq 0 \text{ then } \{\} \text{ else } \text{UNIV})$ 
  using halfspace_eq_empty_lt [of a r]
  by (simp add: open_halfspace_lt affine_hull_open)

lemma affine_hull_halfspace_le:
  fixes a :: 'a::euclidean_space
  shows  $\text{affine hull } \{x. a \cdot x \leq r\} = (\text{if } a = 0 \wedge r < 0 \text{ then } \{\} \text{ else } \text{UNIV})$ 
proof (cases a = 0)
  case True then show ?thesis by simp
next
  case False
  then have  $\text{affine hull closure } \{x. a \cdot x < r\} = \text{UNIV}$ 
    using affine_hull_halfspace_lt closure_same_affine_hull by fastforce
  moreover have  $\{x. a \cdot x < r\} \subseteq \{x. a \cdot x \leq r\}$ 
    by (simp add: Collect_mono)
  ultimately show ?thesis using False antisym_conv hull_mono top_greatest
    by (metis affine_hull_halfspace_lt)
qed

lemma affine_hull_halfspace_gt:
  fixes a :: 'a::euclidean_space
  shows  $\text{affine hull } \{x. a \cdot x > r\} = (\text{if } a = 0 \wedge r \geq 0 \text{ then } \{\} \text{ else } \text{UNIV})$ 
  using halfspace_eq_empty_gt [of r a]
  by (simp add: open_halfspace_gt affine_hull_open)

lemma affine_hull_halfspace_ge:
  fixes a :: 'a::euclidean_space
  shows  $\text{affine hull } \{x. a \cdot x \geq r\} = (\text{if } a = 0 \wedge r > 0 \text{ then } \{\} \text{ else } \text{UNIV})$ 

```

using affine_hull_halfspace_le [of $-a -r$] by simp

lemma aff_dim_halfspace_lt:
 fixes $a :: 'a::euclidean\_space$
 shows $\text{aff\_dim } \{x. a \cdot x < r\} =$
 $(\text{if } a = 0 \wedge r \leq 0 \text{ then } -1 \text{ else } \text{DIM}('a))$
 by simp (metis aff_dim_open_halfspace_eq_empty_lt open_halfspace_lt)

lemma aff_dim_halfspace_le:
 fixes $a :: 'a::euclidean\_space$
 shows $\text{aff\_dim } \{x. a \cdot x \leq r\} =$
 $(\text{if } a = 0 \wedge r < 0 \text{ then } -1 \text{ else } \text{DIM}('a))$
 proof -
 have $\text{int } (\text{DIM}('a)) = \text{aff\_dim } (\text{UNIV}::'a \text{ set})$
 by (simp)
 then have $\text{aff\_dim } (\text{affine hull } \{x. a \cdot x \leq r\}) = \text{DIM}('a)$ if $(a = 0 \longrightarrow r \geq 0)$
 using that by (simp add: affine_hull_halfspace_le not_less)
 then show ?thesis
 by (force)
 qed

lemma aff_dim_halfspace_gt:
 fixes $a :: 'a::euclidean\_space$
 shows $\text{aff\_dim } \{x. a \cdot x > r\} =$
 $(\text{if } a = 0 \wedge r \geq 0 \text{ then } -1 \text{ else } \text{DIM}('a))$
 by simp (metis aff_dim_open_halfspace_eq_empty_gt open_halfspace_gt)

lemma aff_dim_halfspace_ge:
 fixes $a :: 'a::euclidean\_space$
 shows $\text{aff\_dim } \{x. a \cdot x \geq r\} =$
 $(\text{if } a = 0 \wedge r > 0 \text{ then } -1 \text{ else } \text{DIM}('a))$
 using aff_dim_halfspace_le [of $-a -r$] by simp

proposition aff_dim_eq_hyperplane:
 fixes $S :: 'a::euclidean\_space \text{ set}$
 shows $\text{aff\_dim } S = \text{DIM}('a) - 1 \longleftrightarrow (\exists a b. a \neq 0 \wedge \text{affine hull } S = \{x. a \cdot x = b\})$
 (is ?lhs = ?rhs)
 proof (cases $S = \{\}$)
 case True then show ?thesis
 by (auto simp: dest: hyperplane_eq_Ex)
 next
 case False
 then obtain c where $c \in S$ by blast
 show ?thesis
 proof (cases $c = 0$)
 case True
 have $?lhs \longleftrightarrow (\exists a. a \neq 0 \wedge \text{span } ((\lambda x. x - c) ' S) = \{x. a \cdot x = 0\})$

```

    by (simp add: aff_dim_eq_dim [of c] ⟨c ∈ S⟩ hull_inc dim_eq_hyperplane
del: One_nat_def)
    also have ...  $\longleftrightarrow$  ?rhs
    using span_zero [of S] True ⟨c ∈ S⟩ affine_hull_span_0 hull_inc
    by (fastforce simp add: affine_hull_span_gen [of c] ⟨c = 0⟩)
    finally show ?thesis .
next
case False
have xc_im:  $x \in (+) c \text{ ' } \{y. a \cdot y = 0\}$  if  $a \cdot x = a \cdot c$  for  $a$   $x$ 
proof -
  have  $\exists y. a \cdot y = 0 \wedge c + y = x$ 
  by (metis that add.commute diff_add_cancel inner_commute inner_diff_left
right_minus_eq)
  then show  $x \in (+) c \text{ ' } \{y. a \cdot y = 0\}$ 
  by blast
qed
have 2:  $\text{span } ((\lambda x. x - c) \text{ ' } S) = \{x. a \cdot x = 0\}$ 
  if  $(+) c \text{ ' } \text{span } ((\lambda x. x - c) \text{ ' } S) = \{x. a \cdot x = b\}$  for  $a$   $b$ 
proof -
  have  $b = a \cdot c$ 
  using span_0 that by fastforce
  with that have  $(+) c \text{ ' } \text{span } ((\lambda x. x - c) \text{ ' } S) = \{x. a \cdot x = a \cdot c\}$ 
  by simp
  then have  $\text{span } ((\lambda x. x - c) \text{ ' } S) = (\lambda x. x - c) \text{ ' } \{x. a \cdot x = a \cdot c\}$ 
  by (metis (no_types) image_cong translation_galois uminus_add_conv_diff)
  also have ... =  $\{x. a \cdot x = 0\}$ 
  by (force simp: inner_distrib inner_diff_right
intro: image_eqI [where  $x=x+c$  for  $x$ ])
  finally show ?thesis .
qed
have ?lhs =  $(\exists a. a \neq 0 \wedge \text{span } ((\lambda x. x - c) \text{ ' } S) = \{x. a \cdot x = 0\})$ 
  by (simp add: aff_dim_eq_dim [of c] ⟨c ∈ S⟩ hull_inc dim_eq_hyperplane
del: One_nat_def)
  also have ... = ?rhs
  by (fastforce simp add: affine_hull_span_gen [of c] ⟨c ∈ S⟩ hull_inc in-
ner_distrib intro: xc_im intro!: 2)
  finally show ?thesis .
qed
qed

```

corollary *aff_dim_hyperplane* [simp]:

```

  fixes  $a :: 'a::\text{euclidean\_space}$ 
  shows  $a \neq 0 \implies \text{aff\_dim } \{x. a \cdot x = r\} = \text{DIM}('a) - 1$ 
by (metis aff_dim_eq_hyperplane affine_hull_eq affine_hyperplane)

```

5.0.14 Some stepping theorems

lemma *aff_dim_insert*:

```

  fixes  $a :: 'a::\text{euclidean\_space}$ 

```



```

  shows  $\text{aff\_dim } (\text{insert } a \ S) = (\text{if } a \in \text{affine hull } S \text{ then } \text{aff\_dim } S \text{ else } \text{aff\_dim } S + 1)$ 
proof (cases  $S = \{\}$ )
  case True then show ?thesis
    by simp
next
  case False
  then obtain  $x \ s'$  where  $S: S = \text{insert } x \ s' \ x \notin s'$ 
    by (meson Set.set_insert_all_not_in_conv)
  show ?thesis using S
    by (force simp add: affine_hull_insert_span_gen span_zero insert_commute
      [of a] aff_dim_eq_dim [of x] dim_insert)
qed

```

```

lemma affine_dependent_choose:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  assumes  $\neg(\text{affine\_dependent } S)$ 
  shows  $\text{affine\_dependent}(\text{insert } a \ S) \longleftrightarrow a \notin S \wedge a \in \text{affine hull } S$ 
    (is ?lhs = ?rhs)
proof safe
  assume  $\text{affine\_dependent } (\text{insert } a \ S)$  and  $a \in S$ 
  then show False
    using  $\langle a \in S \rangle \text{ assms insert\_absorb}$  by fastforce
next
  assume lhs:  $\text{affine\_dependent } (\text{insert } a \ S)$ 
  then have  $a \notin S$ 
    by (metis (no_types) assms insert_absorb)
  moreover have finite S
    using affine_independent_iff_card assms by blast
  moreover have  $\text{aff\_dim } (\text{insert } a \ S) \neq \text{int } (\text{card } S)$ 
    using  $\langle \text{finite } S \rangle \text{ affine\_independent\_iff\_card } \langle a \notin S \rangle \text{ lhs}$  by fastforce
  ultimately show  $a \in \text{affine hull } S$ 
    by (metis aff_dim_affine_independent aff_dim_insert assms)
next
  assume  $a \notin S$  and  $a \in \text{affine hull } S$ 
  show  $\text{affine\_dependent } (\text{insert } a \ S)$ 
    by (simp add:  $\langle a \in \text{affine hull } S \rangle \langle a \notin S \rangle \text{ affine\_dependent\_def}$ )
qed

```

```

lemma affine_independent_insert:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  shows  $\llbracket \neg \text{affine\_dependent } S; a \notin \text{affine hull } S \rrbracket \implies \neg \text{affine\_dependent}(\text{insert } a \ S)$ 
  by (simp add: affine_dependent_choose)

```

```

lemma subspace_bounded_eq_trivial:
  fixes  $S :: 'a::\text{real\_normed\_vector\_set}$ 
  assumes subspace S
  shows  $\text{bounded } S \longleftrightarrow S = \{0\}$ 

```

```

proof –
  have False if bounded S  $x \in S \ x \neq 0$  for  $x$ 
  proof –
    obtain  $B$  where  $B: \bigwedge y. y \in S \implies \text{norm } y < B \ B > 0$ 
    using  $\langle \text{bounded } S \rangle$  by (force simp: bounded_pos_less)
    have  $(B / \text{norm } x) *_R x \in S$ 
    using assms subspace_mul  $\langle x \in S \rangle$  by auto
    moreover have  $\text{norm } ((B / \text{norm } x) *_R x) = B$ 
    using that B by (simp add: algebra_simps)
    ultimately show False using  $B$  by force
  qed
  then have  $\text{bounded } S \implies S = \{0\}$ 
  using assms subspace_0 by fastforce
  then show ?thesis
  by blast
qed

lemma affine_bounded_eq_trivial:
  fixes  $S :: 'a::\text{real\_normed\_vector\_space}$  set
  assumes affine S
  shows  $\text{bounded } S \longleftrightarrow S = \{0\} \vee (\exists a. S = \{a\})$ 
proof (cases S = {0})
  case True then show ?thesis
  by simp
next
  case False
  then obtain  $b$  where  $b \in S$  by blast
  with False assms
  have  $\text{bounded } S \implies S = \{b\}$ 
  using affine_diffs_subspace [OF assms  $\langle b \in S \rangle$ ]
  by (metis (no_types, lifting) ab_group_add_class.ab_left_minus bounded_translation
image_empty image_insert subspace_bounded_eq_trivial translation_invert)
  then show ?thesis by auto
qed

lemma affine_bounded_eq_lowdim:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes affine S
  shows  $\text{bounded } S \longleftrightarrow \text{aff\_dim } S \leq 0$ 
proof
  show  $\text{aff\_dim } S \leq 0 \implies \text{bounded } S$ 
  by (metis aff_dim_singular aff_dim_subset affine_dim_equal affine_singular all_not_in_conv
assms bounded_empty bounded_insert dual_order.antisym empty_subsetI insert_subset)
qed (use affine_bounded_eq_trivial assms in fastforce)

lemma bounded_hyperplane_eq_trivial_0:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  assumes  $a \neq 0$ 

```

```

  shows bounded  $\{x. a \cdot x = 0\} \longleftrightarrow DIM('a) = 1$ 
proof
  assume bounded  $\{x. a \cdot x = 0\}$ 
  then have aff_dim  $\{x. a \cdot x = 0\} \leq 0$ 
    by (simp add: affine_bounded_eq_lowdim affine_hyperplane)
  with assms show  $DIM('a) = 1$ 
    by (simp add: le_Suc_eq)
next
  assume  $DIM('a) = 1$ 
  then show bounded  $\{x. a \cdot x = 0\}$ 
    by (simp add: affine_bounded_eq_lowdim affine_hyperplane assms)
qed

lemma bounded_hyperplane_eq_trivial:
  fixes a :: 'a::euclidean_space
  shows bounded  $\{x. a \cdot x = r\} \longleftrightarrow (if\ a = 0\ then\ r \neq 0\ else\ DIM('a) = 1)$ 
proof (simp add: bounded_hyperplane_eq_trivial_0, clarify)
  assume  $r \neq 0 \ a \neq 0$ 
  have aff_dim  $\{x. y \cdot x = 0\} = aff\_dim\ \{x. a \cdot x = r\}$  if  $y \neq 0$  for  $y::'a$ 
    by (metis that  $\langle a \neq 0 \rangle$  aff_dim_hyperplane)
  then show bounded  $\{x. a \cdot x = r\} = (DIM('a) = Suc\ 0)$ 
    by (metis One_nat_def  $\langle a \neq 0 \rangle$  affine_bounded_eq_lowdim affine_hyperplane
    bounded_hyperplane_eq_trivial_0)
qed

```

5.0.15 General case without assuming closure and getting non-strict separation

```

proposition separating_hyperplane_closed_point_inset:
  fixes S :: 'a::euclidean_space set
  assumes convex S closed  $S \neq \{\}$   $z \notin S$ 
  obtains a b where  $a \in S \ (a - z) \cdot z < b \wedge x. x \in S \implies b < (a - z) \cdot x$ 
proof -
  obtain y where  $y \in S$  and  $y: \bigwedge u. u \in S \implies dist\ z\ y \leq dist\ z\ u$ 
    using distance_attains_inf [of S z] assms by auto
  then have *:  $(y - z) \cdot z < (y - z) \cdot z + (norm\ (y - z))^2 / 2$ 
    using  $\langle y \in S \rangle \langle z \notin S \rangle$  by auto
  show ?thesis
  proof (rule that [OF  $\langle y \in S \rangle$  *])
    fix x
    assume  $x \in S$ 
    have yz:  $0 < (y - z) \cdot (y - z)$ 
      using  $\langle y \in S \rangle \langle z \notin S \rangle$  by auto
    { assume 0:  $0 < ((z - y) \cdot (x - y))$ 
      with any_closest_point_dot [OF  $\langle convex\ S \rangle \langle closed\ S \rangle$ ]
      have False
        using  $y \langle x \in S \rangle \langle y \in S \rangle$  not_less by blast
    }
    then have  $0 \leq ((y - z) \cdot (x - y))$ 

```

```

    by (force simp: not_less inner_diff_left)
  with yz have  $0 < 2 * ((y - z) \cdot (x - y)) + (y - z) \cdot (y - z)$ 
  by (simp add: algebra_simps)
  then show  $(y - z) \cdot z + (\text{norm } (y - z))^2 / 2 < (y - z) \cdot x$ 
  by (simp add: field_simps inner_diff_left inner_diff_right dot_square_norm
[symmetric])
qed
qed

```

```

lemma separating_hyperplane_closed_0_inset:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes  $\text{convex } S$   $\text{closed } S$   $S \neq \{\}$   $0 \notin S$ 
  obtains  $a$   $b$  where  $a \in S$   $a \neq 0$   $0 < b \wedge x. x \in S \implies a \cdot x > b$ 
  using separating_hyperplane_closed_point_inset [OF assms] by simp (metis  $\langle 0 \notin S \rangle$ )

```

```

proposition separating_hyperplane_set_0_inspan:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes  $\text{convex } S$   $S \neq \{\}$   $0 \notin S$ 
  obtains  $a$  where  $a \in \text{span } S$   $a \neq 0$   $\wedge x. x \in S \implies 0 \leq a \cdot x$ 
proof -
  define  $k$  where [abs_def]:  $k \ c = \{x. 0 \leq c \cdot x\}$  for  $c :: 'a$ 
  have  $\text{span } S \cap \text{frontier } (\text{cball } 0 \ 1) \cap \bigcap f' \neq \{\}$ 
    if  $f': \text{finite } f' \ f' \subseteq k \ 'S$  for  $f'$ 
  proof -
    obtain  $C$  where  $C \subseteq S$   $\text{finite } C$  and  $C: f' = k \ 'C$ 
      using finite_subset_image [OF f'] by blast
    obtain  $a$  where  $a \in S$   $a \neq 0$ 
      using  $\langle S \neq \{\} \rangle \langle 0 \notin S \rangle \text{ex\_in\_conv}$  by blast
    then have  $\text{norm } (a /_R (\text{norm } a)) = 1$ 
      by simp
    moreover have  $a /_R (\text{norm } a) \in \text{span } S$ 
      by (simp add:  $\langle a \in S \rangle \text{span\_scale span\_base}$ )
    ultimately have  $\text{ass: } a /_R (\text{norm } a) \in \text{span } S \cap \text{sphere } 0 \ 1$ 
      by simp
    show ?thesis
  proof (cases  $C = \{\}$ )
    case True with  $C$  ass show ?thesis
      by auto
    next
    case False
      have closed  $(\text{convex hull } C)$ 
      using  $\langle \text{finite } C \rangle \text{compact\_eq\_bounded\_closed finite\_imp\_compact\_convex\_hull}$ 
    by auto
    moreover have  $\text{convex hull } C \neq \{\}$ 
      by (simp add: False)
    moreover have  $0 \notin \text{convex hull } C$ 
      by (metis  $\langle C \subseteq S \rangle \langle \text{convex } S \rangle \langle 0 \notin S \rangle \text{convex\_hull\_subset hull\_same}$ )

```

```

insert_absorb insert_subset)
  ultimately obtain a b
    where a ∈ convex hull C a ≠ 0 0 < b
      and ab:  $\bigwedge x. x \in \text{convex hull } C \implies a \cdot x > b$ 
    using separating_hyperplane_closed_0_inset by blast
  then have a ∈ S
    by (metis  $\langle C \subseteq S \rangle$  assms(1) subsetCE subset_hull)
  moreover have norm (a /R (norm a)) = 1
    using  $\langle a \neq 0 \rangle$  by simp
  moreover have a /R (norm a) ∈ span S
    by (simp add:  $\langle a \in S \rangle$  span_scale span_base)
  ultimately have ass: a /R (norm a) ∈ span S ∩ sphere 0 1
    by simp
  have  $\bigwedge x. \llbracket a \neq 0; x \in C \rrbracket \implies 0 \leq x \cdot a$ 
    using ab  $\langle 0 < b \rangle$  by (metis hull_inc inner_commute less_eq_real_def
less_trans)
  then have aa: a /R (norm a) ∈  $(\bigcap c \in C. \{x. 0 \leq c \cdot x\})$ 
    by (auto simp add: field_split_simps)
  show ?thesis
    unfolding C k_def
    using ass aa Int_iff empty_iff by force
qed
qed
moreover have  $\bigwedge T. T \in k \text{ ' } S \implies \text{closed } T$ 
  using closed_halfspace_ge k_def by blast
ultimately have  $(\text{span } S \cap \text{frontier}(\text{cball } 0 \ 1)) \cap (\bigcap (k \text{ ' } S)) \neq \{\}$ 
  by (metis compact_imp_fip closed_Int_compact closed_span compact_cball
compact_frontier)
then show ?thesis
  unfolding set_eq_iff k_def
  by simp (metis inner_commute norm_eq_zero that zero_neq_one)
qed

```

lemma separating_hyperplane_set_point_inaff:

```

fixes S :: 'a::euclidean_space set
assumes convex S S ≠ {} and zno: z ∉ S
obtains a b where (z + a) ∈ affine hull (insert z S)
  and a ≠ 0 and a · z ≤ b
  and  $\bigwedge x. x \in S \implies a \cdot x \geq b$ 

```

proof –

```

from separating_hyperplane_set_0_inspan [of image  $(\lambda x. -z + x)$  S]
have convex  $((+) (-z) \text{ ' } S)$ 
  using  $\langle \text{convex } S \rangle$  by simp
moreover have  $(+) (-z) \text{ ' } S \neq \{\}$ 
  by (simp add:  $\langle S \neq \{\} \rangle$ )
moreover have  $0 \notin (+) (-z) \text{ ' } S$ 
  using zno by auto
ultimately obtain a where a ∈ span  $((+) (-z) \text{ ' } S)$  a ≠ 0

```

and $a: \bigwedge x. x \in ((+) (-z) 'S) \implies 0 \leq a \cdot x$
 using *separating_hyperplane_set_0_inspan* [of image $(\lambda x. -z + x) S$]
 by *blast*
 then have *sxz*: $\bigwedge x. x \in S \implies a \cdot z \leq a \cdot x$
 by (metis (no_types, lifting) *imageI inner_minus_right inner_right_distrib*
minus_add neg_le_0_iff_le neg_le_iff_le real_add_le_0_iff)
 moreover
 have $z + a \in \text{affine hull insert } z S$
 using $\langle a \in \text{span } ((+) (-z) 'S) \rangle$ *affine_hull_insert_span_gen* by *blast*
 ultimately show ?thesis
 using $\langle a \neq 0 \rangle$ *sxz* that by *auto*
 qed

proposition *supporting_hyperplane_rel_boundary*:

fixes $S :: 'a::\text{euclidean\_space set}$
 assumes *convex* S $x \in S$ and *xno*: $x \notin \text{rel\_interior } S$
 obtains a where $a \neq 0$
 and $\bigwedge y. y \in S \implies a \cdot x \leq a \cdot y$
 and $\bigwedge y. y \in \text{rel\_interior } S \implies a \cdot x < a \cdot y$

proof –

obtain a b where *aff*: $(x + a) \in \text{affine hull (insert } x (\text{rel\_interior } S))$
 and $a \neq 0$ and $a \cdot x \leq b$
 and *ageb*: $\bigwedge u. u \in (\text{rel\_interior } S) \implies a \cdot u \geq b$
 using *separating_hyperplane_set_point_inaff* [of $\text{rel\_interior } S$ x] *assms*
 by (auto simp: *rel_interior_eq_empty convex_rel_interior*)
 have *le_ay*: $a \cdot x \leq a \cdot y$ if $y \in S$ for y

proof –

have *con*: *continuous_on* (*closure* ($\text{rel\_interior } S$)) $((\cdot) a)$
 by (rule *continuous_intros continuous_on_subset* | *blast*) +
 have *y*: $y \in \text{closure } (\text{rel\_interior } S)$
 using $\langle \text{convex } S \rangle$ *closure_def convex_closure_rel_interior* $\langle y \in S \rangle$
 by *fastforce*
 show ?thesis
 using *continuous_ge_on_closure* [OF *con y*] *ageb* $\langle a \cdot x \leq b \rangle$
 by *fastforce*

qed

have $\exists: a \cdot x < a \cdot y$ if $y \in \text{rel\_interior } S$ for y

proof –

obtain e where $0 < e$ $y \in S$ and *e*: $\text{cball } y \cap \text{affine hull } S \subseteq S$
 using $\langle y \in \text{rel\_interior } S \rangle$ by (force simp: *rel_interior_cball*)
 define y' where $y' = y - (e / \text{norm } a) *_{\mathbb{R}} ((x + a) - x)$
 have $y' \in \text{cball } y \cap S$
 unfolding *y'_def* using $\langle 0 < e \rangle$ by *force*
 moreover have $y' \in \text{affine hull } S$
 unfolding *y'_def*
 by (metis $\langle x \in S \rangle \langle y \in S \rangle \langle \text{convex } S \rangle$ *aff affine_affine_hull hull_redundant*
rel_interior_same_affine_hull hull_inc mem_affine_3_minus2)
 ultimately have $y' \in S$
 using *e* by *auto*

```

have  $a \cdot x \leq a \cdot y$ 
  using  $le\_ay \langle a \neq 0 \rangle \langle y \in S \rangle$  by blast
moreover have  $a \cdot x \neq a \cdot y$ 
  using  $le\_ay [OF \langle y' \in S \rangle] \langle a \neq 0 \rangle \langle 0 < e \rangle not\_le$ 
  by (fastforce simp add:  $y'_def inner\_diff dot\_square\_norm power2\_eq\_square$ )
ultimately show ?thesis by force
qed
show ?thesis
  by (rule that [OF  $\langle a \neq 0 \rangle le\_ay 3$ ])
qed

```

```

lemma supporting_hyperplane_relative_frontier:
  fixes  $S :: 'a::euclidean\_space\ set$ 
  assumes  $convex\ S\ x \in closure\ S\ x \notin rel\_interior\ S$ 
  obtains  $a$  where  $a \neq 0$ 
    and  $\bigwedge y. y \in closure\ S \implies a \cdot x \leq a \cdot y$ 
    and  $\bigwedge y. y \in rel\_interior\ S \implies a \cdot x < a \cdot y$ 
using supporting_hyperplane_rel_boundary [of  $closure\ S\ x$ ]
by (metis assms convex_closure convex_rel_interior_closure)

```

5.0.16 Some results on decomposing convex hulls: intersections, simplicial subdivision

```

lemma
  fixes  $S :: 'a::euclidean\_space\ set$ 
  assumes  $\neg affine\_dependent(S \cup T)$ 
  shows  $convex\_hull\_Int\_subset: convex\ hull\ S \cap convex\ hull\ T \subseteq convex\ hull$ 
     $(S \cap T)$  (is ?C)
    and  $affine\_hull\_Int\_subset: affine\ hull\ S \cap affine\ hull\ T \subseteq affine\ hull$ 
     $(S \cap T)$  (is ?A)
proof -
  have [simp]:  $finite\ S\ finite\ T$ 
  using  $aff\_independent\_finite\ assms$  by blast+
  have  $sum\ u\ (S \cap T) = 1 \wedge$ 
     $(\sum_{v \in S \cap T} u\ v *_{\mathbb{R}} v) = (\sum_{v \in S} u\ v *_{\mathbb{R}} v)$ 
  if [simp]:  $sum\ u\ S = 1$ 
     $sum\ v\ T = 1$ 
  and  $eq: (\sum_{x \in T} v\ x *_{\mathbb{R}} x) = (\sum_{x \in S} u\ x *_{\mathbb{R}} x)$  for  $u\ v$ 
proof -
  define  $f$  where  $f\ x = (if\ x \in S\ then\ u\ x\ else\ 0) - (if\ x \in T\ then\ v\ x\ else\ 0)$ 
for  $x$ 
  have  $sum\ f\ (S \cup T) = 0$ 
  by (simp add:  $f\_def sum\_Un sum\_subtractf flip: sum.inter\_restrict$ )
  moreover have  $(\sum_{x \in (S \cup T)} f\ x *_{\mathbb{R}} x) = 0$ 
  by (simp add:  $eq\ f\_def sum\_Un scaleR\_left\_diff\_distrib sum\_subtractf$ 
    if_smult_flip:  $sum.inter\_restrict\ cong: if\_cong$ )
  ultimately have  $\bigwedge v. v \in S \cup T \implies f\ v = 0$ 
  using  $aff\_independent\_finite\ assms$  unfolding  $affine\_dependent\_explicit$ 
  by blast

```

```

    then have u [simp]:  $\bigwedge x. x \in S \implies u\ x = (\text{if } x \in T \text{ then } v\ x \text{ else } 0)$ 
    by (simp add: f_def) presburger
  have sum u (S  $\cap$  T) = sum u S
    by (simp add: sum.inter_restrict)
  then have sum u (S  $\cap$  T) = 1
    using that by linarith
  moreover have  $(\sum v \in S \cap T. u\ v *_{\mathbb{R}} v) = (\sum v \in S. u\ v *_{\mathbb{R}} v)$ 
    by (auto simp: if_smult sum.inter_restrict intro: sum.cong)
  ultimately show ?thesis
    by force
qed
then show ?A ?C
  by (auto simp: convex_hull_finite affine_hull_finite)
qed

```

proposition *affine_hull_Int*:

```

fixes S :: 'a::euclidean_space set
assumes  $\neg$  affine_dependent (S  $\cup$  T)
  shows affine_hull (S  $\cap$  T) = affine_hull S  $\cap$  affine_hull T
by (simp add: affine_hull_Int_subset assms hull_mono subset_antisym)

```

proposition *convex_hull_Int*:

```

fixes S :: 'a::euclidean_space set
assumes  $\neg$  affine_dependent (S  $\cup$  T)
  shows convex_hull (S  $\cap$  T) = convex_hull S  $\cap$  convex_hull T
by (simp add: convex_hull_Int_subset assms hull_mono subset_antisym)

```

proposition

```

fixes S :: 'a::euclidean_space set set
assumes  $\neg$  affine_dependent ( $\bigcup S$ )
  shows affine_hull_Inter: affine_hull ( $\bigcap S$ ) = ( $\bigcap T \in S. \text{affine\_hull } T$ ) (is ?A)
  and convex_hull_Inter: convex_hull ( $\bigcap S$ ) = ( $\bigcap T \in S. \text{convex\_hull } T$ ) (is ?C)
proof -
  have finite S
    using aff_independent_finite assms finite_UnionD by blast
  then have ?A  $\wedge$  ?C using assms
  proof (induction S rule: finite_induct)
    case empty then show ?case by auto
  next
    case (insert T F)
    then show ?case
    proof (cases F={})
      case True then show ?thesis by simp
    next
      case False
      with insert.premis have [simp]:  $\neg$  affine_dependent (T  $\cup \bigcap F$ )
        by (auto intro: affine_dependent_subset)
      have [simp]:  $\neg$  affine_dependent ( $\bigcup F$ )

```



```

    using affine_independent_subset insert.premis by fastforce
  show ?thesis
    by (simp add: affine_hull_Int convex_hull_Int insert.IH)
qed
qed
then show ?A ?C
  by auto
qed

proposition in_convex_hull_exchange_unique:
  fixes S :: 'a::euclidean_space set
  assumes naff:  $\neg$  affine_dependent S and a:  $a \in \text{convex hull } S$ 
    and S:  $T \subseteq S$   $T' \subseteq S$ 
    and x:  $x \in \text{convex hull } (\text{insert } a \ T)$ 
    and x':  $x \in \text{convex hull } (\text{insert } a \ T')$ 
  shows  $x \in \text{convex hull } (\text{insert } a \ (T \cap T'))$ 
proof (cases  $a \in S$ )
  case True
    then have  $\neg$  affine_dependent  $(\text{insert } a \ T \cup \text{insert } a \ T')$ 
      using affine_dependent_subset assms by auto
    then have  $x \in \text{convex hull } (\text{insert } a \ T \cap \text{insert } a \ T')$ 
      by (metis IntI convex_hull_Int x x')
    then show ?thesis
      by simp
  next
  case False
    then have anot:  $a \notin T$   $a \notin T'$ 
      using assms by auto
    have [simp]: finite S
      by (simp add: aff_independent_finite assms)
    then obtain b where b0:  $\bigwedge s. s \in S \implies 0 \leq b \ s$ 
      and b1:  $\text{sum } b \ S = 1$  and aeq:  $a = (\sum s \in S. b \ s *_{\mathbb{R}} s)$ 
      using a by (auto simp: convex_hull_finite)
    have fin [simp]: finite T finite T'
      using assms infinite_super ⟨finite S⟩ by blast+
    then obtain c c' where c0:  $\bigwedge t. t \in \text{insert } a \ T \implies 0 \leq c \ t$ 
      and c1:  $\text{sum } c \ (\text{insert } a \ T) = 1$ 
      and xeq:  $x = (\sum t \in \text{insert } a \ T. c \ t *_{\mathbb{R}} t)$ 
      and c'0:  $\bigwedge t. t \in \text{insert } a \ T' \implies 0 \leq c' \ t$ 
      and c'1:  $\text{sum } c' \ (\text{insert } a \ T') = 1$ 
      and x'eq:  $x = (\sum t \in \text{insert } a \ T'. c' \ t *_{\mathbb{R}} t)$ 
      using x x' by (auto simp: convex_hull_finite)
    with fin anot
    have sumTT':  $\text{sum } c \ T = 1 - c \ a$   $\text{sum } c' \ T' = 1 - c' \ a$ 
      and wsumT:  $(\sum t \in T. c \ t *_{\mathbb{R}} t) = x - c \ a *_{\mathbb{R}} a$ 
      by simp_all
    have wsumT':  $(\sum t \in T'. c' \ t *_{\mathbb{R}} t) = x - c' \ a *_{\mathbb{R}} a$ 
      using x'eq fin anot by simp
    define cc where  $cc \equiv \lambda x. \text{if } x \in T \text{ then } c \ x \text{ else } 0$ 

```

```

define cc' where cc'  $\equiv \lambda x. \text{if } x \in T' \text{ then } c' x \text{ else } 0$ 
define dd where dd  $\equiv \lambda x. cc\ x - cc'\ x + (c\ a - c'\ a) * b\ x$ 
have sumSS':  $sum\ cc\ S = 1 - c\ a$   $sum\ cc'\ S = 1 - c'\ a$ 
  unfolding cc_def cc'_def using S
  by (simp_all add: Int_absorb1 Int_absorb2 sum_subtractf sum.inter_restrict
[symmetric] sumTT')
  have wsumSS:  $(\sum t \in S. cc\ t *_R t) = x - c\ a *_R a$   $(\sum t \in S. cc'\ t *_R t) = x$ 
 $- c'\ a *_R a$ 
  unfolding cc_def cc'_def using S
  by (simp_all add: Int_absorb1 Int_absorb2 if_smult sum.inter_restrict [symmetric]
wsumT wsumT' cong: if_cong)
  have sum_dd0:  $sum\ dd\ S = 0$ 
  unfolding dd_def using S
  by (simp add: sumSS' comm_monoid_add_class.sum.distrib sum_subtractf
algebra_simps sum_distrib_right [symmetric] b1)
have  $(\sum v \in S. (b\ v * x) *_R v) = x *_R (\sum v \in S. b\ v *_R v)$  for x
  by (simp add: pth_5 real_vector.scale_sum_right mult.commute)
then have *:  $(\sum v \in S. (b\ v * x) *_R v) = x *_R a$  for x
  using aeq by blast
have  $(\sum v \in S. dd\ v *_R v) = 0$ 
  unfolding dd_def using S
  by (simp add: * wsumSS sum.distrib sum_subtractf algebra_simps)
then have dd0:  $dd\ v = 0$  if  $v \in S$  for v
  using naff [unfolded affine_dependent_explicit_not_ex, rule_format, of S dd]
  using that sum_dd0 by force
consider  $c'\ a \leq c\ a \mid c\ a \leq c'\ a$  by linarith
then show ?thesis
proof cases
  case 1
  then have  $sum\ cc\ S \leq sum\ cc'\ S$ 
  by (simp add: sumSS')
  then have le:  $cc\ x \leq cc'\ x$  if  $x \in S$  for x
  using dd0 [OF that] 1 b0 mult_left_mono that
  by (fastforce simp add: dd_def algebra_simps)
have cc0:  $cc\ x = 0$  if  $x \in S$   $x \notin T \cap T'$  for x
  using le [OF  $\langle x \in S \rangle$ ] that cc0
  by (force simp: cc_def cc'_def split: if_split_asm)
show ?thesis
proof (simp add: convex_hull_finite, intro exI conjI)
  show  $\forall x \in T \cap T'. 0 \leq (cc(a := c\ a))\ x$ 
  by (simp add: cc0 cc_def)
  show  $0 \leq (cc(a := c\ a))\ a$ 
  by (simp add: cc0)
  have  $sum\ (cc(a := c\ a))\ (insert\ a\ (T \cap T')) = c\ a + sum\ (cc(a := c\ a))\ (T$ 
 $\cap T')$ 
  by (simp add: anot)
  also have  $\dots = c\ a + sum\ (cc(a := c\ a))\ S$ 
  using  $\langle T \subseteq S \rangle$  False cc0 cc_def  $\langle a \notin S \rangle$  by (fastforce intro!: sum.mono_neutral_left
split: if_split_asm)

```

```

also have ... =  $c \ a + (1 - c \ a)$ 
  by (metis  $\langle a \notin S \rangle$  fun_upd_other sum.cong sumSS'(1))
finally show  $\text{sum } (cc(a := c \ a)) \ (\text{insert } a \ (T \cap T')) = 1$ 
  by simp
have  $(\sum_{x \in \text{insert } a \ (T \cap T')} . (cc(a := c \ a)) \ x *_{\mathcal{R}} x) = c \ a *_{\mathcal{R}} a + (\sum_{x \in T \cap T'} . (cc(a := c \ a)) \ x *_{\mathcal{R}} x)$ 
  by (simp add: anot)
also have ... =  $c \ a *_{\mathcal{R}} a + (\sum_{x \in S} . (cc(a := c \ a)) \ x *_{\mathcal{R}} x)$ 
  using  $\langle T \subseteq S \rangle$  False cc0 by (fastforce intro!: sum.mono_neutral_left split:
if_split_asm)
also have ... =  $c \ a *_{\mathcal{R}} a + x - c \ a *_{\mathcal{R}} a$ 
  by (simp add: wsumSS  $\langle a \notin S \rangle$  if_smult sum_delta_notmem)
finally show  $(\sum_{x \in \text{insert } a \ (T \cap T')} . (cc(a := c \ a)) \ x *_{\mathcal{R}} x) = x$ 
  by simp
qed
next
case 2
then have  $\text{sum } cc' \ S \leq \text{sum } cc \ S$ 
  by (simp add: sumSS')
then have  $le: cc' \ x \leq cc \ x$  if  $x \in S$  for  $x$ 
  using dd0 [OF that] 2 b0 mult_left_mono that
  by (fastforce simp add: dd_def algebra_simps)
have  $cc0: cc' \ x = 0$  if  $x \in S \ x \notin T \cap T'$  for  $x$ 
  using le [OF  $\langle x \in S \rangle$ ] that c'0
  by (force simp: cc_def cc'_def split: if_split_asm)
show ?thesis
proof (simp add: convex_hull_finite, intro exI conjI)
  show  $\forall x \in T \cap T'. 0 \leq (cc'(a := c' \ a)) \ x$ 
    by (simp add: c'0 cc'_def)
  show  $0 \leq (cc'(a := c' \ a)) \ a$ 
    by (simp add: c'0)
  have  $\text{sum } (cc'(a := c' \ a)) \ (\text{insert } a \ (T \cap T')) = c' \ a + \text{sum } (cc'(a := c' \ a))$ 
     $(T \cap T')$ 
    by (simp add: anot)
  also have ... =  $c' \ a + \text{sum } (cc'(a := c' \ a)) \ S$ 
    using  $\langle T \subseteq S \rangle$  False cc0 by (fastforce intro!: sum.mono_neutral_left split:
if_split_asm)
  also have ... =  $c' \ a + (1 - c' \ a)$ 
    by (metis  $\langle a \notin S \rangle$  fun_upd_other sum.cong sumSS')
  finally show  $\text{sum } (cc'(a := c' \ a)) \ (\text{insert } a \ (T \cap T')) = 1$ 
    by simp
  have  $(\sum_{x \in \text{insert } a \ (T \cap T')} . (cc'(a := c' \ a)) \ x *_{\mathcal{R}} x) = c' \ a *_{\mathcal{R}} a + (\sum_{x \in T \cap T'} . (cc'(a := c' \ a)) \ x *_{\mathcal{R}} x)$ 
    by (simp add: anot)
  also have ... =  $c' \ a *_{\mathcal{R}} a + (\sum_{x \in S} . (cc'(a := c' \ a)) \ x *_{\mathcal{R}} x)$ 
    using  $\langle T \subseteq S \rangle$  False cc0 by (fastforce intro!: sum.mono_neutral_left split:
if_split_asm)
  also have ... =  $c \ a *_{\mathcal{R}} a + x - c \ a *_{\mathcal{R}} a$ 
    by (simp add: wsumSS  $\langle a \notin S \rangle$  if_smult sum_delta_notmem)

```

```

      finally show  $(\sum x \in \text{insert } a (T \cap T'). (cc'(a := c' a)) x *_R x) = x$ 
      by simp
    qed
  qed
qed

corollary convex_hull_exchange_Int:
  fixes a :: 'a::euclidean_space
  assumes  $\neg \text{affine\_dependent } S$   $a \in \text{convex\_hull } S$   $T \subseteq S$   $T' \subseteq S$ 
  shows  $(\text{convex\_hull } (\text{insert } a T)) \cap (\text{convex\_hull } (\text{insert } a T')) =$ 
     $\text{convex\_hull } (\text{insert } a (T \cap T'))$  (is ?lhs = ?rhs)
proof (rule subset_antisym)
  show ?lhs  $\subseteq$  ?rhs
    using in_convex_hull_exchange_unique assms by blast
  show ?rhs  $\subseteq$  ?lhs
    by (metis hull_mono inf_le1 inf_le2 insert_inter_insert le_inf_iff)
qed

lemma Int_closed_segment:
  fixes b :: 'a::euclidean_space
  assumes  $b \in \text{closed\_segment } a c \vee \neg \text{collinear}\{a,b,c\}$ 
  shows  $\text{closed\_segment } a b \cap \text{closed\_segment } b c = \{b\}$ 
proof (cases  $c = a$ )
  case True
  then show ?thesis
    using assms collinear_3_eq_affine_dependent by fastforce
next
  case False
  from assms show ?thesis
  proof
    assume  $b \in \text{closed\_segment } a c$ 
    moreover have  $\neg \text{affine\_dependent } \{a, c\}$ 
      by (simp)
    ultimately show ?thesis
      using False convex_hull_exchange_Int [of  $\{a,c\}$   $b$   $\{a\}$   $\{c\}$ ]
      by (simp add: segment_convex_hull insert_commute)
  next
    assume ncoll:  $\neg \text{collinear } \{a, b, c\}$ 
    have False if  $\text{closed\_segment } a b \cap \text{closed\_segment } b c \neq \{b\}$ 
    proof -
      have  $b \in \text{closed\_segment } a b$  and  $b \in \text{closed\_segment } b c$ 
        by auto
      with that obtain d where  $b \neq d$   $d \in \text{closed\_segment } a b$   $d \in \text{closed\_segment } b c$ 
        by force
      then have d:  $\text{collinear } \{a, d, b\}$   $\text{collinear } \{b, d, c\}$ 
        by (auto simp: between_mem_segment between_imp_collinear)
      have collinear  $\{a, b, c\}$ 
        by (metis (full_types)  $\langle b \neq d \rangle$  collinear_3_trans d insert_commute)
    qed
  qed

```

```

    with ncoll show False ..
  qed
  then show ?thesis
    by blast
  qed
qed

lemma affine_hull_finite_intersection_hyperplanes:
  fixes S :: 'a::euclidean_space set
  obtains  $\mathcal{F}$  where
    finite  $\mathcal{F}$ 
    of_nat (card  $\mathcal{F}$ ) + aff_dim S = DIM('a)
    affine_hull S =  $\bigcap \mathcal{F}$ 
     $\bigwedge h. h \in \mathcal{F} \implies \exists a b. a \neq 0 \wedge h = \{x. a \cdot x = b\}$ 
proof -
  obtain b where b  $\subseteq$  S
    and indb:  $\neg$  affine_dependent b
    and eq: affine_hull S = affine_hull b
  using affine_basis_exists by blast
  obtain c where indc:  $\neg$  affine_dependent c and b  $\subseteq$  c
    and affc: affine_hull c = UNIV
  by (metis extend_to_affine_basis affine_UNIV hull_same indb subset_UNIV)
  then have finite c
    by (simp add: aff_independent_finite)
  then have fbc: finite b card b  $\leq$  card c
    using  $\langle b \subseteq c \rangle$  infinite_super by (auto simp: card_mono)
  have imeq:  $(\lambda x. \text{affine\_hull } x) \text{ ' } ((\lambda a. c - \{a\}) \text{ ' } (c - b)) = ((\lambda a. \text{affine\_hull } (c - \{a\})) \text{ ' } (c - b))$ 
    by blast
  have card_cb: (card (c - b)) + aff_dim S = DIM('a)
proof -
  have aff: aff_dim (UNIV::'a set) = aff_dim c
    by (metis aff_dim_affine_hull affc)
  have aff_dim b = aff_dim S
    by (metis (no_types) aff_dim_affine_hull eq)
  then have int (card b) = 1 + aff_dim S
    by (simp add: aff_dim_affine_independent indb)
  then show ?thesis
    using fbc aff
    by (simp add:  $\langle \neg \text{affine\_dependent } c \rangle \langle b \subseteq c \rangle$  aff_dim_affine_independent
card_Diff_subset of_nat_diff)
  qed
  show ?thesis
proof (cases c = b)
  case True show ?thesis
proof
  show int (card  $\{\}$ ) + aff_dim S = int DIM('a)
    using True card_cb by auto
  show affine_hull S =  $\bigcap \{\}$ 

```

```

    using True affc eq by blast
  qed auto
next
case False
have ind:  $\neg$  affine_dependent  $(\bigcup a \in c - b. c - \{a\})$ 
  by (rule affine_independent_subset [OF indc]) auto
have *:  $1 + \text{aff\_dim } (c - \{t\}) = \text{int } (\text{DIM}('a))$  if  $t: t \in c$  for  $t$ 
proof -
  have insert  $t \ c = c$ 
  using  $t$  by blast
  then show ?thesis
    by (metis (full_types) add.commute aff_dim_affine_hull aff_dim_insert
aff_dim_UNIV affc affine_dependent_def indc insert_Diff_single  $t$ )
  qed
let ? $\mathcal{F}$  =  $(\lambda x. \text{affine hull } x) \ ' ((\lambda a. c - \{a\}) \ ' (c - b))$ 
show ?thesis
proof
  have card  $((\lambda a. \text{affine hull } (c - \{a\})) \ ' (c - b)) = \text{card } (c - b)$ 
  proof (rule card_image)
    show inj_on  $(\lambda a. \text{affine hull } (c - \{a\})) \ (c - b)$ 
    unfolding inj_on_def
    by (metis Diff_eq_empty_iff Diff_iff indc affine_dependent_def hull_subset
insert_iff)
  qed
  then show  $\text{int } (\text{card } ?\mathcal{F}) + \text{aff\_dim } S = \text{int } \text{DIM}('a)$ 
    by (simp add: imeq card_cb)
  show affine hull  $S = \bigcap ?\mathcal{F}$ 
    by (metis Diff_eq_empty_iff INT_simps(4) UN_singleton order_refl  $\langle b \subseteq$ 
 $c \rangle$  False eq_double_diff_affine_hull_Inter [OF ind])
  have  $\bigwedge a. \llbracket a \in c; a \notin b \rrbracket \implies \text{aff\_dim } (c - \{a\}) = \text{int } (\text{DIM}('a) - \text{Suc } 0)$ 
    by (metis * DIM_ge_Suc0 One_nat_def add_diff_cancel_left' int_ops(2)
of_nat_diff)
  then show  $\bigwedge h. h \in ?\mathcal{F} \implies \exists a \ b. a \neq 0 \wedge h = \{x. a \cdot x = b\}$ 
    by (auto simp only: One_nat_def aff_dim_eq_hyperplane [symmetric])
  qed (use  $\langle \text{finite } c \rangle$  in auto)
qed
qed

lemma affine_hyperplane_sums_eq_UNIV_0:
  fixes  $S :: 'a :: \text{euclidean\_space}$  set
  assumes affine  $S$ 
    and  $0 \in S$  and  $w \in S$ 
    and  $a \cdot w \neq 0$ 
  shows  $\{x + y \mid x \ y. x \in S \wedge a \cdot y = 0\} = \text{UNIV}$ 
proof -
  have subspace  $S$ 
    by (simp add: asms subspace_affine)
  have span1:  $\text{span } \{y. a \cdot y = 0\} \subseteq \text{span } \{x + y \mid x \ y. x \in S \wedge a \cdot y = 0\}$ 
    using  $\langle 0 \in S \rangle$  add.left_neutral by (intro span_mono) force

```

```

have  $w \notin \text{span } \{y. a \cdot y = 0\}$ 
  using  $\langle a \cdot w \neq 0 \rangle$  span_induct subspace_hyperplane by auto
moreover have  $w \in \text{span } \{x + y \mid x y. x \in S \wedge a \cdot y = 0\}$ 
  using  $\langle w \in S \rangle$ 
  by (metis (mono_tags, lifting) inner_zero_right mem_Collect_eq pth_d span_base)
ultimately have span2:  $\text{span } \{y. a \cdot y = 0\} \neq \text{span } \{x + y \mid x y. x \in S \wedge a \cdot$ 
 $y = 0\}$ 
  by blast
have  $a \neq 0$  using assms inner_zero_left by blast
then have  $\text{DIM}('a) - 1 = \dim \{y. a \cdot y = 0\}$ 
  by (simp add: dim_hyperplane)
also have  $\dots < \dim \{x + y \mid x y. x \in S \wedge a \cdot y = 0\}$ 
  using span1 span2 by (blast intro: dim_subset)
finally have  $\text{DIM}('a) - 1 < \dim \{x + y \mid x y. x \in S \wedge a \cdot y = 0\}$  .
then have DD:  $\dim \{x + y \mid x y. x \in S \wedge a \cdot y = 0\} = \text{DIM}('a)$ 
  using antisym dim_subset_UNIV lowdim_subset_hyperplane not_le by fast-
force
have subs: subspace  $\{x + y \mid x y. x \in S \wedge a \cdot y = 0\}$ 
  using subspace_sums [OF  $\langle \text{subspace } S \rangle$  subspace_hyperplane] by simp
moreover have  $\text{span } \{x + y \mid x y. x \in S \wedge a \cdot y = 0\} = \text{UNIV}$ 
  using DD dim_eq_full by blast
ultimately show ?thesis
  by (simp add: subs) (metis (lifting) span_eq_iff subs)
qed

```

proposition *affine_hyperplane_sums_eq_UNIV*:

```

fixes  $S :: 'a :: \text{euclidean\_space}$  set
assumes affine S
  and  $S \cap \{v. a \cdot v = b\} \neq \{\}$ 
  and  $S - \{v. a \cdot v = b\} \neq \{\}$ 
shows  $\{x + y \mid x y. x \in S \wedge a \cdot y = b\} = \text{UNIV}$ 
proof (cases  $a = 0$ )
  case True with assms show ?thesis
    by (auto simp: if_splits)
next
  case False
  obtain  $c$  where  $c \in S$  and  $c: a \cdot c = b$ 
    using assms by force
  with affine_diffs_subspace [OF  $\langle \text{affine } S \rangle$ ]
  have subspace  $((+) (- c) 'S)$  by blast
  then have aff: affine  $((+) (- c) 'S)$ 
    by (simp add: subspace_imp_affine)
  have 0:  $0 \in (+) (- c) 'S$ 
    by (simp add:  $\langle c \in S \rangle$ )
  obtain  $d$  where  $d \in S$  and  $a \cdot d \neq b$  and  $dc: d - c \in (+) (- c) 'S$ 
    using assms by auto
  then have adc:  $a \cdot (d - c) \neq 0$ 
    by (simp add: c inner_diff_right)
  define  $U$  where  $U \equiv \{x + y \mid x y. x \in (+) (- c) 'S \wedge a \cdot y = 0\}$ 

```

```

have  $u + v \in (+) (c+c) \text{ ' } U$ 
  if  $u \in S$   $b = a \cdot v$  for  $u \ v$ 
proof
  show  $u + v = c + c + (u + v - c - c)$ 
    by (simp add: algebra_simps)
  have  $\exists x \ y. u + v - c - c = x + y \wedge (\exists xa \in S. x = xa - c) \wedge a \cdot y = 0$ 
  proof (intro exI conjI)
    show  $u + v - c - c = (u-c) + (v-c) \ a \cdot (v - c) = 0$ 
      by (simp_all add: algebra_simps c that)
    qed (use that in auto)
  then show  $u + v - c - c \in U$ 
    by (auto simp: U_def image_def)
  qed
moreover have  $\llbracket a \cdot v = 0; u \in S \rrbracket$ 
   $\implies \exists x \ ya. v + (u + c) = x + ya \wedge x \in S \wedge a \cdot ya = b$  for  $v \ u$ 
  by (metis add.left_commute c inner_right_distrib pth_d)
ultimately have  $\{x + y \mid x \ y. x \in S \wedge a \cdot y = b\} = (+) (c+c) \text{ ' } U$ 
  by (fastforce simp: algebra_simps U_def)
also have  $\dots = \text{range } ((+) (c + c))$ 
  by (simp only: U_def affine_hyperplane_sums_eq_UNIV_0 [OF aff 0 dc adc])
also have  $\dots = UNIV$ 
  by simp
finally show ?thesis .
qed

```

```

lemma aff_dim_sums_Int_0:
  assumes affine S
    and affine T
    and  $0 \in S$   $0 \in T$ 
  shows  $\text{aff\_dim } \{x + y \mid x \ y. x \in S \wedge y \in T\} = (\text{aff\_dim } S + \text{aff\_dim } T) - \text{aff\_dim}(S \cap T)$ 
proof -
  have  $0 \in \{x + y \mid x \ y. x \in S \wedge y \in T\}$ 
    using assms by force
  then have  $0: 0 \in \text{affine hull } \{x + y \mid x \ y. x \in S \wedge y \in T\}$ 
    by (metis (lifting) hull_inc)
  have sub: subspace S subspace T
    using assms by (auto simp: subspace_affine)
  show ?thesis
    using dim_sums_Int [OF sub] by (simp add: aff_dim_zero assms 0 hull_inc)
qed

```

```

proposition aff_dim_sums_Int:
  assumes affine S
    and affine T
    and  $S \cap T \neq \{\}$ 
  shows  $\text{aff\_dim } \{x + y \mid x \ y. x \in S \wedge y \in T\} = (\text{aff\_dim } S + \text{aff\_dim } T) - \text{aff\_dim}(S \cap T)$ 
proof -

```



```

obtain  $a$  where  $a: a \in S \ a \in T$  using  $assms$  by  $force$ 
have  $aff: affine \ ((+) \ (-a) \ 'S) \ affine \ ((+) \ (-a) \ 'T)$ 
  using  $affine\_translation \ [symmetric, \ of \ - \ a]$   $assms$  by  $(simp\_all \ cong: im-$ 
 $age\_cong\_simp)$ 
have  $zero: 0 \in ((+) \ (-a) \ 'S) \ 0 \in ((+) \ (-a) \ 'T)$ 
  using  $a \ assms$  by  $auto$ 
have  $\{x + y \mid x \ y. \ x \in (+) \ (-a) \ 'S \wedge y \in (+) \ (-a) \ 'T\} =$ 
 $(+) \ (-2 *_{\mathbb{R}} a) \ '\{x + y \mid x \ y. \ x \in S \wedge y \in T\}$ 
  by  $(force \ simp: algebra\_simps \ scaleR\_2)$ 
moreover have  $(+) \ (-a) \ 'S \cap (+) \ (-a) \ 'T = (+) \ (-a) \ '(S \cap T)$ 
  by  $auto$ 
ultimately show  $?thesis$ 
  using  $aff\_dim\_sums\_Int\_0 \ [OF \ aff \ zero] \ aff\_dim\_translation\_eq$ 
  by  $(metis \ (lifting))$ 
qed

```

```

lemma  $aff\_dim\_affine\_Int\_hyperplane:$ 
  fixes  $a :: 'a::euclidean\_space$ 
  assumes  $affine \ S$ 
  shows  $aff\_dim(S \cap \{x. \ a \cdot x = b\}) =$ 
 $(if \ S \cap \{v. \ a \cdot v = b\} = \{\} \ then \ - \ 1$ 
 $\ else \ if \ S \subseteq \{v. \ a \cdot v = b\} \ then \ aff\_dim \ S$ 
 $\ else \ aff\_dim \ S - 1)$ 
proof  $(cases \ a = 0)$ 
  case  $True$  with  $assms$  show  $?thesis$ 
  by  $auto$ 
next
  case  $False$ 
  then have  $aff\_dim \ (S \cap \{x. \ a \cdot x = b\}) = aff\_dim \ S - 1$ 
    if  $x \in S \ a \cdot x \neq b$  and  $non: S \cap \{v. \ a \cdot v = b\} \neq \{\}$  for  $x$ 
  proof  $-$ 
    have  $[simp]: \{x + y \mid x \ y. \ x \in S \wedge a \cdot y = b\} = UNIV$ 
      using  $affine\_hyperplane\_sums\_eq \ UNIV \ [OF \ assms \ non]$  that by  $blast$ 
    show  $?thesis$ 
      using  $aff\_dim\_sums\_Int \ [OF \ assms \ affine\_hyperplane \ non]$ 
      by  $(simp \ add: \ of\_nat\_diff \ False)$ 
  qed
  then show  $?thesis$ 
    by  $(metis \ (mono\_tags, \ lifting) \ inf.orderE \ aff\_dim\_empty\_eq \ mem\_Collect\_eq \$ 
 $subsetI)$ 
qed

```

```

lemma  $aff\_dim\_lt\_full:$ 
  fixes  $S :: 'a::euclidean\_space \ set$ 
  shows  $aff\_dim \ S < DIM('a) \longleftrightarrow (affine \ hull \ S \neq UNIV)$ 
by  $(metis \ (no\_types) \ aff\_dim\_affine\_hull \ aff\_dim\_le\_DIM \ aff\_dim\_UNIV \ affine\_hull\_UNIV \$ 
 $less\_le)$ 

```

```

lemma  $aff\_dim\_openin:$ 

```

```

fixes S :: 'a::euclidean_space set
assumes ope: openin (top_of_set T) S and affine T S ≠ {}
shows aff_dim S = aff_dim T
proof -
  show ?thesis
  proof (rule order_antisym)
    show aff_dim S ≤ aff_dim T
      by (blast intro: aff_dim_subset [OF openin_imp_subset] ope)
  next
    obtain a where a ∈ S
      using ⟨S ≠ {}⟩ by blast
    have S ⊆ T
      using ope openin_imp_subset by auto
    then have a ∈ T
      using ⟨a ∈ S⟩ by auto
    then have subT': subspace ((λx. - a + x) ' T)
      using affine_diffs_subspace ⟨affine T⟩ by auto
    then obtain B where Bsub: B ⊆ ((λx. - a + x) ' T) and po: pairwise
      orthogonal B
      and eq1: ⋀x. x ∈ B ⇒ norm x = 1 and independent B
      and cardB: card B = dim ((λx. - a + x) ' T)
      and spanB: span B = ((λx. - a + x) ' T)
    by (rule orthonormal_basis_subspace) auto
    obtain e where 0 < e and e: cball a e ∩ T ⊆ S
      by (meson ⟨a ∈ S⟩ openin_contains_cball ope)
    have aff_dim T = aff_dim ((λx. - a + x) ' T)
      by (metis aff_dim_translation_eq)
    also have ... = dim ((λx. - a + x) ' T)
      using aff_dim_subspace subT' by blast
    also have ... = card B
      by (simp add: cardB)
    also have ... = card ((λx. e *_R x) ' B)
      using ⟨0 < e⟩ by (force simp: inj_on_def card_image)
    also have ... ≤ dim ((λx. - a + x) ' S)
  proof (simp, rule independent_card_le_dim)
    have e': cball 0 e ∩ (λx. x - a) ' T ⊆ (λx. x - a) ' S
      using e by (auto simp: dist_norm norm_minus_commute subset_eq)
    have (λx. e *_R x) ' B ⊆ cball 0 e ∩ (λx. x - a) ' T
      using Bsub ⟨0 < e⟩ eq1 subT' ⟨a ∈ T⟩ by (auto simp: subspace_def)
    then show (λx. e *_R x) ' B ⊆ (λx. x - a) ' S
      using e' by blast
    have inj_on ((*_R) e) (span B)
      using ⟨0 < e⟩ inj_on_def by fastforce
    then show independent ((λx. e *_R x) ' B)
      using linear_scale_self ⟨independent B⟩ linear_dependent_inj_imageD by
blast
  qed
  also have ... = aff_dim S
    using ⟨a ∈ S⟩ aff_dim_eq_dim_hull_inc by (force cong: image_cong_simp)

```

```

    finally show  $\text{aff\_dim } T \leq \text{aff\_dim } S$  .
  qed
qed

lemma dim_openin:
  fixes  $S :: 'a::\text{euclidean\_space set}$ 
  assumes  $\text{ope: openin (top\_of\_set } T) S \text{ and subspace } T \ S \neq \{\}$ 
  shows  $\text{dim } S = \text{dim } T$ 
proof (rule order_antisym)
  show  $\text{dim } S \leq \text{dim } T$ 
    by (metis ope dim_subset openin_subset topspace_euclidean_subtopology)
next
  have  $\text{dim } T = \text{aff\_dim } S$ 
    using aff_dim_openin
    by (metis aff_dim_subspace  $\langle \text{subspace } T \rangle \langle S \neq \{\} \rangle$  ope subspace_affine)
  also have  $\dots \leq \text{dim } S$ 
    by (metis aff_dim_subset aff_dim_subspace dim_span span_superset
      subspace_span)
  finally show  $\text{dim } T \leq \text{dim } S$  by simp
qed

```

5.0.17 Lower-dimensional affine subsets are nowhere dense

```

proposition dense_complement_subspace:
  fixes  $S :: 'a :: \text{euclidean\_space set}$ 
  assumes  $\text{dim\_less: dim } T < \text{dim } S \text{ and subspace } S \text{ shows closure}(S - T) = S$ 
proof -
  have  $\text{closure}(S - U) = S$  if  $\text{dim } U < \text{dim } S \ U \subseteq S$  for  $U$ 
  proof -
    have  $\text{span } U \subset \text{span } S$ 
      by (metis neq_iff psubsetI span_eq_dim span_mono that)
    then obtain  $a$  where  $a \neq 0 \ a \in \text{span } S$  and  $a: \bigwedge y. y \in \text{span } U \implies \text{orthogonal}$ 
    a y
      using orthogonal_to_subspace_exists_gen by metis
    show ?thesis
  proof
    have closed  $S$ 
      by (simp add:  $\langle \text{subspace } S \rangle$  closed_subspace)
    then show  $\text{closure}(S - U) \subseteq S$ 
      by (simp add: closure_minimal)
    show  $S \subseteq \text{closure}(S - U)$ 
  proof (clarsimp simp: closure_approachable)
    fix  $x$  and  $e::\text{real}$ 
    assume  $x \in S \ 0 < e$ 
    show  $\exists y \in S - U. \text{dist } y \ x < e$ 
  proof (cases  $x \in U$ )
    case True
    let  $?y = x + (e/2 / \text{norm } a) *_{\mathbb{R}} a$ 
    show ?thesis

```

```

proof
  show  $\text{dist } ?y \ x < e$ 
    using  $\langle 0 < e \rangle$  by (simp add: dist_norm)
next
  have  $?y \in S$ 
    by (metis  $\langle a \in \text{span } S \rangle \langle x \in S \rangle \text{assms}(2) \text{span\_eq\_iff\_subspace\_add}$ 
     $\text{subspace\_scale}$ )
  moreover have  $?y \notin U$ 
  proof -
    have  $e/2 / \text{norm } a \neq 0$ 
      using  $\langle 0 < e \rangle \langle a \neq 0 \rangle$  by auto
    then show  $?thesis$ 
      by (metis True  $\langle a \neq 0 \rangle$  a orthogonal_scaleR orthogonal_self
       $\text{real\_vector.scale\_eq\_0\_iff\_span\_add\_eq\_span\_base}$ )
    qed
  ultimately show  $?y \in S - U$  by blast
  qed
next
  case False
  with  $\langle 0 < e \rangle \langle x \in S \rangle$  show  $?thesis$  by force
  qed
qed
qed
qed
moreover have  $S - S \cap T = S - T$ 
  by blast
moreover have  $\dim (S \cap T) < \dim S$ 
  by (metis dim_less dim_subset inf.cobounded2 inf.orderE inf.strict_boundedE
  not_le)
  ultimately show  $?thesis$ 
    by force
qed

corollary dense_complement_affine:
  fixes  $S :: 'a :: \text{euclidean\_space set}$ 
  assumes less:  $\text{aff\_dim } T < \text{aff\_dim } S$  and affine S shows  $\text{closure}(S - T) = S$ 
proof (cases  $S \cap T = \{\}$ )
  case True
    then show  $?thesis$ 
      by (metis Diff_triv affine_hull_eq  $\langle \text{affine } S \rangle$  closure_same_affine_hull closure_subset hull_subset subset_antisym)
  next
    case False
    then obtain  $z$  where  $z: z \in S \cap T$  by blast
    then have  $\text{subspace } ((+) (- z) ' S)$ 
      by (meson IntD1 affine_diffs_subspace  $\langle \text{affine } S \rangle$ )
    moreover have  $\text{int } (\dim ((+) (- z) ' T)) < \text{int } (\dim ((+) (- z) ' S))$ 
  thm aff_dim_eq_dim
    using  $z$  less by (simp add: aff_dim_eq_dim_subtract [of  $z$ ] hull_inc cong:

```

```

image_cong_simp)
ultimately have closure(((+) (- z) ' S) - ((+) (- z) ' T)) = ((+) (- z) ' S)
  by (simp add: dense_complement_subspace)
then show ?thesis
  by (metis closure_translation translation_diff translation_invert)
qed

corollary dense_complement_openin_affine_hull:
  fixes S :: 'a :: euclidean_space set
  assumes less: aff_dim T < aff_dim S
    and ope: openin (top_of_set (affine hull S)) S
  shows closure(S - T) = closure S
proof -
  have affine_hull S - T  $\subseteq$  affine_hull S
  by blast
  then have closure (S  $\cap$  closure (affine hull S - T)) = closure (S  $\cap$  (affine hull
S - T))
  by (rule closure_openin_Int_closure [OF ope])
  then show ?thesis
  by (metis Int_Diff aff_dim_affine_hull affine_affine_hull dense_complement_affine
hull_subset inf.orderE less)
qed

```

```

corollary dense_complement_convex:
  fixes S :: 'a :: euclidean_space set
  assumes aff_dim T < aff_dim S convex S
  shows closure(S - T) = closure S
proof
  show closure (S - T)  $\subseteq$  closure S
  by (simp add: closure_mono)
  have closure (rel_interior S - T) = closure (rel_interior S)
  by (simp add: assms dense_complement_openin_affine_hull openin_rel_interior
rel_interior_aff_dim rel_interior_same_affine_hull)
  then show closure S  $\subseteq$  closure (S - T)
  by (metis Diff_mono convex S closure_mono convex_closure_rel_interior
order_refl rel_interior_subset)
qed

```

```

corollary dense_complement_convex_closed:
  fixes S :: 'a :: euclidean_space set
  assumes aff_dim T < aff_dim S convex S closed S
  shows closure(S - T) = S
  by (simp add: assms dense_complement_convex)

```

5.0.18 Parallel slices, etc

If we take a slice out of a set, we can do it perpendicularly, with the normal vector to the slice parallel to the affine hull.

proposition *affine_parallel_slice*:

```

fixes S :: 'a :: euclidean_space set
assumes affine S
  and S ∩ {x. a · x ≤ b} ≠ {}
  and ¬ (S ⊆ {x. a · x ≤ b})
obtains a' b' where a' ≠ 0
  S ∩ {x. a' · x ≤ b'} = S ∩ {x. a · x ≤ b}
  S ∩ {x. a' · x = b'} = S ∩ {x. a · x = b}
  ∧ w. w ∈ S ⇒ (w + a') ∈ S
proof (cases S ∩ {x. a · x = b} = {})
case True
  then obtain u v where u ∈ S v ∈ S a · u ≤ b a · v > b
  using assms by (auto simp: not_le)
  define η where η = u + ((b - a · u) / (a · v - a · u)) *R (v - u)
  have η ∈ S
    by (simp add: η_def ⟨u ∈ S⟩ ⟨v ∈ S⟩ ⟨affine S⟩ mem_affine_3_minus)
  moreover have a · η = b
    using ⟨a · u ≤ b⟩ ⟨b < a · v⟩
    by (simp add: η_def algebra_simps) (simp add: field_simps)
  ultimately have False
    using True by force
  then show ?thesis ..
next
case False
  then obtain z where z ∈ S and z: a · z = b
  using assms by auto
  with affine_diffs_subspace [OF ⟨affine S⟩]
  have sub: subspace ((+) (- z) ' S) by blast
  then have aff: affine ((+) (- z) ' S) and span: span ((+) (- z) ' S) = ((+)
  (- z) ' S)
  by (auto simp: subspace_imp_affine)
  obtain a' a'' where a': a' ∈ span ((+) (- z) ' S) and a: a = a' + a''
    and ∧ w. w ∈ span ((+) (- z) ' S) ⇒ orthogonal a'' w
  using orthogonal_subspace_decomp_exists [of (+) (- z) ' S a] by metis
  then have ∧ w. w ∈ S ⇒ a'' · (w - z) = 0
    by (simp add: span_base orthogonal_def)
  then have a'': ∧ w. w ∈ S ⇒ a'' · w = (a - a') · z
    by (simp add: a_inner_diff_right)
  then have ba'': ∧ w. w ∈ S ⇒ a'' · w = b - a' · z
    by (simp add: inner_diff_left z)
  show ?thesis
proof (cases a' = 0)
case True
  with a assms True a'' diff_zero less_irrefl show ?thesis
    by auto
next
case False
  show ?thesis
proof
  show S ∩ {x. a' · x ≤ a' · z} = S ∩ {x. a · x ≤ b}

```

```

      S ∩ {x. a' · x = a' · z} = S ∩ {x. a · x = b}
    by (auto simp: a ba'' inner_left_distrib)
  have ∧w. w ∈ (+) (- z) ' S ⇒ (w + a') ∈ (+) (- z) ' S
    by (metis subspace_add a' span_eq_iff sub)
  then show ∧w. w ∈ S ⇒ (w + a') ∈ S
    by fastforce
qed (use False in auto)
qed
qed

```

```

lemma diffs_affine_hull_span:
  assumes a ∈ S
  shows (λx. x - a) ' (affine hull S) = span ((λx. x - a) ' S)
proof -
  have *: ((λx. x - a) ' (S - {a})) = ((λx. x - a) ' S) - {0}
    by (auto simp: algebra_simps)
  show ?thesis
    by (auto simp add: algebra_simps affine_hull_span2 [OF assms] *)
qed

```

```

lemma aff_dim_dim_affine_diffs:
  fixes S :: 'a :: euclidean_space set
  assumes affine S a ∈ S
  shows aff_dim S = dim ((λx. x - a) ' S)
proof -
  obtain B where aff: affine hull B = affine hull S
    and ind: ¬ affine_dependent B
    and card: of_nat (card B) = aff_dim S + 1
    using aff_dim_basis_exists by blast
  then have B ≠ {} using assms
    by (metis affine_hull_eq_empty ex_in_conv)
  then obtain c where c ∈ B by auto
  then have c ∈ S
    by (metis aff_affine_hull_eq ⟨affine S⟩ hull_inc)
  have xy: x - c = y - a ⇔ y = x + 1 *R (a - c) for x y c and a::'a
    by (auto simp: algebra_simps)
  have *: (λx. x - c) ' S = (λx. x - a) ' S
    using assms ⟨c ∈ S⟩
    by (auto simp: image_iff xy; metis mem_affine_3_minus_pth_1)
  have affS: affine hull S = S
    by (simp add: ⟨affine S⟩)
  have aff_dim S = of_nat (card B) - 1
    using card by simp
  also have ... = dim ((λx. x - c) ' B)
    using affine_independent_card_dim_diffs [OF ind ⟨c ∈ B⟩]
    by (simp add: affine_independent_card_dim_diffs [OF ind ⟨c ∈ B⟩])
  also have ... = dim ((λx. x - c) ' (affine hull B))
    by (simp add: diffs_affine_hull_span ⟨c ∈ B⟩)
  also have ... = dim ((λx. x - a) ' S)

```

```

    by (simp add: affS aff *)
  finally show ?thesis .
qed

lemma aff_dim_linear_image_le:
  assumes linear f
  shows aff_dim (f ' S) ≤ aff_dim S
proof -
  have aff_dim (f ' T) ≤ aff_dim T if affine T for T
  proof (cases T = {})
    case True then show ?thesis by (simp add: aff_dim_geq)
  next
    case False
    then obtain a where a ∈ T by auto
    have 1: ((λx. x - f a) ' f ' T) = {x - f a | x. x ∈ f ' T}
      by auto
    have 2: {x - f a | x. x ∈ f ' T} = f ' ((λx. x - a) ' T)
      by (force simp: linear_diff [OF assms] 2)
    have aff_dim (f ' T) = int (dim {x - f a | x. x ∈ f ' T})
      by (simp add: ⟨a ∈ T⟩ hull_inc aff_dim_eq_dim [of f a] 1 cong: im-
age_cong_simp)
    also have ... = int (dim (f ' ((λx. x - a) ' T)))
      by (force simp: linear_diff [OF assms] 2)
    also have ... ≤ int (dim ((λx. x - a) ' T))
      by (simp add: dim_image_le [OF assms])
    also have ... ≤ aff_dim T
      by (simp add: aff_dim_dim_affine_diffs [symmetric] ⟨a ∈ T⟩ ⟨affine T⟩)
    finally show ?thesis .
  qed
then
  have aff_dim (f ' (affine hull S)) ≤ aff_dim (affine hull S)
    using affine_affine_hull [of S] by blast
  then show ?thesis
    using affine_hull_linear_image assms linear_conv_bounded_linear by fast-
force
qed

lemma aff_dim_injective_linear_image [simp]:
  assumes linear f inj f
  shows aff_dim (f ' S) = aff_dim S
proof (rule antisym)
  show aff_dim (f ' S) ≤ aff_dim S
    by (simp add: aff_dim_linear_image_le assms(1))
next
  obtain g where linear g g ∘ f = id
    using assms(1) assms(2) linear_injective_left_inverse by blast
  then have aff_dim S ≤ aff_dim (g ' f ' S)
    by (simp add: image_comp)
  also have ... ≤ aff_dim (f ' S)

```



```

    by (simp add: ⟨linear g⟩ aff_dim_linear_image_le)
  finally show aff_dim S ≤ aff_dim (f ‘ S) .
qed

```

lemma *choose_affine_subset*:

```

  assumes affine S -1 ≤ d and dle: d ≤ aff_dim S
  obtains T where affine T T ⊆ S aff_dim T = d
proof (cases d = -1 ∨ S = {})
  case True with assms show ?thesis
    by (metis aff_dim_empty affine_empty bot.extremum that eq_iff)
  next
  case False
  with assms obtain a where a ∈ S 0 ≤ d by auto
  with assms have ss: subspace ((+) (- a) ‘ S)
    by (simp add: affine_diffs_subspace_subtract cong: image_cong_simp)
  have nat d ≤ dim ((+) (- a) ‘ S)
    by (metis aff_dim_subspace aff_dim_translation_eq dle nat_int nat_mono
ss)
  then obtain T where subspace T and Tsb: T ⊆ span ((+) (- a) ‘ S)
    and Tdim: dim T = nat d
    using choose_subspace_of_subspace [of nat d (+) (- a) ‘ S] by blast
  then have affine T
    using subspace_affine by blast
  then have affine ((+) a ‘ T)
    by (metis affine_hull_eq affine_hull_translation)
  moreover have (+) a ‘ T ⊆ S
proof -
    have T ⊆ (+) (- a) ‘ S
      by (metis (no_types) span_eq_iff Tsb ss)
    then show (+) a ‘ T ⊆ S
      using add_ac by auto
  qed
  moreover have aff_dim ((+) a ‘ T) = d
    by (simp add: aff_dim_subspace Tdim ⟨0 ≤ d⟩ ⟨subspace T⟩ aff_dim_translation_eq)
  ultimately show ?thesis
    by (rule that)
qed

```

5.0.19 Paracompactness

proposition *paracompact*:

```

  fixes S :: 'a :: {metric_space, second_countable_topology} set
  assumes S ⊆ ⋃ C and opC: ⋀ T. T ∈ C ⟹ open T
  obtains C' where S ⊆ ⋃ C'
    and ⋀ U. U ∈ C' ⟹ open U ∧ (∃ T. T ∈ C ∧ U ⊆ T)
    and ⋀ x. x ∈ S
      ⟹ ∃ V. open V ∧ x ∈ V ∧ finite {U. U ∈ C' ∧ (U ∩ V ≠ {})}
proof (cases S = {})

```

```

    case True with that show ?thesis by blast
next
case False
have  $\exists T \ U. \ x \in U \wedge \text{open } U \wedge \text{closure } U \subseteq T \wedge T \in \mathcal{C} \text{ if } x \in S$  for  $x$ 
proof -
  obtain  $T$  where  $x \in T \ T \in \mathcal{C} \ \text{open } T$ 
  using assms  $\langle x \in S \rangle$  by blast
  then obtain  $e$  where  $e > 0 \ \text{cball } x \ e \subseteq T$ 
  by (force simp: open_contains_cball)
  then show ?thesis
  by (meson open_ball  $\langle T \in \mathcal{C} \rangle \ \text{ball\_subset\_cball} \ \text{centre\_in\_ball} \ \text{closed\_cball}$ 
closure_minimal dual_order.trans)
qed
then obtain  $F \ G$  where  $Gin: x \in G \ x$  and  $oG: \text{open } (G \ x)$ 
and  $clos: \text{closure } (G \ x) \subseteq F \ x$  and  $Fin: F \ x \in \mathcal{C}$ 
if  $x \in S$  for  $x$ 
by metis
then obtain  $\mathcal{F}$  where  $\mathcal{F} \subseteq G \ ' S$  countable  $\mathcal{F} \cup \mathcal{F} = \bigcup (G \ ' S)$ 
using Lindelof [of  $G \ ' S$ ] by (metis image_iff)
then obtain  $K$  where  $K: K \subseteq S$  countable  $K$  and  $eq: \bigcup (G \ ' K) = \bigcup (G \ ' S)$ 
by (metis countable_subset_image)
with False  $Gin$  have  $K \neq \{\}$  by force
then obtain  $a :: \text{nat} \Rightarrow 'a$  where  $\text{range } a = K$ 
by (metis range_from_nat_into  $\langle \text{countable } K \rangle$ )
then have  $odif: \bigwedge n. \ \text{open } (F \ (a \ n)) - \bigcup \{\text{closure } (G \ (a \ m)) \mid m. \ m < n\}$ 
using  $\langle K \subseteq S \rangle \ Fin \ opC$  by (fastforce simp add:)
let  $?C = \text{range } (\lambda n. \ F \ (a \ n)) - \bigcup \{\text{closure } (G \ (a \ m)) \mid m. \ m < n\}$ 
have  $\text{enum\_}S: \exists n. \ x \in F \ (a \ n) \wedge x \in G \ (a \ n) \text{ if } x \in S$  for  $x$ 
proof -
  have  $\exists y \in K. \ x \in G \ y$  using  $eq$  that  $Gin$  by fastforce
  then show ?thesis
  using  $clos \ K \ \langle \text{range } a = K \rangle \ \text{closure\_subset}$  by blast
qed
show ?thesis
proof
  show  $S \subseteq \text{Union } ?C$ 
  proof
    fix  $x$  assume  $x \in S$ 
    define  $n$  where  $n \equiv \text{LEAST } n. \ x \in F \ (a \ n)$ 
    have  $n: x \in F \ (a \ n)$ 
    using  $\text{enum\_}S$  [OF  $\langle x \in S \rangle$ ] by (force simp: n_def intro: LeastI)
    have  $\text{notn}: x \notin F \ (a \ m)$  if  $m < n$  for  $m$ 
    using that not_less_Least by (force simp: n_def)
    then have  $x \notin \bigcup \{\text{closure } (G \ (a \ m)) \mid m. \ m < n\}$ 
    using  $n \ \langle K \subseteq S \rangle \ \langle \text{range } a = K \rangle \ clos \ \text{notn}$  by fastforce
    with  $n$  show  $x \in \text{Union } ?C$ 
    by blast
  qed
  show  $\bigwedge U. \ U \in ?C \implies \text{open } U \wedge (\exists T. \ T \in \mathcal{C} \wedge U \subseteq T)$ 

```

```

    using Fin ⟨K ⊆ S⟩ ⟨range a = K⟩ by (auto simp: odif)
  show ∃ V. open V ∧ x ∈ V ∧ finite {U. U ∈ ?C ∧ (U ∩ V ≠ {})} if x ∈ S
for x
  proof -
    obtain n where n: x ∈ F(a n) x ∈ G(a n)
    using ⟨x ∈ S⟩ enum_S by auto
    have {U ∈ ?C. U ∩ G(a n) ≠ {}} ⊆ (λn. F(a n) - ⋃ {closure(G(a m))
|m. m < n}) ‘ atMost n
  proof clarsimp
    fix k assume (F(a k) - ⋃ {closure(G(a m)) |m. m < k}) ∩ G(a n) ≠
{}
    then have k ≤ n
    by auto (metis closure_subset not_le subsetCE)
    then show F(a k) - ⋃ {closure(G(a m)) |m. m < k}
      ∈ (λn. F(a n) - ⋃ {closure(G(a m)) |m. m < n}) ‘ {..n}
    by force
  qed
  moreover have finite ((λn. F(a n) - ⋃ {closure(G(a m)) |m. m < n}) ‘
atMost n)
  by force
  ultimately have *: finite {U ∈ ?C. U ∩ G(a n) ≠ {}}
  using finite_subset by blast
  have a n ∈ S
  using ⟨K ⊆ S⟩ ⟨range a = K⟩ by blast
  then show ?thesis
  by (blast intro: oG n *)
qed
qed
qed

```

corollary paracompact_closedin:

```

  fixes S :: 'a :: {metric_space, second_countable_topology} set
  assumes cin: closedin (top_of_set U) S
    and oin: ⋀ T. T ∈ C ⇒ openin (top_of_set U) T
    and S ⊆ ⋃ C
  obtains C' where S ⊆ ⋃ C'
    and ⋀ V. V ∈ C' ⇒ openin (top_of_set U) V ∧ (∃ T. T ∈ C ∧ V
⊆ T)
    and ⋀ x. x ∈ U
      ⇒ ∃ V. openin (top_of_set U) V ∧ x ∈ V ∧
        finite {X. X ∈ C' ∧ (X ∩ V ≠ {})}
  proof -
    have ∃ Z. open Z ∧ (T = U ∩ Z) if T ∈ C for T
    using oin [OF that] by (auto simp: openin_open)
    then obtain F where opF: open (F T) and intF: U ∩ F T = T if T ∈ C for
T
    by metis
    obtain K where K: closed K U ∩ K = S
    using cin by (auto simp: closedin_closed)

```

```

have 1:  $U \subseteq \bigcup (\text{insert } (- K) (F \text{ ' } \mathcal{C}))$ 
  by clarsimp (metis Int_iff Union_iff  $\langle U \cap K = S \rangle \langle S \subseteq \bigcup \mathcal{C} \rangle \text{subsetD intF}$ )
have 2:  $\bigwedge T. T \in \text{insert } (- K) (F \text{ ' } \mathcal{C}) \implies \text{open } T$ 
  using  $\langle \text{closed } K \rangle$  by (auto simp: opF)
obtain  $\mathcal{D}$  where  $U \subseteq \bigcup \mathcal{D}$ 
  and D1:  $\bigwedge U. U \in \mathcal{D} \implies \text{open } U \wedge (\exists T. T \in \text{insert } (- K) (F \text{ ' } \mathcal{C}) \wedge U \subseteq T)$ 
  and D2:  $\bigwedge x. x \in U \implies \exists V. \text{open } V \wedge x \in V \wedge \text{finite } \{U \in \mathcal{D}. U \cap V \neq \{\}\}$ 
  by (blast intro: paracompact [OF 1 2])
let  $?C = \{U \cap V \mid V. V \in \mathcal{D} \wedge (V \cap K \neq \{\})\}$ 
show ?thesis
proof (rule_tac  $\mathcal{C}' = \{U \cap V \mid V. V \in \mathcal{D} \wedge (V \cap K \neq \{\})\}$  in that)
  show  $S \subseteq \bigcup ?C$ 
    using  $\langle U \cap K = S \rangle \langle U \subseteq \bigcup \mathcal{D} \rangle K$  by (blast dest!: subsetD)
  show  $\bigwedge V. V \in ?C \implies \text{openin } (\text{top\_of\_set } U) V \wedge (\exists T. T \in \mathcal{C} \wedge V \subseteq T)$ 
    using D1 intF by fastforce
  have *:  $\{X. (\exists V. X = U \cap V \wedge V \in \mathcal{D} \wedge V \cap K \neq \{\}) \wedge X \cap (U \cap V) \neq \{\}\} \subseteq$ 
     $(\lambda x. U \cap x) \text{ ' } \{U \in \mathcal{D}. U \cap V \neq \{\}\} \text{ for } V$ 
    by blast
  show  $\exists V. \text{openin } (\text{top\_of\_set } U) V \wedge x \in V \wedge \text{finite } \{X \in ?C. X \cap V \neq \{\}\}$ 
    if  $x \in U$  for  $x$ 
  proof -
    from D2 [OF that] obtain  $V$  where  $\text{open } V \wedge x \in V \wedge \text{finite } \{U \in \mathcal{D}. U \cap V \neq \{\}\}$ 
    by auto
    with * show ?thesis
      by (rule_tac  $x = U \cap V$  in exI) (auto intro: that finite_subset [OF *])
  qed
qed
qed

```

corollary *paracompact_closed:*

```

fixes  $S :: 'a :: \{\text{metric\_space}, \text{second\_countable\_topology}\}$  set
assumes closed S
  and opC:  $\bigwedge T. T \in \mathcal{C} \implies \text{open } T$ 
  and  $S \subseteq \bigcup \mathcal{C}$ 
obtains  $\mathcal{C}'$  where  $S \subseteq \bigcup \mathcal{C}'$ 
  and  $\bigwedge U. U \in \mathcal{C}' \implies \text{open } U \wedge (\exists T. T \in \mathcal{C} \wedge U \subseteq T)$ 
  and  $\bigwedge x. \exists V. \text{open } V \wedge x \in V \wedge$ 
     $\text{finite } \{U. U \in \mathcal{C}' \wedge (U \cap V \neq \{\})\}$ 
by (rule paracompact_closedin [of UNIV S C]) (auto simp: assms)

```

5.0.20 Closed-graph characterization of continuity

lemma *continuous_closed_graph_gen:*

```

fixes  $T :: 'b :: \text{real\_normed\_vector\_space}$  set

```

```

assumes contf: continuous_on S f and fm:  $f \text{ ' } S \subseteq T$ 
shows closedin (top_of_set ( $S \times T$ )) (( $\lambda x. \text{Pair } x (f x)$ ) ' S)
proof -
  have eq: (( $\lambda x. \text{Pair } x (f x)$ ) ' S) = ( $S \times T \cap (\lambda z. (f \circ \text{fst})z - \text{snd } z) - \{0\}$ )
    using fm by auto
  show ?thesis
    unfolding eq
    by (intro continuous_intros continuous_closedin_preimage continuous_on_subset
[OF contf]) auto
qed

```

```

lemma continuous_closed_graph_eq:
fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes compact T and fm:  $f \text{ ' } S \subseteq T$ 
shows continuous_on S  $f \longleftrightarrow$ 
  closedin (top_of_set ( $S \times T$ )) (( $\lambda x. \text{Pair } x (f x)$ ) ' S)
  (is ?lhs = ?rhs)
proof -
  have ?lhs if ?rhs
proof (clarsimp simp add: continuous_on_closed_gen [OF fm])
    fix U
    assume U: closedin (top_of_set T) U
    have eq: ( $S \cap f \text{ ' } U$ ) = fst ' ((( $\lambda x. \text{Pair } x (f x)$ ) ' S)  $\cap (S \times U)$ )
      by (force simp: image_iff)
    show closedin (top_of_set S) ( $S \cap f \text{ ' } U$ )
      by (simp add: U closedin_Int closedin_Times closed_map_fst [OF  $\langle \text{compact } T \rangle$ ] that eq)
    qed
  with continuous_closed_graph_gen assms show ?thesis by blast
qed

```

```

lemma continuous_closed_graph:
fixes f :: 'a::topological_space  $\Rightarrow$  'b::real_normed_vector
assumes closed S and contf: continuous_on S f
shows closed (( $\lambda x. \text{Pair } x (f x)$ ) ' S)
proof (rule closedin_closed_trans)
  show closedin (top_of_set ( $S \times \text{UNIV}$ )) (( $\lambda x. (x, f x)$ ) ' S)
    by (rule continuous_closed_graph_gen [OF contf subset_UNIV])
qed (simp add:  $\langle \text{closed } S \rangle$  closed_Times)

```

```

lemma continuous_from_closed_graph:
fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes compact T and fm:  $f \text{ ' } S \subseteq T$  and clo: closed (( $\lambda x. \text{Pair } x (f x)$ ) ' S)
shows continuous_on S f
  using fm clo
  by (auto intro: closed_subset simp: continuous_closed_graph_eq [OF  $\langle \text{compact } T \rangle$ ] fm])

```

```

lemma continuous_on_Un_local_open:

```

```

assumes opS: openin (top_of_set ( $S \cup T$ ))  $S$ 
and opT: openin (top_of_set ( $S \cup T$ ))  $T$ 
and contf: continuous_on  $S$   $f$  and contg: continuous_on  $T$   $g$ 
shows continuous_on ( $S \cup T$ )  $f$ 
using pasting_lemma [of  $\{S, T\}$  top_of_set ( $S \cup T$ ) id euclidean  $\lambda i. f\ f$ ] contf
contg opS opT
by (simp add: subtopology_subtopology) (metis inf.absorb2 openin_imp_subset)

```

```

lemma continuous_on_cases_local_open:
assumes opS: openin (top_of_set ( $S \cup T$ ))  $S$ 
and opT: openin (top_of_set ( $S \cup T$ ))  $T$ 
and contf: continuous_on  $S$   $f$  and contg: continuous_on  $T$   $g$ 
and fg:  $\bigwedge x. x \in S \wedge \neg P\ x \vee x \in T \wedge P\ x \implies f\ x = g\ x$ 
shows continuous_on ( $S \cup T$ ) ( $\lambda x. \text{if } P\ x \text{ then } f\ x \text{ else } g\ x$ )
proof -
have  $\bigwedge x. x \in S \implies (\text{if } P\ x \text{ then } f\ x \text{ else } g\ x) = f\ x$   $\bigwedge x. x \in T \implies (\text{if } P\ x \text{ then } f\ x \text{ else } g\ x) = g\ x$ 
by (simp_all add: fg)
then have continuous_on  $S$  ( $\lambda x. \text{if } P\ x \text{ then } f\ x \text{ else } g\ x$ ) continuous_on  $T$  ( $\lambda x. \text{if } P\ x \text{ then } f\ x \text{ else } g\ x$ )
by (simp_all add: contf contg cong: continuous_on_cong)
then show ?thesis
by (rule continuous_on_Un_local_open [OF opS opT])
qed

```

5.0.21 The union of two collinear segments is another segment

```

proposition in_convex_hull_exchange:
fixes  $a :: 'a :: \text{euclidean\_space}$ 
assumes  $a: a \in \text{convex\_hull } S$  and  $xS: x \in \text{convex\_hull } S$ 
obtains  $b$  where  $b \in S \wedge x \in \text{convex\_hull } (\text{insert } a\ (S - \{b\}))$ 
proof (cases  $a \in S$ )
case True
with  $xS$  insert_Diff that show ?thesis by fastforce
next
case False
show ?thesis
proof (cases finite  $S \wedge \text{card } S \leq \text{Suc } (\text{DIM } ('a))$ )
case True
then obtain  $u$  where  $u0: \bigwedge i. i \in S \implies 0 \leq u\ i$  and  $u1: \text{sum } u\ S = 1$ 
and  $ua: (\sum_{i \in S. u\ i *_{\mathbb{R}} i) = a$ 
using  $a$  by (auto simp: convex_hull_finite)
obtain  $v$  where  $v0: \bigwedge i. i \in S \implies 0 \leq v\ i$  and  $v1: \text{sum } v\ S = 1$ 
and  $vx: (\sum_{i \in S. v\ i *_{\mathbb{R}} i) = x$ 
using True  $xS$  by (auto simp: convex_hull_finite)
show ?thesis
proof (cases  $\exists b. b \in S \wedge v\ b = 0$ )

```

```

case True
then obtain b where b:  $b \in S \vee b = 0$ 
  by blast
show ?thesis
proof
  have fin: finite (insert a (S - {b}))
    using sum.infinite v1 by fastforce
  show  $x \in \text{convex hull insert a (S - \{b\})}$ 
    unfolding convex_hull_finite [OF fin] mem_Collect_eq
  proof (intro conjI exI ballI)
    have  $(\sum x \in \text{insert a (S - \{b\})}. \text{if } x = a \text{ then } 0 \text{ else } v x) =$ 
       $(\sum x \in S - \{b\}. \text{if } x = a \text{ then } 0 \text{ else } v x)$ 
      using fin by (force intro: sum.mono_neutral_right)
    also have  $\dots = (\sum x \in S - \{b\}. v x)$ 
      using b False by (auto intro!: sum.cong split: if_split_asm)
    also have  $\dots = (\sum_{x \in S}. v x)$ 
      by (metis  $\langle v b = 0 \rangle$  diff_zero sum.infinite sum_diff1 u1 zero_neq_one)
    finally show  $(\sum_{x \in \text{insert a (S - \{b\})}. \text{if } x = a \text{ then } 0 \text{ else } v x) = 1$ 
      by (simp add: v1)
    show  $\bigwedge x. x \in \text{insert a (S - \{b\})} \implies 0 \leq (\text{if } x = a \text{ then } 0 \text{ else } v x)$ 
      by (auto simp: v0)
    have  $(\sum x \in \text{insert a (S - \{b\})}. (\text{if } x = a \text{ then } 0 \text{ else } v x) *_R x) =$ 
       $(\sum x \in S - \{b\}. (\text{if } x = a \text{ then } 0 \text{ else } v x) *_R x)$ 
      using fin by (force intro: sum.mono_neutral_right)
    also have  $\dots = (\sum x \in S - \{b\}. v x *_R x)$ 
      using b False by (auto intro!: sum.cong split: if_split_asm)
    also have  $\dots = (\sum_{x \in S}. v x *_R x)$ 
      by (metis (no_types, lifting) b(2) diff_zero fin finite.emptyI finite_Diff2
finite_insert scale_eq_0_iff sum_diff1)
    finally show  $(\sum_{x \in \text{insert a (S - \{b\})}. (\text{if } x = a \text{ then } 0 \text{ else } v x) *_R x) =$ 
      x
      by (simp add: vx)
    qed
  qed (rule  $\langle b \in S \rangle$ )
next
case False
have le_Max:  $u i / v i \leq \text{Max } ((\lambda i. u i / v i) \text{ ` } S) \text{ if } i \in S \text{ for } i$ 
  by (simp add: True that)
have Max  $((\lambda i. u i / v i) \text{ ` } S) \in (\lambda i. u i / v i) \text{ ` } S$ 
  using True v1 by (auto intro: Max_in)
then obtain b where b  $\in S$  and beq:  $\text{Max } ((\lambda b. u b / v b) \text{ ` } S) = u b / v b$ 
  by blast
then have  $0 \neq u b / v b$ 
  using le_Max beq divide_le_0_iff le_numeral_extra(2) sum_nonpos u1
  by (metis False eq_iff v0)
then have  $0 < u b \wedge 0 < v b$ 
  using False  $\langle b \in S \rangle$  u0 v0 by force+
have fin: finite (insert a (S - {b}))
  using sum.infinite v1 by fastforce

```

```

show ?thesis
proof
  show  $x \in \text{convex\_hull\_insert } a (S - \{b\})$ 
    unfolding convex_hull_finite [OF fin] mem_Collect_eq
    proof (intro conjI exI ballI)
      have  $(\sum x \in \text{insert } a (S - \{b\}). \text{ if } x=a \text{ then } v \, b / u \, b \text{ else } v \, x - (v \, b / u \, b) * u \, x) =$ 
         $v \, b / u \, b + (\sum x \in S - \{b\}. v \, x - (v \, b / u \, b) * u \, x)$ 
        using  $\langle a \notin S \rangle \langle b \in S \rangle \text{ True}$ 
        by (auto intro!: sum.cong split: if_split_asm)
      also have  $\dots = v \, b / u \, b + (\sum x \in S - \{b\}. v \, x) - (v \, b / u \, b) * (\sum x \in S - \{b\}. u \, x)$ 
        by (simp add: Groups_Big.sum_subtractf sum_distrib_left)
      also have  $\dots = (\sum x \in S. v \, x)$ 
        using  $\langle 0 < u \, b \rangle \text{ True}$  by (simp add: Groups_Big.sum_diff1 u1 field_simps)
      finally show  $\text{sum } (\lambda x. \text{ if } x=a \text{ then } v \, b / u \, b \text{ else } v \, x - (v \, b / u \, b) * u \, x) (\text{insert } a (S - \{b\})) = 1$ 
        by (simp add: v1)
      show  $0 \leq (\text{if } i = a \text{ then } v \, b / u \, b \text{ else } v \, i - v \, b / u \, b * u \, i)$ 
        if  $i \in \text{insert } a (S - \{b\})$  for  $i$ 
        using  $\langle 0 < u \, b \rangle \langle 0 < v \, b \rangle v0 \text{ [of } i] \text{ le\_Max [of } i] \text{ beq that False}$ 
        by (auto simp: field_simps split: if_split_asm)
      have  $(\sum x \in \text{insert } a (S - \{b\}). (\text{if } x=a \text{ then } v \, b / u \, b \text{ else } v \, x - v \, b / u \, b * u \, x) *_R x) =$ 
         $(v \, b / u \, b) *_R a + (\sum x \in S - \{b\}. (v \, x - v \, b / u \, b * u \, x) *_R x)$ 
        using  $\langle a \notin S \rangle \langle b \in S \rangle \text{ True}$  by (auto intro!: sum.cong split: if_split_asm)
      also have  $\dots = (v \, b / u \, b) *_R a + (\sum x \in S - \{b\}. v \, x *_R x) - (v \, b / u \, b) *_R (\sum x \in S - \{b\}. u \, x *_R x)$ 
        by (simp add: Groups_Big.sum_subtractf scaleR_left_diff_distrib sum_distrib_left scale_sum_right)
      also have  $\dots = (\sum x \in S. v \, x *_R x)$ 
        using  $\langle 0 < u \, b \rangle \text{ True}$  by (simp add: ua vx Groups_Big.sum_diff1 algebra_simps)
      finally
        show  $(\sum x \in \text{insert } a (S - \{b\}). (\text{if } x=a \text{ then } v \, b / u \, b \text{ else } v \, x - v \, b / u \, b * u \, x) *_R x) = x$ 
        by (simp add: vx)
      qed
    qed (rule  $\langle b \in S \rangle$ )
  qed
next
  case False
  obtain  $T$  where  $\text{finite } T \, T \subseteq S$  and  $\text{ca}T: \text{card } T \leq \text{Suc } (\text{DIM}('a))$  and  $xT: x \in \text{convex\_hull } T$ 
    using  $xS$  by (auto simp: caratheodory [of  $S$ ])
  with False obtain  $b$  where  $b \in S \, b \notin T$ 
    by (metis antisym subsetI)
  show ?thesis

```



```

proof
  show  $x \in \text{convex\_hull insert } a (S - \{b\})$ 
    using  $\langle T \subseteq S \rangle$  b by (blast intro: subsetD [OF hull_mono xT])
  qed (rule  $\langle b \in S \rangle$ )
qed
qed

lemma convex_hull_exchange_Union:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  assumes  $a \in \text{convex\_hull } S$ 
  shows  $\text{convex\_hull } S = (\bigcup b \in S. \text{convex\_hull (insert } a (S - \{b\})))$  (is ?lhs =
  ?rhs)
proof
  show ?lhs  $\subseteq$  ?rhs
    by (blast intro: in_convex_hull_exchange [OF assms])
  show ?rhs  $\subseteq$  ?lhs
    proof clarify
      fix  $x$   $b$ 
      assume  $b \in S$   $x \in \text{convex\_hull insert } a (S - \{b\})$ 
      then show  $x \in \text{convex\_hull } S$  if  $b \in S$ 
        by (metis (no_types) that assms order_refl hull_mono hull_redundant in-
        sert_Diff_single insert_subset subsetCE)
      qed
    qed

lemma Un_closed_segment:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  assumes  $b \in \text{closed\_segment } a$   $c$ 
  shows  $\text{closed\_segment } a \cup \text{closed\_segment } b \ c = \text{closed\_segment } a \ c$ 
proof (cases  $c = a$ )
  case True
    with assms show ?thesis by simp
  next
    case False
    with assms have  $\text{convex\_hull } \{a, b\} \cup \text{convex\_hull } \{b, c\} = (\bigcup ba \in \{a, c\}. \text{convex\_hull insert } b (\{a, c\} - \{ba\}))$ 
    by (auto simp: insert_Diff_if insert_commute)
    then show ?thesis
      using convex_hull_exchange_Union
      by (metis assms segment_convex_hull)
    qed

lemma Un_open_segment:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  assumes  $b \in \text{open\_segment } a \ c$ 
  shows  $\text{open\_segment } a \cup \{b\} \cup \text{open\_segment } b \ c = \text{open\_segment } a \ c$  (is
  ?lhs = ?rhs)
proof -
  have  $b: b \in \text{closed\_segment } a \ c$ 

```

```

    by (simp add: assms open_closed_segment)
  have *: ?rhs  $\subseteq$  insert b (open_segment a b  $\cup$  open_segment b c)
    if {b,c,a}  $\cup$  open_segment a b  $\cup$  open_segment b c = {c,a}  $\cup$  ?rhs
  proof -
    have insert a (insert c (insert b (open_segment a b  $\cup$  open_segment b c))) =
insert a (insert c (?rhs))
    using that by (simp add: insert_commute)
    then show ?thesis
    by (metis (no_types) Diff_cancel Diff_eq_empty_iff Diff_insert2 open_segment_def)
  qed
show ?thesis
proof
  show ?lhs  $\subseteq$  ?rhs
    by (simp add: assms b subset_open_segment)
  show ?rhs  $\subseteq$  ?lhs
    using Un_closed_segment [OF b] *
    by (simp add: closed_segment_eq_open insert_commute)
qed
qed

```

5.0.22 Covering an open set by a countable chain of compact sets

proposition *open_Union_compact_subsets:*
fixes $S :: 'a::\text{euclidean\_space}$ *set*
assumes *open S*
obtains C **where** $\bigwedge n. \text{compact}(C\ n) \wedge n. C\ n \subseteq S$
 $\bigwedge n. C\ n \subseteq \text{interior}(C(\text{Suc } n))$
 $\bigcup (\text{range } C) = S$
 $\bigwedge K. [\![\text{compact } K; K \subseteq S]\!] \implies \exists N. \forall n \geq N. K \subseteq (C\ n)$

proof (*cases S = {}*)
case True
then show ?thesis
by (*rule_tac C = $\lambda n. \{\}$ in that*) *auto*

next
case False
then obtain a where a $\in S$
by auto
let $?C = \lambda n. \text{cball } a (\text{real } n) - (\bigcup x \in -S. \bigcup e \in \text{ball } 0\ (1 / \text{real}(\text{Suc } n)). \{x + e\})$
have $\exists N. \forall n \geq N. K \subseteq (f\ n)$
if $\bigwedge n. \text{compact}(f\ n)$ **and** *sub_int: $\bigwedge n. f\ n \subseteq \text{interior}(f(\text{Suc } n))$*
and *eq: $\bigcup (\text{range } f) = S$* **and** *compact K* $K \subseteq S$ **for** $f\ K$

proof -
have *: $\forall n. f\ n \subseteq (\bigcup n. \text{interior}(f\ n))$
by (*meson Sup_upper2 UNIV_I $\langle \bigwedge n. f\ n \subseteq \text{interior}(f(\text{Suc } n)) \rangle$ image_iff*)
have *mono: $\bigwedge m n. m \leq n \implies f\ m \subseteq f\ n$*
by (*meson dual_order.trans interior_subset lift_Suc_mono_le sub_int*)
obtain I where finite I and I: $K \subseteq (\bigcup i \in I. \text{interior}(f\ i))$

```

proof (rule compactE_image [OF ‹compact K›])
  show  $K \subseteq (\bigcup n. \text{interior } (f\ n))$ 
    using ‹ $K \subseteq S$ › ‹ $\bigcup (f \text{ ` } UNIV) = S$ › by blast
qed auto
{ fix  $n$ 
  assume  $n: \text{Max } I \leq n$ 
  have  $(\bigcup i \in I. \text{interior } (f\ i)) \subseteq f\ n$ 
    by (rule UN_least) (meson dual_order.trans interior_subset mono I Max_ge
[OF ‹finite I›]  $n$ )
  then have  $K \subseteq f\ n$ 
    using  $I$  by auto
}
then show ?thesis
  by blast
qed
moreover have  $\exists f. (\forall n. \text{compact}(f\ n)) \wedge (\forall n. (f\ n) \subseteq S) \wedge (\forall n. (f\ n) \subseteq$ 
interior( $f(Suc\ n)$ ))  $\wedge$ 
   $((\bigcup (\text{range } f) = S))$ 
proof (intro exI conjI allI)
  show  $\bigwedge n. \text{compact } (?C\ n)$ 
    by (auto simp: compact_diff open_sums)
  show  $\bigwedge n. ?C\ n \subseteq S$ 
    by auto
  show  $?C\ n \subseteq \text{interior } (?C\ (Suc\ n))$  for  $n$ 
proof (simp add: interior_diff, rule Diff_mono)
    show  $cball\ a\ (\text{real } n) \subseteq ball\ a\ (1 + \text{real } n)$ 
      by (simp add: cball_subset_ball_iff)
    have  $cl: \text{closed } (\bigcup x \in -S. \bigcup e \in cball\ 0\ (1 / (2 + \text{real } n)). \{x + e\})$ 
      using assms by (auto intro: closed_compact_sums)
    have  $\text{closure } (\bigcup x \in -S. \bigcup y \in ball\ 0\ (1 / (2 + \text{real } n)). \{x + y\})$ 
       $\subseteq (\bigcup x \in -S. \bigcup e \in cball\ 0\ (1 / (2 + \text{real } n)). \{x + e\})$ 
      by (intro closure_minimal UN_mono ball_subset_cball order_refl cl)
    also have  $\dots \subseteq (\bigcup x \in -S. \bigcup y \in ball\ 0\ (1 / (1 + \text{real } n)). \{x + y\})$ 
      by (simp add: cball_subset_ball_iff field_split_simps UN_mono)
    finally show  $\text{closure } (\bigcup x \in -S. \bigcup y \in ball\ 0\ (1 / (2 + \text{real } n)). \{x + y\})$ 
       $\subseteq (\bigcup x \in -S. \bigcup y \in ball\ 0\ (1 / (1 + \text{real } n)). \{x + y\})$  .
qed
have  $S \subseteq \bigcup (\text{range } ?C)$ 
proof
  fix  $x$ 
  assume  $x: x \in S$ 
  then obtain  $e$  where  $e > 0$  and  $e: ball\ x\ e \subseteq S$ 
    using assms open_contains_ball by blast
  then obtain  $N1$  where  $N1 > 0$  and  $N1: \text{real } N1 > 1/e$ 
    using reals_Archimedean2
    by (metis divide_less_0_iff less_eq_real_def neq0_conv not_le of_nat_0
of_nat_1 of_nat_less_0_iff)
  obtain  $N2$  where  $N2: \text{norm}(x - a) \leq \text{real } N2$ 
    by (meson real_arch_simple)

```

```

have N12: inverse((N1 + N2) + 1) ≤ inverse(N1)
  using ⟨N1 > 0⟩ by (auto simp: field_split_simps)
have x ≠ y + z if y ∉ S norm z < 1 / (1 + (real N1 + real N2)) for y z
proof -
  have e * real N1 < e * (1 + (real N1 + real N2))
    by (simp add: ⟨0 < e⟩)
  then have 1 / (1 + (real N1 + real N2)) < e
    using N1 ⟨e > 0⟩
  by (metis divide_less_eq less_trans mult.commute of_nat_add of_nat_less_0_iff
of_nat_Suc)
  then have x - z ∈ ball x e
    using that by simp
  then have x - z ∈ S
    using e by blast
  with that show ?thesis
    by auto
qed
with N2 show x ∈ ⋃ (range ?C)
  by (rule_tac a = N1+N2 in UN_I) (auto simp: dist_norm norm_minus_commute)
qed
then show ⋃ (range ?C) = S by auto
qed
ultimately show ?thesis
  using that by metis
qed

```

5.0.23 Orthogonal complement

definition *orthogonal_comp* ($\_^\perp$ [80] 80)
 where *orthogonal_comp* $W \equiv \{x. \forall y \in W. \text{orthogonal } y \ x\}$

proposition *subspace_orthogonal_comp*: *subspace* ($W^\perp$)
unfolding *subspace_def* *orthogonal_comp_def* *orthogonal_def*
 by (auto simp: inner_right_distrib)

lemma *orthogonal_comp_anti_mono*:

assumes $A \subseteq B$
 shows $B^\perp \subseteq A^\perp$

proof

fix x assume x: $x \in B^\perp$

show $x \in \text{orthogonal\_comp } A$ using x **unfolding** *orthogonal_comp_def*
 by (simp add: *orthogonal_def*, metis *assms* in_mono)

qed

lemma *orthogonal_comp_null* [simp]: $\{0\}^\perp = \text{UNIV}$
 by (auto simp: *orthogonal_comp_def* *orthogonal_def*)

lemma *orthogonal_comp_UNIV* [simp]: $\text{UNIV}^\perp = \{0\}$
unfolding *orthogonal_comp_def* *orthogonal_def*

```

    by auto (use inner_eq_zero_iff in blast)

lemma orthogonal_comp_subset:  $U \subseteq U^{\perp\perp}$ 
  by (auto simp: orthogonal_comp_def orthogonal_def inner_commute)

lemma subspace_sum_minimal:
  assumes  $S \subseteq U$   $T \subseteq U$  subspace  $U$ 
  shows  $S + T \subseteq U$ 
proof
  fix x
  assume  $x \in S + T$ 
  then obtain xs xt where  $xs \in S$   $xt \in T$   $x = xs + xt$ 
  by (meson set_plus_elim)
  then show  $x \in U$ 
  by (meson assms subsetCE subspace_add)
qed

proposition subspace_sum_orthogonal_comp:
  fixes  $U :: 'a :: euclidean\_space$  set
  assumes subspace  $U$ 
  shows  $U + U^{\perp} = UNIV$ 
proof -
  obtain B where  $B \subseteq U$ 
  and ortho: pairwise orthogonal B  $\wedge x. x \in B \implies \text{norm } x = 1$ 
  and independent B  $\text{card } B = \dim U$   $\text{span } B = U$ 
  using orthonormal_basis_subspace [OF assms] by metis
  then have finite B
  by (simp add: indep_card_eq_dim_span)
  have *:  $\forall x \in B. \forall y \in B. x \cdot y = (\text{if } x=y \text{ then } 1 \text{ else } 0)$ 
  using ortho norm_eq_1 by (auto simp: orthogonal_def pairwise_def)
  { fix v
    let ?u =  $\sum_{b \in B. (v \cdot b) *_{\mathbb{R}} b}$ 
    have  $v = ?u + (v - ?u)$ 
    by simp
    moreover have  $?u \in U$ 
    by (metis (no_types, lifting)  $\langle \text{span } B = U \rangle$  assms subspace_sum span_base span_mul)
    moreover have  $(v - ?u) \in U^{\perp}$ 
  }
  proof (clarsimp simp: orthogonal_comp_def orthogonal_def)
    fix y
    assume  $y \in U$ 
    with  $\langle \text{span } B = U \rangle$  span_finite [OF  $\langle \text{finite } B \rangle$ ]
    obtain u where  $u = (\sum_{b \in B. u \cdot b *_{\mathbb{R}} b)$ 
    by auto
    have  $b \cdot (v - ?u) = 0$  if  $b \in B$  for b
    using that  $\langle \text{finite } B \rangle$ 
    by (simp add: * algebra_simps inner_sum_right if_distrib [of (*) v for v]
    inner_commute cong: if_cong)
    then show  $y \cdot (v - ?u) = 0$ 
  end

```

```

      by (simp add: u inner_sum_left)
    qed
    ultimately have  $v \in U + U^\perp$ 
      using set_plus_intro by fastforce
  } then show ?thesis
    by auto
qed

```

```

lemma orthogonal_Int_0:
  assumes subspace U
  shows  $U \cap U^\perp = \{0\}$ 
  using orthogonal_comp_def orthogonal_self
  by (force simp: assms subspace_0 subspace_orthogonal_comp)

```

```

lemma orthogonal_comp_self:
  fixes U :: 'a :: euclidean_space set
  assumes subspace U
  shows  $U^{\perp\perp} = U$ 
proof
  have ssU': subspace ( $U^\perp$ )
    by (simp add: subspace_orthogonal_comp)
  have  $u \in U$  if  $u \in U^{\perp\perp}$  for u
  proof -
    obtain v w where  $u = v + w$   $v \in U$   $w \in U^\perp$ 
      using subspace_sum_orthogonal_comp [OF assms] set_plus_elim by blast
    then have  $u - v \in U^\perp$ 
      by simp
    moreover have  $v \in U^{\perp\perp}$ 
      using  $\langle v \in U \rangle$  orthogonal_comp_subset by blast
    then have  $u - v \in U^{\perp\perp}$ 
      by (simp add: subspace_diff subspace_orthogonal_comp that)
    ultimately have  $u - v = 0$ 
      using orthogonal_Int_0 ssU' by blast
    with  $\langle v \in U \rangle$  show ?thesis
      by auto
  qed
  then show  $U^{\perp\perp} \subseteq U$ 
    by auto
qed (use orthogonal_comp_subset in auto)

```

```

lemma ker_orthogonal_comp_adjoint:
  fixes f :: 'm::euclidean_space  $\Rightarrow$  'n::euclidean_space
  assumes linear f
  shows  $f^{-1} \{0\} = (\text{range } (\text{adjoint } f))^\perp$ 
proof -
  have  $\bigwedge x. [\forall y. y \cdot f x = 0] \implies f x = 0$ 
    using assms inner_commute all_zero_iff by metis
  then show ?thesis
    using assms

```

```

  by (auto simp: orthogonal_comp_def orthogonal_def adjoint_works inner_commute)
qed

```

5.0.24 A non-injective linear function maps into a hyperplane.

```

lemma linear_surj_adj_imp_inj:
  fixes f :: 'm::euclidean_space  $\Rightarrow$  'n::euclidean_space
  assumes linear f surj (adjoint f)
  shows inj f
proof -
  have  $\exists x. y = \text{adjoint } f \ x$  for y
    using assms by (simp add: surjD)
  then show inj f
    using assms unfolding inj_on_def image_def
    by (metis (no_types) adjoint_works euclidean_eqI)
qed

```

— <https://mathonline.wikidot.com/injectivity-and-surjectivity-of-the-adjoint-of-a-linear-map>

```

lemma surj_adjoint_iff_inj [simp]:
  fixes f :: 'm::euclidean_space  $\Rightarrow$  'n::euclidean_space
  assumes linear f
  shows surj (adjoint f)  $\longleftrightarrow$  inj f
proof
  assume surj (adjoint f)
  then show inj f
    by (simp add: assms linear_surj_adj_imp_inj)
next
  assume inj f
  have f - '{0} = {0}
    using assms <inj f> linear_0 linear_injective_0 by fastforce
  moreover have f - '{0} = range (adjoint f)⊥
    by (intro ker_orthogonal_comp_adjoint assms)
  ultimately have range (adjoint f)⊥⊥ = UNIV
    by (metis orthogonal_comp_null)
  then show surj (adjoint f)
    using adjoint_linear <linear f>
    by (subst (asm) orthogonal_comp_self)
    (simp add: adjoint_linear linear_subspace_image)
qed

```

```

lemma inj_adjoint_iff_surj [simp]:
  fixes f :: 'm::euclidean_space  $\Rightarrow$  'n::euclidean_space
  assumes linear f
  shows inj (adjoint f)  $\longleftrightarrow$  surj f
proof
  assume inj (adjoint f)
  have (adjoint f) - '{0} = {0}
    by (metis <inj (adjoint f)> adjoint_linear assms surj_adjoint_iff_inj ker_orthogonal_comp_adjoint

```

```

orthogonal_comp_UNIV)
  then have  $(\text{range}(f))^\perp = \{0\}$ 
    by (metis (no_types, opaque_lifting) adjoint_adjoint adjoint_linear assms
ker_orthogonal_comp_adjoint set_zero)
  then show surj f
    by (metis  $\langle \text{inj} (\text{adjoint } f) \rangle$  adjoint_adjoint adjoint_linear assms surj_adjoint_iff_inj)
next
  assume surj f
  then have  $\text{range } f = (\text{adjoint } f - \{0\})^\perp$ 
    by (simp add: adjoint_adjoint adjoint_linear assms ker_orthogonal_comp_adjoint)
  then have  $\{0\} = \text{adjoint } f - \{0\}$ 
    using  $\langle \text{surj } f \rangle$  adjoint_adjoint adjoint_linear assms ker_orthogonal_comp_adjoint
  by force
  then show inj (adjoint f)
    by (simp add:  $\langle \text{surj } f \rangle$  adjoint_adjoint adjoint_linear assms linear_surj_adj_imp_inj)
qed

```

lemma *linear_singular_into_hyperplane*:

```

fixes f :: 'n::euclidean_space  $\Rightarrow$  'n
assumes linear f
shows  $\neg \text{inj } f \iff (\exists a. a \neq 0 \wedge (\forall x. a \cdot f x = 0))$  (is _ = ?rhs)

```

proof

```

  assume  $\neg \text{inj } f$ 
  then show ?rhs
    using all_zero_iff
    by (metis (no_types, opaque_lifting) adjoint_clauses(2) adjoint_linear assms
linear_injective_0 linear_injective_imp_surjective linear_surj_adj_imp_inj)

```

next

```

  assume ?rhs
  then show  $\neg \text{inj } f$ 
    by (metis assms linear_injective_isomorphism all_zero_iff)

```

qed

lemma *linear_singular_image_hyperplane*:

```

fixes f :: 'n::euclidean_space  $\Rightarrow$  'n
assumes linear f  $\neg \text{inj } f$ 
obtains a where  $a \neq 0 \wedge S. f \restriction S \subseteq \{x. a \cdot x = 0\}$ 
using assms by (fastforce simp add: linear_singular_into_hyperplane)

```

end

5.1 The binary product topology

```

theory Product_Topology
imports Function_Topology
begin

```


5.2 Product Topology

5.2.1 Definition

definition *prod_topology* :: *'a topology* $\Rightarrow$ *'b topology* $\Rightarrow$ (*'a* $\times$ *'b*) *topology* **where**
prod_topology *X Y* $\equiv$ *topology* (*arbitrary union_of* ($\lambda U. U \in \{S \times T \mid S \text{ T. } \text{openin } X S \wedge \text{openin } Y T\}$))

lemma *open_product_open*:

assumes *open A*

shows $\exists \mathcal{U}. \mathcal{U} \subseteq \{S \times T \mid S \text{ T. } \text{open } S \wedge \text{open } T\} \wedge \bigcup \mathcal{U} = A$

proof –

obtain *f g* **where** $\ast: \bigwedge u. u \in A \implies \text{open } (f u) \wedge \text{open } (g u) \wedge u \in (f u) \times (g u) \wedge (f u) \times (g u) \subseteq A$

using *open_prod_def* [*of A*] **assms** **by** *metis*

let $\mathcal{U} = (\lambda u. f u \times g u) \text{ ` } A$

show *?thesis*

by (*rule_tac* $x = \mathcal{U}$ **in** *exI*) (*auto simp: dest: **)

qed

lemma *open_product_open_eq*: (*arbitrary union_of* ($\lambda U. \exists S \text{ T. } U = S \times T \wedge \text{open } S \wedge \text{open } T$)) = *open*

by (*force simp: union_of_def arbitrary_def intro: open_product_open open_Times*)

lemma *openin_prod_topology*:

openin (*prod_topology* *X Y*) = *arbitrary union_of* ($\lambda U. U \in \{S \times T \mid S \text{ T. } \text{openin } X S \wedge \text{openin } Y T\}$)

unfolding *prod_topology_def*

proof (*rule topology_inverse'*)

show *istopology* (*arbitrary union_of* ($\lambda U. U \in \{S \times T \mid S \text{ T. } \text{openin } X S \wedge \text{openin } Y T\}$))

apply (*rule istopology_base, simp*)

by (*metis openin_Int Times_Int Times*)

qed

lemma *topspace_prod_topology* [*simp*]:

topspace (*prod_topology* *X Y*) = *topspace* *X* $\times$ *topspace* *Y*

proof –

have *topspace*(*prod_topology* *X Y*) = $\bigcup$ (*Collect* (*openin* (*prod_topology* *X Y*)))
(is _ = ?Z)

unfolding *topspace_def* ..

also have ... = *topspace* *X* $\times$ *topspace* *Y*

proof

show $?Z \subseteq \text{topspace } X \times \text{topspace } Y$

apply (*auto simp: openin_prod_topology union_of_def arbitrary_def*)

using *openin_subset* **by** *force+*

next

have $\ast: \exists A \text{ B. } \text{topspace } X \times \text{topspace } Y = A \times B \wedge \text{openin } X A \wedge \text{openin } Y B$

B

by *blast*

```

show  $\text{topspace } X \times \text{topspace } Y \subseteq ?Z$ 
apply (rule Union_upper)
using * by (simp add: openin_prod_topology arbitrary_union_of_inc)
qed
finally show ?thesis .
qed

```

```

lemma subtopology_Times:
shows  $\text{subtopology } (\text{prod\_topology } X \ Y) \ (S \times T) = \text{prod\_topology } (\text{subtopology } X \ S) \ (\text{subtopology } Y \ T)$ 
proof -
have  $((\lambda U. \exists S \ T. U = S \times T \wedge \text{openin } X \ S \wedge \text{openin } Y \ T) \text{ relative\_to } S \times T) =$ 
 $(\lambda U. \exists S' \ T'. U = S' \times T' \wedge (\text{openin } X \text{ relative\_to } S) \ S' \wedge (\text{openin } Y \text{ relative\_to } T) \ T')$ 
by (auto simp: relative_to_def Times_Int_Times fun_eq_iff) metis
then show ?thesis
by (simp add: topology_eq openin_prod_topology arbitrary_union_of_relative_to flip: openin_relative_to)
qed

```

```

lemma prod_topology_subtopology:
 $\text{prod\_topology } (\text{subtopology } X \ S) \ Y = \text{subtopology } (\text{prod\_topology } X \ Y) \ (S \times \text{topspace } Y)$ 
 $\text{prod\_topology } X \ (\text{subtopology } Y \ T) = \text{subtopology } (\text{prod\_topology } X \ Y) \ (\text{topspace } X \times T)$ 
by (auto simp: subtopology_Times)

```

```

lemma prod_topology_discrete_topology:
 $\text{discrete\_topology } (S \times T) = \text{prod\_topology } (\text{discrete\_topology } S) \ (\text{discrete\_topology } T)$ 
by (auto simp: discrete_topology_unique openin_prod_topology intro: arbitrary_union_of_inc)

```

```

lemma prod_topology_euclidean [simp]:  $\text{prod\_topology euclidean euclidean} = \text{euclidean}$ 
by (simp add: prod_topology_def open_product_open_eq)

```

```

lemma prod_topology_subtopology_eu [simp]:
 $\text{prod\_topology } (\text{subtopology euclidean } S) \ (\text{subtopology euclidean } T) = \text{subtopology euclidean } (S \times T)$ 
by (simp add: prod_topology_subtopology subtopology_subtopology Times_Int_Times)

```

```

lemma openin_prod_topology_alt:
 $\text{openin } (\text{prod\_topology } X \ Y) \ S \longleftrightarrow$ 
 $(\forall x \ y. (x, y) \in S \longrightarrow (\exists U \ V. \text{openin } X \ U \wedge \text{openin } Y \ V \wedge x \in U \wedge y \in V \wedge U \times V \subseteq S))$ 
apply (auto simp: openin_prod_topology arbitrary_union_of_alt, fastforce)
by (metis mem_Sigma_iff)

```

```

lemma open_map_fst: open_map (prod_topology X Y) X fst
  unfolding open_map_def openin_prod_topology_alt
  by (force simp: openin_subopen [of X fst ' _] intro: subset_fst_imageI)

lemma open_map_snd: open_map (prod_topology X Y) Y snd
  unfolding open_map_def openin_prod_topology_alt
  by (force simp: openin_subopen [of Y snd ' _] intro: subset_snd_imageI)

lemma openin_prod_Times_iff:
  openin (prod_topology X Y) (S × T) ⟷ S = {} ∨ T = {} ∨ openin X S ∧
  openin Y T
proof (cases S = {} ∨ T = {})
  case False
  then show ?thesis
  apply (simp add: openin_prod_topology_alt openin_subopen [of X S] openin_subopen
    [of Y T] times_subset_iff, safe)
  apply (meson|force)+
  done
qed force

lemma closure_of_Times:
  (prod_topology X Y) closure_of (S × T) = (X closure_of S) × (Y closure_of
  T) (is ?lhs = ?rhs)
proof
  show ?lhs ⊆ ?rhs
  by (clarsimp simp: closure_of_def openin_prod_topology_alt) blast
  show ?rhs ⊆ ?lhs
  by (clarsimp simp: closure_of_def openin_prod_topology_alt) (meson SigmaI
  subsetD)
qed

lemma closedin_prod_Times_iff:
  closedin (prod_topology X Y) (S × T) ⟷ S = {} ∨ T = {} ∨ closedin X S
  ∧ closedin Y T
  by (auto simp: closure_of_Times times_eq_iff simp flip: closure_of_eq)

lemma interior_of_Times: (prod_topology X Y) interior_of (S × T) = (X in-
  terior_of S) × (Y interior_of T)
proof (rule interior_of_unique)
  show (X interior_of S) × Y interior_of T ⊆ S × T
  by (simp add: Sigma_mono interior_of_subset)
  show openin (prod_topology X Y) ((X interior_of S) × Y interior_of T)
  by (simp add: openin_prod_Times_iff)
next
  show T' ⊆ (X interior_of S) × Y interior_of T if T' ⊆ S × T openin
  (prod_topology X Y) T' for T'
  proof (clarsimp; intro conjI)
  fix a :: 'a and b :: 'b

```

```

    assume (a, b) ∈ T'
    with that obtain U V where UV: openin X U openin Y V a ∈ U b ∈ V U ×
V ⊆ T'
    by (metis openin_prod_topology_alt)
    then show a ∈ X interior_of S
    using interior_of_maximal_eq that(1) by fastforce
    show b ∈ Y interior_of T
    using UV interior_of_maximal_eq that(1)
    by (metis SigmaI mem_Sigma_iff subset_eq)
qed
qed

```

5.2.2 Continuity

```

lemma continuous_map_pairwise:
  continuous_map Z (prod_topology X Y) f ⟷ continuous_map Z X (fst ∘ f)
  ∧ continuous_map Z Y (snd ∘ f)
  (is ?lhs = ?rhs)
proof -
  let ?g = fst ∘ f and ?h = snd ∘ f
  have f: f x = (?g x, ?h x) for x
  by auto
  show ?thesis
  proof (cases (∀ x ∈ topspace Z. ?g x ∈ topspace X) ∧ (∀ x ∈ topspace Z. ?h x ∈
topspace Y))
    case True
    show ?thesis
    proof safe
      assume continuous_map Z (prod_topology X Y) f
      then have openin Z {x ∈ topspace Z. fst (f x) ∈ U} if openin X U for U
      unfolding continuous_map_def using True that
      apply clarify
      apply (drule_tac x=U in spec)
      by (simp add: openin_prod_Times_iff mem_Times_iff cong: conj_cong)
      with True show continuous_map Z X (fst ∘ f)
      by (auto simp: continuous_map_def)
    next
      assume continuous_map Z (prod_topology X Y) f
      then have openin Z {x ∈ topspace Z. snd (f x) ∈ V} if openin Y V for V
      unfolding continuous_map_def using True that
      apply clarify
      apply (drule_tac x=topspace X × V in spec)
      by (simp add: openin_prod_Times_iff mem_Times_iff cong: conj_cong)
      with True show continuous_map Z Y (snd ∘ f)
      by (auto simp: continuous_map_def)
    next
      assume Z: continuous_map Z X (fst ∘ f) continuous_map Z Y (snd ∘ f)
      have *: openin Z {x ∈ topspace Z. f x ∈ W}
      if ∧ w. w ∈ W ⟹ ∃ U V. openin X U ∧ openin Y V ∧ w ∈ U × V ∧ U

```

```

× V ⊆ W for W
proof (subst openin_subopen, clarify)
  fix x :: 'a
  assume x ∈ topspace Z and f x ∈ W
  with that [OF ⟨f x ∈ W⟩]
  obtain U V where UV: openin X U openin Y V f x ∈ U × V U × V ⊆ W
  by auto
  with Z UV show ∃ T. openin Z T ∧ x ∈ T ∧ T ⊆ {x ∈ topspace Z. f x
∈ W}
  apply (rule_tac x={x ∈ topspace Z. ?g x ∈ U} ∩ {x ∈ topspace Z. ?h x
∈ V} in exI)
  apply (auto simp: ⟨x ∈ topspace Z⟩ continuous_map_def)
  done
qed
show continuous_map Z (prod_topology X Y) f
  using True by (simp add: continuous_map_def openin_prod_topology_alt
mem_Times_iff *)
qed
qed (auto simp: continuous_map_def)
qed

```

lemma continuous_map_paired:

$$\text{continuous\_map } Z \text{ (prod\_topology } X \ Y) \ (\lambda x. (f \ x, g \ x)) \longleftrightarrow \text{continuous\_map } Z \ X \ f \wedge \text{continuous\_map } Z \ Y \ g$$

by (simp add: continuous_map_pairwise o_def)

lemma continuous_map_pairedI [continuous_intros]:

$$\llbracket \text{continuous\_map } Z \ X \ f; \text{continuous\_map } Z \ Y \ g \rrbracket \Longrightarrow \text{continuous\_map } Z \text{ (prod\_topology } X \ Y) \ (\lambda x. (f \ x, g \ x))$$

by (simp add: continuous_map_pairwise o_def)

lemma continuous_map_fst [continuous_intros]: continuous_map (prod_topology X Y) X fst

using continuous_map_pairwise [of prod_topology X Y X Y id]

by (simp add: continuous_map_pairwise)

lemma continuous_map_snd [continuous_intros]: continuous_map (prod_topology X Y) Y snd

using continuous_map_pairwise [of prod_topology X Y X Y id]

by (simp add: continuous_map_pairwise)

lemma continuous_map_fst_of [continuous_intros]:

$$\text{continuous\_map } Z \text{ (prod\_topology } X \ Y) \ f \Longrightarrow \text{continuous\_map } Z \ X \ (fst \circ f)$$

by (simp add: continuous_map_pairwise)

lemma continuous_map_snd_of [continuous_intros]:

$$\text{continuous\_map } Z \text{ (prod\_topology } X \ Y) \ f \Longrightarrow \text{continuous\_map } Z \ Y \ (snd \circ f)$$

by (simp add: continuous_map_pairwise)

lemma *continuous_map_prod_fst*:

$i \in I \implies \text{continuous\_map } (\text{prod\_topology } (\text{product\_topology } (\lambda i. Y) I) X) Y$
 $(\lambda x. \text{fst } x \ i)$

using *continuous_map_componentwise_UNIV continuous_map_fst* **by** *fastforce*

lemma *continuous_map_prod_snd*:

$i \in I \implies \text{continuous\_map } (\text{prod\_topology } X (\text{product\_topology } (\lambda i. Y) I)) Y$
 $(\lambda x. \text{snd } x \ i)$

using *continuous_map_componentwise_UNIV continuous_map_snd* **by** *fastforce*

lemma *continuous_map_if_iff* [*simp*]: *continuous_map* $X \ Y \ (\lambda x. \text{if } P \text{ then } f \ x \text{ else } g \ x) \longleftrightarrow \text{continuous\_map } X \ Y \ (\text{if } P \text{ then } f \text{ else } g)$

by *simp*

lemma *continuous_map_if* [*continuous_intros*]: $\llbracket P \implies \text{continuous\_map } X \ Y \ f; \sim P \implies \text{continuous\_map } X \ Y \ g \rrbracket$

$\implies \text{continuous\_map } X \ Y \ (\lambda x. \text{if } P \text{ then } f \ x \text{ else } g \ x)$

by *simp*

lemma *continuous_map_subtopology_fst* [*continuous_intros*]: *continuous_map* $(\text{subtopology } (\text{prod\_topology } X \ Y) \ Z) \ X \ \text{fst}$

using *continuous_map_from_subtopology continuous_map_fst* **by** *force*

lemma *continuous_map_subtopology_snd* [*continuous_intros*]: *continuous_map* $(\text{subtopology } (\text{prod\_topology } X \ Y) \ Z) \ Y \ \text{snd}$

using *continuous_map_from_subtopology continuous_map_snd* **by** *force*

lemma *quotient_map_fst* [*simp*]:

quotient_map $(\text{prod\_topology } X \ Y) \ X \ \text{fst} \longleftrightarrow (\text{topspace } Y = \{\} \longrightarrow \text{topspace } X = \{\})$

by (*auto simp: continuous_open_quotient_map open_map_fst continuous_map_fst*)

lemma *quotient_map_snd* [*simp*]:

quotient_map $(\text{prod\_topology } X \ Y) \ Y \ \text{snd} \longleftrightarrow (\text{topspace } X = \{\} \longrightarrow \text{topspace } Y = \{\})$

by (*auto simp: continuous_open_quotient_map open_map_snd continuous_map_snd*)

lemma *retraction_map_fst*:

retraction_map $(\text{prod\_topology } X \ Y) \ X \ \text{fst} \longleftrightarrow (\text{topspace } Y = \{\} \longrightarrow \text{topspace } X = \{\})$

proof (*cases topspace Y = {}*)

case *True*

then show *?thesis*

using *continuous_map_image_subset_topspace*

by (*fastforce simp: retraction_map_def retraction_maps_def continuous_map_fst continuous_map_on_empty*)

next

case *False*

```

have  $\exists g. \text{continuous\_map } X (\text{prod\_topology } X \ Y) \ g \wedge (\forall x \in \text{topspace } X. \text{fst } (g \ x) = x)$ 
  if  $y: y \in \text{topspace } Y$  for  $y$ 
  by  $(\text{rule\_tac } x = \lambda x. (x, y) \text{ in } exI) (\text{auto simp: } y \text{ continuous\_map\_paired})$ 
with  $\text{False}$  have  $\text{retraction\_map } (\text{prod\_topology } X \ Y) \ X \ \text{fst}$ 
  by  $(\text{fastforce simp: retraction\_map\_def retraction\_maps\_def continuous\_map\_fst})$ 
with  $\text{False}$  show  $?thesis$ 
  by  $\text{simp}$ 
qed

```

```

lemma  $\text{retraction\_map\_snd}$ :
   $\text{retraction\_map } (\text{prod\_topology } X \ Y) \ Y \ \text{snd} \longleftrightarrow (\text{topspace } X = \{\} \longrightarrow \text{topspace } Y = \{\})$ 
proof  $(\text{cases } \text{topspace } X = \{\})$ 
  case  $\text{True}$ 
  then show  $?thesis$ 
    using  $\text{continuous\_map\_image\_subset\_topspace}$ 
    by  $(\text{fastforce simp: retraction\_map\_def retraction\_maps\_def continuous\_map\_fst continuous\_map\_on\_empty})$ 
  next
  case  $\text{False}$ 
  have  $\exists g. \text{continuous\_map } Y (\text{prod\_topology } X \ Y) \ g \wedge (\forall y \in \text{topspace } Y. \text{snd } (g \ y) = y)$ 
  if  $x: x \in \text{topspace } X$  for  $x$ 
  by  $(\text{rule\_tac } x = \lambda y. (x, y) \text{ in } exI) (\text{auto simp: } x \text{ continuous\_map\_paired})$ 
with  $\text{False}$  have  $\text{retraction\_map } (\text{prod\_topology } X \ Y) \ Y \ \text{snd}$ 
  by  $(\text{fastforce simp: retraction\_map\_def retraction\_maps\_def continuous\_map\_snd})$ 
with  $\text{False}$  show  $?thesis$ 
  by  $\text{simp}$ 
qed

```

```

lemma  $\text{continuous\_map\_of\_fst}$ :
   $\text{continuous\_map } (\text{prod\_topology } X \ Y) \ Z \ (f \circ \text{fst}) \longleftrightarrow \text{topspace } Y = \{\} \vee \text{continuous\_map } X \ Z \ f$ 
proof  $(\text{cases } \text{topspace } Y = \{\})$ 
  case  $\text{True}$ 
  then show  $?thesis$ 
    by  $(\text{simp add: continuous\_map\_on\_empty})$ 
  next
  case  $\text{False}$ 
  then show  $?thesis$ 
    by  $(\text{simp add: continuous\_compose\_quotient\_map\_eq})$ 
qed

```

```

lemma  $\text{continuous\_map\_of\_snd}$ :
   $\text{continuous\_map } (\text{prod\_topology } X \ Y) \ Z \ (f \circ \text{snd}) \longleftrightarrow \text{topspace } X = \{\} \vee \text{continuous\_map } Y \ Z \ f$ 
proof  $(\text{cases } \text{topspace } X = \{\})$ 

```

```

    case True
    then show ?thesis
      by (simp add: continuous_map_on_empty)
next
    case False
    then show ?thesis
      by (simp add: continuous_compose_quotient_map_eq)
qed

lemma continuous_map_prod_top:
  continuous_map (prod_topology X Y) (prod_topology X' Y') ( $\lambda(x,y). (f\ x, g\ y)$ )
 $\longleftrightarrow$ 
  topspace (prod_topology X Y) = {}  $\vee$  continuous_map X X' f  $\wedge$  continuous_map Y Y' g
proof (cases topspace (prod_topology X Y) = {})
  case True
  then show ?thesis
    by (simp add: continuous_map_on_empty)
next
  case False
  then show ?thesis
    by (simp add: continuous_map_paired case_prod_unfold continuous_map_of_fst
      [unfolded o_def] continuous_map_of_snd [unfolded o_def])
qed

lemma in_prod_topology_closure_of:
  assumes  $z \in (\text{prod\_topology } X\ Y)\ \text{closure\_of } S$ 
  shows  $\text{fst } z \in X\ \text{closure\_of } (\text{fst } 'S)$   $\text{snd } z \in Y\ \text{closure\_of } (\text{snd } 'S)$ 
  using assms continuous_map_eq_image_closure_subset continuous_map_fst
  apply fastforce
  using assms continuous_map_eq_image_closure_subset continuous_map_snd
  apply fastforce
  done

proposition compact_space_prod_topology:
  compact_space (prod_topology X Y)  $\longleftrightarrow$  topspace (prod_topology X Y) = {}  $\vee$ 
  compact_space X  $\wedge$  compact_space Y
proof (cases topspace (prod_topology X Y) = {})
  case True
  then show ?thesis
    using compact_space_topspace_empty by blast
next
  case False
  then have non_mt: topspace X  $\neq$  {} topspace Y  $\neq$  {}
    by auto
  have compact_space X compact_space Y if compact_space (prod_topology X Y)
  proof -
    have compactin X (fst ' (topspace X  $\times$  topspace Y))

```



```

    by (metis compact_space_def continuous_map_fst image_compactin that
topspace_prod_topology)
  moreover
    have compactin Y (snd ' (topspace X × topspace Y))
    by (metis compact_space_def continuous_map_snd image_compactin that
topspace_prod_topology)
  ultimately show compact_space X compact_space Y
  by (simp_all add: non_mt compact_space_def)
qed
moreover
define  $\mathcal{X}$  where  $\mathcal{X} \equiv (\lambda V. \text{topspace } X \times V) \text{ ' Collect (openin } Y)$ 
define  $\mathcal{Y}$  where  $\mathcal{Y} \equiv (\lambda U. U \times \text{topspace } Y) \text{ ' Collect (openin } X)$ 
have compact_space (prod_topology X Y) if compact_space X compact_space Y
proof (rule Alexander_subbase_alt)
  show toptop: topspace X × topspace Y  $\subseteq \bigcup (\mathcal{X} \cup \mathcal{Y})$ 
  unfolding  $\mathcal{X\_def}$   $\mathcal{Y\_def}$  by auto
  fix C :: ('a × 'b) set set
  assume C: C  $\subseteq \mathcal{X} \cup \mathcal{Y}$  topspace X × topspace Y  $\subseteq \bigcup C$ 
  then obtain  $\mathcal{X}' \mathcal{Y}'$  where XY:  $\mathcal{X}' \subseteq \mathcal{X}$   $\mathcal{Y}' \subseteq \mathcal{Y}$  and Ceq: C =  $\mathcal{X}' \cup \mathcal{Y}'$ 
  using subset_UnE by metis
  then have sub: topspace X × topspace Y  $\subseteq \bigcup (\mathcal{X}' \cup \mathcal{Y}')$ 
  using C by simp
  obtain  $\mathcal{U} \mathcal{V}$  where  $\mathcal{U}$ :  $\bigwedge U. U \in \mathcal{U} \implies \text{openin } X U \mathcal{Y}' = (\lambda U. U \times \text{topspace } Y) \text{ ' } \mathcal{U}$ 
  and  $\mathcal{V}$ :  $\bigwedge V. V \in \mathcal{V} \implies \text{openin } Y V \mathcal{X}' = (\lambda V. \text{topspace } X \times V) \text{ ' } \mathcal{V}$ 
  using XY by (clarsimp simp add:  $\mathcal{X\_def}$   $\mathcal{Y\_def}$  subset_image_iff) (force
simp add: subset_iff)
  have  $\exists \mathcal{D}. \text{finite } \mathcal{D} \wedge \mathcal{D} \subseteq \mathcal{X}' \cup \mathcal{Y}' \wedge \text{topspace } X \times \text{topspace } Y \subseteq \bigcup \mathcal{D}$ 
  proof -
    have topspace X  $\subseteq \bigcup \mathcal{U} \vee \text{topspace } Y \subseteq \bigcup \mathcal{V}$ 
    using  $\mathcal{U} \mathcal{V} C \text{ Ceq}$  by auto
    then have *:  $\exists \mathcal{D}. \text{finite } \mathcal{D} \wedge$ 
      ( $\forall x \in \mathcal{D}. x \in (\lambda V. \text{topspace } X \times V) \text{ ' } \mathcal{V} \vee x \in (\lambda U. U \times \text{topspace } Y) \text{ ' } \mathcal{U}) \wedge$ 
      ( $\text{topspace } X \times \text{topspace } Y \subseteq \bigcup \mathcal{D}$ )
  proof
    assume topspace X  $\subseteq \bigcup \mathcal{U}$ 
    with  $\langle \text{compact\_space } X \rangle \mathcal{U}$  obtain  $\mathcal{F}$  where finite  $\mathcal{F}$   $\mathcal{F} \subseteq \mathcal{U}$  topspace X
 $\subseteq \bigcup \mathcal{F}$ 
    by (meson compact_space_alt)
    with that show ?thesis
    by (rule_tac x=( $\lambda D. D \times \text{topspace } Y$ ) '  $\mathcal{F}$  in exI) auto
  next
    assume topspace Y  $\subseteq \bigcup \mathcal{V}$ 
    with  $\langle \text{compact\_space } Y \rangle \mathcal{V}$  obtain  $\mathcal{F}$  where finite  $\mathcal{F}$   $\mathcal{F} \subseteq \mathcal{V}$  topspace Y
 $\subseteq \bigcup \mathcal{F}$ 
    by (meson compact_space_alt)
    with that show ?thesis
    by (rule_tac x=( $\lambda C. \text{topspace } X \times C$ ) '  $\mathcal{F}$  in exI) auto
  end
end

```

```

qed
then show ?thesis
  using that  $\mathcal{U} \mathcal{V}$  by blast
qed
then show  $\exists \mathcal{D}. \text{finite } \mathcal{D} \wedge \mathcal{D} \subseteq \mathcal{C} \wedge \text{topspace } X \times \text{topspace } Y \subseteq \bigcup \mathcal{D}$ 
  using  $\mathcal{C} \text{ Ceq}$  by blast
next
  have (finite intersection_of  $(\lambda x. x \in \mathcal{X} \vee x \in \mathcal{Y})$  relative_to  $\text{topspace } X \times \text{topspace } Y$ )
    =  $(\lambda U. \exists S T. U = S \times T \wedge \text{openin } X S \wedge \text{openin } Y T)$ 
    (is ?lhs = ?rhs)
  proof -
    have ?rhs U if ?lhs U for U
    proof -
      have  $\text{topspace } X \times \text{topspace } Y \cap \bigcap T \in \{A \times B \mid A B. A \in \text{Collect } (\text{openin } X) \wedge B \in \text{Collect } (\text{openin } Y)\}$ 
        if finite T  $T \subseteq \mathcal{X} \cup \mathcal{Y}$  for T
        using that
      proof induction
        case (insert B  $\mathcal{B}$ )
        then show ?case
          unfolding  $\mathcal{X\_def} \mathcal{Y\_def}$ 
          apply (simp add: Int_ac subset_eq image_def)
          apply (metis (no_types) openin_Int openin_topspace Times_Int_Times)
          done
      qed auto
    then show ?thesis
      using that
      by (auto simp: subset_eq elim!: relative_toE intersection_ofE)
  qed
moreover
  have ?lhs Z if Z: ?rhs Z for Z
  proof -
    obtain U V where  $Z = U \times V$  openin X U openin Y V
    using Z by blast
    then have UV:  $U \times V = (\text{topspace } X \times \text{topspace } Y) \cap (U \times V)$ 
      by (simp add: Sigma_mono inf_absorb2 openin_subset)
    moreover
      have ?lhs  $((\text{topspace } X \times \text{topspace } Y) \cap (U \times V))$ 
      proof (rule relative_to_inc)
        show (finite intersection_of  $(\lambda x. x \in \mathcal{X} \vee x \in \mathcal{Y})$ )  $(U \times V)$ 
          apply (simp add: intersection_of_def  $\mathcal{X\_def} \mathcal{Y\_def}$ )
          apply (rule_tac x= $\{(U \times \text{topspace } Y), (\text{topspace } X \times V)\}$  in exI)
          using  $\langle \text{openin } X U \rangle \langle \text{openin } Y V \rangle$  openin_subset UV apply (fastforce
simp add:)
          done
      qed
    ultimately show ?thesis
      using  $\langle Z = U \times V \rangle$  by auto

```

```

qed
ultimately show ?thesis
  by meson
qed
then show topology (arbitrary_union_of (finite_intersection_of ( $\lambda x. x \in \mathcal{X} \cup \mathcal{Y}$ )
  relative_to (topspace  $X \times \text{topspace } Y$ ))) =
  prod_topology  $X \ Y$ 
  by (simp add: prod_topology_def)
qed
ultimately show ?thesis
  using False by blast
qed

```

lemma compactin_Times:

```

compactin (prod_topology  $X \ Y$ ) ( $S \times T$ )  $\longleftrightarrow S = \{\}$   $\vee T = \{\}$   $\vee$  compactin  $X \ S \wedge$  compactin  $Y \ T$ 
by (auto simp: compactin_subspace subtopology_Times compact_space_prod_topology)

```

5.2.3 Homeomorphic maps

lemma homeomorphic_maps_prod:

```

homeomorphic_maps (prod_topology  $X \ Y$ ) (prod_topology  $X' \ Y'$ ) ( $\lambda(x,y). (f \ x,$ 
 $g \ y)$ ) ( $\lambda(x,y). (f' \ x, g' \ y)$ )  $\longleftrightarrow$ 
  topspace(prod_topology  $X \ Y$ ) =  $\{\}$   $\wedge$ 
  topspace(prod_topology  $X' \ Y'$ ) =  $\{\}$   $\vee$ 
  homeomorphic_maps  $X \ X' \ f \ f' \wedge$ 
  homeomorphic_maps  $Y \ Y' \ g \ g'$ 
unfolding homeomorphic_maps_def continuous_map_prod_top
by (auto simp: continuous_map_def homeomorphic_maps_def continuous_map_prod_top)

```

lemma homeomorphic_maps_swap:

```

homeomorphic_maps (prod_topology  $X \ Y$ ) (prod_topology  $Y \ X$ )
  ( $\lambda(x,y). (y,x)$ ) ( $\lambda(y,x). (x,y)$ )
by (auto simp: homeomorphic_maps_def case_prod_unfold continuous_mapfst
continuous_map_pairedI continuous_map_snd)

```

lemma homeomorphic_map_swap:

```

homeomorphic_map (prod_topology  $X \ Y$ ) (prod_topology  $Y \ X$ ) ( $\lambda(x,y). (y,x)$ )
using homeomorphic_map_maps homeomorphic_maps_swap by metis

```

lemma embedding_map_graph:

```

embedding_map  $X$  (prod_topology  $X \ Y$ ) ( $\lambda x. (x, f \ x)$ )  $\longleftrightarrow$  continuous_map  $X \ Y \ f$ 
(is ?lhs = ?rhs)

```

proof

```

assume  $L$ : ?lhs
have  $\text{snd} \circ (\lambda x. (x, f \ x)) = f$ 
by force

```

```

moreover have continuous_map X Y (snd ∘ (λx. (x, f x)))
using L
unfolding embedding_map_def
by (meson continuous_map_in_subtopology continuous_map_snd_of_homeo-
morphis_imp_continuous_map)
ultimately show ?rhs
by simp
next
assume R: ?rhs
then show ?lhs
unfolding homeomorphic_map_maps embedding_map_def homeomorphic_maps_def
by (rule_tac x=fst in exI)
(auto simp: continuous_map_in_subtopology continuous_map_paired con-
tinuous_map_from_subtopology
continuous_map_fst)
qed

```

```

lemma homeomorphic_space_prod_topology:
   $\llbracket X \text{ homeomorphic\_space } X''; Y \text{ homeomorphic\_space } Y' \rrbracket$ 
 $\implies \text{prod\_topology } X \ Y \text{ homeomorphic\_space } \text{prod\_topology } X'' \ Y'$ 
using homeomorphic_maps_prod unfolding homeomorphic_space_def by blast

```

```

lemma prod_topology_homeomorphic_space_left:
   $\text{topspace } Y = \{b\} \implies \text{prod\_topology } X \ Y \text{ homeomorphic\_space } X$ 
unfolding homeomorphic_space_def
by (rule_tac x=fst in exI) (simp add: homeomorphic_map_def inj_on_def flip:
homeomorphic_map_maps)

```

```

lemma prod_topology_homeomorphic_space_right:
   $\text{topspace } X = \{a\} \implies \text{prod\_topology } X \ Y \text{ homeomorphic\_space } Y$ 
unfolding homeomorphic_space_def
by (rule_tac x=snd in exI) (simp add: homeomorphic_map_def inj_on_def
flip: homeomorphic_map_maps)

```

```

lemma homeomorphic_space_prod_topology_sing1:
   $b \in \text{topspace } Y \implies X \text{ homeomorphic\_space } (\text{prod\_topology } X \ (\text{subtopology } Y \ \{b\}))$ 
by (metis empty_subsetI homeomorphic_space_sym inf.absorb_iff2 insert_subset
prod_topology_homeomorphic_space_left topspace_subtopology)

```

```

lemma homeomorphic_space_prod_topology_sing2:
   $a \in \text{topspace } X \implies Y \text{ homeomorphic\_space } (\text{prod\_topology } (\text{subtopology } X \ \{a\}) \ Y)$ 
by (metis empty_subsetI homeomorphic_space_sym inf.absorb_iff2 insert_subset
prod_topology_homeomorphic_space_right topspace_subtopology)

```

```

lemma topological_property_of_prod_component:
assumes major:  $P(\text{prod\_topology } X \ Y)$ 

```

```

    and X:  $\bigwedge x. \llbracket x \in \text{topspace } X; P(\text{prod\_topology } X \ Y) \rrbracket \implies P(\text{subtopology}$ 
      ( $\text{prod\_topology } X \ Y$ ) ( $\{x\} \times \text{topspace } Y$ ))
    and Y:  $\bigwedge y. \llbracket y \in \text{topspace } Y; P(\text{prod\_topology } X \ Y) \rrbracket \implies P(\text{subtopology}$ 
      ( $\text{prod\_topology } X \ Y$ ) ( $\text{topspace } X \times \{y\}$ ))
    and PQ:  $\bigwedge X \ X'. X \text{ homeomorphic\_space } X' \implies (P \ X \longleftrightarrow Q \ X')$ 
    and PR:  $\bigwedge X \ X'. X \text{ homeomorphic\_space } X' \implies (P \ X \longleftrightarrow R \ X')$ 
    shows  $\text{topspace}(\text{prod\_topology } X \ Y) = \{\} \vee Q \ X \wedge R \ Y$ 
  proof -
    have  $Q \ X \wedge R \ Y$  if  $\text{topspace}(\text{prod\_topology } X \ Y) \neq \{\}$ 
  proof -
    from that obtain a b where a:  $a \in \text{topspace } X$  and b:  $b \in \text{topspace } Y$ 
    by force
    show ?thesis
    using X [OF a major] and Y [OF b major] homeomorphic_space_prod_topology_sing1
      [OF b, of X] homeomorphic_space_prod_topology_sing2 [OF a, of Y]
    by (simp add: subtopology_Times) (meson PQ PR homeomorphic_space_prod_topology_sing2
      homeomorphic_space_sym)
  qed
  then show ?thesis by metis
qed

lemma limitin_pairwise:
   $\text{limitin } (\text{prod\_topology } X \ Y) \ f \ l \ F \longleftrightarrow \text{limitin } X \ (fst \circ f) \ (fst \ l) \ F \wedge \text{limitin } Y$ 
   $(snd \circ f) \ (snd \ l) \ F$ 
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain a b where ev:  $\bigwedge U. \llbracket (a,b) \in U; \text{openin } (\text{prod\_topology } X \ Y) \ U \rrbracket$ 
 $\implies \forall_F x \text{ in } F. f \ x \in U$ 
    and a:  $a \in \text{topspace } X$  and b:  $b \in \text{topspace } Y$  and l:  $l = (a,b)$ 
  by (auto simp: limitin_def)
  moreover have  $\forall_F x \text{ in } F. fst \ (f \ x) \in U$  if  $\text{openin } X \ U \ a \in U$  for U
  proof -
    have  $\forall_F c \text{ in } F. f \ c \in U \times \text{topspace } Y$ 
    using b that ev [of  $U \times \text{topspace } Y$ ] by (auto simp: openin_prod_topology_alt)
    then show ?thesis
    by (rule eventually_mono) (metis (mono_tags, lifting) SigmaE2 prod.collapse)
  qed
  moreover have  $\forall_F x \text{ in } F. snd \ (f \ x) \in U$  if  $\text{openin } Y \ U \ b \in U$  for U
  proof -
    have  $\forall_F c \text{ in } F. f \ c \in \text{topspace } X \times U$ 
    using a that ev [of  $\text{topspace } X \times U$ ] by (auto simp: openin_prod_topology_alt)
    then show ?thesis
    by (rule eventually_mono) (metis (mono_tags, lifting) SigmaE2 prod.collapse)
  qed
  ultimately show ?rhs
  by (simp add: limitin_def)
next
  have  $\text{limitin } (\text{prod\_topology } X \ Y) \ f \ (a,b) \ F$ 

```

```

    if limitin X (fst ∘ f) a F limitin Y (snd ∘ f) b F for a b
    using that
  proof (clarisimp simp: limitin_def)
    fix Z :: ('a × 'b) set
    assume a: a ∈ topspace X ∀ U. openin X U ∧ a ∈ U ⟶ (∀F x in F. fst (f
x) ∈ U)
    and b: b ∈ topspace Y ∀ U. openin Y U ∧ b ∈ U ⟶ (∀F x in F. snd (f x)
∈ U)
    and Z: openin (prod_topology X Y) Z (a, b) ∈ Z
    then obtain U V where openin X U openin Y V a ∈ U b ∈ V U × V ⊆ Z
    using Z by (force simp: openin_prod_topology_alt)
    then have ∀F x in F. fst (f x) ∈ U ∀F x in F. snd (f x) ∈ V
    by (simp_all add: a b)
    then show ∀F x in F. f x ∈ Z
    by (rule eventually_elim2) (use ⟨U × V ⊆ Z⟩ subsetD in auto)
  qed
  then show ?rhs ⟹ ?lhs
  by (metis prod.collapse)
qed
end

```

5.3 T1 and Hausdorff spaces

```

theory T1_Spaces
imports Product_Topology
begin

```

5.4 T1 spaces with equivalences to many naturally "nice" properties.

```

definition t1_space where
  t1_space X ≡ ∀ x ∈ topspace X. ∀ y ∈ topspace X. x ≠ y ⟶ (∃ U. openin X U ∧
x ∈ U ∧ y ∉ U)

```

```

lemma t1_space_expansive:
  ⟦topspace Y = topspace X; ⋀ U. openin X U ⟹ openin Y U⟧ ⟹ t1_space X
  ⟹ t1_space Y
  by (metis t1_space_def)

```

```

lemma t1_space_alt:
  t1_space X ⟷ (∀ x ∈ topspace X. ∀ y ∈ topspace X. x ≠ y ⟶ (∃ U. closedin
X U ∧ x ∈ U ∧ y ∉ U))
  by (metis DiffE DiffI closedin_def openin_closedin_eq t1_space_def)

```

```

lemma t1_space_empty: topspace X = {} ⟹ t1_space X
  by (simp add: t1_space_def)

```

lemma *t1_space_derived_set_of_singleton:*

$t1\_space\ X \longleftrightarrow (\forall x \in topspace\ X. X\ derived\_set\_of\ \{x\} = \{\})$

apply (*simp add: t1_space_def derived_set_of_def, safe*)

apply (*metis openin_topspace*)

by *force*

lemma *t1_space_derived_set_of_finite:*

$t1\_space\ X \longleftrightarrow (\forall S. finite\ S \longrightarrow X\ derived\_set\_of\ S = \{\})$

proof (*intro iffI allI impI*)

fix *S :: 'a set*

assume *finite S*

then have *fin: finite (($\lambda x. \{x\}$) ` (*topspace X* $\cap$ *S*))*

by *blast*

assume *t1_space X*

then have *X derived_set_of ($\bigcup x \in topspace\ X \cap S. \{x\}$) = {}*

unfolding *derived_set_of_Union [OF fin]*

by (*auto simp: t1_space_derived_set_of_singleton*)

then have *X derived_set_of (topspace X $\cap$ S) = {}*

by *simp*

then show *X derived_set_of S = {}*

by *simp*

qed (*auto simp: t1_space_derived_set_of_singleton*)

lemma *t1_space_closedin_singleton:*

$t1\_space\ X \longleftrightarrow (\forall x \in topspace\ X. closedin\ X\ \{x\})$

apply (*rule iffI*)

apply (*simp add: closedin_contains_derived_set t1_space_derived_set_of_singleton*)

using *t1_space_alt* **by** *auto*

lemma *closedin_t1_singleton:*

$\llbracket t1\_space\ X; a \in topspace\ X \rrbracket \Longrightarrow closedin\ X\ \{a\}$

by (*simp add: t1_space_closedin_singleton*)

lemma *t1_space_closedin_finite:*

$t1\_space\ X \longleftrightarrow (\forall S. finite\ S \wedge S \subseteq topspace\ X \longrightarrow closedin\ X\ S)$

apply (*rule iffI*)

apply (*simp add: closedin_contains_derived_set t1_space_derived_set_of_finite*)

by (*simp add: t1_space_closedin_singleton*)

lemma *closure_of_singleton:*

$t1\_space\ X \Longrightarrow X\ closure\_of\ \{a\} = (if\ a \in topspace\ X\ then\ \{a\}\ else\ \{\})$

by (*simp add: closure_of_eq t1_space_closedin_singleton closure_of_eq_empty_gen*)

lemma *separated_in_singleton:*

assumes *t1_space X*

shows *separatedin X {a} S $\longleftrightarrow a \in topspace\ X \wedge S \subseteq topspace\ X \wedge (a \notin X\ closure\_of\ S)$*

separatedin X S {a} $\longleftrightarrow a \in topspace\ X \wedge S \subseteq topspace\ X \wedge (a \notin X\ closure\_of\ S)$

unfolding *separatedin_def*
using *assms closure_of closure_of_singleton* **by** *fastforce+*

lemma *t1_space_openin_delete*:

$t1\_space\ X \longleftrightarrow (\forall\ U\ x.\ openin\ X\ U \wedge x \in U \longrightarrow openin\ X\ (U - \{x\}))$
apply (*rule iffI*)
apply (*meson closedin_t1_singleton_in_mono openin_diff openin_subset*)
by (*simp add: closedin_def t1_space_closedin_singleton*)

lemma *t1_space_openin_delete_alt*:

$t1\_space\ X \longleftrightarrow (\forall\ U\ x.\ openin\ X\ U \longrightarrow openin\ X\ (U - \{x\}))$
by (*metis Diff_empty Diff_insert0 t1_space_openin_delete*)

lemma *t1_space_singleton_Inter_open*:

$t1\_space\ X \longleftrightarrow (\forall\ x \in topspace\ X.\ \bigcap \{U.\ openin\ X\ U \wedge x \in U\} = \{x\})$ (**is**
 $?P = ?Q$)

and *t1_space_Inter_open_supersets*:

$t1\_space\ X \longleftrightarrow (\forall\ S.\ S \subseteq topspace\ X \longrightarrow \bigcap \{U.\ openin\ X\ U \wedge S \subseteq U\} = S)$ (**is** $?P = ?R$)

proof –

have $?R \implies ?Q$
apply *clarify*
apply (*drule_tac x={x} in spec, simp*)
done

moreover have $?Q \implies ?P$

apply (*clarsimp simp add: t1_space_def*)
apply (*drule_tac x=x in bspec*)
apply (*simp_all add: set_eq_iff*)
by (*metis (no_types, lifting)*)

moreover have $?P \implies ?R$

proof (*clarsimp simp add: t1_space_closedin_singleton, rule subset_antisym*)
fix S

assume $S: \forall x \in topspace\ X.\ closedin\ X\ \{x\}\ S \subseteq topspace\ X$

then show $\bigcap \{U.\ openin\ X\ U \wedge S \subseteq U\} \subseteq S$

apply *clarsimp*

by (*metis Diff_insert_absorb Set.set_insert closedin_def openin_topspace subset_insert*)

qed *force*

ultimately show $?P = ?Q\ ?P = ?R$

by *auto*

qed

lemma *t1_space_derived_set_of_infinite_openin*:

$t1\_space\ X \longleftrightarrow$
 $(\forall\ S.\ X\ derived\_set\_of\ S =$
 $\{x \in topspace\ X.\ \forall\ U.\ x \in U \wedge openin\ X\ U \longrightarrow infinite(S \cap U)\})$
(is $\_ = ?rhs$ **)**

proof


```

assume  $t1\_space\ X$ 
show  $?rhs$ 
proof safe
  fix  $S\ x\ U$ 
  assume  $x \in X\ derived\_set\_of\ S\ x \in U\ openin\ X\ U\ finite\ (S \cap U)$ 
  with  $\langle t1\_space\ X \rangle$  show False
  apply (simp add: t1_space_derived_set_of_finite)
  by (metis IntI empty_iff empty_subsetI inf_commute openin_Int_derived_set_of_subset
subset_antisym)
next
  fix  $S\ x$ 
  have  $eq: (\exists y. (y \neq x) \wedge y \in S \wedge y \in T) \longleftrightarrow \sim((S \cap T) \subseteq \{x\})$  for  $x\ S\ T$ 
  by blast
  assume  $x \in topspace\ X\ \forall U. x \in U \wedge openin\ X\ U \longrightarrow infinite\ (S \cap U)$ 
  then show  $x \in X\ derived\_set\_of\ S$ 
  apply (clarsimp simp add: derived_set_of_def eq)
  by (meson finite.emptyI finite.insertI finite_subset)
qed (auto simp: in_derived_set_of)
qed (auto simp: t1_space_derived_set_of_singleton)

lemma finite_t1_space_imp_discrete_topology:
   $\llbracket topspace\ X = U; finite\ U; t1\_space\ X \rrbracket \Longrightarrow X = discrete\_topology\ U$ 
by (metis discrete_topology_unique_derived_set t1_space_derived_set_of_finite)

lemma t1_space_subtopology:  $t1\_space\ X \Longrightarrow t1\_space(subtopology\ X\ U)$ 
by (simp add: derived_set_of_subtopology t1_space_derived_set_of_finite)

lemma closedin_derived_set_of_gen:
   $t1\_space\ X \Longrightarrow closedin\ X\ (X\ derived\_set\_of\ S)$ 
apply (clarsimp simp add: in_derived_set_of closedin_contains_derived_set
derived_set_of_subset_topspace)
by (metis DiffD2 insert_Diff insert_iff t1_space_openin_delete)

lemma derived_set_of_derived_set_subset_gen:
   $t1\_space\ X \Longrightarrow X\ derived\_set\_of\ (X\ derived\_set\_of\ S) \subseteq X\ derived\_set\_of\ S$ 
by (meson closedin_contains_derived_set closedin_derived_set_of_gen)

lemma subtopology_eq_discrete_topology_gen_finite:
   $\llbracket t1\_space\ X; finite\ S \rrbracket \Longrightarrow subtopology\ X\ S = discrete\_topology(tospace\ X \cap S)$ 
by (simp add: subtopology_eq_discrete_topology_gen t1_space_derived_set_of_finite)

lemma subtopology_eq_discrete_topology_finite:
   $\llbracket t1\_space\ X; S \subseteq topspace\ X; finite\ S \rrbracket$ 
   $\Longrightarrow subtopology\ X\ S = discrete\_topology\ S$ 
by (simp add: subtopology_eq_discrete_topology_eq t1_space_derived_set_of_finite)

lemma t1_space_closed_map_image:
   $\llbracket closed\_map\ X\ Y\ f; f\ ' (topspace\ X) = topspace\ Y; t1\_space\ X \rrbracket \Longrightarrow t1\_space$ 

```

```

Y
  by (metis closed_map_def finite_subset_image t1_space_closedin_finite)

lemma homeomorphic_t1_space:  $X \text{ homeomorphic\_space } Y \implies (t1\_space\ X \longleftrightarrow t1\_space\ Y)$ 
  apply (clarsimp simp add: homeomorphic_space_def)
  by (meson homeomorphic_eq_everything_map homeomorphic_maps_map t1_space_closed_map_image)

proposition t1_space_product_topology:
   $t1\_space\ (product\_topology\ X\ I) \longleftrightarrow topspace(product\_topology\ X\ I) = \{\}$   $\vee (\forall i \in I. t1\_space\ (X\ i))$ 
proof (cases topspace(product_topology X I) =  $\{\}$ )
  case True
  then show ?thesis
    using True t1_space_empty by blast
next
  case False
  then obtain f where  $f: f \in (\Pi_{i \in I}. topspace(X\ i))$ 
  by fastforce
  have  $t1\_space\ (product\_topology\ X\ I) \longleftrightarrow (\forall i \in I. t1\_space\ (X\ i))$ 
  proof (intro iffI ballI)
    show  $t1\_space\ (X\ i)$  if  $t1\_space\ (product\_topology\ X\ I)$  and  $i \in I$  for  $i$ 
    proof -
      have  $clo: \bigwedge h. h \in (\Pi_{i \in I}. topspace\ (X\ i)) \implies closedin\ (product\_topology\ X\ I)\ \{h\}$ 
      using that by (simp add: t1_space_closedin_singleton)
      show ?thesis
      unfolding t1_space_closedin_singleton
      proof clarify
        show  $closedin\ (X\ i)\ \{xi\}$  if  $xi \in topspace\ (X\ i)$  for  $xi$ 
        using  $clo$  [of  $\lambda j \in I. \text{if } i=j \text{ then } xi \text{ else } f\ j$ ]  $f$  that  $\langle i \in I \rangle$ 
        by (fastforce simp add: closedin_product_topology_singleton)
      qed
    qed
  next
  next
  show  $t1\_space\ (product\_topology\ X\ I)$  if  $\forall i \in I. t1\_space\ (X\ i)$ 
  using that
  by (simp add: t1_space_closedin_singleton Ball_def PiE_iff closedin_product_topology_singleton)
  qed
  then show ?thesis
  using False by blast
qed

lemma t1_space_prod_topology:
   $t1\_space(prod\_topology\ X\ Y) \longleftrightarrow topspace(prod\_topology\ X\ Y) = \{\}$   $\vee t1\_space\ X \wedge t1\_space\ Y$ 
proof (cases topspace (prod_topology X Y) =  $\{\}$ )
  case True then show ?thesis

```

```

    by (auto simp: t1_space_empty)
next
case False
have eq:  $\{(x,y)\} = \{x\} \times \{y\}$  for  $x y$ 
  by simp
have t1_space (prod_topology X Y)  $\longleftrightarrow$  (t1_space X  $\wedge$  t1_space Y)
  using False
  by (force simp: t1_space_closedin_singleton closedin_prod_Times_iff eq simp
del: insert_Times_insert)
  with False show ?thesis
  by simp
qed

```

5.4.1 Hausdorff Spaces

definition *Hausdorff_space*

where

Hausdorff_space $X \equiv$
 $\forall x y. x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge (x \neq y)$
 $\longrightarrow (\exists U V. \text{openin } X U \wedge \text{openin } X V \wedge x \in U \wedge y \in V \wedge \text{disjnt } U$
 $V)$

lemma *Hausdorff_space_expansive:*

$\llbracket \text{Hausdorff\_space } X; \text{topspace } X = \text{topspace } Y; \bigwedge U. \text{openin } X U \implies \text{openin } Y U \rrbracket \implies \text{Hausdorff\_space } Y$
 by (metis Hausdorff_space_def)

lemma *Hausdorff_space_topspace_empty:*

$\text{topspace } X = \{\} \implies \text{Hausdorff\_space } X$
 by (simp add: Hausdorff_space_def)

lemma *Hausdorff_imp_t1_space:*

$\text{Hausdorff\_space } X \implies \text{t1\_space } X$
 by (metis Hausdorff_space_def disjnt_iff t1_space_def)

lemma *closedin_derived_set_of:*

$\text{Hausdorff\_space } X \implies \text{closedin } X (X \text{ derived\_set\_of } S)$
 by (simp add: Hausdorff_imp_t1_space closedin_derived_set_of_gen)

lemma *t1_or_Hausdorff_space:*

$\text{t1\_space } X \vee \text{Hausdorff\_space } X \longleftrightarrow \text{t1\_space } X$
 using Hausdorff_imp_t1_space by blast

lemma *Hausdorff_space_sing_Inter_opens:*

$\llbracket \text{Hausdorff\_space } X; a \in \text{topspace } X \rrbracket \implies \bigcap \{u. \text{openin } X u \wedge a \in u\} = \{a\}$
 using Hausdorff_imp_t1_space t1_space_singleton_Inter_open by force

lemma *Hausdorff_space_subtopology:*

assumes *Hausdorff_space* X **shows** *Hausdorff_space*(subtopology $X S$)

```

proof –
  have *:  $\text{disjnt } U \ V \implies \text{disjnt } (S \cap U) \ (S \cap V)$  for  $U \ V$ 
    by (simp add: disjnt_iff)
  from assms show ?thesis
    apply (simp add: Hausdorff_space_def openin_subtopology_alt)
    apply (fast intro: * elim!: all_forward)
    done
qed

lemma Hausdorff_space_compact_separation:
  assumes  $X$ : Hausdorff_space  $X$  and  $S$ : compactin  $X$   $S$  and  $T$ : compactin  $X$   $T$ 
and  $\text{disjnt } S \ T$ 
  obtains  $U \ V$  where  $\text{openin } X \ U \ \text{openin } X \ V \ S \subseteq U \ T \subseteq V \ \text{disjnt } U \ V$ 
proof (cases  $S = \{\}$ )
  case True
    then show thesis
      by (metis  $\langle \text{compactin } X \ T \rangle \text{compactin\_subset\_topspace disjnt\_empty1 empty\_subsetI}$ 
         $\text{openin\_empty openin\_topspace that}$ )
  next
    case False
    have  $\forall x \in S. \exists U \ V. \text{openin } X \ U \wedge \text{openin } X \ V \wedge x \in U \wedge T \subseteq V \wedge \text{disjnt } U \ V$ 
    proof
      fix  $a$ 
      assume  $a \in S$ 
      then have  $a \notin T$ 
        by (meson assms(4) disjnt_iff)
      have  $a$ :  $a \in \text{topspace } X$ 
        using  $S \ \langle a \in S \rangle \text{compactin\_subset\_topspace}$  by blast
      show  $\exists U \ V. \text{openin } X \ U \wedge \text{openin } X \ V \wedge a \in U \wedge T \subseteq V \wedge \text{disjnt } U \ V$ 
      proof (cases  $T = \{\}$ )
      case True
        then show ?thesis
          using  $a \ \text{disjnt\_empty2 openin\_empty}$  by blast
      next
        case False
        have  $\forall x \in \text{topspace } X - \{a\}. \exists U \ V. \text{openin } X \ U \wedge \text{openin } X \ V \wedge x \in U \wedge$ 
 $a \in V \wedge \text{disjnt } U \ V$ 
          using  $X \ a$  by (simp add: Hausdorff_space_def)
          then obtain  $U \ V$  where  $UV$ :  $\forall x \in \text{topspace } X - \{a\}. \text{openin } X \ (U \ x) \wedge$ 
 $\text{openin } X \ (V \ x) \wedge x \in U \ x \wedge a \in V \ x \wedge \text{disjnt } (U \ x) \ (V \ x)$ 
          by metis
          with  $\langle a \notin T \rangle \text{compactin\_subset\_topspace [OF } T]$ 
          have  $Topen$ :  $\forall W \in U \ ' T. \text{openin } X \ W$  and  $Tsub$ :  $T \subseteq \bigcup (U \ ' T)$ 
          by (auto simp: )
          then obtain  $\mathcal{F}$  where  $\mathcal{F}$ : finite  $\mathcal{F} \ \mathcal{F} \subseteq U \ ' T$  and  $T \subseteq \bigcup \mathcal{F}$ 
          using  $T$  unfolding compactin_def by meson
          then obtain  $F$  where  $F$ : finite  $F \ F \subseteq T \ \mathcal{F} = U \ ' F$  and  $SUF$ :  $T \subseteq \bigcup (U \ ' F)$ 
and  $a \notin F$ 
    
```

```

    using finite_subset_image [OF  $\mathcal{F}$ ]  $\langle a \notin T \rangle$  by (metis subsetD)
  have  $U: \bigwedge x. \llbracket x \in \text{topspace } X; x \neq a \rrbracket \implies \text{openin } X (U x)$ 
    and  $V: \bigwedge x. \llbracket x \in \text{topspace } X; x \neq a \rrbracket \implies \text{openin } X (V x)$ 
    and  $\text{disj}: \bigwedge x. \llbracket x \in \text{topspace } X; x \neq a \rrbracket \implies \text{disjnt } (U x) (V x)$ 
    using UV by blast+
  show ?thesis
  proof (intro exI conjI)
    have  $F \neq \{\}$ 
      using False SUF by blast
    with  $\langle a \notin F \rangle$  show  $\text{openin } X (\bigcap (V ' F))$ 
      using F compactin_subset_topospace [OF T] by (force intro: V)
    show  $\text{openin } X (\bigcup (U ' F))$ 
      using F Topen Tsub by (force intro: U)
    show  $\text{disjnt } (\bigcap (V ' F)) (\bigcup (U ' F))$ 
      using disj
      apply (auto simp: disjnt_def)
      using  $\langle F \subseteq T \rangle \langle a \notin F \rangle$  compactin_subset_topospace [OF T] by blast
    show  $a \in (\bigcap (V ' F))$ 
      using  $\langle F \subseteq T \rangle T UV \langle a \notin T \rangle$  compactin_subset_topospace by blast
    qed (auto simp: SUF)
  qed
  qed
  then obtain U V where  $UV: \forall x \in S. \text{openin } X (U x) \wedge \text{openin } X (V x) \wedge x \in U x \wedge T \subseteq V x \wedge \text{disjnt } (U x) (V x)$ 
    by metis
  then have  $S \subseteq \bigcup (U ' S)$ 
    by auto
  moreover have  $\forall W \in U ' S. \text{openin } X W$ 
    using UV by blast
  ultimately obtain I where  $I: S \subseteq \bigcup (U ' I) \ I \subseteq S$  finite I
    by (metis S compactin_def finite_subset_image)
  show thesis
  proof
    show  $\text{openin } X (\bigcup (U ' I))$ 
      using  $\langle I \subseteq S \rangle UV$  by blast
    show  $\text{openin } X (\bigcap (V ' I))$ 
      using False UV  $\langle I \subseteq S \rangle \langle S \subseteq \bigcup (U ' I) \rangle \langle \text{finite } I \rangle$  by blast
    show  $\text{disjnt } (\bigcup (U ' I)) (\bigcap (V ' I))$ 
      by simp (meson UV  $\langle I \subseteq S \rangle$  disjnt_subset2 in_mono le_INF_iff order_refl)
    qed (use UV I in auto)
  qed

```

lemma Hausdorff_space_compact_sets:

```

  Hausdorff_space X  $\longleftrightarrow$ 
    ( $\forall S T. \text{compactin } X S \wedge \text{compactin } X T \wedge \text{disjnt } S T$ 
       $\longrightarrow (\exists U V. \text{openin } X U \wedge \text{openin } X V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U$ 
         $V))$ 
  (is ?lhs = ?rhs)

```

```

proof
  assume ?lhs
  then show ?rhs
    by (meson Hausdorff_space_compact_separation)
next
  assume R [rule_format]: ?rhs
  show ?lhs
  proof (clarsimp simp add: Hausdorff_space_def)
    fix x y
    assume  $x \in \text{topspace } X$   $y \in \text{topspace } X$   $x \neq y$ 
    then show  $\exists U. \text{openin } X \ U \wedge (\exists V. \text{openin } X \ V \wedge x \in U \wedge y \in V \wedge \text{disjnt } U \ V)$ 
      using R [of {x} {y}] by auto
    qed
  qed

```

```

lemma compactin_imp_closedin:
  assumes X: Hausdorff_space X and S: compactin X S shows closedin X S
proof -
  have  $S \subseteq \text{topspace } X$ 
    by (simp add: assms compactin_subset_topspace)
  moreover
  have  $\exists T. \text{openin } X \ T \wedge x \in T \wedge T \subseteq \text{topspace } X - S$  if  $x \in \text{topspace } X$   $x \notin S$ 
for x
    using Hausdorff_space_compact_separation [OF X _ S, of {x}] that
    apply (simp add: disjnt_def)
    by (metis Diff_mono Diff_triv openin_subset)
  ultimately show ?thesis
    using closedin_def openin_subopen by force
qed

```

```

lemma closedin_Hausdorff_singleton:
   $\llbracket \text{Hausdorff\_space } X; x \in \text{topspace } X \rrbracket \implies \text{closedin } X \ \{x\}$ 
  by (simp add: Hausdorff_imp_t1_space closedin_t1_singleton)

```

```

lemma closedin_Hausdorff_sing_eq:
  Hausdorff_space X  $\implies \text{closedin } X \ \{x\} \longleftrightarrow x \in \text{topspace } X$ 
  by (meson closedin_Hausdorff_singleton closedin_subset insert_subset)

```

```

lemma Hausdorff_space_discrete_topology [simp]:
  Hausdorff_space (discrete_topology U)
  unfolding Hausdorff_space_def
  apply safe
  by (metis discrete_topology_unique_alt disjnt_empty2 disjnt_insert2 insert_iff
mk_disjoint_insert topspace_discrete_topology)

```

```

lemma compactin_Int:
   $\llbracket \text{Hausdorff\_space } X; \text{compactin } X \ S; \text{compactin } X \ T \rrbracket \implies \text{compactin } X \ (S \cap T)$ 
  by (simp add: closed_Int_compactin compactin_imp_closedin)

```

lemma *finite_topspace_imp_discrete_topology*:

$\llbracket \text{topspace } X = U; \text{finite } U; \text{Hausdorff\_space } X \rrbracket \implies X = \text{discrete\_topology } U$
using *Hausdorff_imp_t1_space finite_t1_space_imp_discrete_topology* **by** *blast*

lemma *derived_set_of_finite*:

$\llbracket \text{Hausdorff\_space } X; \text{finite } S \rrbracket \implies X \text{ derived\_set\_of } S = \{\}$
using *Hausdorff_imp_t1_space t1_space_derived_set_of_finite* **by** *auto*

lemma *derived_set_of_singleton*:

$\text{Hausdorff\_space } X \implies X \text{ derived\_set\_of } \{x\} = \{\}$
by (*simp add: derived_set_of_finite*)

lemma *closedin_Hausdorff_finite*:

$\llbracket \text{Hausdorff\_space } X; S \subseteq \text{topspace } X; \text{finite } S \rrbracket \implies \text{closedin } X \ S$
by (*simp add: compactin_imp_closedin finite_imp_compactin_eq*)

lemma *open_in_Hausdorff_delete*:

$\llbracket \text{Hausdorff\_space } X; \text{openin } X \ S \rrbracket \implies \text{openin } X \ (S - \{x\})$
using *Hausdorff_imp_t1_space t1_space_openin_delete_alt* **by** *auto*

lemma *closedin_Hausdorff_finite_eq*:

$\llbracket \text{Hausdorff\_space } X; \text{finite } S \rrbracket \implies \text{closedin } X \ S \longleftrightarrow S \subseteq \text{topspace } X$
by (*meson closedin_Hausdorff_finite closedin_def*)

lemma *derived_set_of_infinite_openin*:

$\text{Hausdorff\_space } X$
 $\implies X \text{ derived\_set\_of } S =$
 $\{x \in \text{topspace } X. \forall U. x \in U \wedge \text{openin } X \ U \longrightarrow \text{infinite}(S \cap U)\}$
using *Hausdorff_imp_t1_space t1_space_derived_set_of_infinite_openin* **by**
fastforce

lemma *Hausdorff_space_discrete_compactin*:

$\text{Hausdorff\_space } X$
 $\implies S \cap X \text{ derived\_set\_of } S = \{\} \wedge \text{compactin } X \ S \longleftrightarrow S \subseteq \text{topspace } X \wedge$
 $\text{finite } S$
using *derived_set_of_finite discrete_compactin_eq_finite* **by** *fastforce*

lemma *Hausdorff_space_finite_topspace*:

$\text{Hausdorff\_space } X \implies X \text{ derived\_set\_of } (\text{topspace } X) = \{\} \wedge \text{compact\_space}$
 $X \longleftrightarrow \text{finite}(\text{topspace } X)$
using *derived_set_of_finite discrete_compact_space_eq_finite* **by** *auto*

lemma *derived_set_of_derived_set_subset*:

$\text{Hausdorff\_space } X \implies X \text{ derived\_set\_of } (X \text{ derived\_set\_of } S) \subseteq X \text{ derived\_set\_of}$
 S
by (*simp add: Hausdorff_imp_t1_space derived_set_of_derived_set_subset_gen*)

lemma *Hausdorff_space_injective_preimage:*

assumes *Hausdorff_space* Y **and** *cmf*: *continuous_map* $X\ Y\ f$ **and** *inj_on* f
(topspace X)
shows *Hausdorff_space* X
unfolding *Hausdorff_space_def*
proof *clarify*
fix $x\ y$
assume $x \in \text{topspace } X$ **and** $y \in \text{topspace } X$ **and** $x \neq y$
then obtain $U\ V$ **where** *openin* $Y\ U$ *openin* $Y\ V$ $f\ x \in U$ $f\ y \in V$ *disjnt* $U\ V$
using *assms* **unfolding** *Hausdorff_space_def* *continuous_map_def* **by** (*meson inj_onD*)
show $\exists\ U\ V. \text{openin } X\ U \wedge \text{openin } X\ V \wedge x \in U \wedge y \in V \wedge \text{disjnt } U\ V$
proof (*intro exI conjI*)
show *openin* $X\ \{x \in \text{topspace } X. f\ x \in U\}$
using $\langle \text{openin } Y\ U \rangle$ *cmf* *continuous_map* **by** *fastforce*
show *openin* $X\ \{x \in \text{topspace } X. f\ x \in V\}$
using $\langle \text{openin } Y\ V \rangle$ *cmf* *openin_continuous_map_preimage* **by** *blast*
show *disjnt* $\{x \in \text{topspace } X. f\ x \in U\}\ \{x \in \text{topspace } X. f\ x \in V\}$
using $\langle \text{disjnt } U\ V \rangle$ **by** (*auto simp add: disjnt_def*)
qed (*use* $x\ \langle f\ x \in U \rangle\ y\ \langle f\ y \in V \rangle$ **in** *auto*)
qed

lemma *homeomorphic_Hausdorff_space:*

$X\ \text{homeomorphic\_space } Y \implies \text{Hausdorff\_space } X \longleftrightarrow \text{Hausdorff\_space } Y$
unfolding *homeomorphic_space_def* *homeomorphic_maps_map*
by (*auto simp: homeomorphic_eq everything_map Hausdorff_space_injective_preimage*)

lemma *Hausdorff_space_retraction_map_image:*

$\llbracket \text{retraction\_map } X\ Y\ r; \text{Hausdorff\_space } X \rrbracket \implies \text{Hausdorff\_space } Y$
unfolding *retraction_map_def*
using *Hausdorff_space_subtopology* *homeomorphic_Hausdorff_space* *retraction_maps_section_image*
by *blast*

lemma *compact_Hausdorff_space_optimal:*

assumes *eq*: *topspace* $Y = \text{topspace } X$ **and** XY : $\bigwedge U. \text{openin } X\ U \implies \text{openin } Y\ U$
and *Hausdorff_space* X *compact_space* Y
shows $Y = X$
proof $-$
have $\bigwedge U. \text{closedin } X\ U \implies \text{closedin } Y\ U$
using XY **using** *topology_finer_closedin* [*OF eq*]
by *metis*
have *openin* $Y\ S = \text{openin } X\ S$ **for** S
by (*metis* XY *assms*(3) *assms*(4) *closedin_compact_space* *compactin_contractive* *compactin_imp_closedin* *eq* *openin_closedin_eq*)
then show *?thesis*
by (*simp add: topology_eq*)
qed


```

lemma continuous_map_imp_closed_graph:
  assumes  $f$ : continuous_map  $X$   $Y$   $f$  and  $Y$ : Hausdorff_space  $Y$ 
  shows closedin (prod_topology  $X$   $Y$ ) (( $\lambda x$ . ( $x, f x$ )) 'topspace  $X$ )
  unfolding closedin_def
proof
  show ( $\lambda x$ . ( $x, f x$ )) 'topspace  $X$   $\subseteq$  topspace (prod_topology  $X$   $Y$ )
    using continuous_map_def  $f$  by fastforce
  show openin (prod_topology  $X$   $Y$ ) (topspace (prod_topology  $X$   $Y$ ) - ( $\lambda x$ . ( $x, f$ 
 $x$ )) 'topspace  $X$ )
    unfolding openin_prod_topology_alt
  proof (intro allI impI)
    show  $\exists U V$ . openin  $X$   $U$   $\wedge$  openin  $Y$   $V$   $\wedge$   $x \in U$   $\wedge$   $y \in V$   $\wedge$   $U \times V \subseteq$  topspace
    (prod_topology  $X$   $Y$ ) - ( $\lambda x$ . ( $x, f x$ )) 'topspace  $X$ 
      if ( $x, y$ )  $\in$  topspace (prod_topology  $X$   $Y$ ) - ( $\lambda x$ . ( $x, f x$ )) 'topspace  $X$ 
      for  $x y$ 
    proof -
      have  $x \in$  topspace  $X$   $y \in$  topspace  $Y$   $y \neq f x$ 
        using that by auto
      moreover have  $f x \in$  topspace  $Y$ 
        by (meson  $\langle x \in$  topspace  $X \rangle$  continuous_map_def  $f$ )
      ultimately obtain  $U V$  where  $UV$ : openin  $Y$   $U$  openin  $Y$   $V$   $f x \in U$   $y \in V$ 
        disjoint  $U V$ 
      using  $Y$  Hausdorff_space_def by metis
      show ?thesis
    proof (intro exI conjI)
      show openin  $X$  { $x \in$  topspace  $X$ .  $f x \in U$ }
        using  $\langle$ openin  $Y$   $U \rangle$   $f$  openin_continuous_map_preimage by blast
      show { $x \in$  topspace  $X$ .  $f x \in U$ }  $\times V \subseteq$  topspace (prod_topology  $X$   $Y$ ) -
      ( $\lambda x$ . ( $x, f x$ )) 'topspace  $X$ 
        using  $UV$  by (auto simp: disjoint_iff dest: openin_subset)
      qed (use  $UV$   $\langle x \in$  topspace  $X \rangle$  in auto)
    qed
  qed
qed

```

```

lemma continuous_imp_closed_map:
   $\llbracket$ continuous_map  $X$   $Y$   $f$ ; compact_space  $X$ ; Hausdorff_space  $Y$  $\rrbracket \implies$  closed_map
 $X$   $Y$   $f$ 
  by (meson closed_map_def closedin_compact_space compactin_imp_closedin
  image_compactin)

```

```

lemma continuous_imp_quotient_map:
   $\llbracket$ continuous_map  $X$   $Y$   $f$ ; compact_space  $X$ ; Hausdorff_space  $Y$ ;  $f$  ' (topspace
 $X$ ) = topspace  $Y$  $\rrbracket$ 
   $\implies$  quotient_map  $X$   $Y$   $f$ 
  by (simp add: continuous_imp_closed_map continuous_closed_imp_quotient_map)

```

```

lemma continuous_imp_homeomorphic_map:
   $\llbracket$ continuous_map  $X$   $Y$   $f$ ; compact_space  $X$ ; Hausdorff_space  $Y$ ;

```

$f^{-1}(\text{topspace } X) = \text{topspace } Y; \text{inj\_on } f(\text{topspace } X) \parallel$
 $\implies \text{homeomorphic\_map } X \ Y \ f$
by (simp add: continuous_imp_closed_map bijective_closed_imp_homeomorphic_map)

lemma continuous_imp_embedding_map:
 $\parallel \text{continuous\_map } X \ Y \ f; \text{compact\_space } X; \text{Hausdorff\_space } Y; \text{inj\_on } f(\text{topspace } X) \parallel$
 $\implies \text{embedding\_map } X \ Y \ f$
by (simp add: continuous_imp_closed_map injective_closed_imp_embedding_map)

lemma continuous_inverse_map:
assumes compact_space X Hausdorff_space Y
and cmf: continuous_map $X \ Y \ f$ **and** gf: $\bigwedge x. x \in \text{topspace } X \implies g(f\ x) = x$
and Sf: $S \subseteq f^{-1}(\text{topspace } X)$
shows continuous_map (subtopology $Y \ S$) $X \ g$
proof (rule continuous_map_from_subtopology_mono [OF $\langle S \subseteq f^{-1}(\text{topspace } X) \rangle$])
show continuous_map (subtopology $Y (f^{-1}(\text{topspace } X))) \ X \ g$
unfolding continuous_map_closedin
proof (intro conjI ballI allI impI)
fix x
assume $x \in \text{topspace } (\text{subtopology } Y (f^{-1}(\text{topspace } X)))$
then show $g\ x \in \text{topspace } X$
by (auto simp: gf)
next
fix C
assume C : closedin $X \ C$
show closedin (subtopology $Y (f^{-1}(\text{topspace } X)))$
 $\{x \in \text{topspace } (\text{subtopology } Y (f^{-1}(\text{topspace } X))). g\ x \in C\}$
proof (rule compactin_imp_closedin)
show Hausdorff_space (subtopology $Y (f^{-1}(\text{topspace } X)))$
using Hausdorff_space_subtopology [OF $\langle \text{Hausdorff\_space } Y \rangle$] **by** blast
have compactin $Y (f^{-1} C)$
using C cmf image_compactin closedin_compact_space [OF $\langle \text{compact\_space } X \rangle$] **by** blast
moreover have $\{x \in \text{topspace } Y. x \in f^{-1}(\text{topspace } X) \wedge g\ x \in C\} = f^{-1} C$
using closedin_subset [OF C] cmf **by** (auto simp: gf continuous_map_def)
ultimately have compactin $Y \{x \in \text{topspace } Y. x \in f^{-1}(\text{topspace } X) \wedge g\ x \in C\}$
by simp
then show compactin (subtopology $Y (f^{-1}(\text{topspace } X)))$
 $\{x \in \text{topspace } (\text{subtopology } Y (f^{-1}(\text{topspace } X))). g\ x \in C\}$
by (auto simp add: compactin_subtopology)
qed
qed
qed

lemma closed_map_paired_continuous_map_right:
 $\parallel \text{continuous\_map } X \ Y \ f; \text{Hausdorff\_space } Y \parallel \implies \text{closed\_map } X \ (\text{prod\_topology } Y \ f)$

$X\ Y) (\lambda x. (x, f\ x))$
by (simp add: continuous_map_imp_closed_graph embedding_map_graph embedding_imp_closed_map)

lemma closed_map_paired_continuous_map_left:
assumes f : continuous_map $X\ Y\ f$ **and** Y : Hausdorff_space Y
shows closed_map X (prod_topology $Y\ X$) $(\lambda x. (f\ x, x))$
proof –
have eq: $(\lambda x. (f\ x, x)) = (\lambda(a, b). (b, a)) \circ (\lambda x. (x, f\ x))$
by auto
show ?thesis
unfolding eq
proof (rule closed_map_compose)
show closed_map X (prod_topology $X\ Y$) $(\lambda x. (x, f\ x))$
using Y closed_map_paired_continuous_map_right f **by** blast
show closed_map (prod_topology $X\ Y$) (prod_topology $Y\ X$) $(\lambda(a, b). (b, a))$
by (metis homeomorphic_map_swap homeomorphic_imp_closed_map)
qed
qed

lemma proper_map_paired_continuous_map_right:
 $\llbracket$ continuous_map $X\ Y\ f$; Hausdorff_space Y $\rrbracket$
 $\implies$ proper_map X (prod_topology $X\ Y$) $(\lambda x. (x, f\ x))$
using closed_injective_imp_proper_map closed_map_paired_continuous_map_right
by (metis (mono_tags, lifting) Pair_inject inj_onI)

lemma proper_map_paired_continuous_map_left:
 $\llbracket$ continuous_map $X\ Y\ f$; Hausdorff_space Y $\rrbracket$
 $\implies$ proper_map X (prod_topology $Y\ X$) $(\lambda x. (f\ x, x))$
using closed_injective_imp_proper_map closed_map_paired_continuous_map_left
by (metis (mono_tags, lifting) Pair_inject inj_onI)

lemma Hausdorff_space_prod_topology:
Hausdorff_space (prod_topology $X\ Y$) $\longleftrightarrow$ topspace (prod_topology $X\ Y$) = $\{\}$ $\vee$
Hausdorff_space $X \wedge$ Hausdorff_space Y
(is ?lhs = ?rhs)
proof
assume ?lhs
then show ?rhs
by (rule topological_property_of_prod_component) (auto simp: Hausdorff_space_subtopology homeomorphic_Hausdorff_space)
next
assume R : ?rhs
show ?lhs
proof (cases (topspace $X \times$ topspace Y) = $\{\}$)
case False
with R **have** ne : topspace $X \neq \{\}$ topspace $Y \neq \{\}$ **and** X : Hausdorff_space
 X **and** Y : Hausdorff_space Y
by auto

```

show ?thesis
  unfolding Hausdorff_space_def
proof clarify
  fix x y x' y'
  assume xy: (x, y) ∈ topspace (prod_topology X Y)
  and xy': (x', y') ∈ topspace (prod_topology X Y)
  and *:  $\nexists U V. \text{openin} (\text{prod\_topology } X \ Y) \ U \wedge \text{openin} (\text{prod\_topology } X \ Y) \ V$ 
   $\wedge (x, y) \in U \wedge (x', y') \in V \wedge \text{disjnt } U \ V$ 
  have False if  $x \neq x' \vee y \neq y'$ 
  using that
  proof
    assume  $x \neq x'$ 
    then obtain U V where openin X U openin X V  $x \in U \ x' \in V \ \text{disjnt } U \ V$ 
    by (metis Hausdorff_space_def X mem_Sigma_iff topspace_prod_topology
xy xy')
    let ?U =  $U \times \text{topspace } Y$ 
    let ?V =  $V \times \text{topspace } Y$ 
    have openin (prod_topology X Y) ?U openin (prod_topology X Y) ?V
    by (simp_all add: openin_prod_Times_iff ⟨openin X U⟩ ⟨openin X V⟩)
    moreover have disjnt ?U ?V
    by (simp add: ⟨disjnt U V⟩)
    ultimately show False
  using * ⟨ $x \in U$ ⟩ ⟨ $x' \in V$ ⟩ xy xy' by (metis SigmaD2 SigmaI topspace_prod_topology)
  next
    assume  $y \neq y'$ 
    then obtain U V where openin Y U openin Y V  $y \in U \ y' \in V \ \text{disjnt } U \ V$ 
    by (metis Hausdorff_space_def Y mem_Sigma_iff topspace_prod_topology
xy xy')
    let ?U =  $\text{topspace } X \times U$ 
    let ?V =  $\text{topspace } X \times V$ 
    have openin (prod_topology X Y) ?U openin (prod_topology X Y) ?V
    by (simp_all add: openin_prod_Times_iff ⟨openin Y U⟩ ⟨openin Y V⟩)
    moreover have disjnt ?U ?V
    by (simp add: ⟨disjnt U V⟩)
    ultimately show False
  using * ⟨ $y \in U$ ⟩ ⟨ $y' \in V$ ⟩ xy xy' by (metis SigmaD1 SigmaI topspace_prod_topology)
  qed
  then show  $x = x' \wedge y = y'$ 
  by blast
  qed
qed (simp add: Hausdorff_space_topspace_empty)
qed

```

lemma Hausdorff_space_product_topology:

$\text{Hausdorff\_space } (\text{product\_topology } X \ I) \longleftrightarrow (\Pi_E \ i \in I. \text{topspace}(X \ i)) = \{\} \vee$
 $(\forall i \in I. \text{Hausdorff\_space } (X \ i))$
(is ?lhs = ?rhs)

```

proof
  assume ?lhs
  then show ?rhs
    apply (rule topological_property_of_product_component)
    apply (blast dest: Hausdorff_space_subtopology_homeomorphic_Hausdorff_space)+
    done
next
  assume R: ?rhs
  show ?lhs
  proof (cases ( $\prod_{E \in I} \text{topspace}(X \ i) = \{\}$ ))
    case True
    then show ?thesis
      by (simp add: Hausdorff_space_topspace_empty)
    next
    case False
    have  $\exists U \ V. \text{openin}(\text{product\_topology } X \ I) \ U \wedge \text{openin}(\text{product\_topology } X \ I) \ V \wedge f \in U \wedge g \in V \wedge \text{disjnt } U \ V$ 
    if f:  $f \in (\prod_{E \in I} \text{topspace}(X \ i))$  and g:  $g \in (\prod_{E \in I} \text{topspace}(X \ i))$  and  $f \neq g$ 
      for  $f \ g :: 'a \Rightarrow 'b$ 
    proof -
      obtain  $m$  where  $f \ m \neq g \ m$ 
      using  $\langle f \neq g \rangle$  by blast
      then have  $m \in I$ 
      using  $f \ g$  by fastforce
      then have Hausdorff_space  $(X \ m)$ 
      using False that R by blast
      then obtain  $U \ V$  where  $U: \text{openin}(X \ m) \ U$  and  $V: \text{openin}(X \ m) \ V$  and  $f \ m \in U \ g \ m \in V \text{ disjnt } U \ V$ 
      by (metis Hausdorff_space_def PiE_mem  $\langle f \ m \neq g \ m \rangle \langle m \in I \rangle f \ g$ )
      show ?thesis
      proof (intro exI conjI)
        let ?U =  $(\prod_{E \in I} \text{topspace}(X \ i)) \cap \{x. x \ m \in U\}$ 
        let ?V =  $(\prod_{E \in I} \text{topspace}(X \ i)) \cap \{x. x \ m \in V\}$ 
        show  $\text{openin}(\text{product\_topology } X \ I) \ ?U \ \text{openin}(\text{product\_topology } X \ I) \ ?V$ 
        using  $\langle m \in I \rangle \ U \ V$ 
        by (force simp add: openin_product_topology intro: arbitrary_union_of_inc
            relative_to_inc finite_intersection_of_inc)+
        show  $f \in ?U$ 
        using  $\langle f \ m \in U \rangle \ f$  by blast
        show  $g \in ?V$ 
        using  $\langle g \ m \in V \rangle \ g$  by blast
        show  $\text{disjnt } ?U \ ?V$ 
        using  $\langle \text{disjnt } U \ V \rangle$  by (auto simp: PiE_def Pi_def disjnt_def)
      qed
    qed
  then show ?thesis
    by (simp add: Hausdorff_space_def)
qed

```

1114

qed

end

5.5 Path-Connectedness

```
theory Path_Connected
imports
  Starlike
  T1_Spaces
begin
```

5.5.1 Paths and Arcs

```
definition path :: (real  $\Rightarrow$  'a::topological_space)  $\Rightarrow$  bool
  where path g  $\longleftrightarrow$  continuous_on {0..1} g
```

```
definition pathstart :: (real  $\Rightarrow$  'a::topological_space)  $\Rightarrow$  'a
  where pathstart g = g 0
```

```
definition pathfinish :: (real  $\Rightarrow$  'a::topological_space)  $\Rightarrow$  'a
  where pathfinish g = g 1
```

```
definition path_image :: (real  $\Rightarrow$  'a::topological_space)  $\Rightarrow$  'a set
  where path_image g = g ` {0 .. 1}
```

```
definition reversepath :: (real  $\Rightarrow$  'a::topological_space)  $\Rightarrow$  real  $\Rightarrow$  'a
  where reversepath g = ( $\lambda x$ . g(1 - x))
```

```
definition joinpaths :: (real  $\Rightarrow$  'a::topological_space)  $\Rightarrow$  (real  $\Rightarrow$  'a)  $\Rightarrow$  real  $\Rightarrow$  'a
  (infixr +++ 75)
  where g1 +++ g2 = ( $\lambda x$ . if x  $\leq$  1/2 then g1 (2 * x) else g2 (2 * x - 1))
```

```
definition simple_path :: (real  $\Rightarrow$  'a::topological_space)  $\Rightarrow$  bool
  where simple_path g  $\longleftrightarrow$ 
    path g  $\wedge$  ( $\forall x \in \{0..1\}. \forall y \in \{0..1\}. g\ x = g\ y \longrightarrow x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0$ )
```

```
definition arc :: (real  $\Rightarrow$  'a::topological_space)  $\Rightarrow$  bool
  where arc g  $\longleftrightarrow$  path g  $\wedge$  inj_on g {0..1}
```

5.5.2 Invariance theorems

```
lemma path_eq: path p  $\implies$  ( $\bigwedge t. t \in \{0..1\} \implies p\ t = q\ t$ )  $\implies$  path q
  using continuous_on_eq path_def by blast
```

```
lemma path_continuous_image: path g  $\implies$  continuous_on (path_image g) f  $\implies$ 
  path(f  $\circ$  g)
  unfolding path_def path_image_def
```

using continuous_on_compose by blast

lemma continuous_on_translation_eq:

fixes $g :: 'a :: \text{real\_normed\_vector} \Rightarrow 'b :: \text{real\_normed\_vector}$

shows continuous_on $A ((+) a \circ g) = \text{continuous\_on } A g$

proof -

have $g: g = (\lambda x. -a + x) \circ ((\lambda x. a + x) \circ g)$

by (rule ext) simp

show ?thesis

by (metis (no_types, opaque_lifting) g continuous_on_compose homeomorphism_def homeomorphism_translation)

qed

lemma path_translation_eq:

fixes $g :: \text{real} \Rightarrow 'a :: \text{real\_normed\_vector}$

shows $\text{path}((\lambda x. a + x) \circ g) = \text{path } g$

using continuous_on_translation_eq path_def by blast

lemma path_linear_image_eq:

fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$

assumes linear f inj f

shows $\text{path}(f \circ g) = \text{path } g$

proof -

from linear_injective_left_inverse [OF assms]

obtain h where $h: \text{linear } h \ h \circ f = \text{id}$

by blast

then have $g: g = h \circ (f \circ g)$

by (metis comp_assoc id_comp)

show ?thesis

unfolding path_def

using h assms

by (metis g continuous_on_compose linear_continuous_on linear_conv_bounded_linear)

qed

lemma pathstart_translation: $\text{pathstart}((\lambda x. a + x) \circ g) = a + \text{pathstart } g$

by (simp add: pathstart_def)

lemma pathstart_linear_image_eq: $\text{linear } f \implies \text{pathstart}(f \circ g) = f(\text{pathstart } g)$

by (simp add: pathstart_def)

lemma pathfinish_translation: $\text{pathfinish}((\lambda x. a + x) \circ g) = a + \text{pathfinish } g$

by (simp add: pathfinish_def)

lemma pathfinish_linear_image: $\text{linear } f \implies \text{pathfinish}(f \circ g) = f(\text{pathfinish } g)$

by (simp add: pathfinish_def)

lemma path_image_translation: $\text{path\_image}((\lambda x. a + x) \circ g) = (\lambda x. a + x) \text{ ` } (\text{path\_image } g)$

by (simp add: image_comp path_image_def)

lemma *path_image_linear_image*: $\text{linear } f \implies \text{path\_image}(f \circ g) = f'(\text{path\_image } g)$

by (*simp add: image_comp path_image_def*)

lemma *reversepath_translation*: $\text{reversepath}((\lambda x. a + x) \circ g) = (\lambda x. a + x) \circ \text{reversepath } g$

by (*rule ext*) (*simp add: reversepath_def*)

lemma *reversepath_linear_image*: $\text{linear } f \implies \text{reversepath}(f \circ g) = f \circ \text{reversepath } g$

by (*rule ext*) (*simp add: reversepath_def*)

lemma *joinpaths_translation*:

$((\lambda x. a + x) \circ g1) +++ ((\lambda x. a + x) \circ g2) = (\lambda x. a + x) \circ (g1 +++ g2)$

by (*rule ext*) (*simp add: joinpaths_def*)

lemma *joinpaths_linear_image*: $\text{linear } f \implies (f \circ g1) +++ (f \circ g2) = f \circ (g1 +++ g2)$

by (*rule ext*) (*simp add: joinpaths_def*)

lemma *simple_path_translation_eq*:

fixes $g :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$

shows $\text{simple\_path}((\lambda x. a + x) \circ g) = \text{simple\_path } g$

by (*simp add: simple_path_def path_translation_eq*)

lemma *simple_path_linear_image_eq*:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$

assumes $\text{linear } f \text{ inj } f$

shows $\text{simple\_path}(f \circ g) = \text{simple\_path } g$

using *assms inj_on_eq_iff* [of f]

by (*auto simp: path_linear_image_eq simple_path_def path_translation_eq*)

lemma *arc_translation_eq*:

fixes $g :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$

shows $\text{arc}((\lambda x. a + x) \circ g) = \text{arc } g$

by (*auto simp: arc_def inj_on_def path_translation_eq*)

lemma *arc_linear_image_eq*:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$

assumes $\text{linear } f \text{ inj } f$

shows $\text{arc}(f \circ g) = \text{arc } g$

using *assms inj_on_eq_iff* [of f]

by (*auto simp: arc_def inj_on_def path_linear_image_eq*)

5.5.3 Basic lemmas about paths

lemma *path_of_real*: $\text{path\_complex\_of\_real}$

unfolding *path_def* **by** (*intro continuous_intros*)


```

lemma path_const: path ( $\lambda t. a$ ) for  $a::'a::\text{real\_normed\_vector}$ 
  unfolding path_def by (intro continuous_intros)

lemma path_minus: path  $g \implies \text{path } (\lambda t. - g\ t)$  for  $g::\text{real} \Rightarrow 'a::\text{real\_normed\_vector}$ 
  unfolding path_def by (intro continuous_intros)

lemma path_add:  $\llbracket \text{path } f; \text{path } g \rrbracket \implies \text{path } (\lambda t. f\ t + g\ t)$  for  $f::\text{real} \Rightarrow 'a::\text{real\_normed\_vector}$ 
  unfolding path_def by (intro continuous_intros)

lemma path_diff:  $\llbracket \text{path } f; \text{path } g \rrbracket \implies \text{path } (\lambda t. f\ t - g\ t)$  for  $f::\text{real} \Rightarrow 'a::\text{real\_normed\_vector}$ 
  unfolding path_def by (intro continuous_intros)

lemma path_mult:  $\llbracket \text{path } f; \text{path } g \rrbracket \implies \text{path } (\lambda t. f\ t * g\ t)$  for  $f::\text{real} \Rightarrow 'a::\text{real\_normed\_field}$ 
  unfolding path_def by (intro continuous_intros)

lemma pathin_iff_path_real [simp]: pathin euclideanreal  $g \longleftrightarrow \text{path } g$ 
  by (simp add: pathin_def path_def)

lemma continuous_on_path: path  $f \implies t \subseteq \{0..1\} \implies \text{continuous\_on } t\ f$ 
  using continuous_on_subset path_def by blast

lemma arc_imp_simple_path: arc  $g \implies \text{simple\_path } g$ 
  by (simp add: arc_def inj_on_def simple_path_def)

lemma arc_imp_path: arc  $g \implies \text{path } g$ 
  using arc_def by blast

lemma arc_imp_inj_on: arc  $g \implies \text{inj\_on } g\ \{0..1\}$ 
  by (auto simp: arc_def)

lemma simple_path_imp_path: simple_path  $g \implies \text{path } g$ 
  using simple_path_def by blast

lemma simple_path_cases: simple_path  $g \implies \text{arc } g \vee \text{pathfinish } g = \text{pathstart } g$ 
  unfolding simple_path_def arc_def inj_on_def pathfinish_def pathstart_def
  by force

lemma simple_path_imp_arc: simple_path  $g \implies \text{pathfinish } g \neq \text{pathstart } g \implies$ 
  arc  $g$ 
  using simple_path_cases by auto

lemma arc_distinct_ends: arc  $g \implies \text{pathfinish } g \neq \text{pathstart } g$ 
  unfolding arc_def inj_on_def pathfinish_def pathstart_def
  by fastforce

lemma arc_simple_path: arc  $g \longleftrightarrow \text{simple\_path } g \wedge \text{pathfinish } g \neq \text{pathstart } g$ 
  using arc_distinct_ends arc_imp_simple_path simple_path_cases by blast

```

lemma *simple_path_eq_arc*: $\text{pathfinish } g \neq \text{pathstart } g \implies (\text{simple\_path } g = \text{arc } g)$

by (*simp add: arc_simple_path*)

lemma *path_image_const* [*simp*]: $\text{path\_image } (\lambda t. a) = \{a\}$

by (*force simp: path_image_def*)

lemma *path_image_nonempty* [*simp*]: $\text{path\_image } g \neq \{\}$

unfolding *path_image_def image_is_empty box_eq_empty*

by *auto*

lemma *pathstart_in_path_image*[*intro*]: $\text{pathstart } g \in \text{path\_image } g$

unfolding *pathstart_def path_image_def*

by *auto*

lemma *pathfinish_in_path_image*[*intro*]: $\text{pathfinish } g \in \text{path\_image } g$

unfolding *pathfinish_def path_image_def*

by *auto*

lemma *connected_path_image*[*intro*]: $\text{path } g \implies \text{connected } (\text{path\_image } g)$

unfolding *path_def path_image_def*

using *connected_continuous_image connected_Icc* **by** *blast*

lemma *compact_path_image*[*intro*]: $\text{path } g \implies \text{compact } (\text{path\_image } g)$

unfolding *path_def path_image_def*

using *compact_continuous_image connected_Icc* **by** *blast*

lemma *reversepath_reversepath*[*simp*]: $\text{reversepath } (\text{reversepath } g) = g$

unfolding *reversepath_def*

by *auto*

lemma *pathstart_reversepath*[*simp*]: $\text{pathstart } (\text{reversepath } g) = \text{pathfinish } g$

unfolding *pathstart_def reversepath_def pathfinish_def*

by *auto*

lemma *pathfinish_reversepath*[*simp*]: $\text{pathfinish } (\text{reversepath } g) = \text{pathstart } g$

unfolding *pathstart_def reversepath_def pathfinish_def*

by *auto*

lemma *reversepath_o*: $\text{reversepath } g = g \circ (-)1$

by (*auto simp: reversepath_def*)

lemma *pathstart_join*[*simp*]: $\text{pathstart } (g1 +++ g2) = \text{pathstart } g1$

unfolding *pathstart_def joinpaths_def pathfinish_def*

by *auto*

lemma *pathfinish_join*[*simp*]: $\text{pathfinish } (g1 +++ g2) = \text{pathfinish } g2$

unfolding *pathstart_def joinpaths_def pathfinish_def*

by *auto*

```

lemma path_image_reversepath[simp]: path_image (reversepath g) = path_image
g
proof -
  have *:  $\bigwedge g. \text{path\_image } (\text{reversepath } g) \subseteq \text{path\_image } g$ 
    unfolding path_image_def subset_eq reversepath_def Ball_def image_iff
    by force
  show ?thesis
    using *[of g] *[of reversepath g]
    unfolding reversepath_reversepath
    by auto
qed

```

```

lemma path_reversepath [simp]: path (reversepath g)  $\longleftrightarrow$  path g
proof -
  have *:  $\bigwedge g. \text{path } g \implies \text{path } (\text{reversepath } g)$ 
    unfolding path_def reversepath_def
    apply (rule continuous_on_compose[unfolded o_def, of  $\lambda x. 1 - x$ ])
    apply (auto intro: continuous_intros continuous_on_subset[of  $\{0..1\}$ ])
    done
  show ?thesis
    using * by force
qed

```

```

lemma arc_reversepath:
  assumes arc g shows arc(reversepath g)
proof -
  have injg: inj_on g  $\{0..1\}$ 
    using assms
    by (simp add: arc_def)
  have **:  $\bigwedge x y::\text{real}. 1-x = 1-y \implies x = y$ 
    by simp
  show ?thesis
    using assms by (clarsimp simp: arc_def intro!: inj_onI) (simp add: inj_onD
reversepath_def **)
qed

```

```

lemma simple_path_reversepath: simple_path g  $\implies$  simple_path (reversepath g)
  apply (simp add: simple_path_def)
  apply (force simp: reversepath_def)
  done

```

```

lemmas reversepath_simps =
  path_reversepath path_image_reversepath pathstart_reversepath pathfinish_reversepath

```

```

lemma path_join[simp]:
  assumes pathfinish g1 = pathstart g2
  shows path (g1 +++ g2)  $\longleftrightarrow$  path g1  $\wedge$  path g2
  unfolding path_def pathfinish_def pathstart_def

```

```

proof safe
  assume cont: continuous_on {0..1} (g1 +++ g2)
  have g1: continuous_on {0..1} g1  $\longleftrightarrow$  continuous_on {0..1} ((g1 +++ g2)  $\circ$ 
    ( $\lambda x. x / 2$ ))
    by (intro continuous_on_cong refl) (auto simp: joinpaths_def)
  have g2: continuous_on {0..1} g2  $\longleftrightarrow$  continuous_on {0..1} ((g1 +++ g2)  $\circ$ 
    ( $\lambda x. x / 2 + 1/2$ ))
    using assms
    by (intro continuous_on_cong refl) (auto simp: joinpaths_def pathfinish_def
pathstart_def)
  show continuous_on {0..1} g1 and continuous_on {0..1} g2
    unfolding g1 g2
    by (auto intro!: continuous_intros continuous_on_subset[OF cont] simp del:
o_apply)
next
  assume g1g2: continuous_on {0..1} g1 continuous_on {0..1} g2
  have 01: {0 .. 1} = {0..1/2}  $\cup$  {1/2 .. 1::real}
    by auto
  {
    fix x :: real
    assume  $0 \leq x$  and  $x \leq 1$ 
    then have  $x \in (\lambda x. x * 2) \text{ ' } \{0..1 / 2\}$ 
      by (intro image_eqI[where x=x/2]) auto
  }
  note 1 = this
  {
    fix x :: real
    assume  $0 \leq x$  and  $x \leq 1$ 
    then have  $x \in (\lambda x. x * 2 - 1) \text{ ' } \{1 / 2..1\}$ 
      by (intro image_eqI[where x=x/2 + 1/2]) auto
  }
  note 2 = this
  show continuous_on {0..1} (g1 +++ g2)
    using assms
    unfolding joinpaths_def 01
    apply (intro continuous_on_cases closed_atLeastAtMost g1g2[THEN contin-
uous_on_compose2] continuous_intros)
    apply (auto simp: field_simps pathfinish_def pathstart_def intro!: 1 2)
    done
qed

```

5.5.4 Path Images

```

lemma bounded_path_image: path g  $\implies$  bounded(path_image g)
  by (simp add: compact_imp_bounded compact_path_image)

```

```

lemma closed_path_image:
  fixes g :: real  $\Rightarrow$  'a::t2_space
  shows path g  $\implies$  closed(path_image g)

```

```

  by (metis compact_path_image compact_imp_closed)

lemma connected_simple_path_image: simple_path g  $\implies$  connected(path_image g)
  by (metis connected_path_image simple_path_imp_path)

lemma compact_simple_path_image: simple_path g  $\implies$  compact(path_image g)
  by (metis compact_path_image simple_path_imp_path)

lemma bounded_simple_path_image: simple_path g  $\implies$  bounded(path_image g)
  by (metis bounded_path_image simple_path_imp_path)

lemma closed_simple_path_image:
  fixes g :: real  $\Rightarrow$  'a::t2_space
  shows simple_path g  $\implies$  closed(path_image g)
  by (metis closed_path_image simple_path_imp_path)

lemma connected_arc_image: arc g  $\implies$  connected(path_image g)
  by (metis connected_path_image arc_imp_path)

lemma compact_arc_image: arc g  $\implies$  compact(path_image g)
  by (metis compact_path_image arc_imp_path)

lemma bounded_arc_image: arc g  $\implies$  bounded(path_image g)
  by (metis bounded_path_image arc_imp_path)

lemma closed_arc_image:
  fixes g :: real  $\Rightarrow$  'a::t2_space
  shows arc g  $\implies$  closed(path_image g)
  by (metis closed_path_image arc_imp_path)

lemma path_image_join_subset: path_image (g1 +++ g2)  $\subseteq$  path_image g1  $\cup$ 
  path_image g2
  unfolding path_image_def joinpaths_def
  by auto

lemma subset_path_image_join:
  assumes path_image g1  $\subseteq$  s
  and path_image g2  $\subseteq$  s
  shows path_image (g1 +++ g2)  $\subseteq$  s
  using path_image_join_subset[of g1 g2] and assms
  by auto

lemma path_image_join:
  assumes pathfinish g1 = pathstart g2
  shows path_image(g1 +++ g2) = path_image g1  $\cup$  path_image g2
proof -
  have path_image g1  $\subseteq$  path_image (g1 +++ g2)
  proof (clarsimp simp: path_image_def joinpaths_def)

```

```

    fix u::real
    assume 0 ≤ u u ≤ 1
    then show g1 u ∈ (λx. g1 (2 * x)) ‘ ({0..1} ∩ {x. x * 2 ≤ 1})
      by (rule_tac x=u/2 in image_eqI) auto
  qed
  moreover
  have §: g2 u ∈ (λx. g2 (2 * x - 1)) ‘ ({0..1} ∩ {x. ¬ x * 2 ≤ 1})
    if 0 < u u ≤ 1 for u
    using that assms
    by (rule_tac x=(u+1)/2 in image_eqI) (auto simp: field_simps pathfinish_def
pathstart_def)
  have g2 0 ∈ (λx. g1 (2 * x)) ‘ ({0..1} ∩ {x. x * 2 ≤ 1})
    using assms
    by (rule_tac x=1/2 in image_eqI) (auto simp: pathfinish_def pathstart_def)
  then have path_image g2 ⊆ path_image (g1 +++ g2)
    by (auto simp: path_image_def joinpaths_def intro!: §)
  ultimately show ?thesis
    using path_image_join_subset by blast
  qed

lemma not_in_path_image_join:
  assumes x ∉ path_image g1
  and x ∉ path_image g2
  shows x ∉ path_image (g1 +++ g2)
  using assms and path_image_join_subset[of g1 g2]
  by auto

lemma pathstart_compose: pathstart(f ∘ p) = f(pathstart p)
  by (simp add: pathstart_def)

lemma pathfinish_compose: pathfinish(f ∘ p) = f(pathfinish p)
  by (simp add: pathfinish_def)

lemma path_image_compose: path_image (f ∘ p) = f ‘ (path_image p)
  by (simp add: image_comp path_image_def)

lemma path_compose_join: f ∘ (p +++ q) = (f ∘ p) +++ (f ∘ q)
  by (rule ext) (simp add: joinpaths_def)

lemma path_compose_reversepath: f ∘ reversepath p = reversepath(f ∘ p)
  by (rule ext) (simp add: reversepath_def)

lemma joinpaths_eq:
  (∧ t. t ∈ {0..1} ⇒ p t = p' t) ⇒
  (∧ t. t ∈ {0..1} ⇒ q t = q' t)
  ⇒ t ∈ {0..1} ⇒ (p +++ q) t = (p' +++ q') t
  by (auto simp: joinpaths_def)

lemma simple_path_inj_on: simple_path g ⇒ inj_on g {0<..<1}

```

by (auto simp: simple_path_def path_image_def inj_on_def less_eq_real_def Ball_def)

5.5.5 Simple paths with the endpoints removed

lemma *simple_path_endless*:

assumes *simple_path c*

shows $\text{path\_image } c - \{\text{pathstart } c, \text{pathfinish } c\} = c \restriction \{0 < .. < 1\}$ (is ?lhs = ?rhs)

proof

show ?lhs $\subseteq$?rhs

using less_eq_real_def by (auto simp: path_image_def pathstart_def pathfinish_def)

show ?rhs $\subseteq$?lhs

using assms

apply (auto simp: simple_path_def path_image_def pathstart_def pathfinish_def Ball_def)

using less_eq_real_def zero_le_one by blast+

qed

lemma *connected_simple_path_endless*:

assumes *simple_path c*

shows *connected*($\text{path\_image } c - \{\text{pathstart } c, \text{pathfinish } c\}$)

proof –

have *continuous_on* $\{0 < .. < 1\}$ *c*

using assms by (simp add: simple_path_def continuous_on_path path_def subset_iff)

then have *connected* ($c \restriction \{0 < .. < 1\}$)

using *connected_Ioo_connected_continuous_image* by blast

then show ?thesis

using assms by (simp add: simple_path_endless)

qed

lemma *nonempty_simple_path_endless*:

simple_path c $\implies \text{path\_image } c - \{\text{pathstart } c, \text{pathfinish } c\} \neq \{\}$

by (simp add: simple_path_endless)

5.5.6 The operations on paths

lemma *path_image_subset_reversepath*: $\text{path\_image}(\text{reversepath } g) \leq \text{path\_image } g$

by *simp*

lemma *path_imp_reversepath*: $\text{path } g \implies \text{path}(\text{reversepath } g)$

by *simp*

lemma *half_bounded_equal*: $1 \leq x * 2 \implies x * 2 \leq 1 \longleftrightarrow x = (1/2::\text{real})$

by *simp*

lemma *continuous_on_joinpaths*:

```

    assumes continuous_on {0..1} g1 continuous_on {0..1} g2 pathfinish g1 =
    pathstart g2
    shows continuous_on {0..1} (g1 +++ g2)
  proof -
    have {0..1::real} = {0..1/2} ∪ {1/2..1}
    by auto
    then show ?thesis
    using assms by (metis path_def path_join)
  qed

```

```

lemma path_join_imp:  $\llbracket \text{path } g1; \text{ path } g2; \text{ pathfinish } g1 = \text{pathstart } g2 \rrbracket \implies$ 
  path(g1 +++ g2)
  by simp

```

```

lemma simple_path_join_loop:
  assumes arc g1 arc g2
    pathfinish g1 = pathstart g2 pathfinish g2 = pathstart g1
    path_image g1 ∩ path_image g2 ⊆ {pathstart g1, pathstart g2}
  shows simple_path(g1 +++ g2)
  proof -
    have injg1: inj_on g1 {0..1}
    using assms
    by (simp add: arc_def)
    have injg2: inj_on g2 {0..1}
    using assms
    by (simp add: arc_def)
    have g12: g1 1 = g2 0
    and g21: g2 1 = g1 0
    and sb: g1 ‘ {0..1} ∩ g2 ‘ {0..1} ⊆ {g1 0, g2 0}
    using assms
    by (simp_all add: arc_def pathfinish_def pathstart_def path_image_def)
    { fix x and y::real
      assume g2_eq: g2 (2 * x - 1) = g1 (2 * y)
      and xyI: x ≠ 1 ∨ y ≠ 0
      and xy: x ≤ 1 0 ≤ y y * 2 ≤ 1 ∧ x * 2 ≤ 1
      then consider g1 (2 * y) = g1 0 | g1 (2 * y) = g2 0
      using sb by force
      then have False
    proof cases
      case 1
      then have y = 0
      using xy g2_eq by (auto dest!: inj_onD [OF injg1])
      then show ?thesis
      using xy g2_eq xyI by (auto dest: inj_onD [OF injg2] simp flip: g21)
    next
      case 2
      then have 2*x = 1
      using g2_eq g12 inj_onD [OF injg2] atLeastAtMost_iff xy(1) xy(4) by
      fastforce
    }
  }

```



```

    with  $xy$  show False by auto
  qed
} note * = this
{ fix  $x$  and  $y::real$ 
  assume  $xy: g1 (2 * x) = g2 (2 * y - 1) \ y \leq 1 \ 0 \leq x \neg y * 2 \leq 1 \ x * 2 \leq 1$ 
  then have  $x = 0 \wedge y = 1$ 
    using *  $xy$  by force
  } note ** = this
show ?thesis
  using assms
  apply (simp add: arc_def simple_path_def)
  apply (auto simp: joinpaths_def split: if_split_asm
    dest!: * ** dest: inj_onD [OF injg1] inj_onD [OF injg2])
  done
qed

lemma arc_join:
  assumes arc  $g1$  arc  $g2$ 
    pathfinish  $g1 = \text{pathstart } g2$ 
    path_image  $g1 \cap \text{path\_image } g2 \subseteq \{\text{pathstart } g2\}$ 
  shows arc( $g1 \mathrel{+++} g2$ )
proof -
  have injg1: inj_on  $g1 \ \{0..1\}$ 
    using assms
    by (simp add: arc_def)
  have injg2: inj_on  $g2 \ \{0..1\}$ 
    using assms
    by (simp add: arc_def)
  have g11:  $g1 \ 1 = g2 \ 0$ 
  and sb:  $g1 \ ' \{0..1\} \cap g2 \ ' \{0..1\} \subseteq \{g2 \ 0\}$ 
    using assms
    by (simp_all add: arc_def pathfinish_def pathstart_def path_image_def)
  { fix  $x$  and  $y::real$ 
    assume  $xy: g2 (2 * x - 1) = g1 (2 * y) \ x \leq 1 \ 0 \leq y \ y * 2 \leq 1 \neg x * 2 \leq 1$ 
    then have  $g1 (2 * y) = g2 \ 0$ 
      using sb by force
    then have False
      using  $xy$  inj_onD injg2 by fastforce
  } note * = this
show ?thesis
  using assms
  apply (simp add: arc_def inj_on_def)
  apply (auto simp: joinpaths_def arc_imp_path split: if_split_asm
    dest: * *[OF sym] inj_onD [OF injg1] inj_onD [OF injg2])
  done
qed

lemma reversepath_joinpaths:
  pathfinish  $g1 = \text{pathstart } g2 \implies \text{reversepath}(g1 \mathrel{+++} g2) = \text{reversepath } g2$ 

```

```

+++ reversepath g1
  unfolding reversepath_def pathfinish_def pathstart_def joinpaths_def
  by (rule ext) (auto simp: mult.commute)

```

5.5.7 Some reversed and "if and only if" versions of joining theorems

```

lemma path_join_path_ends:
  fixes g1 :: real  $\Rightarrow$  'a::metric_space
  assumes path(g1 +++ g2) path g2
  shows pathfinish g1 = pathstart g2
proof (rule ccontr)
  define e where e = dist (g1 1) (g2 0)
  assume Neg: pathfinish g1  $\neq$  pathstart g2
  then have 0 < dist (pathfinish g1) (pathstart g2)
    by auto
  then have e > 0
    by (metis e_def pathfinish_def pathstart_def)
  then have  $\forall e > 0. \exists d > 0. \forall x' \in \{0..1\}. \text{dist } x' 0 < d \longrightarrow \text{dist } (g2 \ x') (g2 \ 0) < e$ 
    using  $\langle \text{path } g2 \rangle$  atLeastAtMost_iff zero_le_one unfolding path_def continuous_on_iff
    by blast
  then obtain d1 where d1 > 0
    and d1:  $\bigwedge x'. \llbracket x' \in \{0..1\}; \text{norm } x' < d1 \rrbracket \Longrightarrow \text{dist } (g2 \ x') (g2 \ 0) < e/2$ 
    by (metis  $\langle 0 < e \rangle$  half_gt_zero_iff norm_conv_dist)
  obtain d2 where d2 > 0
    and d2:  $\bigwedge x'. \llbracket x' \in \{0..1\}; \text{dist } x' (1/2) < d2 \rrbracket \Longrightarrow \text{dist } ((g1 \ +++ \ g2) \ x') (g1 \ 1) < e/2$ 
    using assms(1)  $\langle e > 0 \rangle$  unfolding path_def continuous_on_iff
    apply (drule_tac x=1/2 in bspec, simp)
    apply (drule_tac x=e/2 in spec, force simp: joinpaths_def)
    done
  have int01_1:  $\min (1/2) (\min d1 d2) / 2 \in \{0..1\}$ 
    using  $\langle d1 > 0 \rangle \langle d2 > 0 \rangle$  by (simp add: min_def)
  have dist1:  $\text{norm } (\min (1 / 2) (\min d1 d2) / 2) < d1$ 
    using  $\langle d1 > 0 \rangle \langle d2 > 0 \rangle$  by (simp add: min_def dist_norm)
  have int01_2:  $1/2 + \min (1/2) (\min d1 d2) / 4 \in \{0..1\}$ 
    using  $\langle d1 > 0 \rangle \langle d2 > 0 \rangle$  by (simp add: min_def)
  have dist2:  $\text{dist } (1 / 2 + \min (1 / 2) (\min d1 d2) / 4) (1 / 2) < d2$ 
    using  $\langle d1 > 0 \rangle \langle d2 > 0 \rangle$  by (simp add: min_def dist_norm)
  have [simp]:  $\neg \min (1 / 2) (\min d1 d2) \leq 0$ 
    using  $\langle d1 > 0 \rangle \langle d2 > 0 \rangle$  by (simp add: min_def)
  have dist (g2 (min (1 / 2) (min d1 d2) / 2)) (g1 1) < e/2
    dist (g2 (min (1 / 2) (min d1 d2) / 2)) (g2 0) < e/2
    using d1 [OF int01_1 dist1] d2 [OF int01_2 dist2] by (simp_all add: joinpaths_def)
  then have dist (g1 1) (g2 0) < e/2 + e/2
    using dist_triangle_half_r e_def by blast
  then show False

```

by (simp add: e_def [symmetric])
qed

lemma path_join_eq [simp]:
fixes $g1 :: \text{real} \Rightarrow 'a::\text{metric\_space}$
assumes path $g1$ path $g2$
shows $\text{path}(g1 +++ g2) \longleftrightarrow \text{pathfinish } g1 = \text{pathstart } g2$
using assms by (metis path_join_path_ends path_join_imp)

lemma simple_path_joinE:
assumes simple_path($g1 +++ g2$) and $\text{pathfinish } g1 = \text{pathstart } g2$
obtains arc $g1$ arc $g2$
 $\text{path\_image } g1 \cap \text{path\_image } g2 \subseteq \{\text{pathstart } g1, \text{pathstart } g2\}$

proof -
have *: $\bigwedge x y. \llbracket 0 \leq x; x \leq 1; 0 \leq y; y \leq 1; (g1 +++ g2) x = (g1 +++ g2) y \rrbracket$
 $\implies x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0$
using assms by (simp add: simple_path_def)
have path $g1$
using assms path_join simple_path_imp_path by blast
moreover have inj_on $g1$ $\{0..1\}$
proof (clarsimp simp: inj_on_def)
fix $x y$
assume $g1 x = g1 y$ $0 \leq x \leq 1$ $0 \leq y \leq 1$
then show $x = y$
using * [of $x/2$ $y/2$] by (simp add: joinpaths_def split_ifs)

qed

ultimately have arc $g1$
using assms by (simp add: arc_def)
have [simp]: $g2 0 = g1 1$
using assms by (metis pathfinish_def pathstart_def)
have path $g2$
using assms path_join simple_path_imp_path by blast
moreover have inj_on $g2$ $\{0..1\}$
proof (clarsimp simp: inj_on_def)
fix $x y$
assume $g2 x = g2 y$ $0 \leq x \leq 1$ $0 \leq y \leq 1$
then show $x = y$
using * [of $(x + 1) / 2$ $(y + 1) / 2$]
by (force simp: joinpaths_def split_ifs field_split_simps)

qed

ultimately have arc $g2$
using assms by (simp add: arc_def)
have $g2 y = g1 0 \vee g2 y = g1 1$
if $g1 x = g2 y$ $0 \leq x \leq 1$ $0 \leq y \leq 1$ for $x y$
using * [of $x / 2$ $(y + 1) / 2$] that
by (auto simp: joinpaths_def split_ifs field_split_simps)
then have $\text{path\_image } g1 \cap \text{path\_image } g2 \subseteq \{\text{pathstart } g1, \text{pathstart } g2\}$
by (fastforce simp: pathstart_def pathfinish_def path_image_def)
with $\langle \text{arc } g1 \rangle \langle \text{arc } g2 \rangle$ show ?thesis using that by blast

qed

lemma *simple_path_join_loop_eq*:
assumes *pathfinish* *g2* = *pathstart* *g1* *pathfinish* *g1* = *pathstart* *g2*
shows *simple_path*(*g1* +++ *g2*) $\longleftrightarrow$
 $\text{arc } g1 \wedge \text{arc } g2 \wedge \text{path\_image } g1 \cap \text{path\_image } g2 \subseteq \{\text{pathstart } g1,$
 $\text{pathstart } g2\}$
by (*metis* *assms* *simple_path_joinE* *simple_path_join_loop*)

lemma *arc_join_eq*:
assumes *pathfinish* *g1* = *pathstart* *g2*
shows *arc*(*g1* +++ *g2*) $\longleftrightarrow$
 $\text{arc } g1 \wedge \text{arc } g2 \wedge \text{path\_image } g1 \cap \text{path\_image } g2 \subseteq \{\text{pathstart } g2\}$
(is ?lhs = ?rhs)

proof

assume *?lhs*
then have *simple_path*(*g1* +++ *g2*) **by** (*rule* *arc_imp_simple_path*)
then have *: $\bigwedge x y. \llbracket 0 \leq x; x \leq 1; 0 \leq y; y \leq 1; (g1 \text{ +++ } g2) \ x = (g1 \text{ +++ } g2) \ y \rrbracket$
 $\implies x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0$
using *assms* **by** (*simp* *add*: *simple_path_def*)
have *False* **if** *g1* 0 = *g2* *u* 0 $\leq u$ *u* ≤ 1 **for** *u*
using * [*of* 0 (*u* + 1) / 2] **that** *assms* *arc_distinct_ends* [*OF* $\langle ?lhs \rangle$]
by (*auto* *simp*: *joinpaths_def* *pathstart_def* *pathfinish_def* *split_ifs* *field_split_simps*)
then have *n1*: *pathstart* *g1* $\notin$ *path_image* *g2*
unfolding *pathstart_def* *path_image_def*
using *atLeastAtMost_iff* **by** *blast*
show *?rhs* **using** $\langle ?lhs \rangle$
using $\langle \text{simple\_path } (g1 \text{ +++ } g2) \rangle$ *assms* *n1* *simple_path_joinE* **by** *auto*
next
assume *?rhs* **then show** *?lhs*
using *assms*
by (*fastforce* *simp*: *pathfinish_def* *pathstart_def* *intro!*: *arc_join*)
qed

lemma *arc_join_eq_alt*:
 $\text{pathfinish } g1 = \text{pathstart } g2$
 $\implies (\text{arc}(g1 \text{ +++ } g2) \longleftrightarrow$
 $\text{arc } g1 \wedge \text{arc } g2 \wedge$
 $\text{path\_image } g1 \cap \text{path\_image } g2 = \{\text{pathstart } g2\})$
using *pathfinish_in_path_image* **by** (*fastforce* *simp*: *arc_join_eq*)

5.5.8 The joining of paths is associative

lemma *path_assoc*:
 $\llbracket \text{pathfinish } p = \text{pathstart } q; \text{pathfinish } q = \text{pathstart } r \rrbracket$
 $\implies \text{path}(p \text{ +++ } (q \text{ +++ } r)) \longleftrightarrow \text{path}((p \text{ +++ } q) \text{ +++ } r)$
by *simp*

```

lemma simple_path_assoc:
  assumes pathfinish  $p = \text{pathstart } q$  pathfinish  $q = \text{pathstart } r$ 
  shows simple_path  $(p +++ (q +++ r)) \longleftrightarrow \text{simple\_path } ((p +++ q) +++ r)$ 
proof (cases pathstart  $p = \text{pathfinish } r$ )
  case True show ?thesis
  proof
    assume simple_path  $(p +++ q +++ r)$ 
    with assms True show simple_path  $((p +++ q) +++ r)$ 
    by (fastforce simp add: simple_path_join_loop_eq arc_join_eq path_image_join
      dest: arc_distinct_ends [of r])
  next
    assume 0: simple_path  $((p +++ q) +++ r)$ 
    with assms True have  $q: \text{pathfinish } r \notin \text{path\_image } q$ 
    using arc_distinct_ends
    by (fastforce simp add: simple_path_join_loop_eq arc_join_eq path_image_join)
    have pathstart  $r \notin \text{path\_image } p$ 
    using assms
    by (metis 0 IntI arc_distinct_ends arc_join_eq_alt empty_iff insert_iff
      pathfinish_in_path_image pathfinish_join simple_path_joinE)
    with assms 0  $q$  True show simple_path  $(p +++ q +++ r)$ 
    by (auto simp: simple_path_join_loop_eq arc_join_eq path_image_join
      dest!: subsetD [OF IntI])
  qed
next
  case False
  { fix  $x :: 'a$ 
    assume  $a: \text{path\_image } p \cap \text{path\_image } q \subseteq \{\text{pathstart } q\}$ 
       $(\text{path\_image } p \cup \text{path\_image } q) \cap \text{path\_image } r \subseteq \{\text{pathstart } r\}$ 
       $x \in \text{path\_image } p \implies x \in \text{path\_image } r$ 
    have pathstart  $r \in \text{path\_image } q$ 
    by (metis assms(2) pathfinish_in_path_image)
    with  $a$  have  $x = \text{pathstart } q$ 
    by blast
  }
  with False assms show ?thesis
  by (auto simp: simple_path_eq_arc simple_path_join_loop_eq arc_join_eq
    path_image_join)
qed

```

```

lemma arc_assoc:
   $\llbracket \text{pathfinish } p = \text{pathstart } q; \text{pathfinish } q = \text{pathstart } r \rrbracket$ 
   $\implies \text{arc}(p +++ (q +++ r)) \longleftrightarrow \text{arc}((p +++ q) +++ r)$ 
by (simp add: arc_simple_path simple_path_assoc)

```

Symmetry and loops

```

lemma path_sym:
   $\llbracket \text{pathfinish } p = \text{pathstart } q; \text{pathfinish } q = \text{pathstart } p \rrbracket \implies \text{path}(p +++ q) \longleftrightarrow$ 

```

path(*q* +++ *p*)
 by *auto*

lemma *simple_path_sym*:

$\llbracket \text{pathfinish } p = \text{pathstart } q; \text{pathfinish } q = \text{pathstart } p \rrbracket$
 $\implies \text{simple\_path}(p +++ q) \longleftrightarrow \text{simple\_path}(q +++ p)$

by (*metis* (*full_types*) *inf_commute insert_commute simple_path_joinE simple_path_join_loop*)

lemma *path_image_sym*:

$\llbracket \text{pathfinish } p = \text{pathstart } q; \text{pathfinish } q = \text{pathstart } p \rrbracket$
 $\implies \text{path\_image}(p +++ q) = \text{path\_image}(q +++ p)$

by (*simp add: path_image_join sup_commute*)

5.5.9 Subpath

definition *subpath* :: *real* $\Rightarrow$ *real* $\Rightarrow$ (*real* $\Rightarrow$ 'a) $\Rightarrow$ *real* $\Rightarrow$ 'a::*real_normed_vector*
 where *subpath* *a b g* $\equiv \lambda x. g((b - a) * x + a)$

lemma *path_image_subpath_gen*:

fixes *g* :: *real* $\Rightarrow$ 'a::*real_normed_vector*

shows *path_image*(*subpath* *u v g*) = *g* ‘ (*closed_segment* *u v*)

by (*auto simp add: closed_segment_real_eq path_image_def subpath_def*)

lemma *path_image_subpath*:

fixes *g* :: *real* $\Rightarrow$ 'a::*real_normed_vector*

shows *path_image*(*subpath* *u v g*) = (if *u* $\leq$ *v* then *g* ‘ {*u*..*v*} else *g* ‘ {*v*..*u*})

by (*simp add: path_image_subpath_gen closed_segment_eq_real_ivl*)

lemma *path_image_subpath_commute*:

fixes *g* :: *real* $\Rightarrow$ 'a::*real_normed_vector*

shows *path_image*(*subpath* *u v g*) = *path_image*(*subpath* *v u g*)

by (*simp add: path_image_subpath_gen closed_segment_eq_real_ivl*)

lemma *path_subpath* [*simp*]:

fixes *g* :: *real* $\Rightarrow$ 'a::*real_normed_vector*

assumes *path* *g u* $\in$ {0..*1*} *v* $\in$ {0..*1*}

shows *path*(*subpath* *u v g*)

proof –

have *continuous_on* {0..*1*} (*g* $\circ$ ($\lambda x. ((v - u) * x + u)$))

using *assms*

apply (*intro continuous_intros; simp add: image_affinity_atLeastAtMost* [**where** *c=u*])

apply (*auto simp: path_def continuous_on_subset*)

done

then show ?*thesis*

by (*simp add: path_def subpath_def*)

qed

lemma *pathstart_subpath* [*simp*]: *pathstart*(*subpath* *u v g*) = *g*(*u*)

```

  by (simp add: pathstart_def subpath_def)

lemma pathfinish_subpath [simp]: pathfinish(subpath u v g) = g(v)
  by (simp add: pathfinish_def subpath_def)

lemma subpath_trivial [simp]: subpath 0 1 g = g
  by (simp add: subpath_def)

lemma subpath_reversepath: subpath 1 0 g = reversepath g
  by (simp add: reversepath_def subpath_def)

lemma reversepath_subpath: reversepath(subpath u v g) = subpath v u g
  by (simp add: reversepath_def subpath_def algebra_simps)

lemma subpath_translation: subpath u v (( $\lambda x. a + x$ )  $\circ$  g) = ( $\lambda x. a + x$ )  $\circ$  subpath
u v g
  by (rule ext) (simp add: subpath_def)

lemma subpath_image: subpath u v (f  $\circ$  g) = f  $\circ$  subpath u v g
  by (rule ext) (simp add: subpath_def)

lemma affine_ineq:
  fixes x :: 'a::linordered_idom
  assumes x  $\leq$  1 v  $\leq$  u
  shows v + x * u  $\leq$  u + x * v
proof -
  have (1-x)*(u-v)  $\geq$  0
  using assms by auto
  then show ?thesis
  by (simp add: algebra_simps)
qed

lemma sum_le_prod1:
  fixes a::real shows  $\llbracket a \leq 1; b \leq 1 \rrbracket \implies a + b \leq 1 + a * b$ 
  by (metis add.commute affine_ineq mult.right_neutral)

lemma simple_path_subpath_eq:
  simple_path(subpath u v g)  $\longleftrightarrow$ 
  path(subpath u v g)  $\wedge$  u  $\neq$  v  $\wedge$ 
  ( $\forall x y. x \in \text{closed\_segment } u \ v \wedge y \in \text{closed\_segment } u \ v \wedge g \ x = g \ y$ 
 $\longrightarrow x = y \vee x = u \wedge y = v \vee x = v \wedge y = u$ )
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then have p: path ( $\lambda x. g ((v - u) * x + u)$ )
    and sim: ( $\bigwedge x y. \llbracket x \in \{0..1\}; y \in \{0..1\}; g ((v - u) * x + u) = g ((v - u) * y + u) \rrbracket$ 
 $\implies x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0$ )
  by (auto simp: simple_path_def subpath_def)

```

```

{ fix x y
  assume  $x \in \text{closed\_segment } u \ v \ y \in \text{closed\_segment } u \ v \ g \ x = g \ y$ 
  then have  $x = y \vee x = u \wedge y = v \vee x = v \wedge y = u$ 
    using sim [of  $(x-u)/(v-u) \ (y-u)/(v-u)$ ] p
  by (auto split: if_split_asm simp add: closed_segment_real_eq image_affinity_atLeastAtMost)
    (simp_all add: field_split_simps)
} moreover
have path(subpath u v g)  $\wedge u \neq v$ 
  using sim [of 1/3 2/3] p
  by (auto simp: subpath_def)
ultimately show ?rhs
  by metis
next
  assume ?rhs
  then
    have d1:  $\bigwedge x \ y. \llbracket g \ x = g \ y; u \leq x; x \leq v; u \leq y; y \leq v \rrbracket \implies x = y \vee x = u \wedge y = v \vee x = v \wedge y = u$ 
    and d2:  $\bigwedge x \ y. \llbracket g \ x = g \ y; v \leq x; x \leq u; v \leq y; y \leq u \rrbracket \implies x = y \vee x = u \wedge y = v \vee x = v \wedge y = u$ 
    and ne:  $u < v \vee v < u$ 
    and psp: path (subpath u v g)
    by (auto simp: closed_segment_real_eq image_affinity_atLeastAtMost)
  have [simp]:  $\bigwedge x. u + x * v = v + x * u \longleftrightarrow u=v \vee x=1$ 
    by algebra
  show ?lhs using psp ne
    unfolding simple_path_def subpath_def
    by (fastforce simp add: algebra_simps affine_ineq mult_left_mono crossprod-uct_eq dest: d1 d2)
qed

lemma arc_subpath_eq:
  arc(subpath u v g)  $\longleftrightarrow$  path(subpath u v g)  $\wedge u \neq v \wedge \text{inj\_on } g \ (\text{closed\_segment } u \ v)$ 
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then have p: path ( $\lambda x. g \ ((v - u) * x + u)$ )
    and sim: ( $\bigwedge x \ y. \llbracket x \in \{0..1\}; y \in \{0..1\}; g \ ((v - u) * x + u) = g \ ((v - u) * y + u) \rrbracket \implies x = y$ )
    by (auto simp: arc_def inj_on_def subpath_def)
  { fix x y
    assume  $x \in \text{closed\_segment } u \ v \ y \in \text{closed\_segment } u \ v \ g \ x = g \ y$ 
    then have  $x = y$ 
      using sim [of  $(x-u)/(v-u) \ (y-u)/(v-u)$ ] p
      by (cases v = u)
    (simp_all split: if_split_asm add: inj_on_def closed_segment_real_eq image_affinity_atLeastAtMost,
      simp add: field_simps)
  }

```



```

} moreover
have path(subpath u v g)  $\wedge$   $u \neq v$ 
  using sim [of 1/3 2/3] p
  by (auto simp: subpath_def)
ultimately show ?rhs
  unfolding inj_on_def
  by metis
next
assume ?rhs
then
have d1:  $\bigwedge x y. \llbracket g\ x = g\ y; u \leq x; x \leq v; u \leq y; y \leq v \rrbracket \implies x = y$ 
  and d2:  $\bigwedge x y. \llbracket g\ x = g\ y; v \leq x; x \leq u; v \leq y; y \leq u \rrbracket \implies x = y$ 
  and ne:  $u < v \vee v < u$ 
  and psp: path (subpath u v g)
  by (auto simp: inj_on_def closed_segment_real_eq image_affinity_atLeastAtMost)
show ?lhs using psp ne
  unfolding arc_def subpath_def inj_on_def
  by (auto simp: algebra_simps affine_ineq mult_left_mono crossproduct_eq
dest: d1 d2)
qed

```

```

lemma simple_path_subpath:
  assumes simple_path g  $u \in \{0..1\}$   $v \in \{0..1\}$   $u \neq v$ 
  shows simple_path(subpath u v g)
  using assms
  apply (simp add: simple_path_subpath_eq simple_path_imp_path)
  apply (simp add: simple_path_def closed_segment_real_eq image_affinity_atLeastAtMost,
fastforce)
  done

```

```

lemma arc_simple_path_subpath:
   $\llbracket \text{simple\_path } g; u \in \{0..1\}; v \in \{0..1\}; g\ u \neq g\ v \rrbracket \implies \text{arc}(\text{subpath } u\ v\ g)$ 
  by (force intro: simple_path_subpath simple_path_imp_arc)

```

```

lemma arc_subpath_arc:
   $\llbracket \text{arc } g; u \in \{0..1\}; v \in \{0..1\}; u \neq v \rrbracket \implies \text{arc}(\text{subpath } u\ v\ g)$ 
  by (meson arc_def arc_imp_simple_path arc_simple_path_subpath inj_onD)

```

```

lemma arc_simple_path_subpath_interior:
   $\llbracket \text{simple\_path } g; u \in \{0..1\}; v \in \{0..1\}; u \neq v; |u-v| < 1 \rrbracket \implies \text{arc}(\text{subpath } u\ v\ g)$ 
  by (force simp: simple_path_def intro: arc_simple_path_subpath)

```

```

lemma path_image_subpath_subset:
   $\llbracket u \in \{0..1\}; v \in \{0..1\} \rrbracket \implies \text{path\_image}(\text{subpath } u\ v\ g) \subseteq \text{path\_image } g$ 
  by (metis atLeastAtMost_iff atLeastatMost_subset_iff path_image_def path_image_subpath
subset_image_iff)

```

```

lemma join_subpaths_middle: subpath (0) ((1 / 2)) p +++ subpath ((1 / 2)) 1
p = p
by (rule ext) (simp add: joinpaths_def subpath_def field_split_simps)

```

5.5.10 There is a subpath to the frontier

```

lemma subpath_to_frontier_explicit:
  fixes S :: 'a::metric_space set
  assumes g: path g and pathfinish g ∉ S
  obtains u where 0 ≤ u u ≤ 1
    ∧ x. 0 ≤ x ∧ x < u ⟹ g x ∈ interior S
    (g u ∉ interior S) (u = 0 ∨ g u ∈ closure S)

proof -
  have gcon: continuous_on {0..1} g
  using g by (simp add: path_def)
  moreover have bounded ({u. g u ∈ closure (- S)} ∩ {0..1})
  using compact_eq_bounded_closed by fastforce
  ultimately have com: compact ({0..1} ∩ {u. g u ∈ closure (- S)})
  using closed_vimage_Int
  by (metis (full_types) Int_commute closed_atLeastAtMost closed_closure com-
    pact_eq_bounded_closed vimage_def)
  have 1 ∈ {u. g u ∈ closure (- S)}
  using assms by (simp add: pathfinish_def closure_def)
  then have dis: {0..1} ∩ {u. g u ∈ closure (- S)} ≠ {}
  using atLeastAtMost_iff zero_le_one by blast
  then obtain u where 0 ≤ u u ≤ 1 and gu: g u ∈ closure (- S)
    and umin: ∧t. [0 ≤ t; t ≤ 1; g t ∈ closure (- S)] ⟹ u ≤ t
  using compact_attains_inf [OF com dis] by fastforce
  then have umin': ∧t. [0 ≤ t; t ≤ 1; t < u] ⟹ g t ∈ S
  using closure_def by fastforce
  have §: g u ∈ closure S if u ≠ 0
  proof -
    have u > 0 using that ⟨0 ≤ u⟩ by auto
    { fix e::real assume e > 0
      obtain d where d>0 and d: ∧x'. [x' ∈ {0..1}; dist x' u ≤ d] ⟹ dist (g
        x') (g u) < e
      using continuous_onE [OF gcon _ ⟨e > 0⟩] ⟨0 ≤ _⟩ ⟨_ ≤ 1⟩ atLeastAt-
        Most_iff by auto
      have *: dist (max 0 (u - d / 2)) u ≤ d
      using ⟨0 ≤ u⟩ ⟨u ≤ 1⟩ ⟨d > 0⟩ by (simp add: dist_real_def)
      have ∃y∈S. dist y (g u) < e
      using ⟨0 < u⟩ ⟨u ≤ 1⟩ ⟨d > 0⟩
      by (force intro: d [OF _ *] umin')
    }
    then show ?thesis
    by (simp add: frontier_def closure_approachable)
  qed
  show ?thesis
proof

```

```

    show  $\bigwedge x. 0 \leq x \wedge x < u \implies g\ x \in \text{interior } S$ 
      using  $\langle u \leq 1 \rangle$  interior_closure uin by fastforce
    show  $g\ u \notin \text{interior } S$ 
      by (simp add: gu interior_closure)
  qed (use  $\langle 0 \leq u \rangle \langle u \leq 1 \rangle$  § in auto)
qed

lemma subpath_to_frontier_strong:
  assumes  $g: \text{path } g$  and  $\text{pathfinish } g \notin S$ 
  obtains  $u$  where  $0 \leq u \leq 1$   $g\ u \notin \text{interior } S$ 
     $u = 0 \vee (\forall x. 0 \leq x \wedge x < 1 \longrightarrow \text{subpath } 0\ u\ g\ x \in \text{interior } S)$ 
 $\wedge g\ u \in \text{closure } S$ 
proof -
  obtain  $u$  where  $0 \leq u \leq 1$ 
    and  $gxin: \bigwedge x. 0 \leq x \wedge x < u \implies g\ x \in \text{interior } S$ 
    and  $gunot: (g\ u \notin \text{interior } S)$  and  $u0: (u = 0 \vee g\ u \in \text{closure } S)$ 
  using subpath_to_frontier_explicit [OF assms] by blast
  show ?thesis
  proof
    show  $g\ u \notin \text{interior } S$ 
      using gunot by blast
    qed (use  $\langle 0 \leq u \rangle \langle u \leq 1 \rangle$  u0 in  $\langle (\text{force simp: subpath\_def } gxin) + \rangle$ )
  qed

lemma subpath_to_frontier:
  assumes  $g: \text{path } g$  and  $g0: \text{pathstart } g \in \text{closure } S$  and  $g1: \text{pathfinish } g \notin S$ 
  obtains  $u$  where  $0 \leq u \leq 1$   $g\ u \in \text{frontier } S$   $\text{path\_image}(\text{subpath } 0\ u\ g) - \{g\ u\} \subseteq \text{interior } S$ 
proof -
  obtain  $u$  where  $0 \leq u \leq 1$ 
    and  $\text{notin}: g\ u \notin \text{interior } S$ 
    and  $\text{disj}: u = 0 \vee$ 
       $(\forall x. 0 \leq x \wedge x < 1 \longrightarrow \text{subpath } 0\ u\ g\ x \in \text{interior } S) \wedge g\ u \in$ 
 $\text{closure } S$ 
    (is  $\_ \vee ?P$ )
  using subpath_to_frontier_strong [OF g g1] by blast
  show ?thesis
  proof
    show  $g\ u \in \text{frontier } S$ 
      by (metis DiffI disj frontier_def g0 notin pathstart_def)
    show  $\text{path\_image}(\text{subpath } 0\ u\ g) - \{g\ u\} \subseteq \text{interior } S$ 
      using disj
    proof
      assume  $u = 0$ 
      then show ?thesis
        by (simp add: path_image_subpath)
    next
      assume  $P: ?P$ 
      show ?thesis

```

```

proof (clarsimp simp add: path_image_subpath_gen)
  fix y
  assume y:  $y \in \text{closed\_segment } 0 \ u \ g \ y \notin \text{interior } S$ 
  with  $\langle 0 \leq u \rangle$  have  $0 \leq y \ y \leq u$ 
    by (auto simp: closed_segment_eq_real_ivl split: if_split_asm)
  then have  $y=u \vee \text{subpath } 0 \ u \ g \ (y/u) \in \text{interior } S$ 
    using P less_eq_real_def by force
  then show  $g \ y = g \ u$ 
    using y by (auto simp: subpath_def split: if_split_asm)
qed
qed
qed (use  $\langle 0 \leq u \rangle \ \langle u \leq 1 \rangle$  in auto)
qed

lemma exists_path_subpath_to_frontier:
  fixes  $S :: 'a::\text{real\_normed\_vector}$  set
  assumes  $\text{path } g \ \text{pathstart } g \in \text{closure } S \ \text{pathfinish } g \notin S$ 
  obtains  $h$  where  $\text{path } h \ \text{pathstart } h = \text{pathstart } g \ \text{path\_image } h \subseteq \text{path\_image } g$ 
     $\text{path\_image } h - \{\text{pathfinish } h\} \subseteq \text{interior } S$ 
     $\text{pathfinish } h \in \text{frontier } S$ 
proof -
  obtain  $u$  where  $0 \leq u \ u \leq 1 \ g \ u \in \text{frontier } S \ (\text{path\_image}(\text{subpath } 0 \ u \ g) - \{g \ u\}) \subseteq \text{interior } S$ 
    using subpath_to_frontier [OF assms] by blast
  show ?thesis
  proof
    show  $\text{path\_image}(\text{subpath } 0 \ u \ g) \subseteq \text{path\_image } g$ 
      by (simp add: path_image_subpath_subset u)
    show  $\text{pathstart}(\text{subpath } 0 \ u \ g) = \text{pathstart } g$ 
      by (metis pathstart_def pathstart_subpath)
    qed (use assms u in  $\langle \text{auto simp: path\_image\_subpath} \rangle$ )
  qed

lemma exists_path_subpath_to_frontier_closed:
  fixes  $S :: 'a::\text{real\_normed\_vector}$  set
  assumes  $S: \text{closed } S$  and  $g: \text{path } g$  and  $g0: \text{pathstart } g \in S$  and  $g1: \text{pathfinish } g \notin S$ 
  obtains  $h$  where  $\text{path } h \ \text{pathstart } h = \text{pathstart } g \ \text{path\_image } h \subseteq \text{path\_image } g$ 
     $\text{pathfinish } h \in \text{frontier } S$ 
proof -
  obtain  $h$  where  $\text{path } h \ \text{pathstart } h = \text{pathstart } g \ \text{path\_image } h \subseteq \text{path\_image } g$ 
     $\text{path\_image } h - \{\text{pathfinish } h\} \subseteq \text{interior } S$ 
     $\text{pathfinish } h \in \text{frontier } S$ 
    using exists_path_subpath_to_frontier [OF  $g \ g1$ ] closure_closed [OF  $S$ ]  $g0$ 
by auto
  show ?thesis

```

```

proof
  show  $\text{path\_image } h \subseteq \text{path\_image } g \cap S$ 
    using  $\text{assms } h \text{ interior\_subset } [\text{of } S]$  by ( $\text{auto simp: frontier\_def}$ )
  qed ( $\text{use } h \text{ in auto}$ )
qed

```

5.5.11 Shift Path to Start at Some Given Point

definition $\text{shiftpath} :: \text{real} \Rightarrow (\text{real} \Rightarrow 'a::\text{topological\_space}) \Rightarrow \text{real} \Rightarrow 'a$
where $\text{shiftpath } a f = (\lambda x. \text{if } (a + x) \leq 1 \text{ then } f(a + x) \text{ else } f(a + x - 1))$

lemma shiftpath_alt_def : $\text{shiftpath } a f = (\lambda x. \text{if } x \leq 1 - a \text{ then } f(a + x) \text{ else } f(a + x - 1))$
by ($\text{auto simp: shiftpath\_def}$)

lemma $\text{pathstart\_shiftpath}$: $a \leq 1 \implies \text{pathstart } (\text{shiftpath } a g) = g a$
unfolding $\text{pathstart\_def shiftpath\_def}$ **by** auto

lemma $\text{pathfinish\_shiftpath}$:
assumes $0 \leq a$
and $\text{pathfinish } g = \text{pathstart } g$
shows $\text{pathfinish } (\text{shiftpath } a g) = g a$
using assms
unfolding $\text{pathstart\_def pathfinish\_def shiftpath\_def}$
by auto

lemma $\text{endpoints\_shiftpath}$:
assumes $\text{pathfinish } g = \text{pathstart } g$
and $a \in \{0 \dots 1\}$
shows $\text{pathfinish } (\text{shiftpath } a g) = g a$
and $\text{pathstart } (\text{shiftpath } a g) = g a$
using assms
by ($\text{auto intro!: pathfinish\_shiftpath pathstart\_shiftpath}$)

lemma closed_shiftpath :
assumes $\text{pathfinish } g = \text{pathstart } g$
and $a \in \{0..1\}$
shows $\text{pathfinish } (\text{shiftpath } a g) = \text{pathstart } (\text{shiftpath } a g)$
using $\text{endpoints\_shiftpath[OF assms]}$
by auto

lemma path_shiftpath :
assumes $\text{path } g$
and $\text{pathfinish } g = \text{pathstart } g$
and $a \in \{0..1\}$
shows $\text{path } (\text{shiftpath } a g)$

proof –
have $\ast: \{0 \dots 1\} = \{0 \dots 1 - a\} \cup \{1 - a \dots 1\}$
using $\text{assms}(3)$ **by** auto

```

have **:  $\bigwedge x. x + a = 1 \implies g (x + a - 1) = g (x + a)$ 
  using assms(2)[unfolded pathfinish_def pathstart_def]
  by auto
show ?thesis
  unfolding path_def shiftpath_def *
proof (rule continuous_on_closed_Un)
  have contg: continuous_on {0..1} g
    using ⟨path g⟩ path_def by blast
  show continuous_on {0..1-a} ( $\lambda x. \text{if } a + x \leq 1 \text{ then } g (a + x) \text{ else } g (a + x - 1)$ )
  proof (rule continuous_on_eq)
    show continuous_on {0..1-a} ( $g \circ (+) a$ )
    by (intro continuous_intros continuous_on_subset [OF contg]) (use ⟨a ∈ {0..1}⟩ in auto)
  qed auto
  show continuous_on {1-a..1} ( $\lambda x. \text{if } a + x \leq 1 \text{ then } g (a + x) \text{ else } g (a + x - 1)$ )
  proof (rule continuous_on_eq)
    show continuous_on {1-a..1} ( $g \circ (+) (a - 1)$ )
    by (intro continuous_intros continuous_on_subset [OF contg]) (use ⟨a ∈ {0..1}⟩ in auto)
  qed (auto simp: ** add commute add_diff_eq)
qed auto
qed

```

```

lemma shiftpath_shiftpath:
  assumes pathfinish g = pathstart g
  and a ∈ {0..1}
  and x ∈ {0..1}
  shows shiftpath (1 - a) (shiftpath a g) x = g x
  using assms
  unfolding pathfinish_def pathstart_def shiftpath_def
  by auto

```

```

lemma path_image_shiftpath:
  assumes a: a ∈ {0..1}
  and pathfinish g = pathstart g
  shows path_image (shiftpath a g) = path_image g
proof -
  { fix x
    assume g: g 1 = g 0 x ∈ {0..1::real} and gne:  $\bigwedge y. y \in \{0..1\} \cap \{x. \neg a + x \leq 1\} \implies g x \neq g (a + y - 1)$ 
    then have  $\exists y \in \{0..1\} \cap \{x. a + x \leq 1\}. g x = g (a + y)$ 
    proof (cases a ≤ x)
    case False
    then show ?thesis
      apply (rule_tac x=1 + x - a in bexI)
      using g gne[of 1 + x - a] a by (force simp: field_simps)+
    next

```

```

      case True
      then show ?thesis
        using g a by (rule_tac x=x - a in bexI) (auto simp: field_simps)
    qed
  }
  then show ?thesis
    using assms
    unfolding shiftpath_def path_image_def pathfinish_def pathstart_def
    by (auto simp: image_iff)
qed

lemma simple_path_shiftpath:
  assumes simple_path g pathfinish g = pathstart g and a:  $0 \leq a \wedge a \leq 1$ 
  shows simple_path (shiftpath a g)
  unfolding simple_path_def
proof (intro conjI impI ballI)
  show path (shiftpath a g)
    by (simp add: assms path_shiftpath simple_path_imp_path)
  have *:  $\bigwedge x y. \llbracket g\ x = g\ y; x \in \{0..1\}; y \in \{0..1\} \rrbracket \implies x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0$ 
  using assms by (simp add: simple_path_def)
  show  $x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0$ 
    if  $x \in \{0..1\}$   $y \in \{0..1\}$  shiftpath a g  $x = \text{shiftpath } a\ g\ y$  for  $x\ y$ 
    using that a unfolding shiftpath_def
    by (force split: if_split_asm dest!: *)
qed

```

5.5.12 Straight-Line Paths

definition *linepath* :: $'a::\text{real\_normed\_vector} \Rightarrow 'a \Rightarrow \text{real} \Rightarrow 'a$
 where *linepath* $a\ b = (\lambda x. (1 - x) *_R a + x *_R b)$

lemma *pathstart_linepath*[simp]: *pathstart* (*linepath* $a\ b$) = a
 unfolding *pathstart_def* *linepath_def*
 by *auto*

lemma *pathfinish_linepath*[simp]: *pathfinish* (*linepath* $a\ b$) = b
 unfolding *pathfinish_def* *linepath_def*
 by *auto*

lemma *linepath_inner*: *linepath* $a\ b\ x \cdot v = \text{linepath } (a \cdot v) (b \cdot v)\ x$
 by (simp add: *linepath_def* *algebra_simps*)

lemma *Re_linepath'*: *Re* (*linepath* $a\ b\ x$) = *linepath* (*Re* a) (*Re* b) x
 by (simp add: *linepath_def*)

lemma *Im_linepath'*: *Im* (*linepath* $a\ b\ x$) = *linepath* (*Im* a) (*Im* b) x
 by (simp add: *linepath_def*)

lemma *linepath_0'*: *linepath* *a b 0* = *a*
by (*simp add: linepath_def*)

lemma *linepath_1'*: *linepath* *a b 1* = *b*
by (*simp add: linepath_def*)

lemma *continuous_linepath_at*[*intro*]: *continuous* (*at x*) (*linepath a b*)
unfolding *linepath_def*
by (*intro continuous_intros*)

lemma *continuous_on_linepath* [*intro, continuous_intros*]: *continuous_on s* (*linepath a b*)
using *continuous_linepath_at*
by (*auto intro!: continuous_at_imp_continuous_on*)

lemma *path_linepath*[*iff*]: *path* (*linepath a b*)
unfolding *path_def*
by (*rule continuous_on_linepath*)

lemma *path_image_linepath*[*simp*]: *path_image* (*linepath a b*) = *closed_segment a b*
unfolding *path_image_def segment_linepath_def*
by *auto*

lemma *reversepath_linepath*[*simp*]: *reversepath* (*linepath a b*) = *linepath b a*
unfolding *reversepath_def linepath_def*
by *auto*

lemma *linepath_0* [*simp*]: *linepath 0 b x* = *x *_R b*
by (*simp add: linepath_def*)

lemma *linepath_cnj*: *cnj* (*linepath a b x*) = *linepath (cnj a) (cnj b) x*
by (*simp add: linepath_def*)

lemma *arc_linepath*:
assumes *a ≠ b* **shows** [*simp*]: *arc* (*linepath a b*)
proof –
{
 fix *x y* :: *real*
 assume *x *_R b + y *_R a = x *_R a + y *_R b*
 then have *(x – y) *_R a = (x – y) *_R b*
 by (*simp add: algebra_simps*)
 with *assms* **have** *x = y*
 by *simp*
}
then show *?thesis*
unfolding *arc_def inj_on_def*
by (*fastforce simp: algebra_simps linepath_def*)
qed


```

lemma simple_path_linepath[intro]:  $a \neq b \implies \text{simple\_path } (\text{linepath } a \ b)$ 
  by (simp add: arc_imp_simple_path)

lemma linepath_trivial [simp]:  $\text{linepath } a \ a \ x = a$ 
  by (simp add: linepath_def real_vector.scale_left_diff_distrib)

lemma linepath_refl:  $\text{linepath } a \ a = (\lambda x. \ a)$ 
  by auto

lemma subpath_refl:  $\text{subpath } a \ a \ g = \text{linepath } (g \ a) \ (g \ a)$ 
  by (simp add: subpath_def linepath_def algebra_simps)

lemma linepath_of_real:  $(\text{linepath } (\text{of\_real } a) \ (\text{of\_real } b) \ x) = \text{of\_real } ((1 - x) * a + x * b)$ 
  by (simp add: scaleR_conv_of_real linepath_def)

lemma of_real_linepath:  $\text{of\_real } (\text{linepath } a \ b \ x) = \text{linepath } (\text{of\_real } a) \ (\text{of\_real } b) \ x$ 
  by (metis linepath_of_real mult.right_neutral of_real_def real_scaleR_def)

lemma inj_on_linepath:
  assumes  $a \neq b$  shows  $\text{inj\_on } (\text{linepath } a \ b) \ \{0..1\}$ 
proof (clarsimp simp: inj_on_def linepath_def)
  fix  $x \ y$ 
  assume  $(1 - x) *_{\mathbb{R}} a + x *_{\mathbb{R}} b = (1 - y) *_{\mathbb{R}} a + y *_{\mathbb{R}} b \ 0 \leq x \leq 1 \ 0 \leq y \leq 1$ 
  then have  $x *_{\mathbb{R}} (a - b) = y *_{\mathbb{R}} (a - b)$ 
  by (auto simp: algebra_simps)
  then show  $x = y$ 
  using assms by auto
qed

lemma linepath_le_1:
  fixes  $a::'a::\text{linordered\_idom}$  shows  $\llbracket a \leq 1; b \leq 1; 0 \leq u; u \leq 1 \rrbracket \implies (1 - u) * a + u * b \leq 1$ 
  using mult_left_le [of a 1 - u] mult_left_le [of b u] by auto

lemma linepath_in_path:
  shows  $x \in \{0..1\} \implies \text{linepath } a \ b \ x \in \text{closed\_segment } a \ b$ 
  by (auto simp: segment linepath_def)

lemma linepath_image_01:  $\text{linepath } a \ b \ ` \{0..1\} = \text{closed\_segment } a \ b$ 
  by (auto simp: segment linepath_def)

lemma linepath_in_convex_hull:
  fixes  $x::\text{real}$ 
  assumes  $a: a \in \text{convex\_hull } S$ 
  and  $b: b \in \text{convex\_hull } S$ 

```

```

    and  $x: 0 \leq x \leq 1$ 
  shows  $\text{linepath } a \ b \ x \in \text{convex hull } S$ 
proof -
  have  $\text{linepath } a \ b \ x \in \text{closed\_segment } a \ b$ 
    using  $x$  by (auto simp flip:  $\text{linepath\_image\_01}$ )
  then show ?thesis
    using  $a \ b \text{ convex\_contains\_segment}$  by blast
qed

lemma  $\text{Re\_linepath}$ :  $\text{Re}(\text{linepath } (\text{of\_real } a) (\text{of\_real } b) \ x) = (1 - x) * a + x * b$ 
  by (simp add:  $\text{linepath\_def}$ )

lemma  $\text{Im\_linepath}$ :  $\text{Im}(\text{linepath } (\text{of\_real } a) (\text{of\_real } b) \ x) = 0$ 
  by (simp add:  $\text{linepath\_def}$ )

lemma  $\text{bounded\_linear\_linepath}$ :
  assumes  $\text{bounded\_linear } f$ 
  shows  $f (\text{linepath } a \ b \ x) = \text{linepath } (f \ a) \ (f \ b) \ x$ 
proof -
  interpret  $f$ :  $\text{bounded\_linear } f$  by fact
  show ?thesis by (simp add:  $\text{linepath\_def } f.\text{add } f.\text{scale}$ )
qed

lemma  $\text{bounded\_linear\_linepath'}$ :
  assumes  $\text{bounded\_linear } f$ 
  shows  $f \circ \text{linepath } a \ b = \text{linepath } (f \ a) \ (f \ b)$ 
  using  $\text{bounded\_linear\_linepath} [OF \ \text{assms}]$  by (simp add:  $\text{fun\_eq\_iff}$ )

lemma  $\text{linepath\_cnj'}$ :  $\text{cnj} \circ \text{linepath } a \ b = \text{linepath } (\text{cnj } a) \ (\text{cnj } b)$ 
  by (simp add:  $\text{linepath\_def } \text{fun\_eq\_iff}$ )

lemma  $\text{differentiable\_linepath}$  [intro]:  $\text{linepath } a \ b$  differentiable at  $x$  within  $A$ 
  by (auto simp:  $\text{linepath\_def}$ )

lemma  $\text{has\_vector\_derivative\_linepath\_within}$ :
  ( $\text{linepath } a \ b$  has_vector_derivative  $(b - a)$ ) (at  $x$  within  $S$ )
  by (force intro:  $\text{derivative\_eq\_intros}$  simp add:  $\text{linepath\_def}$  has_vector_derivative_def
    algebra_simps)

```

5.5.13 Segments via convex hulls

```

lemma  $\text{segments\_subset\_convex\_hull}$ :
   $\text{closed\_segment } a \ b \subseteq (\text{convex hull } \{a, b, c\})$ 
   $\text{closed\_segment } a \ c \subseteq (\text{convex hull } \{a, b, c\})$ 
   $\text{closed\_segment } b \ c \subseteq (\text{convex hull } \{a, b, c\})$ 
   $\text{closed\_segment } b \ a \subseteq (\text{convex hull } \{a, b, c\})$ 
   $\text{closed\_segment } c \ a \subseteq (\text{convex hull } \{a, b, c\})$ 
   $\text{closed\_segment } c \ b \subseteq (\text{convex hull } \{a, b, c\})$ 
by (auto simp:  $\text{segment\_convex\_hull } \text{linepath\_of\_real}$  elim!:  $\text{rev\_subsetD} [OF \ \_$ 

```

hull_mono])

lemma *midpoints_in_convex_hull*:
assumes $x \in \text{convex\_hull } s$ $y \in \text{convex\_hull } s$
shows $\text{midpoint } x \ y \in \text{convex\_hull } s$
proof –
have $(1 - \text{inverse}(2)) * x + \text{inverse}(2) * y \in \text{convex\_hull } s$
by (*rule* *convexD_alt*) (*use* *assms* **in** *auto*)
then show *?thesis*
by (*simp* *add*: *midpoint_def* *algebra_simps*)
qed

lemma *not_in_interior_convex_hull_3*:
fixes $a :: \text{complex}$
shows $a \notin \text{interior}(\text{convex\_hull } \{a, b, c\})$
 $b \notin \text{interior}(\text{convex\_hull } \{a, b, c\})$
 $c \notin \text{interior}(\text{convex\_hull } \{a, b, c\})$
by (*auto* *simp*: *card_insert_le_m1* *not_in_interior_convex_hull*)

lemma *midpoint_in_closed_segment* [*simp*]: $\text{midpoint } a \ b \in \text{closed\_segment } a \ b$
using *midpoints_in_convex_hull* *segment_convex_hull* **by** *blast*

lemma *midpoint_in_open_segment* [*simp*]: $\text{midpoint } a \ b \in \text{open\_segment } a \ b \longleftrightarrow a \neq b$
by (*simp* *add*: *open_segment_def*)

lemma *continuous_IVT_local_extremum*:
fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow \text{real}$
assumes *contf*: *continuous_on* (*closed_segment* $a \ b$) f
and $a \neq b$ $f \ a = f \ b$
obtains z **where** $z \in \text{open\_segment } a \ b$
 $(\forall w \in \text{closed\_segment } a \ b. (f \ w) \leq (f \ z)) \vee$
 $(\forall w \in \text{closed\_segment } a \ b. (f \ z) \leq (f \ w))$

proof –
obtain c **where** $c \in \text{closed\_segment } a \ b$ **and** $c: \bigwedge y. y \in \text{closed\_segment } a \ b \implies f \ y \leq f \ c$
using *continuous_attains_sup* [*of* *closed_segment* $a \ b$ f] *contf* **by** *auto*
obtain d **where** $d \in \text{closed\_segment } a \ b$ **and** $d: \bigwedge y. y \in \text{closed\_segment } a \ b \implies f \ d \leq f \ y$
using *continuous_attains_inf* [*of* *closed_segment* $a \ b$ f] *contf* **by** *auto*
show *?thesis*
proof (*cases* $c \in \text{open\_segment } a \ b \vee d \in \text{open\_segment } a \ b$)
case *True*
then show *?thesis*
using $c \ d$ **that** **by** *blast*
next
case *False*
then have $(c = a \vee c = b) \wedge (d = a \vee d = b)$
by (*simp* *add*: $\langle c \in \text{closed\_segment } a \ b \rangle \langle d \in \text{closed\_segment } a \ b \rangle \text{open\_segment\_def}$)

```

with ⟨a ≠ b⟩ ⟨f a = f b⟩ c d show ?thesis
by (rule_tac z = midpoint a b in that) (fastforce+)
qed
qed

```

An injective map into $\mathbb{R}$ is also an open map w.r.T. the universe, and conversely.

proposition *injective_eq_1d_open_map_UNIV*:

fixes $f :: \text{real} \Rightarrow \text{real}$

assumes *contf*: *continuous_on S f* **and** *S*: *is_interval S*

shows *inj_on f S* $\longleftrightarrow (\forall T. \text{open } T \wedge T \subseteq S \longrightarrow \text{open}(f \text{ ` } T))$

(**is** *?lhs = ?rhs*)

proof *safe*

fix T

assume *injf*: *?lhs* **and** *open T* **and** $T \subseteq S$

have $\exists U. \text{open } U \wedge f x \in U \wedge U \subseteq f \text{ ` } T$ **if** $x \in T$ **for** x

proof $-$

obtain δ **where** $\delta > 0$ **and** δ : *cball x δ ⊆ T*

using $\langle \text{open } T \rangle \langle x \in T \rangle$ *open_contains_cball_eq* **by** *blast*

show *?thesis*

proof (*intro exI conjI*)

have *closed_segment* $(x-\delta) (x+\delta) = \{x-\delta..x+\delta\}$

using $\langle 0 < \delta \rangle$ **by** (*auto simp: closed_segment_eq_real_ivl*)

also have $\dots \subseteq S$

using $\delta \text{ ` } T \subseteq S$ **by** (*auto simp: dist_norm subset_eq*)

finally have $f \text{ ` } (\text{open\_segment } (x-\delta) (x+\delta)) = \text{open\_segment } (f (x-\delta)) (f (x+\delta))$

using *continuous_injective_image_open_segment_1*

by (*metis continuous_on_subset [OF contf] inj_on_subset [OF injf]*)

then show *open* $(f \text{ ` } \{x-\delta < .. < x+\delta\})$

using $\langle 0 < \delta \rangle$ **by** (*simp add: open_segment_eq_real_ivl*)

show $f x \in f \text{ ` } \{x - \delta < .. < x + \delta\}$

by (*auto simp: δ > 0*)

show $f \text{ ` } \{x - \delta < .. < x + \delta\} \subseteq f \text{ ` } T$

using δ **by** (*auto simp: dist_norm subset_iff*)

qed

qed

with *open_subopen* **show** *open* $(f \text{ ` } T)$

by *blast*

next

assume *R*: *?rhs*

have *False* **if** *xy*: $x \in S \ y \in S$ **and** $f x = f y$ $x \neq y$ **for** $x \ y$

proof $-$

have *open* $(f \text{ ` } \text{open\_segment } x \ y)$

using *R*

by (*metis S convex_contains_open_segment is_interval_convex open_greaterThanLessThan open_segment_eq_real_ivl xy*)

moreover

have *continuous_on* $(\text{closed\_segment } x \ y) \ f$

```

  by (meson S closed_segment_subset contf continuous_on_subset is_interval_convex
that)
  then obtain  $\xi$  where  $\xi \in \text{open\_segment } x \ y$ 
    and  $\xi$ :  $(\forall w \in \text{closed\_segment } x \ y. (f \ w) \leq (f \ \xi)) \vee$ 
       $(\forall w \in \text{closed\_segment } x \ y. (f \ \xi) \leq (f \ w))$ 
    using continuous_IVT_local_extremum [of x y f]  $\langle f \ x = f \ y \rangle \langle x \neq y \rangle$  by blast
  ultimately obtain e where  $e > 0$  and  $e$ :  $\bigwedge u. \text{dist } u \ (f \ \xi) < e \implies u \in f \ `$ 
    open_segment x y
    using open_dist by (metis image_eqI)
  have fin:  $f \ \xi + (e/2) \in f \ ` \text{open\_segment } x \ y \ f \ \xi - (e/2) \in f \ ` \text{open\_segment}$ 
    x y
    using e [of f  $\xi + (e/2)$ ] e [of f  $\xi - (e/2)$ ]  $\langle e > 0 \rangle$  by (auto simp: dist_norm)
  show ?thesis
    using  $\xi \ \langle 0 < e \rangle$  fin open_closed_segment by fastforce
qed
then show ?lhs
  by (force simp: inj_on_def)
qed

```

5.5.14 Bounding a point away from a path

```

lemma not_on_path_ball:
  fixes g :: real  $\Rightarrow$  'a::heine_borel
  assumes path g
    and z:  $z \notin \text{path\_image } g$ 
  shows  $\exists e > 0. \text{ball } z \ e \cap \text{path\_image } g = \{\}$ 
proof -
  have closed (path_image g)
    by (simp add:  $\langle \text{path } g \rangle$  closed_path_image)
  then obtain a where  $a \in \text{path\_image } g \ \forall y \in \text{path\_image } g. \text{dist } z \ a \leq \text{dist } z \ y$ 
    by (auto intro: distance_attains_inf[OF path_image_nonempty, of g z])
  then show ?thesis
    by (rule_tac x=dist z a in exI) (use dist_commute z in auto)
qed

```

```

lemma not_on_path_cball:
  fixes g :: real  $\Rightarrow$  'a::heine_borel
  assumes path g
    and z  $\notin \text{path\_image } g$ 
  shows  $\exists e > 0. \text{cball } z \ e \cap (\text{path\_image } g) = \{\}$ 
proof -
  obtain e where  $\text{ball } z \ e \cap \text{path\_image } g = \{\}$   $e > 0$ 
    using not_on_path_ball[OF assms] by auto
  moreover have  $\text{cball } z \ (e/2) \subseteq \text{ball } z \ e$ 
    using  $\langle e > 0 \rangle$  by auto
  ultimately show ?thesis
    by (rule_tac x=e/2 in exI) auto
qed

```

5.5.15 Path component

Original formalization by Tom Hales

definition $\text{path\_component } S \ x \ y \equiv$
 $(\exists g. \text{path } g \wedge \text{path\_image } g \subseteq S \wedge \text{pathstart } g = x \wedge \text{pathfinish } g = y)$

abbreviation

$\text{path\_component\_set } S \ x \equiv \text{Collect } (\text{path\_component } S \ x)$

lemmas $\text{path\_defs} = \text{path\_def pathstart\_def pathfinish\_def path\_image\_def path\_component\_def}$

lemma $\text{path\_component\_mem}$:

assumes $\text{path\_component } S \ x \ y$
shows $x \in S$ **and** $y \in S$
using assms
unfolding path_defs
by auto

lemma $\text{path\_component\_refl}$:

assumes $x \in S$
shows $\text{path\_component } S \ x \ x$
using assms
unfolding path_defs
by $(\text{metis } (\text{full\_types}) \ \text{assms } \text{continuous\_on\_const } \text{image\_subset\_iff } \text{path\_image\_def})$

lemma $\text{path\_component\_refl\_eq}$: $\text{path\_component } S \ x \ x \longleftrightarrow x \in S$

by $(\text{auto intro!} : \text{path\_component\_mem } \text{path\_component\_refl})$

lemma $\text{path\_component\_sym}$: $\text{path\_component } S \ x \ y \Longrightarrow \text{path\_component } S \ y \ x$

unfolding $\text{path\_component\_def}$
by $(\text{metis } (\text{no\_types}) \ \text{path\_image\_reversepath } \text{path\_reversepath } \text{pathfinish\_reversepath } \text{pathstart\_reversepath})$

lemma $\text{path\_component\_trans}$:

assumes $\text{path\_component } S \ x \ y$ **and** $\text{path\_component } S \ y \ z$
shows $\text{path\_component } S \ x \ z$
using assms
unfolding $\text{path\_component\_def}$
by $(\text{metis } \text{path\_join } \text{pathfinish\_join } \text{pathstart\_join } \text{subset\_path\_image\_join})$

lemma $\text{path\_component\_of\_subset}$: $S \subseteq T \Longrightarrow \text{path\_component } S \ x \ y \Longrightarrow \text{path\_component } T \ x \ y$

unfolding $\text{path\_component\_def}$ **by** auto

lemma $\text{path\_component\_linepath}$:

fixes $S :: 'a :: \text{real\_normed\_vector\_set}$
shows $\text{closed\_segment } a \ b \subseteq S \Longrightarrow \text{path\_component } S \ a \ b$
unfolding $\text{path\_component\_def}$
by $(\text{rule\_tac } x = \text{linepath } a \ b \text{ in } \text{exI}, \text{ auto})$

Path components as sets

lemma *path_component_set*:

path_component_set S $x =$
 $\{y. (\exists g. \text{path } g \wedge \text{path\_image } g \subseteq S \wedge \text{pathstart } g = x \wedge \text{pathfinish } g = y)\}$
by (*auto simp: path_component_def*)

lemma *path_component_subset*: *path_component_set* S $x \subseteq S$

by (*auto simp: path_component_mem(2)*)

lemma *path_component_eq_empty*: *path_component_set* S $x = \{\}$ $\longleftrightarrow x \notin S$

using *path_component_mem path_component_refl_eq*
by *fastforce*

lemma *path_component_mono*:

$S \subseteq T \implies (\text{path\_component\_set } S \ x) \subseteq (\text{path\_component\_set } T \ x)$

by (*simp add: Collect_mono path_component_of_subset*)

lemma *path_component_eq*:

$y \in \text{path\_component\_set } S \ x \implies \text{path\_component\_set } S \ y = \text{path\_component\_set } S \ x$

by (*metis (no_types, lifting) Collect_cong mem_Collect_eq path_component_sym path_component_trans*)

5.5.16 Path connectedness of a space

definition *path_connected* $S \longleftrightarrow$

$(\forall x \in S. \forall y \in S. \exists g. \text{path } g \wedge \text{path\_image } g \subseteq S \wedge \text{pathstart } g = x \wedge \text{pathfinish } g = y)$

lemma *path_connectedin_iff_path_connected_real* [*simp*]:

path_connectedin euclideanreal $S \longleftrightarrow \text{path\_connected } S$

by (*simp add: path_connectedin path_connected_def path_defs*)

lemma *path_connected_component*: *path_connected* $S \longleftrightarrow (\forall x \in S. \forall y \in S. \text{path\_component } S \ x \ y)$

unfolding *path_connected_def path_component_def* **by** *auto*

lemma *path_connected_component_set*: *path_connected* $S \longleftrightarrow (\forall x \in S. \text{path\_component\_set } S \ x = S)$

unfolding *path_connected_component path_component_subset*

using *path_component_mem* **by** *blast*

lemma *path_component_maximal*:

$\llbracket x \in T; \text{path\_connected } T; T \subseteq S \rrbracket \implies T \subseteq (\text{path\_component\_set } S \ x)$

by (*metis path_component_mono path_connected_component_set*)

lemma *convex_imp_path_connected*:

fixes $S :: 'a::\text{real\_normed\_vector\_set}$

assumes *convex* S

```

shows path_connected S
unfolding path_connected_def
using assms convex_contains_segment by fastforce

```

```

lemma path_connected_UNIV [iff]: path_connected (UNIV :: 'a::real_normed_vector
set)
  by (simp add: convex_imp_path_connected)

```

```

lemma path_component_UNIV: path_component_set UNIV x = (UNIV :: 'a::real_normed_vector
set)
  using path_connected_component_set by auto

```

```

lemma path_connected_imp_connected:
  assumes path_connected S
  shows connected S
proof (rule connectedI)
  fix e1 e2
  assume as: open e1 open e2  $S \subseteq e1 \cup e2$   $e1 \cap e2 \cap S = \{\}$   $e1 \cap S \neq \{\}$   $e2 \cap S \neq \{\}$ 
  then obtain x1 x2 where obt:  $x1 \in e1 \cap S$   $x2 \in e2 \cap S$ 
  by auto
  then obtain g where g: path g path_image g  $\subseteq S$  pathstart g = x1 pathfinish g = x2
  using assms[unfolded path_connected_def, rule_format, of x1 x2] by auto
  have *: connected {0..1::real}
  by (auto intro!: convex_connected)
  have {0..1}  $\subseteq \{x \in \{0..1\}. g\ x \in e1\} \cup \{x \in \{0..1\}. g\ x \in e2\}$ 
  using as(3) g(2)[unfolded path_defs] by blast
  moreover have  $\{x \in \{0..1\}. g\ x \in e1\} \cap \{x \in \{0..1\}. g\ x \in e2\} = \{\}$ 
  using as(4) g(2)[unfolded path_defs]
  unfolding subset_eq
  by auto
  moreover have  $\{x \in \{0..1\}. g\ x \in e1\} \neq \{\} \wedge \{x \in \{0..1\}. g\ x \in e2\} \neq \{\}$ 
  using g(3,4)[unfolded path_defs]
  using obt
  by (simp add: ex_in_conv [symmetric], metis zero_le_one order_refl)
ultimately show False
  using *[unfolded connected_local_not_ex, rule_format,
    of {0..1}  $\cap g - 'e1$  {0..1}  $\cap g - 'e2$ ]
  using continuous_openin_preimage_gen[OF g(1)[unfolded path_def] as(1)]
  using continuous_openin_preimage_gen[OF g(1)[unfolded path_def] as(2)]
  by auto
qed

```

```

lemma open_path_component:
  fixes S :: 'a::real_normed_vector set
  assumes open S
  shows open (path_component_set S x)
  unfolding open_contains_ball

```



```

proof
  fix  $y$ 
  assume  $as: y \in \text{path\_component\_set } S \ x$ 
  then have  $y \in S$ 
  by (simp add: path_component_mem(2))
  then obtain  $e$  where  $e: e > 0 \ \text{ball } y \ e \subseteq S$ 
  using assms openE by blast
have  $\bigwedge u. \text{dist } y \ u < e \implies \text{path\_component } S \ x \ u$ 
  by (metis (full_types) as centre_in_ball convex_ball convex_imp_path_connected
e mem_Collect_eq mem_ball path_component_eq path_component_of_subset path_connected_component)
  then show  $\exists e > 0. \text{ball } y \ e \subseteq \text{path\_component\_set } S \ x$ 
  using  $\langle e > 0 \rangle$  by auto
qed

```

```

lemma open_non_path_component:
  fixes  $S :: 'a::\text{real\_normed\_vector\_set}$ 
  assumes open S
  shows open (S - path_component_set S x)
  unfolding open_contains_ball
proof
  fix  $y$ 
  assume  $y: y \in S - \text{path\_component\_set } S \ x$ 
  then obtain  $e$  where  $e: e > 0 \ \text{ball } y \ e \subseteq S$ 
  using assms openE by auto
  show  $\exists e > 0. \text{ball } y \ e \subseteq S - \text{path\_component\_set } S \ x$ 
  proof (intro exI conjI subsetI DiffI notI)
    show  $\bigwedge x. x \in \text{ball } y \ e \implies x \in S$ 
    using  $e$  by blast
    show False if  $z \in \text{ball } y \ e \ z \in \text{path\_component\_set } S \ x$  for  $z$ 
    proof -
      have  $y \in \text{path\_component\_set } S \ z$ 
      by (meson assms convex_ball convex_imp_path_connected e open_contains_ball_eq
open_path_component path_component_maximal that(1))
      then have  $y \in \text{path\_component\_set } S \ x$ 
      using path_component_eq that(2) by blast
      then show False
      using  $y$  by blast
    qed
  qed (use e in auto)
qed

```

```

lemma connected_open_path_connected:
  fixes  $S :: 'a::\text{real\_normed\_vector\_set}$ 
  assumes open S
  and connected S
  shows path_connected S
  unfolding path_connected_component_set
proof (rule, rule, rule path_component_subset, rule)
  fix  $x \ y$ 

```

```

assume  $x \in S$  and  $y \in S$ 
show  $y \in \text{path\_component\_set } S \ x$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  moreover have  $\text{path\_component\_set } S \ x \cap S \neq \{\}$ 
    using  $\langle x \in S \rangle \text{ path\_component\_eq\_empty path\_component\_subset[of } S \ x]$ 
    by auto
  ultimately
  show False
  using  $\langle y \in S \rangle \text{ open\_non\_path\_component[OF assms(1)] open\_path\_component[OF}$ 
assms(1)]
    using assms(2)[unfolded connected_def not_ex, rule_format,
of path\_component\_set } S \ x \ S - \text{path\_component\_set } S \ x]
    by auto
qed
qed

```

```

lemma path_connected_continuous_image:
  assumes contf: continuous_on } S f
  and path_connected } S
  shows path_connected (f ` } S)
  unfolding path_connected_def
proof (rule, rule)
  fix  $x' \ y'$ 
  assume  $x' \in f ` } S \ y' \in f ` } S$ 
  then obtain  $x \ y$  where  $x: x \in S$  and  $y: y \in S$  and  $x': x' = f \ x$  and  $y': y' = f \ y$ 
    by auto
  from  $x \ y$  obtain  $g$  where  $\text{path } g \wedge \text{path\_image } g \subseteq S \wedge \text{pathstart } g = x \wedge$ 
pathfinish } g = y
    using assms(2)[unfolded path_connected_def] by fast
  then show  $\exists g. \text{path } g \wedge \text{path\_image } g \subseteq f ` } S \wedge \text{pathstart } g = x' \wedge \text{pathfinish } g$ 
= y'
    unfolding } x' y' path_defs
    by (fastforce intro: continuous_on_compose continuous_on_subset[OF contf])
qed

```

```

lemma path_connected_translationI:
  fixes  $a :: 'a :: \text{topological\_group\_add}$ 
  assumes path_connected } S shows path_connected ((\x. a + x) ` } S)
  by (intro path_connected_continuous_image assms continuous_intros)

```

```

lemma path_connected_translation:
  fixes  $a :: 'a :: \text{topological\_group\_add}$ 
  shows path_connected ((\x. a + x) ` } S) = path_connected } S
proof -
  have  $\forall x \ y. (+) (x::'a) ` (+) (0 - x) ` y = y$ 
    by (simp add: image_image)
  then show ?thesis

```

```

    by (metis (no_types) path_connected_translationI)
qed

lemma path_connected_segment [simp]:
  fixes a :: 'a::real_normed_vector
  shows path_connected (closed_segment a b)
  by (simp add: convex_imp_path_connected)

lemma path_connected_open_segment [simp]:
  fixes a :: 'a::real_normed_vector
  shows path_connected (open_segment a b)
  by (simp add: convex_imp_path_connected)

lemma homeomorphic_path_connectedness:
  S homeomorphic T  $\implies$  path_connected S  $\longleftrightarrow$  path_connected T
  unfolding homeomorphic_def homeomorphism_def by (metis path_connected_continuous_image)

lemma path_connected_empty [simp]: path_connected {}
  unfolding path_connected_def by auto

lemma path_connected_singleton [simp]: path_connected {a}
  unfolding path_connected_def pathstart_def pathfinish_def path_image_def
  using path_def by fastforce

lemma path_connected_Un:
  assumes path_connected S
  and path_connected T
  and S  $\cap$  T  $\neq$  {}
  shows path_connected (S  $\cup$  T)
  unfolding path_connected_component
proof (intro ballI)
  fix x y
  assume x: x  $\in$  S  $\cup$  T and y: y  $\in$  S  $\cup$  T
  from assms obtain z where z: z  $\in$  S z  $\in$  T
  by auto
  show path_component (S  $\cup$  T) x y
  using x y
proof safe
  assume x  $\in$  S y  $\in$  S
  then show path_component (S  $\cup$  T) x y
  by (meson Un_upper1  $\langle$ path_connected S $\rangle$  path_component_of_subset path_connected_component)
next
  assume x  $\in$  S y  $\in$  T
  then show path_component (S  $\cup$  T) x y
  by (metis z assms(1-2) le_sup_iff order_refl path_component_of_subset
    path_component_trans path_connected_component)
next
  assume x  $\in$  T y  $\in$  S
  then show path_component (S  $\cup$  T) x y

```

```

    by (metis z assms(1-2) le_sup_iff order_refl path_component_of_subset
    path_component_trans path_connected_component)
  next
    assume  $x \in T$   $y \in T$ 
    then show  $\text{path\_component } (S \cup T) \ x \ y$ 
    by (metis Un_upper1 assms(2) path_component_of_subset path_connected_component
    sup_commute)
  qed
qed

```

lemma *path_connected_UNION*:

```

  assumes  $\bigwedge i. i \in A \implies \text{path\_connected } (S \ i)$ 
  and  $\bigwedge i. i \in A \implies z \in S \ i$ 
  shows  $\text{path\_connected } (\bigcup_{i \in A}. S \ i)$ 
  unfolding path_connected_component
proof clarify
  fix  $x \ i \ y \ j$ 
  assume *:  $i \in A \ x \in S \ i \ j \in A \ y \in S \ j$ 
  then have  $\text{path\_component } (S \ i) \ x \ z$  and  $\text{path\_component } (S \ j) \ z \ y$ 
    using assms by (simp_all add: path_connected_component)
  then have  $\text{path\_component } (\bigcup_{i \in A}. S \ i) \ x \ z$  and  $\text{path\_component } (\bigcup_{i \in A}. S \ i) \ z \ y$ 
    using *(1,3) by (auto elim!: path_component_of_subset [rotated])
  then show  $\text{path\_component } (\bigcup_{i \in A}. S \ i) \ x \ y$ 
    by (rule path_component_trans)
qed

```

lemma *path_component_path_image_pathstart*:

```

  assumes  $p$ :  $\text{path } p$  and  $x$ :  $x \in \text{path\_image } p$ 
  shows  $\text{path\_component } (\text{path\_image } p) \ (\text{pathstart } p) \ x$ 
proof -
  obtain  $y$  where  $x$ :  $x = p \ y$  and  $y$ :  $0 \leq y \leq 1$ 
    using  $x$  by (auto simp: path_image_def)
  show ?thesis
    unfolding path_component_def
  proof (intro exI conjI)
    have continuous_on ((* )  $y$  '  $\{0..1\}$ )  $p$ 
      by (simp add: continuous_on_path image_mult_atLeastAtMost_if  $p \ y$ )
    then have continuous_on  $\{0..1\}$  ( $p \circ ((* ) \ y)$ )
      using continuous_on_compose continuous_on_mult_const by blast
    then show  $\text{path } (\lambda u. p \ (y * u))$ 
      by (simp add: path_def)
    show  $\text{path\_image } (\lambda u. p \ (y * u)) \subseteq \text{path\_image } p$ 
      using  $y$  mult_le_one by (fastforce simp: path_image_def image_iff)
    qed (auto simp: pathstart_def pathfinish_def  $x$ )
  qed

```

lemma *path_connected_path_image*: $\text{path } p \implies \text{path\_connected}(\text{path\_image } p)$
 unfolding path_connected_component

by (meson path_component_path_image_pathstart path_component_sym path_component_trans)

lemma path_connected_path_component [simp]:

path_connected (path_component_set s x)

proof –

{ fix y z

assume pa: path_component s x y path_component s x z

then have pae: path_component_set s x = path_component_set s y

using path_component_eq by auto

have yz: path_component s y z

using pa path_component_sym path_component_trans by blast

then have $\exists g. \text{path } g \wedge \text{path\_image } g \subseteq \text{path\_component\_set } s \wedge \text{pathstart}$

$g = y \wedge \text{pathfinish } g = z$

apply (simp add: path_component_def)

by (metis pae path_component_maximal path_connected_path_image pathstart_in_path_image)

}

then show ?thesis

by (simp add: path_connected_def)

qed

lemma path_component: path_component S x y $\longleftrightarrow (\exists t. \text{path\_connected } t \wedge t \subseteq$

$S \wedge x \in t \wedge y \in t)$

apply (intro iffI)

apply (metis path_connected_path_image path_defs(5) pathfinish_in_path_image pathstart_in_path_image)

using path_component_of_subset path_connected_component by blast

lemma path_component_path_component [simp]:

path_component_set (path_component_set S x) x = path_component_set S x

proof (cases x $\in$ S)

case True show ?thesis

by (metis True mem_Collect_eq path_component_refl path_connected_component_set path_connected_path_component)

next

case False then show ?thesis

by (metis False empty_iff path_component_eq_empty)

qed

lemma path_component_subset_connected_component:

(path_component_set S x) $\subseteq$ (connected_component_set S x)

proof (cases x $\in$ S)

case True show ?thesis

by (simp add: True connected_component_maximal path_component_refl path_component_subset path_connected_imp_connected)

next

case False then show ?thesis

using path_component_eq_empty by auto

qed

5.5.17 Lemmas about path-connectedness

```

lemma path_connected_linear_image:
  fixes  $f :: 'a::\text{real\_normed\_vector} \Rightarrow 'b::\text{real\_normed\_vector}$ 
  assumes path_connected  $S$  bounded_linear  $f$ 
  shows path_connected( $f \, ' \, S$ )
by (auto simp: linear_continuous_on assms path_connected_continuous_image)

lemma is_interval_path_connected: is_interval  $S \implies \text{path\_connected } S$ 
  by (simp add: convex_imp_path_connected is_interval_convex)

lemma path_connected_Ioi[simp]: path_connected  $\{a<..\}$  for  $a :: \text{real}$ 
  by (simp add: convex_imp_path_connected)

lemma path_connected_Ici[simp]: path_connected  $\{a.. \}$  for  $a :: \text{real}$ 
  by (simp add: convex_imp_path_connected)

lemma path_connected_Iio[simp]: path_connected  $\{.. $a\}$  for  $a :: \text{real}$ 
  by (simp add: convex_imp_path_connected)

lemma path_connected_Iic[simp]: path_connected  $\{.. $a\}$  for  $a :: \text{real}$ 
  by (simp add: convex_imp_path_connected)

lemma path_connected_Ioo[simp]: path_connected  $\{a<.. $b\}$  for  $a \, b :: \text{real}$ 
  by (simp add: convex_imp_path_connected)

lemma path_connected_Ioc[simp]: path_connected  $\{a<.. $b\}$  for  $a \, b :: \text{real}$ 
  by (simp add: convex_imp_path_connected)

lemma path_connected_Ico[simp]: path_connected  $\{a.. $b\}$  for  $a \, b :: \text{real}$ 
  by (simp add: convex_imp_path_connected)

lemma path_connectedin_path_image:
  assumes pathin  $X \, g$  shows path_connectedin  $X \, (g \, ' \, (\{0..1\}))$ 
  unfolding pathin_def
proof (rule path_connectedin_continuous_map_image)
  show continuous_map (subtopology_euclideanreal  $\{0..1\}$ )  $X \, g$ 
    using assms pathin_def by blast
qed (auto simp: is_interval_1 is_interval_path_connected)

lemma path_connected_space_subconnected:
  path_connected_space  $X \longleftrightarrow$ 
  ( $\forall x \in \text{topspace } X. \forall y \in \text{topspace } X. \exists S. \text{path\_connectedin } X \, S \wedge x \in S \wedge y \in S$ )
  by (metis path_connectedin_path_connectedin_topspace path_connected_space_def)

lemma connectedin_path_image: pathin  $X \, g \implies \text{connectedin } X \, (g \, ' \, (\{0..1\}))$ 
  by (simp add: path_connectedin_imp_connectedin path_connectedin_path_image)$$$$$ 
```

```

lemma compactin_path_image: pathin  $X$   $g \implies$  compactin  $X$  ( $g \text{ ' } (\{0..1\})$ )
  unfolding pathin_def
  by (rule image_compactin [of top_of_set  $\{0..1\}$ ]) auto

lemma linear_homeomorphism_image:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes linear  $f$  inj  $f$ 
  obtains  $g$  where homeomorphism ( $f \text{ ' } S$ )  $S$   $g$   $f$ 
proof –
  obtain  $g$  where linear  $g$   $g \circ f = \text{id}$ 
    using assms linear_injective_left_inverse by blast
  then have homeomorphism ( $f \text{ ' } S$ )  $S$   $g$   $f$ 
    using assms unfolding homeomorphism_def
    by (auto simp: eq_id_iff [symmetric] image_comp linear_conv_bounded_linear
linear_continuous_on)
  then show thesis ..
qed

lemma linear_homeomorphic_image:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes linear  $f$  inj  $f$ 
  shows  $S$  homeomorphic  $f \text{ ' } S$ 
by (meson homeomorphic_def homeomorphic_sym linear_homeomorphism_image
[OF assms])

lemma path_connected_Times:
  assumes path_connected  $s$  path_connected  $t$ 
  shows path_connected ( $s \times t$ )
proof (simp add: path_connected_def Sigma_def, clarify)
  fix  $x1\ y1\ x2\ y2$ 
  assume  $x1 \in s\ y1 \in t\ x2 \in s\ y2 \in t$ 
  obtain  $g$  where path  $g$  and  $g$ : path_image  $g \subseteq s$  and  $gs$ : pathstart  $g = x1$  and
 $gf$ : pathfinish  $g = x2$ 
    using  $\langle x1 \in s \rangle \langle x2 \in s \rangle$  assms by (force simp: path_connected_def)
  obtain  $h$  where path  $h$  and  $h$ : path_image  $h \subseteq t$  and  $hs$ : pathstart  $h = y1$  and
 $hf$ : pathfinish  $h = y2$ 
    using  $\langle y1 \in t \rangle \langle y2 \in t \rangle$  assms by (force simp: path_connected_def)
  have path ( $\lambda z. (x1, h\ z)$ )
    using  $\langle \text{path } h \rangle$ 
    unfolding path_def
    by (intro continuous_intros continuous_on_compose2 [where  $g = \text{Pair } \_$ ];
force)
  moreover have path ( $\lambda z. (g\ z, y2)$ )
    using  $\langle \text{path } g \rangle$ 
    unfolding path_def
    by (intro continuous_intros continuous_on_compose2 [where  $g = \text{Pair } \_$ ];
force)
  ultimately have 1: path ( $((\lambda z. (x1, h\ z)) +++ (\lambda z. (g\ z, y2)))$ )
    by (metis  $hf\ gs$  path_join_imp pathstart_def pathfinish_def)

```

```

have path_image ((λz. (x1, h z)) +++ (λz. (g z, y2))) ⊆ path_image (λz. (x1,
h z)) ∪ path_image (λz. (g z, y2))
by (rule Path_Connected.path_image_join_subset)
also have ... ⊆ (⋃ x∈s. ⋃ x1∈t. {(x, x1)})
using g h ⟨x1 ∈ s⟩ ⟨y2 ∈ t⟩ by (force simp: path_image_def)
finally have 2: path_image ((λz. (x1, h z)) +++ (λz. (g z, y2))) ⊆ (⋃ x∈s.
⋃ x1∈t. {(x, x1)}) .
show ∃ g. path g ∧ path_image g ⊆ (⋃ x∈s. ⋃ x1∈t. {(x, x1)}) ∧
pathstart g = (x1, y1) ∧ pathfinish g = (x2, y2)
using 1 2 gf hs
by (metis (no_types, lifting) pathfinish_def pathfinish_join pathstart_def path-
start_join)
qed

```

```

lemma is_interval_path_connected_1:
  fixes s :: real set
  shows is_interval s ⟷ path_connected s
using is_interval_connected_1 is_interval_path_connected path_connected_imp_connected
by blast

```

5.5.18 Path components

```

lemma Union_path_component [simp]:
  Union {path_component_set S x | x. x ∈ S} = S
apply (rule subset_antisym)
using path_component_subset apply force
using path_component_refl by auto

lemma path_component_disjoint:
  disjoint (path_component_set S a) (path_component_set S b) ⟷
  (a ⊄ path_component_set S b)
unfolding disjoint_iff
using path_component_sym path_component_trans by blast

```

```

lemma path_component_eq_eq:
  path_component S x = path_component S y ⟷
  (x ∉ S) ∧ (y ∉ S) ∨ x ∈ S ∧ y ∈ S ∧ path_component S x y
  (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
  by (metis (no_types) path_component_mem(1) path_component_refl)
next
  assume ?rhs then show ?lhs
  proof
    assume x ∉ S ∧ y ∉ S then show ?lhs
    by (metis Collect_empty_eq_bot path_component_eq_empty)
  next
    assume S: x ∈ S ∧ y ∈ S ∧ path_component S x y show ?lhs
    by (rule ext) (metis S path_component_trans path_component_sym)
  end
end

```


qed
qed

lemma *path_component_unique*:

assumes $x \in c \subseteq S$ *path_connected* c

$\bigwedge c'. \llbracket x \in c'; c' \subseteq S; \text{path\_connected } c' \rrbracket \implies c' \subseteq c$

shows *path_component_set* $S \ x = c$

(**is** *?lhs = ?rhs*)

proof

show *?lhs* $\subseteq$ *?rhs*

using *assms*

by (*metis* *mem_Collect_eq* *path_component_refl* *path_component_subset* *path_connected_path_component_subsetD*)

qed (*simp* *add: assms path_component_maximal*)

lemma *path_component_intermediate_subset*:

path_component_set $u \ a \subseteq t \wedge t \subseteq u$

$\implies \text{path\_component\_set } t \ a = \text{path\_component\_set } u \ a$

by (*metis* (*no_types*) *path_component_mono* *path_component_path_component_subset_antisym*)

lemma *complement_path_component_Union*:

fixes $x :: 'a :: \text{topological\_space}$

shows $S - \text{path\_component\_set } S \ x =$

$\bigcup (\{\text{path\_component\_set } S \ y \mid y. y \in S\} - \{\text{path\_component\_set } S \ x\})$

proof –

have *: $(\bigwedge x. x \in S - \{a\} \implies \text{disjnt } a \ x) \implies \bigcup S - a = \bigcup (S - \{a\})$

for $a :: 'a$ *set* **and** S

by (*auto simp: disjnt_def*)

have $\bigwedge y. y \in \{\text{path\_component\_set } S \ x \mid x. x \in S\} - \{\text{path\_component\_set } S \ x\}$

$\implies \text{disjnt } (\text{path\_component\_set } S \ x) \ y$

using *path_component_disjoint* *path_component_eq* **by** *fastforce*

then have $\bigcup \{\text{path\_component\_set } S \ x \mid x. x \in S\} - \text{path\_component\_set } S \ x$

$=$

$\bigcup (\{\text{path\_component\_set } S \ y \mid y. y \in S\} - \{\text{path\_component\_set } S \ x\})$

by (*meson* *)

then show *?thesis* **by** *simp*

qed

5.5.19 Path components

definition *path_component_of*

where *path_component_of* $X \ x \ y \equiv \exists g. \text{pathin } X \ g \wedge g \ 0 = x \wedge g \ 1 = y$

abbreviation *path_component_of_set*

where *path_component_of_set* $X \ x \equiv \text{Collect } (\text{path\_component\_of } X \ x)$

definition *path_components_of* $:: 'a \text{ topology} \Rightarrow 'a \text{ set set}$

where $\text{path\_components\_of } X \equiv \text{path\_component\_of\_set } X \text{ ' } \text{topspace } X$

lemma *pathin_canon_iff*: $\text{pathin } (\text{top\_of\_set } T) \ g \longleftrightarrow \text{path } g \wedge g \text{ ' } \{0..1\} \subseteq T$
by (*simp add: path_def pathin_def*)

lemma *path_component_of_canon_iff* [*simp*]:
 $\text{path\_component\_of } (\text{top\_of\_set } T) \ a \ b \longleftrightarrow \text{path\_component } T \ a \ b$
by (*simp add: path_component_of_def pathin_canon_iff path_defs*)

lemma *path_component_in_topspace*:
 $\text{path\_component\_of } X \ x \ y \implies x \in \text{topspace } X \wedge y \in \text{topspace } X$
by (*auto simp: path_component_of_def pathin_def continuous_map_def*)

lemma *path_component_of_refl*:
 $\text{path\_component\_of } X \ x \ x \longleftrightarrow x \in \text{topspace } X$
by (*metis path_component_in_topspace path_component_of_def pathin_const*)

lemma *path_component_of_sym*:
assumes $\text{path\_component\_of } X \ x \ y$
shows $\text{path\_component\_of } X \ y \ x$
using *assms*
apply (*clarsimp simp: path_component_of_def pathin_def*)
apply (*rule_tac x=g* $(\lambda t. 1 - t)$ **in** *exI*)
apply (*auto intro!: continuous_map_compose simp: continuous_map_in_subtopology continuous_on_op_minus*)
done

lemma *path_component_of_sym_iff*:
 $\text{path\_component\_of } X \ x \ y \longleftrightarrow \text{path\_component\_of } X \ y \ x$
by (*metis path_component_of_sym*)

lemma *continuous_map_cases_le*:
assumes *contp*: $\text{continuous\_map } X \ \text{euclideanreal } p$
and *contq*: $\text{continuous\_map } X \ \text{euclideanreal } q$
and *contf*: $\text{continuous\_map } (\text{subtopology } X \ \{x. x \in \text{topspace } X \wedge p \ x \leq q \ x\})$
 $Y \ f$
and *contg*: $\text{continuous\_map } (\text{subtopology } X \ \{x. x \in \text{topspace } X \wedge q \ x \leq p \ x\})$
 $Y \ g$
and *fg*: $\bigwedge x. \llbracket x \in \text{topspace } X; p \ x = q \ x \rrbracket \implies f \ x = g \ x$
shows $\text{continuous\_map } X \ Y \ (\lambda x. \text{if } p \ x \leq q \ x \text{ then } f \ x \text{ else } g \ x)$
proof –
have $\text{continuous\_map } X \ Y \ (\lambda x. \text{if } q \ x - p \ x \in \{0..\} \text{ then } f \ x \text{ else } g \ x)$
proof (*rule continuous_map_cases_function*)
show $\text{continuous\_map } X \ \text{euclideanreal } (\lambda x. q \ x - p \ x)$
by (*intro contp contq continuous_intros*)
show $\text{continuous\_map } (\text{subtopology } X \ \{x \in \text{topspace } X. q \ x - p \ x \in \text{euclideanreal closure\_of } \{0..\}\}) \ Y \ f$
by (*simp add: contf*)
show $\text{continuous\_map } (\text{subtopology } X \ \{x \in \text{topspace } X. q \ x - p \ x \in \text{euclideanreal$

```

closure_of (topspace euclideanreal - {0..})) Y g
  by (simp add: contg flip: Compl_eq_Diff_UNIV)
qed (auto simp: fg)
then show ?thesis
  by simp
qed

```

```

lemma continuous_map_cases_lt:
  assumes contp: continuous_map X euclideanreal p
  and contq: continuous_map X euclideanreal q
  and contf: continuous_map (subtopology X {x. x ∈ topspace X ∧ p x ≤ q x})
Y f
  and contg: continuous_map (subtopology X {x. x ∈ topspace X ∧ q x ≤ p x})
Y g
  and fg: ∧x. [x ∈ topspace X; p x = q x] ⇒ f x = g x
  shows continuous_map X Y (λx. if p x < q x then f x else g x)
proof -
  have continuous_map X Y (λx. if q x - p x ∈ {0<..} then f x else g x)
  proof (rule continuous_map_cases_function)
    show continuous_map X euclideanreal (λx. q x - p x)
    by (intro contp contq continuous_intros)
    show continuous_map (subtopology X {x ∈ topspace X. q x - p x ∈ euclideanreal
closure_of {0<..}}) Y f
    by (simp add: contf)
    show continuous_map (subtopology X {x ∈ topspace X. q x - p x ∈ euclideanreal
closure_of (topspace euclideanreal - {0<..})) Y g
    by (simp add: contg flip: Compl_eq_Diff_UNIV)
  qed (auto simp: fg)
  then show ?thesis
    by simp
qed

```

```

lemma path_component_of_trans:
  assumes path_component_of X x y and path_component_of X y z
  shows path_component_of X x z
  unfolding path_component_of_def pathin_def
proof -
  let ?T01 = top_of_set {0..1::real}
  obtain g1 g2 where g1: continuous_map ?T01 X g1 x = g1 0 y = g1 1
  and g2: continuous_map ?T01 X g2 g2 0 = g1 1 z = g2 1
  using assms unfolding path_component_of_def pathin_def by blast
  let ?g = λx. if x ≤ 1/2 then (g1 ∘ (λt. 2 * t)) x else (g2 ∘ (λt. 2 * t - 1)) x
  show ∃g. continuous_map ?T01 X g ∧ g 0 = x ∧ g 1 = z
  proof (intro exI conjI)
    show continuous_map (subtopology euclideanreal {0..1}) X ?g
    proof (intro continuous_map_cases_le continuous_map_compose, force, force)
      show continuous_map (subtopology ?T01 {x ∈ topspace ?T01. x ≤ 1/2})
?T01 ((*) 2)
      by (auto simp: continuous_map_in_subtopology continuous_map_from_subtopology)
    qed
  qed

```

```

have continuous_map
  (subtopology (top_of_set {0..1}) {x. 0 ≤ x ∧ x ≤ 1 ∧ 1 ≤ x * 2})
  euclideanreal (λt. 2 * t - 1)
by (intro continuous_intros) (force intro: continuous_map_from_subtopology)
then show continuous_map (subtopology ?T01 {x ∈ topspace ?T01. 1/2 ≤
x}) ?T01 (λt. 2 * t - 1)
  by (force simp: continuous_map_in_subtopology)
  show (g1 ∘ (*) 2) x = (g2 ∘ (λt. 2 * t - 1)) x if x ∈ topspace ?T01 x =
1/2 for x
  using that by (simp add: g2(2) mult.commute continuous_map_from_subtopology)
qed (auto simp: g1 g2)
qed (auto simp: g1 g2)
qed

```

```

lemma path_component_of_mono:
  [[path_component_of (subtopology X S) x y; S ⊆ T]] ⇒ path_component_of
(subtopology X T) x y
  unfolding path_component_of_def
  by (metis subsetD pathin_subtopology)

```

```

lemma path_component_of:
  path_component_of X x y ⇔ (∃ T. path_connectedin X T ∧ x ∈ T ∧ y ∈ T)
  (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
    by (metis atLeastAtMost_iff image_eqI order_refl path_component_of_def
path_connectedin_path_image zero_le_one)
  next
    assume ?rhs then show ?lhs
      by (metis path_component_of_def path_connectedin)
qed

```

```

lemma path_component_of_set:
  path_component_of X x y ⇔ (∃ g. pathin X g ∧ g 0 = x ∧ g 1 = y)
  by (auto simp: path_component_of_def)

```

```

lemma path_component_of_subset_topspace:
  Collect(path_component_of X x) ⊆ topspace X
  using path_component_in_topspace by fastforce

```

```

lemma path_component_of_eq_empty:
  Collect(path_component_of X x) = {} ⇔ (x ∉ topspace X)
  using path_component_in_topspace path_component_of_refl by fastforce

```

```

lemma path_connected_space_iff_path_component:
  path_connected_space X ⇔ (∀ x ∈ topspace X. ∀ y ∈ topspace X. path_component_of
X x y)
  by (simp add: path_component_of path_connected_space_subconnected)

```

```

lemma path_connected_space_imp_path_component_of:
   $\llbracket \text{path\_connected\_space } X; a \in \text{topspace } X; b \in \text{topspace } X \rrbracket$ 
     $\implies \text{path\_component\_of } X \ a \ b$ 
by (simp add: path_connected_space_iff_path_component)

lemma path_connected_space_path_component_set:
   $\text{path\_connected\_space } X \longleftrightarrow (\forall x \in \text{topspace } X. \text{Collect}(\text{path\_component\_of } X \ x) = \text{topspace } X)$ 
using path_component_of_subset_topspace path_connected_space_iff_path_component
by fastforce

lemma path_component_of_maximal:
   $\llbracket \text{path\_connectedin } X \ s; x \in s \rrbracket \implies s \subseteq \text{Collect}(\text{path\_component\_of } X \ x)$ 
using path_component_of by fastforce

lemma path_component_of_equiv:
   $\text{path\_component\_of } X \ x \ y \longleftrightarrow x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge \text{path\_component\_of } X \ x = \text{path\_component\_of } X \ y$ 
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    apply (simp add: fun_eq_iff path_component_in_topspace)
    apply (meson path_component_of_sym path_component_of_trans)
    done
qed (simp add: path_component_of_refl)

lemma path_component_of_disjoint:
   $\text{disjnt} (\text{Collect} (\text{path\_component\_of } X \ x)) (\text{Collect} (\text{path\_component\_of } X \ y))$ 
 $\longleftrightarrow$ 
   $\sim (\text{path\_component\_of } X \ x \ y)$ 
by (force simp: disjnt_def path_component_of_eq_empty path_component_of_equiv)

lemma path_component_of_eq:
   $\text{path\_component\_of } X \ x = \text{path\_component\_of } X \ y \longleftrightarrow$ 
   $(x \notin \text{topspace } X) \wedge (y \notin \text{topspace } X) \vee$ 
   $x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge \text{path\_component\_of } X \ x \ y$ 
by (metis Collect_empty_eq_bot path_component_of_eq_empty path_component_of_equiv)

lemma path_component_of_aux:
   $\text{path\_component\_of } X \ x \ y$ 
 $\implies \text{path\_component\_of} (\text{subtopology } X (\text{Collect} (\text{path\_component\_of } X \ x))) \ x \ y$ 
by (meson path_component_of_path_component_of_maximal path_connectedin_subtopology)

lemma path_connectedin_path_component_of:
   $\text{path\_connectedin } X (\text{Collect} (\text{path\_component\_of } X \ x))$ 
proof –
  have  $\text{topspace} (\text{subtopology } X (\text{path\_component\_of\_set } X \ x)) = \text{path\_component\_of\_set}$ 

```

```

X x
  by (meson path_component_of_subset_topspace topspace_subtopology_subset)
  then have path_connected_space (subtopology X (path_component_of_set X
x))
  by (metis (full_types) path_component_of_aux mem_Collect_eq path_component_of_equiv
path_connected_space_iff_path_component)
  then show ?thesis
  by (simp add: path_component_of_subset_topspace path_connectedin_def)
qed

```

```

lemma path_connectedin_euclidean [simp]:
  path_connectedin euclidean S  $\longleftrightarrow$  path_connected S
  by (auto simp: path_connectedin_def path_connected_space_iff_path_component
path_connected_component)

```

```

lemma path_connected_space_euclidean_subtopology [simp]:
  path_connected_space (subtopology euclidean S)  $\longleftrightarrow$  path_connected S
  using path_connectedin_topspace by force

```

```

lemma Union_path_components_of:
   $\bigcup (\text{path\_components\_of } X) = \text{topspace } X$ 
  by (auto simp: path_components_of_def path_component_of_equiv)

```

```

lemma path_components_of_maximal:
   $\llbracket C \in \text{path\_components\_of } X; \text{path\_connectedin } X \ S; \sim \text{disjnt } C \ S \rrbracket \implies S \subseteq C$ 
  apply (auto simp: path_components_of_def path_component_of_equiv)
  using path_component_of_maximal path_connectedin_def apply fastforce
  by (meson disjnt_subset2 path_component_of_disjoint path_component_of_equiv
path_component_of_maximal)

```

```

lemma pairwise_disjoint_path_components_of:
  pairwise disjnt (path_components_of X)
  by (auto simp: path_components_of_def pairwise_def path_component_of_disjoint
path_component_of_equiv)

```

```

lemma complement_path_components_of_Union:
   $C \in \text{path\_components\_of } X$ 
   $\implies \text{topspace } X - C = \bigcup (\text{path\_components\_of } X - \{C\})$ 
  by (metis Diff_cancel Diff_subset Union_path_components_of cSup_singleton
diff_Union_pairwise_disjoint insert_subset pairwise_disjoint_path_components_of)

```

```

lemma nonempty_path_components_of:
  assumes  $C \in \text{path\_components\_of } X$  shows  $C \neq \{\}$ 
proof -
  have  $C \in \text{path\_component\_of\_set } X \text{ ' } \text{topspace } X$ 
  using assms path_components_of_def by blast
  then show ?thesis
  using path_component_of_refl by fastforce
qed

```

lemma *path_components_of_subset*: $C \in \text{path\_components\_of } X \implies C \subseteq \text{topspace } X$

by (*auto simp: path_components_of_def path_component_of_equiv*)

lemma *path_connectedin_path_components_of*:

$C \in \text{path\_components\_of } X \implies \text{path\_connectedin } X \ C$

by (*auto simp: path_components_of_def path_connectedin_path_component_of*)

lemma *path_component_in_path_components_of*:

$\text{Collect } (\text{path\_component\_of } X \ a) \in \text{path\_components\_of } X \longleftrightarrow a \in \text{topspace } X$

by (*metis imageI nonempty_path_components_of path_component_of_eq_empty path_components_of_def*)

lemma *path_connectedin_Union*:

assumes $\mathcal{A}$: $\bigwedge S. S \in \mathcal{A} \implies \text{path\_connectedin } X \ S \cap \mathcal{A} \neq \{\}$

shows $\text{path\_connectedin } X \ (\bigcup \mathcal{A})$

proof –

obtain a **where** $\bigwedge S. S \in \mathcal{A} \implies a \in S$

using *assms* **by** *blast*

then have $\bigwedge x. x \in \text{topspace } (\text{subtopology } X \ (\bigcup \mathcal{A})) \implies \text{path\_component\_of } (\text{subtopology } X \ (\bigcup \mathcal{A})) \ a \ x$

by *simp (meson Union_upper \mathcal{A} path_component_of path_connectedin_subtopology)*

then show *?thesis*

using $\mathcal{A}$ **unfolding** *path_connectedin_def*

by (*metis Sup_le_iff path_component_of_equiv path_connected_space_iff_path_component*)

qed

lemma *path_connectedin_Un*:

$\llbracket \text{path\_connectedin } X \ S; \text{path\_connectedin } X \ T; S \cap T \neq \{\} \rrbracket$

$\implies \text{path\_connectedin } X \ (S \cup T)$

by (*blast intro: path_connectedin_Union [of \{S,T\}, simplified]*)

lemma *path_connected_space_iff_components_eq*:

$\text{path\_connected\_space } X \longleftrightarrow$

$(\forall C \in \text{path\_components\_of } X. \forall C' \in \text{path\_components\_of } X. C = C')$

unfolding *path_components_of_def*

proof (*intro iffI ballI*)

assume $\forall C \in \text{path\_component\_of\_set } X \ ' \text{topspace } X.$

$\forall C' \in \text{path\_component\_of\_set } X \ ' \text{topspace } X. C = C'$

then show $\text{path\_connected\_space } X$

using *path_component_of_refl path_connected_space_iff_path_component* **by**

fastforce

qed (*auto simp: path_connected_space_path_component_set*)

lemma *path_components_of_eq_empty*:

$\text{path\_components\_of } X = \{\} \longleftrightarrow \text{topspace } X = \{\}$

using *Union_path_components_of nonempty_path_components_of* **by** *fastforce*

lemma *path_components_of_empty_space:*

topspace X = {} $\implies$ path_components_of X = {}

by (*simp add: path_components_of_eq_empty*)

lemma *path_components_of_subset_singleton:*

path_components_of X $\subseteq$ {S} $\longleftrightarrow$

path_connected_space X $\wedge$ (topspace X = {} $\vee$ topspace X = S)

proof (*cases topspace X = {}*)

case *True*

then show *?thesis*

by (*auto simp: path_components_of_empty_space path_connected_space_topspace_empty*)

next

case *False*

have (*path_components_of X = {S}*) $\longleftrightarrow$ (*path_connected_space X $\wedge$ topspace X = S*)

proof (*intro iffI conjI*)

assume *L: path_components_of X = {S}*

then show *path_connected_space X*

by (*simp add: path_connected_space_iff_components_eq*)

show *topspace X = S*

by (*metis L ccpo_Sup_singleton [of S] Union_path_components_of*)

next

assume *R: path_connected_space X $\wedge$ topspace X = S*

then show *path_components_of X = {S}*

using *ccpo_Sup_singleton [of S]*

by (*metis False all_not_in_conv insert_iff mk_disjoint_insert path_component_in_path_components path_connected_space_iff_components_eq path_connected_space_path_component_set*)

qed

with *False show ?thesis*

by (*simp add: path_components_of_eq_empty subset_singleton_iff*)

qed

lemma *path_connected_space_iff_components_subset_singleton:*

path_connected_space X $\longleftrightarrow$ ($\exists a.$ path_components_of X $\subseteq$ {a})

by (*simp add: path_components_of_subset_singleton*)

lemma *path_components_of_eq_singleton:*

path_components_of X = {S} $\longleftrightarrow$ path_connected_space X $\wedge$ topspace X $\neq$ {} $\wedge$ S = topspace X

by (*metis cSup_singleton insert_not_empty path_components_of_subset_singleton subset_singleton_iff*)

lemma *path_components_of_path_connected_space:*

path_connected_space X $\implies$ path_components_of X = (if topspace X = {} then {} else {topspace X})

by (*simp add: path_components_of_eq_empty path_components_of_eq_singleton*)

lemma *path_component_subset_connected_component_of:*


```

  path_component_of_set X x  $\subseteq$  connected_component_of_set X x
proof (cases x  $\in$  topspace X)
  case True
  then show ?thesis
    by (simp add: connected_component_of_maximal_path_component_of_refl
  path_connectedin_imp_connectedin path_connectedin_path_component_of)
  next
  case False
  then show ?thesis
    using path_component_of_eq_empty by fastforce
qed

```

```

lemma exists_path_component_of_superset:
  assumes S: path_connectedin X S and ne: topspace X  $\neq$  {}
  obtains C where C  $\in$  path_components_of X S  $\subseteq$  C
proof (cases S = {})
  case True
  then show ?thesis
    using ne path_components_of_eq_empty that by fastforce
  next
  case False
  then obtain a where a  $\in$  S
    by blast
  show ?thesis
proof
    show Collect (path_component_of X a)  $\in$  path_components_of X
      by (meson  $\langle a \in S \rangle$  S subsetD path_component_in_path_components_of
  path_connectedin_subset_topspace)
    show S  $\subseteq$  Collect (path_component_of X a)
      by (simp add: S  $\langle a \in S \rangle$  path_component_of_maximal)
    qed
  qed

```

```

lemma path_component_of_eq_overlap:
  path_component_of X x = path_component_of X y  $\longleftrightarrow$ 
    (x  $\notin$  topspace X)  $\wedge$  (y  $\notin$  topspace X)  $\vee$ 
    Collect (path_component_of X x)  $\cap$  Collect (path_component_of X y)  $\neq$  {}
  by (metis disjnt_def empty_iff inf_bot_right mem_Collect_eq path_component_of_disjoint
  path_component_of_eq path_component_of_eq_empty)

```

```

lemma path_component_of_nonoverlap:
  Collect (path_component_of X x)  $\cap$  Collect (path_component_of X y) = {}
 $\longleftrightarrow$ 
    (x  $\notin$  topspace X)  $\vee$  (y  $\notin$  topspace X)  $\vee$ 
    path_component_of X x  $\neq$  path_component_of X y
  by (metis inf.idem path_component_of_eq_empty path_component_of_eq_overlap)

```

```

lemma path_component_of_overlap:
  Collect (path_component_of X x)  $\cap$  Collect (path_component_of X y)  $\neq$  {}

```

$\longleftrightarrow$
 $x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge \text{path\_component\_of } X \ x = \text{path\_component\_of } X \ y$
by (meson path_component_of_nonoverlap)

lemma path_components_of_disjoint:

$\llbracket C \in \text{path\_components\_of } X; C' \in \text{path\_components\_of } X \rrbracket \implies \text{disjnt } C \ C'$
 $\longleftrightarrow C \neq C'$
by (auto simp: path_components_of_def path_component_of_disjoint path_component_of_equiv)

lemma path_components_of_overlap:

$\llbracket C \in \text{path\_components\_of } X; C' \in \text{path\_components\_of } X \rrbracket \implies C \cap C' \neq \{\}$
 $\longleftrightarrow C = C'$
by (auto simp: path_components_of_def path_component_of_equiv)

lemma path_component_of_unique:

$\llbracket x \in C; \text{path\_connectedin } X \ C; \bigwedge C'. \llbracket x \in C'; \text{path\_connectedin } X \ C' \rrbracket \implies C' \subseteq C \rrbracket$
 $\implies \text{Collect } (\text{path\_component\_of } X \ x) = C$
by (meson subsetD eq_iff path_component_of_maximal path_connectedin_path_component_of)

lemma path_component_of_discrete_topology [simp]:

$\text{Collect } (\text{path\_component\_of } (\text{discrete\_topology } U) \ x) = (\text{if } x \in U \text{ then } \{x\} \text{ else } \{\})$

proof –

have $\bigwedge C'. \llbracket x \in C'; \text{path\_connectedin } (\text{discrete\_topology } U) \ C' \rrbracket \implies C' \subseteq \{x\}$
by (metis path_connectedin_discrete_topology subsetD singletonD)
then have $x \in U \implies \text{Collect } (\text{path\_component\_of } (\text{discrete\_topology } U) \ x) = \{x\}$
by (simp add: path_component_of_unique)
then show ?thesis
using path_component_in_topspace **by** fastforce
qed

lemma path_component_of_discrete_topology_iff [simp]:

$\text{path\_component\_of } (\text{discrete\_topology } U) \ x \ y \longleftrightarrow x \in U \wedge y = x$
by (metis empty_iff insertI1 mem_Collect_eq path_component_of_discrete_topology singletonD)

lemma path_components_of_discrete_topology [simp]:

$\text{path\_components\_of } (\text{discrete\_topology } U) = (\lambda x. \{x\}) \text{ ' } U$
by (auto simp: path_components_of_def image_def fun_eq_iff)

lemma homeomorphic_map_path_component_of:

assumes $f: \text{homeomorphic\_map } X \ Y \ f$ **and** $x: x \in \text{topspace } X$
shows $\text{Collect } (\text{path\_component\_of } Y \ (f \ x)) = f \text{ ' } \text{Collect}(\text{path\_component\_of } X \ x)$
proof –
obtain g **where** $g: \text{homeomorphic\_maps } X \ Y \ f \ g$

```

    using f homeomorphic_map_maps by blast
  show ?thesis
  proof
    have Collect (path_component_of Y (f x))  $\subseteq$  topspace Y
    by (simp add: path_component_of_subset_topspace)
    moreover have g ' Collect (path_component_of Y (f x))  $\subseteq$  Collect (path_component_of
X (g (f x)))
    using g x unfolding homeomorphic_maps_def
    by (metis f homeomorphic_imp_surjective_map imageI mem_Collect_eq
path_component_of_maximal path_component_of_refl path_connectedin_continuous_map_image
path_connectedin_path_component_of)
    ultimately show Collect (path_component_of Y (f x))  $\subseteq$  f ' Collect (path_component_of
X x)
    using g x unfolding homeomorphic_maps_def continuous_map_def im-
age_iff subset_iff
    by metis
    show f ' Collect (path_component_of X x)  $\subseteq$  Collect (path_component_of Y
(f x))
    proof (rule path_component_of_maximal)
      show path_connectedin Y (f ' Collect (path_component_of X x))
      by (meson f homeomorphic_map_path_connectedness_eq path_connectedin_path_component_of)
    qed (simp add: path_component_of_refl x)
  qed
qed
qed

lemma homeomorphic_map_path_components_of:
  assumes homeomorphic_map X Y f
  shows path_components_of Y = (image f) ' (path_components_of X)
  (is ?lhs = ?rhs)
  unfolding path_components_of_def homeomorphic_imp_surjective_map [OF
assms, symmetric]
  using assms homeomorphic_map_path_component_of by fastforce

```

5.5.20 Sphere is path-connected

```

lemma path_connected_punctured_universe:
  assumes  $2 \leq DIM('a::euclidean\_space)$ 
  shows path_connected (- {a::'a})
  proof -
    let ?A = {x::'a.  $\exists i \in Basis. x \cdot i < a \cdot i$ }
    let ?B = {x::'a.  $\exists i \in Basis. a \cdot i < x \cdot i$ }

    have A: path_connected ?A
      unfolding Collect_bex_eq
    proof (rule path_connected_UNION)
      fix i :: 'a
      assume i  $\in$  Basis
      then show  $(\sum i \in Basis. (a \cdot i - 1) * i) \in \{x::'a. x \cdot i < a \cdot i\}$ 
      by simp
    qed
  qed

```

```

    show path_connected {x. x · i < a · i}
      using convex_imp_path_connected [OF convex_halfspace_lt, of i a · i]
      by (simp add: inner_commute)
  qed
  have B: path_connected ?B
    unfolding Collect_bex_eq
  proof (rule path_connected_UNION)
    fix i :: 'a
    assume i ∈ Basis
    then show (∑ i ∈ Basis. (a · i + 1) *R i) ∈ {x :: 'a. a · i < x · i}
      by simp
    show path_connected {x. a · i < x · i}
      using convex_imp_path_connected [OF convex_halfspace_gt, of a · i i]
      by (simp add: inner_commute)
  qed
  obtain S :: 'a set where S ⊆ Basis and card S = Suc (Suc 0)
    using ex_card [OF assms]
    by auto
  then obtain b0 b1 :: 'a where b0 ∈ Basis and b1 ∈ Basis and b0 ≠ b1
    unfolding card_Suc_eq by auto
  then have a + b0 - b1 ∈ ?A ∩ ?B
    by (auto simp: inner_simps inner_Basis)
  then have ?A ∩ ?B ≠ {}
    by fast
  with A B have path_connected (?A ∪ ?B)
    by (rule path_connected_Un)
  also have ?A ∪ ?B = {x. ∃ i ∈ Basis. x · i ≠ a · i}
    unfolding neq_iff bex_disj_distrib Collect_disj_eq ..
  also have ... = {x. x ≠ a}
    unfolding euclidean_eq_iff [where 'a = 'a]
    by (simp add: Bex_def)
  also have ... = - {a}
    by auto
  finally show ?thesis .
  qed

corollary connected_punctured_universe:
  2 ≤ DIM('N::euclidean_space) ⇒ connected(- {a :: 'N})
  by (simp add: path_connected_punctured_universe path_connected_imp_connected)

proposition path_connected_sphere:
  fixes a :: 'a :: euclidean_space
  assumes 2 ≤ DIM('a)
  shows path_connected(sphere a r)
  proof (cases r 0 :: real rule: linorder_cases)
    case less
    then show ?thesis
      by (simp)
  next

```

```

    case equal
    then show ?thesis
      by (simp)
  next
    case greater
    then have eq: (sphere (0::'a) r) = ( $\lambda x. (r / \text{norm } x) *_R x$ ) ' ( $-\{0::'a\}$ )
      by (force simp: image_iff split: if_split_asm)
    have continuous_on ( $-\{0::'a\}$ ) ( $\lambda x. (r / \text{norm } x) *_R x$ )
      by (intro continuous_intros) auto
    then have path_connected (( $\lambda x. (r / \text{norm } x) *_R x$ ) ' ( $-\{0::'a\}$ ))
      by (intro path_connected_continuous_image path_connected_punctured_universe
    assms)
    with eq have path_connected (sphere (0::'a) r)
      by auto
    then have path_connected((+) a ' (sphere (0::'a) r))
      by (simp add: path_connected_translation)
    then show ?thesis
      by (metis add.right_neutral sphere_translation)
  qed

lemma connected_sphere:
  fixes a :: 'a :: euclidean_space
  assumes 2  $\leq$  DIM('a)
  shows connected(sphere a r)
  using path_connected_sphere [OF assms]
  by (simp add: path_connected_imp_connected)

corollary path_connected_complement_bounded_convex:
  fixes S :: 'a :: euclidean_space set
  assumes bounded S convex S and 2: 2  $\leq$  DIM('a)
  shows path_connected ( $-\ S$ )
proof (cases S = {})
  case True then show ?thesis
    using convex_imp_path_connected by auto
  next
    case False
    then obtain a where a  $\in S$  by auto
    have § [rule_format]:  $\forall y \in S. \forall u. 0 \leq u \wedge u \leq 1 \longrightarrow (1 - u) *_R a + u *_R y \in S$ 
      using <convex S> <a  $\in S$ > by (simp add: convex_alt)
    { fix x y assume x  $\notin S$  y  $\notin S$ 
      then have x  $\neq$  a y  $\neq$  a using <a  $\in S$ > by auto
      then have bxy: bounded(insert x (insert y S))
        by (simp add: <bounded S>)
      then obtain B::real where B: 0 < B and Bx: norm (a - x) < B and By:
        norm (a - y) < B
        and S  $\subseteq$  ball a B
      using bounded_subset_ballD [OF bxy, of a] by (auto simp: dist_norm)
    }
  qed

```

```

define  $C$  where  $C = B / \text{norm}(x - a)$ 
let  $?Cxa = a + C *_R (x - a)$ 
{ fix  $u$ 
  assume  $u: (1 - u) *_R x + u *_R ?Cxa \in S$  and  $0 \leq u \leq 1$ 
  have  $CC: 1 \leq 1 + (C - 1) * u$ 
  using  $\langle x \neq a \rangle \langle 0 \leq u \rangle Bx$ 
  by (auto simp add: C_def norm_minus_commute)
  have  $*$ :  $\bigwedge v. (1 - u) *_R x + u *_R (a + v *_R (x - a)) = a + (1 + (v - 1) * u) *_R (x - a)$ 
  by (simp add: algebra_simps)
  have  $a + ((1 / (1 + C * u - u)) *_R x + ((u / (1 + C * u - u)) *_R a + (C * u / (1 + C * u - u)) *_R x)) =$ 
 $(1 + (u / (1 + C * u - u))) *_R a + ((1 / (1 + C * u - u)) + (C * u / (1 + C * u - u))) *_R x$ 
  by (simp add: algebra_simps)
  also have  $\dots = (1 + (u / (1 + C * u - u))) *_R a + (1 + (u / (1 + C * u - u))) *_R x$ 
  using  $CC$  by (simp add: field_simps)
  also have  $\dots = x + (1 + (u / (1 + C * u - u))) *_R a + (u / (1 + C * u - u)) *_R x$ 
  by (simp add: algebra_simps)
  also have  $\dots = x + ((1 / (1 + C * u - u)) *_R a + ((u / (1 + C * u - u)) *_R x + (C * u / (1 + C * u - u)) *_R a))$ 
  using  $CC$  by (simp add: field_simps) (simp add: add_divide_distrib scaleR_add_left)
  finally have  $req: (1 - 1 / (1 + (C - 1) * u)) *_R a + (1 / (1 + (C - 1) * u)) *_R (a + (1 + (C - 1) * u) *_R (x - a)) = x$ 
  by (simp add: algebra_simps)
  have  $False$ 
  using  $\S$  [of  $a + (1 + (C - 1) * u) *_R (x - a)$   $1 / (1 + (C - 1) * u)$ ]
  using  $u \langle x \neq a \rangle \langle x \notin S \rangle \langle 0 \leq u \rangle CC$ 
  by (auto simp: req *)
}
then have  $pcx: \text{path\_component } (- S) x ?Cxa$ 
by (force simp: closed_segment_def intro!: path_component_linepath)
define  $D$  where  $D = B / \text{norm}(y - a)$  — massive duplication with the proof above
let  $?Dya = a + D *_R (y - a)$ 
{ fix  $u$ 
  assume  $u: (1 - u) *_R y + u *_R ?Dya \in S$  and  $0 \leq u \leq 1$ 
  have  $DD: 1 \leq 1 + (D - 1) * u$ 
  using  $\langle y \neq a \rangle \langle 0 \leq u \rangle By$ 
  by (auto simp add: D_def norm_minus_commute)
  have  $*$ :  $\bigwedge v. (1 - u) *_R y + u *_R (a + v *_R (y - a)) = a + (1 + (v - 1) * u) *_R (y - a)$ 
  by (simp add: algebra_simps)
  have  $a + ((1 / (1 + D * u - u)) *_R y + ((u / (1 + D * u - u)) *_R a + (D * u / (1 + D * u - u)) *_R y)) =$ 
 $(1 + (u / (1 + D * u - u))) *_R a + ((1 / (1 + D * u - u)) + (D * u$ 

```

```

/ (1 + D * u - u))) *R y
  by (simp add: algebra_simps)
  also have ... = (1 + (u / (1 + D * u - u))) *R a + (1 + (u / (1 + D *
u - u))) *R y
    using DD by (simp add: field_simps)
  also have ... = y + (1 + (u / (1 + D * u - u))) *R a + (u / (1 + D * u
- u)) *R y
    by (simp add: algebra_simps)
  also have ... = y + ((1 / (1 + D * u - u)) *R a +
    ((u / (1 + D * u - u)) *R y + (D * u / (1 + D * u - u)) *R a))
    using DD by (simp add: field_simps) (simp add: add_divide_distrib
scaleR_add_left)
  finally have xeq: (1 - 1 / (1 + (D - 1) * u)) *R a + (1 / (1 + (D - 1)
* u)) *R (a + (1 + (D - 1) * u) *R (y - a)) = y
    by (simp add: algebra_simps)
  have False
    using § [of a + (1 + (D - 1) * u) *R (y - a) 1 / (1 + (D - 1) * u)]
    using u <y ≠ a> <y ∉ S> <0 ≤ u> DD
    by (auto simp: xeq *)
}
then have pdy: path_component (- S) y ?Dya
  by (force simp: closed_segment_def intro!: path_component_linepath)
have pyx: path_component (- S) ?Dya ?Cxa
proof (rule path_component_of_subset)
  show sphere a B ⊆ - S
  using <S ⊆ ball a B> by (force simp: ball_def dist_norm norm_minus_commute)
  have aB: ?Dya ∈ sphere a B ?Cxa ∈ sphere a B
    using <x ≠ a> using <y ≠ a> B by (auto simp: dist_norm C_def D_def)
  then show path_component (sphere a B) ?Dya ?Cxa
    using path_connected_sphere [OF 2] path_connected_component by blast
qed
have path_component (- S) x y
  by (metis path_component_trans path_component_sym pcr pdy pyx)
}
then show ?thesis
  by (auto simp: path_connected_component)
qed

```

lemma *connected_complement_bounded_convex*:

```

  fixes S :: 'a :: euclidean_space set
  assumes bounded S convex S 2 ≤ DIM('a)
  shows connected (- S)
  using path_connected_complement_bounded_convex [OF assms] path_connected_imp_connected
  by blast

```

lemma *connected_diff_ball*:

```

  fixes S :: 'a :: euclidean_space set
  assumes connected S cball a r ⊆ S 2 ≤ DIM('a)
  shows connected (S - ball a r)

```

```

proof (rule connected_diff_open_from_closed [OF ball_subset_cball])
  show connected (cball a r - ball a r)
    using assms_connected_sphere by (auto simp: cball_diff_eq_sphere)
qed (auto simp: assms_dist_norm)

```

```

proposition connected_open_delete:
  assumes open S connected S and  $2 \leq \text{DIM}('N::\text{euclidean\_space})$ 
  shows connected(S - {a::'N})
proof (cases a ∈ S)
  case True
    with ⟨open S⟩ obtain  $\varepsilon$  where  $\varepsilon > 0$  and  $\varepsilon$ : cball a  $\varepsilon \subseteq S$ 
      using open_contains_cball_eq by blast
    define b where  $b \equiv a + \varepsilon *_{\mathbb{R}} (\text{SOME } i. i \in \text{Basis})$ 
    have dist a b =  $\varepsilon$ 
      by (simp add: b_def dist_norm SOME_Basis ⟨ $0 < \varepsilon$ ⟩ less_imp_le)
    with  $\varepsilon$  have  $b \in \bigcap \{S - \text{ball } a \ r \mid r. 0 < r \wedge r < \varepsilon\}$ 
      by auto
    then have nonemp:  $(\bigcap \{S - \text{ball } a \ r \mid r. 0 < r \wedge r < \varepsilon\}) = \{b\} \implies \text{False}$ 
      by auto
    have con:  $\bigwedge r. r < \varepsilon \implies \text{connected } (S - \text{ball } a \ r)$ 
      using  $\varepsilon$  by (force intro: connected_diff_ball [OF ⟨connected S⟩ _ 2])
    have  $x \in \bigcup \{S - \text{ball } a \ r \mid r. 0 < r \wedge r < \varepsilon\}$  if  $x \in S - \{a\}$  for x
      using that ⟨ $0 < \varepsilon$ ⟩
      by (intro UnionI [of S - ball a ( $\min \varepsilon (\text{dist } a \ x) / 2$ )] auto)
    then have  $S - \{a\} = \bigcup \{S - \text{ball } a \ r \mid r. 0 < r \wedge r < \varepsilon\}$ 
      by auto
    then show ?thesis
      by (auto intro: connected_Union con dest!: nonemp)
  next
    case False then show ?thesis
      by (simp add: ⟨connected S⟩)
qed

```

```

corollary path_connected_open_delete:
  assumes open S connected S and  $2 \leq \text{DIM}('N::\text{euclidean\_space})$ 
  shows path_connected(S - {a::'N})
  by (simp add: assms_connected_open_delete connected_open_path_connected
open_delete)

```

```

corollary path_connected_punctured_ball:
   $2 \leq \text{DIM}('N::\text{euclidean\_space}) \implies \text{path\_connected}(\text{ball } a \ r - \{a::'N\})$ 
  by (simp add: path_connected_open_delete)

```

```

corollary connected_punctured_ball:
   $2 \leq \text{DIM}('N::\text{euclidean\_space}) \implies \text{connected}(\text{ball } a \ r - \{a::'N\})$ 
  by (simp add: connected_open_delete)

```

```

corollary connected_open_delete_finite:
  fixes S T::'a::euclidean_space set

```



```

  assumes  $S$ : open  $S$  connected  $S$  and 2:  $2 \leq \text{DIM}('a)$  and finite  $T$ 
  shows connected  $(S - T)$ 
  using  $\langle \text{finite } T \rangle S$ 
proof (induct  $T$ )
  case empty
  show ?case using  $\langle \text{connected } S \rangle$  by simp
next
  case (insert  $x F$ )
  then have connected  $(S - F)$  by auto
  moreover have open  $(S - F)$  using finite_imp_closed[OF  $\langle \text{finite } F \rangle$ ]  $\langle \text{open } S \rangle$ 
  by auto
  ultimately have connected  $(S - F - \{x\})$  using connected_open_delete[OF
  _ 2] by auto
  thus ?case by (metis Diff_insert)
qed

```

```

lemma sphere_1D_doubleton_zero:
  assumes 1:  $\text{DIM}('a) = 1$  and  $r > 0$ 
  obtains  $x y::'a::\text{euclidean\_space}$ 
  where sphere 0  $r = \{x, y\} \wedge \text{dist } x y = 2*r$ 
proof -
  obtain  $b::'a$  where  $b$ : Basis =  $\{b\}$ 
  using 1 card_1_singletonE by blast
  show ?thesis
proof (intro that conjI)
  have  $x = \text{norm } x *_R b \vee x = - \text{norm } x *_R b$  if  $r = \text{norm } x$  for  $x$ 
  proof -
    have  $xb$ :  $(x \cdot b) *_R b = x$ 
    using euclidean_representation [of  $x$ , unfolded  $b$ ] by force
    then have  $\text{norm } ((x \cdot b) *_R b) = \text{norm } x$ 
    by simp
    with  $b$  have  $|x \cdot b| = \text{norm } x$ 
    using norm_Basis by (simp add:  $b$ )
    with  $xb$  show ?thesis
    by (metis (mono_tags, opaque_lifting) abs_eq_iff abs_norm_cancel)
  qed
  with  $\langle r > 0 \rangle b$  show sphere 0  $r = \{r *_R b, - r *_R b\}$ 
  by (force simp: sphere_def dist_norm)
  have  $\text{dist } (r *_R b) (- r *_R b) = \text{norm } (r *_R b + r *_R b)$ 
  by (simp add: dist_norm)
  also have  $\dots = \text{norm } ((2*r) *_R b)$ 
  by (metis mult_2 scaleR_add_left)
  also have  $\dots = 2*r$ 
  using  $\langle r > 0 \rangle b$  norm_Basis by fastforce
  finally show  $\text{dist } (r *_R b) (- r *_R b) = 2*r$  .
qed
qed

```

```

lemma sphere_1D_doubleton:

```

```

fixes a :: 'a :: euclidean_space
assumes DIM('a) = 1 and r > 0
obtains x y where sphere a r = {x,y} ∧ dist x y = 2*r
proof -
  have sphere a r = (+) a ' sphere 0 r
    by (metis add.right_neutral sphere_translation)
  then show ?thesis
    using sphere_1D_doubleton_zero [OF assms]
    by (metis (mono_tags, lifting) dist_add_cancel image_empty image_insert
that)
qed

```

```

lemma psubset_sphere_Compl_connected:
fixes S :: 'a::euclidean_space set
assumes S: S ⊆ sphere a r and 0 < r and 2: 2 ≤ DIM('a)
shows connected(− S)
proof -
  have S ⊆ sphere a r
    using S by blast
  obtain b where dist a b = r and b ∉ S
    using S mem_sphere by blast
  have CS: − S = {x. dist a x ≤ r ∧ (x ∉ S)} ∪ {x. r ≤ dist a x ∧ (x ∉ S)}
    by auto
  have {x. dist a x ≤ r ∧ x ∉ S} ∩ {x. r ≤ dist a x ∧ x ∉ S} ≠ {}
    using ⟨b ∉ S⟩ ⟨dist a b = r⟩ by blast
  moreover have connected {x. dist a x ≤ r ∧ x ∉ S}
    using assms
    by (force intro: connected_intermediate_closure [of ball a r])
  moreover
  have connected {x. r ≤ dist a x ∧ x ∉ S}
  proof (rule connected_intermediate_closure [of − cball a r])
    show {x. r ≤ dist a x ∧ x ∉ S} ⊆ closure (− cball a r)
      using interior_closure by (force intro: connected_complement_bounded_convex)
  qed (use assms connected_complement_bounded_convex in auto)
  ultimately show ?thesis
    by (simp add: CS connected_Un)
qed

```

5.5.21 Every annulus is a connected set

```

lemma path_connected_2DIM_I:
fixes a :: 'N::euclidean_space
assumes 2: 2 ≤ DIM('N) and pc: path_connected {r. 0 ≤ r ∧ P r}
shows path_connected {x. P(norm(x − a))}
proof -
  have {x. P(norm(x − a))} = (+) a ' {x. P(norm x)}
    by force
  moreover have path_connected {x::'N. P(norm x)}
  proof -

```

```

let ?D = {x. 0 ≤ x ∧ P x} × sphere (0::'N) 1
have x ∈ (λz. fst z *R snd z) ' ?D
  if P (norm x) for x::'N
proof (cases x=0)
case True
  with that show ?thesis
    apply (simp add: image_iff)
    by (metis (no_types) mem_sphere_0 order_refl vector_choose_size zero_le_one)
next
case False
  with that show ?thesis
    by (rule_tac x=(norm x, x /R norm x) in image_eqI) auto
qed
then have *: {x::'N. P(norm x)} = (λz. fst z *R snd z) ' ?D
  by auto
have continuous_on ?D (λz::real×'N. fst z *R snd z)
  by (intro continuous_intros)
moreover have path_connected ?D
  by (metis path_connected_Times [OF pc] path_connected_sphere 2)
ultimately show ?thesis
  by (simp add: * path_connected_continuous_image)
qed
ultimately show ?thesis
  using path_connected_translation by metis
qed

```

proposition *path_connected_annulus:*

```

fixes a :: 'N::euclidean_space
assumes 2 ≤ DIM('N)
shows path_connected {x. r1 < norm(x - a) ∧ norm(x - a) < r2}
  path_connected {x. r1 < norm(x - a) ∧ norm(x - a) ≤ r2}
  path_connected {x. r1 ≤ norm(x - a) ∧ norm(x - a) < r2}
  path_connected {x. r1 ≤ norm(x - a) ∧ norm(x - a) ≤ r2}
by (auto simp: is_interval_def intro!: is_interval_convex convex_imp_path_connected
path_connected_2DIM_I [OF assms])

```

proposition *connected_annulus:*

```

fixes a :: 'N::euclidean_space
assumes 2 ≤ DIM('N::euclidean_space)
shows connected {x. r1 < norm(x - a) ∧ norm(x - a) < r2}
  connected {x. r1 < norm(x - a) ∧ norm(x - a) ≤ r2}
  connected {x. r1 ≤ norm(x - a) ∧ norm(x - a) < r2}
  connected {x. r1 ≤ norm(x - a) ∧ norm(x - a) ≤ r2}
by (auto simp: path_connected_annulus [OF assms] path_connected_imp_connected)

```

5.5.22 Relations between components and path components

lemma *open_connected_component:*

```

fixes S :: 'a::real_normed_vector set

```

```

    assumes open S
    shows open (connected_component_set S x)
  proof (clarsimp simp: open_contains_ball)
    fix y
    assume xy: connected_component S x y
    then obtain e where e>0 ball y e  $\subseteq$  S
    using assms connected_component_in openE by blast
    then show  $\exists e>0. \text{ball } y \ e \subseteq \text{connected\_component\_set } S \ x$ 
    by (metis xy centre_in_ball connected_ball connected_component_eq_eq connected_component_in connected_component_maximal)
  qed

```

corollary open_components:

```

    fixes S :: 'a::real_normed_vector set
    shows  $\llbracket \text{open } u; S \in \text{components } u \rrbracket \implies \text{open } S$ 
    by (simp add: components_iff) (metis open_connected_component)

```

lemma in_closure_connected_component:

```

    fixes S :: 'a::real_normed_vector set
    assumes x:  $x \in S$  and S: open S
    shows  $x \in \text{closure } (\text{connected\_component\_set } S \ y) \longleftrightarrow x \in \text{connected\_component\_set } S \ y$ 
  proof -
    { assume  $x \in \text{closure } (\text{connected\_component\_set } S \ y)$ 
      moreover have  $x \in \text{connected\_component\_set } S \ x$ 
      using x by simp
      ultimately have  $x \in \text{connected\_component\_set } S \ y$ 
      using S by (meson Compl_disjoint closure_iff_nhds_not_empty connected_component_disjoint disjoint_eq_subset_Compl open_connected_component)
    }
    then show ?thesis
    by (auto simp: closure_def)
  qed

```

lemma connected_disjoint_Union_open_pick:

```

    assumes pairwise disjoint B
     $\bigwedge S. S \in A \implies \text{connected } S \wedge S \neq \{\}$ 
     $\bigwedge S. S \in B \implies \text{open } S$ 
     $\bigcup A \subseteq \bigcup B$ 
     $S \in A$ 
    obtains T where  $T \in B \ S \subseteq T \ S \cap \bigcup (B - \{T\}) = \{\}$ 

```

proof -

```

    have  $S \subseteq \bigcup B \ \text{connected } S \ S \neq \{\}$ 
    using assms  $\langle S \in A \rangle$  by blast+
    then obtain T where  $T \in B \ S \cap T \neq \{\}$ 
    by (metis Sup_inf_eq_bot_iff inf.absorb_iff2 inf_commute)
    have 1: open T by (simp add:  $\langle T \in B \rangle$  assms)
    have 2: open  $(\bigcup (B - \{T\}))$  using assms by blast
    have 3:  $S \subseteq T \cup \bigcup (B - \{T\})$  using  $\langle S \subseteq \bigcup B \rangle$  by blast

```

```

have  $T \cap \bigcup (B - \{T\}) = \{\}$  using  $\langle T \in B \rangle \langle \text{pairwise\_disjnt } B \rangle$ 
  by (auto simp: pairwise_def disjnt_def)
then have  $4: T \cap \bigcup (B - \{T\}) \cap S = \{\}$  by auto
from connectedD [OF  $\langle \text{connected } S \rangle$  1 2 4 3]
have  $S \cap \bigcup (B - \{T\}) = \{\}$ 
  by (auto simp: Int_commute  $\langle S \cap T \neq \{\} \rangle$ )
with  $\langle T \in B \rangle$  have  $S \subseteq T$ 
  using 3 by auto
show ?thesis
  using  $\langle S \cap \bigcup (B - \{T\}) = \{\} \rangle \langle S \subseteq T \rangle \langle T \in B \rangle$  that by auto
qed

```

lemma connected_disjoint_Union_open_subset:

```

assumes A: pairwise_disjnt A and B: pairwise_disjnt B
  and SA:  $\bigwedge S. S \in A \implies \text{open } S \wedge \text{connected } S \wedge S \neq \{\}$ 
  and SB:  $\bigwedge S. S \in B \implies \text{open } S \wedge \text{connected } S \wedge S \neq \{\}$ 
  and eq [simp]:  $\bigcup A = \bigcup B$ 
shows  $A \subseteq B$ 
proof
  fix S
  assume  $S \in A$ 
  obtain T where  $T \in B \ S \subseteq T \ S \cap \bigcup (B - \{T\}) = \{\}$ 
    using SA SB  $\langle S \in A \rangle$  connected_disjoint_Union_open_pick [OF B, of A] eq
    order_refl by blast
  moreover obtain S' where  $S' \in A \ T \subseteq S' \ T \cap \bigcup (A - \{S'\}) = \{\}$ 
    using SA SB  $\langle T \in B \rangle$  connected_disjoint_Union_open_pick [OF A, of B] eq
    order_refl by blast
  ultimately have  $S' = S$ 
    by (metis A Int_subset_iff SA  $\langle S \in A \rangle$  disjnt_def inf.orderE pairwise_def)
  with  $\langle T \subseteq S' \rangle$  have  $T \subseteq S$  by simp
  with  $\langle S \subseteq T \rangle$  have  $S = T$  by blast
  with  $\langle T \in B \rangle$  show  $S \in B$  by simp
qed

```

lemma connected_disjoint_Union_open_unique:

```

assumes A: pairwise_disjnt A and B: pairwise_disjnt B
  and SA:  $\bigwedge S. S \in A \implies \text{open } S \wedge \text{connected } S \wedge S \neq \{\}$ 
  and SB:  $\bigwedge S. S \in B \implies \text{open } S \wedge \text{connected } S \wedge S \neq \{\}$ 
  and eq [simp]:  $\bigcup A = \bigcup B$ 
shows  $A = B$ 
by (rule subset_antisym; metis connected_disjoint_Union_open_subset assms)

```

proposition components_open_unique:

```

fixes S :: 'a::real_normed_vector set
assumes pairwise_disjnt A  $\bigcup A = S$ 
   $\bigwedge X. X \in A \implies \text{open } X \wedge \text{connected } X \wedge X \neq \{\}$ 
shows components S = A
proof –
  have open S using assms by blast

```

```

show ?thesis
proof (rule connected_disjoint_Union_open_unique)
  show disjoint (components S)
    by (simp add: components_eq disjnt_def pairwise_def)
qed (use ⟨open S⟩ in ⟨simp_all add: assms open_components in_components_connected
in_components_nonempty⟩)
qed

```

5.5.23 Existence of unbounded components

```

lemma cobounded_unbounded_component:
  fixes S :: 'a :: euclidean_space set
  assumes bounded (¬S)
  shows  $\exists x. x \in S \wedge \neg \text{bounded} (\text{connected\_component\_set } S \ x)$ 
proof -
  obtain i :: 'a where i: i ∈ Basis
    using nonempty_Basis by blast
  obtain B where B: B > 0 ∧ ¬S ⊆ ball 0 B
    using bounded_subset_ballD [OF assms, of 0] by auto
  then have *:  $\bigwedge x. B \leq \text{norm } x \implies x \in S$ 
    by (force simp: ball_def dist_norm)
  have bounded_inner:  $\neg \text{bounded} \{x. \text{inner } i \ x \geq B\}$ 
  proof (clarsimp simp: bounded_def dist_norm)
    fix e x
    show  $\exists y. B \leq i \cdot y \wedge \neg \text{norm } (x - y) \leq e$ 
    using i
    by (rule_tac x = x + (max B e + 1 + |i · x|) *R i in exI) (auto simp:
inner_right_distrib)
  qed
  have §:  $\bigwedge x. B \leq i \cdot x \implies x \in S$ 
    using * Basis_le_norm [OF i] by (metis abs_ge_self inner_commute order_trans)
  have {x. B ≤ i · x} ⊆ connected_component_set S (B *R i)
    by (intro connected_component_maximal) (auto simp: i intro: convex_connected
convex_halfspace_ge [of B] §)
  then have  $\neg \text{bounded} (\text{connected\_component\_set } S \ (B *_{\mathbb{R}} i))$ 
    using bounded_subset_unbounded_inner by blast
  moreover have B *R i ∈ S
    by (rule *) (simp add: norm_Basis [OF i])
  ultimately show ?thesis
    by blast
qed

```

```

lemma cobounded_unique_unbounded_component:
  fixes S :: 'a :: euclidean_space set
  assumes bs: bounded (¬S) and 2 ≤ DIM('a)
    and bo:  $\neg \text{bounded} (\text{connected\_component\_set } S \ x)$ 
       $\neg \text{bounded} (\text{connected\_component\_set } S \ y)$ 
  shows connected_component_set S x = connected_component_set S y

```

```

proof -
  obtain i::'a where i: i ∈ Basis
    using nonempty_Basis by blast
  obtain B where B: B>0 -S ⊆ ball 0 B
    using bounded_subset_ballD [OF bs, of 0] by auto
  then have *: ⋀x. B ≤ norm x ⟹ x ∈ S
    by (force simp: ball_def dist_norm)
  obtain x' where x': connected_component S x x' norm x' > B
    using bo [unfolded bounded_def dist_norm, simplified, rule_format]
    by (metis diff_zero norm_minus_commute not_less)
  obtain y' where y': connected_component S y y' norm y' > B
    using bo [unfolded bounded_def dist_norm, simplified, rule_format]
    by (metis diff_zero norm_minus_commute not_less)
  have x'y': connected_component S x' y'
    unfolding connected_component_def
  proof (intro exI conjI)
    show connected (- ball 0 B :: 'a set)
      using assms by (auto intro: connected_complement_bounded_convex)
    qed (use x' y' dist_norm * in auto)
  show ?thesis
  proof (rule connected_component_eq)
    show x ∈ connected_component_set S y
      using x' y' x'y'
      by (metis (no_types) connected_component_eq_eq connected_component_in
mem_Collect_eq)
    qed
  qed

```

```

lemma cobounded_unbounded_components:
  fixes S :: 'a :: euclidean_space set
  shows bounded (-S) ⟹ ∃ c. c ∈ components S ∧ ¬bounded c
  by (metis cobounded_unbounded_component components_def imageI)

```

```

lemma cobounded_unique_unbounded_components:
  fixes S :: 'a :: euclidean_space set
  shows ‖bounded (-S); c ∈ components S; ¬bounded c; c' ∈ components S;
¬bounded c'; 2 ≤ DIM('a)‗ ⟹ c' = c
  unfolding components_iff
  by (metis cobounded_unique_unbounded_component)

```

```

lemma cobounded_has_bounded_component:
  fixes S :: 'a :: euclidean_space set
  assumes bounded (-S) ∧ connected S 2 ≤ DIM('a)
  obtains C where C ∈ components S bounded C
  by (meson cobounded_unique_unbounded_components connected_eq_connected_components_eq
assms)

```

5.5.24 The *inside* and *outside* of a Set

The *inside* comprises the points in a bounded connected component of the set's complement. The *outside* comprises the points in unbounded connected component of the complement.

definition *inside* **where**

$$\text{inside } S \equiv \{x. (x \notin S) \wedge \text{bounded}(\text{connected\_component\_set } (- S) x)\}$$

definition *outside* **where**

$$\text{outside } S \equiv -S \cap \{x. \neg \text{bounded}(\text{connected\_component\_set } (- S) x)\}$$

lemma *outside*: $\text{outside } S = \{x. \neg \text{bounded}(\text{connected\_component\_set } (- S) x)\}$

by (*auto simp: outside_def*) (*metis Compl_iff bounded_empty connected_component_eq_empty*)

lemma *inside_no_overlap* [*simp*]: $\text{inside } S \cap S = \{\}$

by (*auto simp: inside_def*)

lemma *outside_no_overlap* [*simp*]:

$$\text{outside } S \cap S = \{\}$$

by (*auto simp: outside_def*)

lemma *inside_Int_outside* [*simp*]: $\text{inside } S \cap \text{outside } S = \{\}$

by (*auto simp: inside_def outside_def*)

lemma *inside_Un_outside* [*simp*]: $\text{inside } S \cup \text{outside } S = (- S)$

by (*auto simp: inside_def outside_def*)

lemma *inside_eq_outside*:

$$\text{inside } S = \text{outside } S \longleftrightarrow S = \text{UNIV}$$

by (*auto simp: inside_def outside_def*)

lemma *inside_outside*: $\text{inside } S = (- (S \cup \text{outside } S))$

by (*force simp: inside_def outside*)

lemma *outside_inside*: $\text{outside } S = (- (S \cup \text{inside } S))$

by (*auto simp: inside_outside*) (*metis IntI equals0D outside_no_overlap*)

lemma *union_with_inside*: $S \cup \text{inside } S = - \text{outside } S$

by (*auto simp: inside_outside*) (*simp add: outside_inside*)

lemma *union_with_outside*: $S \cup \text{outside } S = - \text{inside } S$

by (*simp add: inside_outside*)

lemma *outside_mono*: $S \subseteq T \implies \text{outside } T \subseteq \text{outside } S$

by (*auto simp: outside bounded_subset connected_component_mono*)

lemma *inside_mono*: $S \subseteq T \implies \text{inside } S - T \subseteq \text{inside } T$

by (*auto simp: inside_def bounded_subset connected_component_mono*)


```

lemma segment_bound_lemma:
  fixes  $u::real$ 
  assumes  $x \geq B$   $y \geq B$   $0 \leq u$   $u \leq 1$ 
  shows  $(1 - u) * x + u * y \geq B$ 
proof -
  obtain  $dx$   $dy$  where  $dx \geq 0$   $dy \geq 0$   $x = B + dx$   $y = B + dy$ 
  using assms by auto (metis add.commute diff_add_cancel)
  with  $\langle 0 \leq u \rangle \langle u \leq 1 \rangle$  show ?thesis
  by (simp add: add_increasing2 mult_left_le field_simps)
qed

lemma cobounded_outside:
  fixes  $S :: 'a :: real\_normed\_vector\_set$ 
  assumes bounded  $S$  shows bounded  $(- \text{outside } S)$ 
proof -
  obtain  $B$  where  $B > 0$   $S \subseteq \text{ball } 0 B$ 
  using bounded_subset_ballD [OF assms, of 0] by auto
  { fix  $x::'a$  and  $C::real$ 
    assume Bno:  $B \leq \text{norm } x$  and  $C: 0 < C$ 
    have  $\exists y. \text{connected\_component } (- S) x y \wedge \text{norm } y > C$ 
    proof (cases  $x = 0$ )
      case True with  $B$  Bno show ?thesis by force
    next
      case False
      have closed_segment  $x (((B + C) / \text{norm } x) *_R x) \subseteq - \text{ball } 0 B$ 
      proof
        fix  $w$ 
        assume  $w \in \text{closed\_segment } x (((B + C) / \text{norm } x) *_R x)$ 
        then obtain  $u$  where
           $w = (1 - u + u * ((B + C) / \text{norm } x) *_R x)$   $0 \leq u$   $u \leq 1$ 
        by (auto simp add: closed_segment_def real_vector_class.scaleR_add_left
            [symmetric])
        with False  $B$   $C$  have  $B \leq (1 - u) * \text{norm } x + u * (B + C)$ 
        using segment_bound_lemma [of  $B$   $\text{norm } x$   $B + C$   $u$ ] Bno
        by simp
        with False  $B$   $C$  show  $w \in - \text{ball } 0 B$ 
        using distrib_right [of  $u$   $u$   $\text{norm } x$ ]
        by (simp add: ball_def w not_less)
      qed
      qed
    also have  $\dots \subseteq -S$ 
    by (simp add: B)
    finally have  $\exists T. \text{connected } T \wedge T \subseteq -S \wedge x \in T \wedge ((B + C) / \text{norm } x) *_R x \in T$ 
    by (rule_tac  $x = \text{closed\_segment } x (((B + C) / \text{norm } x) *_R x)$  in exI) simp
    with False  $B$ 
    show ?thesis
    by (rule_tac  $x = ((B + C) / \text{norm } x) *_R x$  in exI) (simp add: connected_component_def)
  qed
}

```

```

then show ?thesis
  apply (simp add: outside_def assms)
  apply (rule bounded_subset [OF bounded_ball [of 0 B]])
  apply (force simp: dist_norm not_less bounded_pos)
  done
qed

```

```

lemma unbounded_outside:
  fixes S :: 'a::{real_normed_vector, perfect_space} set
  shows bounded S  $\implies$   $\neg$  bounded(outside S)
  using cobounded_imp_unbounded cobounded_outside by blast

```

```

lemma bounded_inside:
  fixes S :: 'a::{real_normed_vector, perfect_space} set
  shows bounded S  $\implies$  bounded(inside S)
  by (simp add: bounded_Int cobounded_outside inside_outside)

```

```

lemma connected_outside:
  fixes S :: 'a::euclidean_space set
  assumes bounded S  $\wedge$  2  $\leq$  DIM('a)
  shows connected(outside S)
  apply (clarsimp simp add: connected_iff_connected_component outside)
  apply (rule_tac S=connected_component_set (- S) x in connected_component_of_subset)
  apply (metis (no_types) assms cobounded_unbounded_component cobounded_unique_unbounded_component
connected_component_eq_eq connected_component_idemp double_complement mem_Collect_eq)
  by (simp add: Collect_mono connected_component_eq)

```

```

lemma outside_connected_component_lt:
  outside S = {x.  $\forall$  B.  $\exists$  y. B < norm(y)  $\wedge$  connected_component (- S) x y}
  apply (auto simp: outside_def dist_norm)
  apply (metis diff_0 norm_minus_cancel not_less)
  by (metis less_diff_eq norm_minus_commute norm_triangle_ineq2 order.trans
pinf(6))

```

```

lemma outside_connected_component_le:
  outside S = {x.  $\forall$  B.  $\exists$  y. B  $\leq$  norm(y)  $\wedge$  connected_component (- S) x y}
  apply (simp add: outside_connected_component_lt Set.set_eq_iff)
  by (meson gt_ex leD le_less_linear less_imp_le order.trans)

```

```

lemma not_outside_connected_component_lt:
  fixes S :: 'a::euclidean_space set
  assumes S: bounded S and 2  $\leq$  DIM('a)
  shows  $\neg$  (outside S) = {x.  $\forall$  B.  $\exists$  y. B < norm(y)  $\wedge$   $\neg$  connected_component
(- S) x y}
proof -
  obtain B::real where B: 0 < B and Bno:  $\bigwedge$  x. x  $\in$  S  $\implies$  norm x  $\leq$  B
  using S [simplified bounded_pos] by auto
  { fix y::'a and z::'a
    assume yz: B < norm z B < norm y

```

```

have connected_component ( $\neg$  cball 0 B) y z
using assms yz
by (force simp: dist_norm intro: connected_componentI [OF __ subset_refl]
connected_complement_bounded_convex)
then have connected_component ( $\neg$  S) y z
by (metis connected_component_of_subset Bno Compl_anti_mono mem_cball_0
subset_iff)
} note cyz = this
show ?thesis
apply (auto simp: outside_bounded_pos)
apply (metis Compl_iff bounded_iff cobounded_imp_unbounded mem_Collect_eq
not_le)
by (metis B connected_component_trans cyz not_le)
qed

```

```

lemma not_outside_connected_component_le:
fixes S :: 'a::euclidean_space set
assumes S: bounded S  $2 \leq \text{DIM}('a)$ 
shows  $\neg$  (outside S) =  $\{x. \forall B. \exists y. B \leq \text{norm}(y) \wedge \neg \text{connected\_component } (\neg$ 
S) x y $\}$ 
apply (auto intro: less_imp_le simp: not_outside_connected_component_lt [OF
assms])
by (meson gt_ex less_le_trans)

```

```

lemma inside_connected_component_lt:
fixes S :: 'a::euclidean_space set
assumes S: bounded S  $2 \leq \text{DIM}('a)$ 
shows inside S =  $\{x. (x \notin S) \wedge (\forall B. \exists y. B < \text{norm}(y) \wedge \neg \text{connected\_component } (\neg$ 
S) x y $\}$ 
by (auto simp: inside_outside not_outside_connected_component_lt [OF assms])

```

```

lemma inside_connected_component_le:
fixes S :: 'a::euclidean_space set
assumes S: bounded S  $2 \leq \text{DIM}('a)$ 
shows inside S =  $\{x. (x \notin S) \wedge (\forall B. \exists y. B \leq \text{norm}(y) \wedge \neg \text{connected\_component } (\neg$ 
S) x y $\}$ 
by (auto simp: inside_outside not_outside_connected_component_le [OF assms])

```

```

lemma inside_subset:
assumes connected U and  $\neg$  bounded U and  $T \cup U = \neg$  S
shows inside S  $\subseteq$  T
apply (auto simp: inside_def)
by (metis bounded_subset [of connected_component_set ( $\neg$  S) _] connected_component_maximal
Compl_iff Un_iff assms subsetI)

```

```

lemma frontier_not_empty:
fixes S :: 'a :: real_normed_vector set
shows  $\llbracket S \neq \{\}; S \neq \text{UNIV} \rrbracket \implies \text{frontier } S \neq \{\}$ 
using connected_Int_frontier [of UNIV S] by auto

```

```

lemma frontier_eq_empty:
  fixes S :: 'a :: real_normed_vector set
  shows frontier S = {}  $\longleftrightarrow$  S = {}  $\vee$  S = UNIV
using frontier_UNIV frontier_empty frontier_not_empty by blast

lemma frontier_of_connected_component_subset:
  fixes S :: 'a :: real_normed_vector set
  shows frontier (connected_component_set S x)  $\subseteq$  frontier S
proof -
  { fix y
    assume y1: y  $\in$  closure (connected_component_set S x)
    and y2: y  $\notin$  interior (connected_component_set S x)
    have y  $\in$  closure S
    using y1 closure_mono connected_component_subset by blast
    moreover have z  $\in$  interior (connected_component_set S x)
    if 0 < e ball y e  $\subseteq$  interior S dist y z < e for e z
    proof -
      have ball y e  $\subseteq$  connected_component_set S y
      using connected_component_maximal that interior_subset
      by (metis centre_in_ball connected_ball subset_trans)
      then show ?thesis
      using y1 apply (simp add: closure_approachable open_contains_ball_eq
[OF open_interior])
      by (metis connected_component_eq dist_commute mem_Collect_eq mem_ball
mem_interior subsetD <0 < e> y2)
    qed
    then have y  $\notin$  interior S
    using y2 by (force simp: open_contains_ball_eq [OF open_interior])
    ultimately have y  $\in$  frontier S
    by (auto simp: frontier_def)
  }
  then show ?thesis by (auto simp: frontier_def)
qed

lemma frontier_Union_subset_closure:
  fixes F :: 'a :: real_normed_vector set set
  shows frontier ( $\bigcup$  F)  $\subseteq$  closure ( $\bigcup$  t  $\in$  F. frontier t)
proof -
  have  $\exists y \in F. \exists y \in \text{frontier } y. \text{dist } y x < e$ 
  if T  $\in$  F y  $\in$  T dist y x < e
  x  $\notin$  interior ( $\bigcup$  F) 0 < e for x y  $\in$  T
proof (cases x  $\in$  T)
  case True with that show ?thesis
  by (metis Diff_iff Sup_upper closure_subset contra_subsetD dist_self frontier_def interior_mono)
next
  case False
  have 1: closed_segment x y  $\cap$  T  $\neq$  {}

```

```

    using ⟨y ∈ T⟩ by blast
  have 2: closed_segment x y - T ≠ {}
    using False by blast
  obtain c where c ∈ closed_segment x y c ∈ frontier T
    using False connected_Int_frontier [OF connected_segment 1 2] by auto
  then show ?thesis
  proof -
    have norm (y - x) < e
      by (metis dist_norm ⟨dist y x < e⟩)
    moreover have norm (c - x) ≤ norm (y - x)
      by (simp add: ⟨c ∈ closed_segment x y⟩ segment_bound(1))
    ultimately have norm (c - x) < e
      by linarith
    then show ?thesis
      by (metis (no_types) ⟨c ∈ frontier T⟩ dist_norm that(1))
  qed
qed
then show ?thesis
  by (fastforce simp add: frontier_def closure_approachable)
qed

lemma frontier_Union_subset:
  fixes F :: 'a::real_normed_vector set set
  shows finite F ⟹ frontier(⋃ F) ⊆ (⋃ t ∈ F. frontier t)
by (rule order_trans [OF frontier_Union_subset_closure])
  (auto simp: closure_subset_eq)

lemma frontier_of_components_subset:
  fixes S :: 'a::real_normed_vector set
  shows C ∈ components S ⟹ frontier C ⊆ frontier S
by (metis Path_Connected.frontier_of_connected_component_subset components_iff)

lemma frontier_of_components_closed_complement:
  fixes S :: 'a::real_normed_vector set
  shows ⟦closed S; C ∈ components (- S)⟧ ⟹ frontier C ⊆ S
  using frontier_complement frontier_of_components_subset frontier_subset_eq
by blast

lemma frontier_minimal_separating_closed:
  fixes S :: 'a::real_normed_vector set
  assumes closed S
    and nconn: ¬ connected(- S)
    and C: C ∈ components (- S)
    and conn: ⋀ T. ⟦closed T; T ⊂ S⟧ ⟹ connected(- T)
  shows frontier C = S
proof (rule ccontr)
  assume frontier C ≠ S
  then have frontier C ⊂ S
    using frontier_of_components_closed_complement [OF ⟨closed S⟩ C] by blast

```

```

then have connected(¬ (frontier C))
  by (simp add: conn)
have ¬ connected(¬ (frontier C))
  unfolding connected_def not_not
proof (intro exI conjI)
  show open C
    using C ⟨closed S⟩ open_components by blast
  show open (¬ closure C)
    by blast
  show C ∩ ¬ closure C ∩ ¬ frontier C = {}
    using closure_subset by blast
  show C ∩ ¬ frontier C ≠ {}
    using C ⟨open C⟩ components_eq frontier_disjoint_eq by fastforce
  show ¬ frontier C ⊆ C ∪ ¬ closure C
    by (simp add: ⟨open C⟩ closed_Compl frontier_closures)
  then show ¬ closure C ∩ ¬ frontier C ≠ {}
    by (metis (no_types, lifting) C Compl_subset_Compl_iff ⟨frontier C ⊆
S⟩ compl_sup frontier_closures in_components_subset psubsetE sup.absorb_iff2
sup.boundedE sup_bot.right_neutral sup_inf_absorb)
  qed
  then show False
    using ⟨connected (¬ frontier C)⟩ by blast
qed

```

```

lemma connected_component_UNIV [simp]:
  fixes x :: 'a::real_normed_vector
  shows connected_component_set UNIV x = UNIV
using connected_iff_eq_connected_component_set [of UNIV::'a set] connected_UNIV
by auto

```

```

lemma connected_component_eq_UNIV:
  fixes x :: 'a::real_normed_vector
  shows connected_component_set s x = UNIV ⟷ s = UNIV
using connected_component_in connected_component_UNIV by blast

```

```

lemma components_UNIV [simp]: components UNIV = {UNIV :: 'a::real_normed_vector
set}
  by (auto simp: components_eq_sing_iff)

```

```

lemma interior_inside_frontier:
  fixes S :: 'a::real_normed_vector set
  assumes bounded S
  shows interior S ⊆ inside (frontier S)
proof -
  { fix x y
    assume x: x ∈ interior S and y: y ∉ S
    and cc: connected_component (¬ frontier S) x y
    have connected_component_set (¬ frontier S) x ∩ frontier S ≠ {}
    proof (rule connected_Int_frontier; simp add: set_eq_iff)

```

```

    show  $\exists u. \text{connected\_component } (- \text{frontier } S) \ x \ u \wedge u \in S$ 
    by (meson cc connected_component_in connected_component_refl_eq interior_subset subsetD x)
    show  $\exists u. \text{connected\_component } (- \text{frontier } S) \ x \ u \wedge u \notin S$ 
    using y cc by blast
  qed
  then have bounded (connected_component_set (- frontier S) x)
    using connected_component_in by auto
}
then show ?thesis
  apply (auto simp: inside_def frontier_def)
  apply (rule classical)
  apply (rule bounded_subset [OF assms], blast)
  done
qed

lemma inside_empty [simp]: inside {} = ({} :: 'a :: {real_normed_vector, perfect_space} set)
  by (simp add: inside_def)

lemma outside_empty [simp]: outside {} = (UNIV :: 'a :: {real_normed_vector, perfect_space} set)
  using inside_empty inside_Un_outside by blast

lemma inside_same_component:
   $\llbracket \text{connected\_component } (- S) \ x \ y; x \in \text{inside } S \rrbracket \implies y \in \text{inside } S$ 
  using connected_component_eq connected_component_in
  by (fastforce simp add: inside_def)

lemma outside_same_component:
   $\llbracket \text{connected\_component } (- S) \ x \ y; x \in \text{outside } S \rrbracket \implies y \in \text{outside } S$ 
  using connected_component_eq connected_component_in
  by (fastforce simp add: outside_def)

lemma convex_in_outside:
  fixes S :: 'a :: {real_normed_vector, perfect_space} set
  assumes S: convex S and z:  $z \notin S$ 
  shows  $z \in \text{outside } S$ 
proof (cases S={})
  case True then show ?thesis by simp
next
  case False then obtain a where  $a \in S$  by blast
  with z have zna:  $z \neq a$  by auto
  { assume bounded (connected_component_set (- S) z)
    with bounded_pos_less obtain B where  $B > 0$  and B:  $\bigwedge x. \text{connected\_component } (- S) \ z \ x \implies \text{norm } x < B$ 
    by (metis mem_Collect_eq)
    define C where  $C = (B + 1 + \text{norm } z) / \text{norm } (z - a)$ 
    have  $C > 0$ 
```

```

using ⟨0 < B⟩ zna by (simp add: C_def field_split_simps add_strict_increasing)
have |norm (z + C *R (z-a)) - norm (C *R (z-a))| ≤ norm z
  by (metis add_diff_cancel norm_triangle_ineq3)
moreover have norm (C *R (z-a)) > norm z + B
  using zna ⟨B>0⟩ by (simp add: C_def le_max_iff_disj)
ultimately have C: norm (z + C *R (z-a)) > B by linarith
{ fix u::real
  assume u: 0 ≤ u ≤ 1 and ins: (1 - u) *R z + u *R (z + C *R (z - a)) ∈ S
  then have Cpos: 1 + u * C > 0
    by (meson ⟨0 < C⟩ add_pos_nonneg less_eq_real_def zero_le_mult_iff
zero_less_one)
  then have *: (1 / (1 + u * C)) *R z + (u * C / (1 + u * C)) *R z = z
    by (simp add: scaleR_add_left [symmetric] field_split_simps)
  then have False
    using convexD_alt [OF S ⟨a ∈ S⟩ ins, of 1/(u*C + 1)] ⟨C>0⟩ ⟨z ∉ S⟩
Cpos u
    by (simp add: * field_split_simps)
  } note contra = this
have connected_component (- S) z (z + C *R (z-a))
proof (rule connected_componentI [OF connected_segment])
  show closed_segment z (z + C *R (z - a)) ⊆ - S
    using contra by (force simp add: closed_segment_def)
qed auto
then have False
  using zna B [of z + C *R (z-a)] C
  by (auto simp: field_split_simps max_mult_distrib_right)
}
then show ?thesis
  by (auto simp: outside_def z)
qed

```

```

lemma outside_convex:
  fixes S :: 'a :: {real_normed_vector, perfect_space} set
  assumes convex S
  shows outside S = - S
  by (metis ComplD assms convex_in_outside equalityI inside_Un_outside subsetI
sup.cobounded2)

```

```

lemma outside_singleton [simp]:
  fixes x :: 'a :: {real_normed_vector, perfect_space}
  shows outside {x} = -{x}
  by (auto simp: outside_convex)

```

```

lemma inside_convex:
  fixes S :: 'a :: {real_normed_vector, perfect_space} set
  shows convex S ⟹ inside S = {}
  by (simp add: inside_outside outside_convex)

```

```

lemma inside_singleton [simp]:

```



```

fixes  $x :: 'a :: \{\text{real\_normed\_vector}, \text{perfect\_space}\}$ 
shows  $\text{inside } \{x\} = \{\}$ 
by (auto simp: inside_convex)

lemma outside_subset_convex:
fixes  $S :: 'a :: \{\text{real\_normed\_vector}, \text{perfect\_space}\}$  set
shows  $\llbracket \text{convex } T; S \subseteq T \rrbracket \implies - T \subseteq \text{outside } S$ 
using outside_convex outside_mono by blast

lemma outside_Un_outside_Un:
fixes  $S :: 'a :: \text{real\_normed\_vector}$  set
assumes  $S \cap \text{outside}(T \cup U) = \{\}$ 
shows  $\text{outside}(T \cup U) \subseteq \text{outside}(T \cup S)$ 
proof
fix  $x$ 
assume  $x: x \in \text{outside}(T \cup U)$ 
have  $Y \subseteq - S$  if  $\text{connected } Y$   $Y \subseteq - T$   $Y \subseteq - U$   $x \in Y$   $u \in Y$  for  $u \in Y$ 
proof -
have  $Y \subseteq \text{connected\_component\_set } (- (T \cup U))$   $x$ 
by (simp add: connected_component_maximal that)
also have  $\dots \subseteq \text{outside}(T \cup U)$ 
by (metis (mono_tags, lifting) Collect_mono mem_Collect_eq outside_outside_same_component x)
finally have  $Y \subseteq \text{outside}(T \cup U)$  .
with assms show ?thesis by auto
qed
with  $x$  show  $x \in \text{outside}(T \cup S)$ 
by (simp add: outside_connected_component_lt connected_component_def)
meson
qed

lemma outside_frontier_misses_closure:
fixes  $S :: 'a :: \text{real\_normed\_vector}$  set
assumes bounded S
shows  $\text{outside}(\text{frontier } S) \subseteq - \text{closure } S$ 
unfolding outside_inside boolean_algebra_class.compl_le_compl_iff
proof -
{ assume  $\text{interior } S \subseteq \text{inside } (\text{frontier } S)$ 
hence  $\text{interior } S \cup \text{inside } (\text{frontier } S) = \text{inside } (\text{frontier } S)$ 
by (simp add: subset_Un_eq)
then have  $\text{closure } S \subseteq \text{frontier } S \cup \text{inside } (\text{frontier } S)$ 
using frontier_def by auto
}
then show  $\text{closure } S \subseteq \text{frontier } S \cup \text{inside } (\text{frontier } S)$ 
using interior_inside_frontier [OF assms] by blast
qed

lemma outside_frontier_eq_complement_closure:
fixes  $S :: 'a :: \{\text{real\_normed\_vector}, \text{perfect\_space}\}$  set

```

```

    assumes bounded S convex S
    shows outside(frontier S) = - closure S
  by (metis Diff_subset assms convex_closure frontier_def outside_frontier_misses_closure
      outside_subset_convex subset_antisym)

```

```

lemma inside_frontier_eq_interior:
  fixes S :: 'a :: {real_normed_vector, perfect_space} set
  shows  $\llbracket \text{bounded } S; \text{convex } S \rrbracket \implies \text{inside}(\text{frontier } S) = \text{interior } S$ 
  apply (simp add: inside_outside outside_frontier_eq_complement_closure)
  using closure_subset interior_subset
  apply (auto simp: frontier_def)
  done

```

```

lemma open_inside:
  fixes S :: 'a::real_normed_vector set
  assumes closed S
  shows open (inside S)
proof -
  { fix x assume x: x ∈ inside S
    have open (connected_component_set (- S) x)
      using assms open_connected_component by blast
    then obtain e where e: e>0 and e:  $\bigwedge y. \text{dist } y \ x < e \longrightarrow \text{connected\_component}(- S) \ x \ y$ 
      using dist_not_less_zero
      apply (simp add: open_dist)
      by (metis (no_types, lifting) Compl_iff connected_component_refl_eq inside_def mem_Collect_eq x)
    then have  $\exists e>0. \text{ball } x \ e \subseteq \text{inside } S$ 
      by (metis e dist_commute inside_same_component mem_ball subsetI x)
  }
  then show ?thesis
    by (simp add: open_contains_ball)
qed

```

```

lemma open_outside:
  fixes S :: 'a::real_normed_vector set
  assumes closed S
  shows open (outside S)
proof -
  { fix x assume x: x ∈ outside S
    have open (connected_component_set (- S) x)
      using assms open_connected_component by blast
    then obtain e where e: e>0 and e:  $\bigwedge y. \text{dist } y \ x < e \longrightarrow \text{connected\_component}(- S) \ x \ y$ 
      using dist_not_less_zero x
      by (auto simp add: open_dist outside_def intro: connected_component_refl)
    then have  $\exists e>0. \text{ball } x \ e \subseteq \text{outside } S$ 
      by (metis e dist_commute outside_same_component mem_ball subsetI x)
  }

```

```

then show ?thesis
  by (simp add: open_contains_ball)
qed

lemma closure_inside_subset:
  fixes  $S :: 'a::\text{real\_normed\_vector}$  set
  assumes closed  $S$ 
  shows  $\text{closure}(\text{inside } S) \subseteq S \cup \text{inside } S$ 
by (metis assms closure_minimal open_closed open_outside sup.cobounded2 union_with_inside)

lemma frontier_inside_subset:
  fixes  $S :: 'a::\text{real\_normed\_vector}$  set
  assumes closed  $S$ 
  shows  $\text{frontier}(\text{inside } S) \subseteq S$ 
proof –
  have  $\text{closure}(\text{inside } S) \cap - \text{inside } S = \text{closure}(\text{inside } S) - \text{interior}(\text{inside } S)$ 
  by (metis (no_types) Diff_Compl assms closure_closed interior_closure open_closed open_inside)
  moreover have  $- \text{inside } S \cap - \text{outside } S = S$ 
  by (metis (no_types) compl_sup double_compl inside_Un_outside)
  moreover have  $\text{closure}(\text{inside } S) \subseteq - \text{outside } S$ 
  by (metis (no_types) assms closure_inside_subset union_with_inside)
  ultimately have  $\text{closure}(\text{inside } S) - \text{interior}(\text{inside } S) \subseteq S$ 
  by blast
  then show ?thesis
  by (simp add: frontier_def open_inside interior_open)
qed

lemma closure_outside_subset:
  fixes  $S :: 'a::\text{real\_normed\_vector}$  set
  assumes closed  $S$ 
  shows  $\text{closure}(\text{outside } S) \subseteq S \cup \text{outside } S$ 
by (metis assms closed_open closure_minimal inside_outside open_inside sup_ge2)

lemma frontier_outside_subset:
  fixes  $S :: 'a::\text{real\_normed\_vector}$  set
  assumes closed  $S$ 
  shows  $\text{frontier}(\text{outside } S) \subseteq S$ 
  unfolding frontier_def
  by (metis Diff_subset_conv assms closure_outside_subset interior_eq open_outside sup_aci(1))

lemma inside_complement_unbounded_connected_empty:
   $\llbracket \text{connected } (- S); \neg \text{bounded } (- S) \rrbracket \implies \text{inside } S = \{\}$ 
  using inside_subset by blast

lemma inside_bounded_complement_connected_empty:
  fixes  $S :: 'a::\{\text{real\_normed\_vector}, \text{perfect\_space}\}$  set
  shows  $\llbracket \text{connected } (- S); \text{bounded } S \rrbracket \implies \text{inside } S = \{\}$ 

```

by (metis inside_complement_unbounded_connected_empty_cobounded_imp_unbounded)

lemma *inside_inside*:

assumes $S \subseteq \text{inside } T$

shows $\text{inside } S - T \subseteq \text{inside } T$

unfolding *inside_def*

proof *clarify*

fix x

assume $x: x \notin T \ x \notin S$ and $bo: \text{bounded } (\text{connected\_component\_set } (- S) x)$

show $\text{bounded } (\text{connected\_component\_set } (- T) x)$

proof (cases $S \cap \text{connected\_component\_set } (- T) x = \{\}$)

case *True* then show ?thesis

by (metis bounded_subset [OF bo] compl_le_compl_iff connected_component_idemp
connected_component_mono disjoint_eq_subset_Compl double_compl)

next

case *False*

then obtain y where $y: y \in S \ y \in \text{connected\_component\_set } (- T) x$

by (meson disjoint_iff)

then have $\text{bounded } (\text{connected\_component\_set } (- T) y)$

using *assms* [unfolding *inside_def*] by blast

with y show ?thesis

by (metis connected_component_eq)

qed

qed

lemma *inside_inside_subset*: $\text{inside}(\text{inside } S) \subseteq S$

using *inside_inside union_with_outside* by fastforce

lemma *inside_outside_intersect_connected*:

$\llbracket \text{connected } T; \text{inside } S \cap T \neq \{\}; \text{outside } S \cap T \neq \{\} \rrbracket \implies S \cap T \neq \{\}$

apply (simp add: *inside_def outside_def* *ex_in_conv* [symmetric] *disjoint_eq_subset_Compl*,
clarify)

by (metis (no_types, opaque_lifting) Compl_anti_mono connected_component_eq
connected_component_maximal contra_subsetD double_compl)

lemma *outside_bounded_nonempty*:

fixes $S :: 'a :: \{\text{real\_normed\_vector}, \text{perfect\_space}\}$ set

assumes $\text{bounded } S$ shows $\text{outside } S \neq \{\}$

by (metis (no_types, lifting) Collect_empty_eq Collect_mem_eq Compl_eq_Diff_UNIV
Diff_cancel

$\text{Diff\_disjoint UNIV\_I assms ball\_eq\_empty bounded\_diff}$
cobounded_outside convex_ball

$\text{double\_complement order_refl outside\_convex outside\_def}$)

lemma *outside_compact_in_open*:

fixes $S :: 'a :: \{\text{real\_normed\_vector}, \text{perfect\_space}\}$ set

assumes $S: \text{compact } S$ and $T: \text{open } T$ and $S \subseteq T$ $T \neq \{\}$

shows $\text{outside } S \cap T \neq \{\}$

proof –

```

have outside S ≠ {}
  by (simp add: compact_imp_bounded outside_bounded_nonempty S)
with assms obtain a b where a: a ∈ outside S and b: b ∈ T by auto
show ?thesis
proof (cases a ∈ T)
  case True with a show ?thesis by blast
next
  case False
    have front: frontier T ⊆ - S
      using ⟨S ⊆ T⟩ frontier_disjoint_eq T by auto
    { fix γ
      assume path γ and pimg_sbs: path_image γ - {pathfinish γ} ⊆ interior
    (- T)
      and pf: pathfinish γ ∈ frontier T and ps: pathstart γ = a
      define c where c = pathfinish γ
      have c ∈ -S unfolding c_def using front pf by blast
      moreover have open (-S) using S compact_imp_closed by blast
      ultimately obtain ε::real where ε > 0 and ε: cball c ε ⊆ -S
        using open_contains_cball[of -S] S by blast
      then obtain d where d ∈ T and d: dist d c < ε
        using closure_approachable [of c T] pf unfolding c_def
        by (metis Diff_iff frontier_def)
      then have d ∈ -S using ε
        using dist_commute by (metis contra_subsetD mem_cball not_le
not_less_iff_gr_or_eq)
      have pimg_sbs_cos: path_image γ ⊆ -S
        using ⟨c ∈ -S⟩ ⟨S ⊆ T⟩ c_def interior_subset pimg_sbs by fastforce
      have closed_segment c d ≤ cball c ε
        by (metis ⟨0 < ε⟩ centre_in_cball closed_segment_subset convex_cball d
dist_commute less_eq_real_def mem_cball)
      with ε have closed_segment c d ⊆ -S by blast
      moreover have con_gcd: connected (path_image γ ∪ closed_segment c d)
        by (rule connected_Un) (auto simp: c_def ⟨path γ⟩ connected_path_image)
      ultimately have connected_component (- S) a d
        unfolding connected_component_def using pimg_sbs_cos ps by blast
      then have outside S ∩ T ≠ {}
        using outside_same_component [OF _ a] by (metis IntI ⟨d ∈ T⟩
empty_iff)
    } note * = this
    have pal: pathstart (linepath a b) ∈ closure (- T)
      by (auto simp: False closure_def)
    show ?thesis
      by (rule exists_path_subpath_to_frontier [OF path_linepath pal _ *]) (auto
simp: b)
  qed
qed

```

lemma *inside_inside_compact_connected:*
fixes $S :: 'a :: \text{euclidean\_space}$ *set*

```

    assumes  $S$ : closed  $S$  and  $T$ : compact  $T$  and connected  $T \subseteq \text{inside } T$ 
    shows  $\text{inside } S \subseteq \text{inside } T$ 
  proof (cases  $\text{inside } T = \{\}$ )
    case True with assms show ?thesis by auto
  next
    case False
    consider  $\text{DIM}('a) = 1 \mid \text{DIM}('a) \geq 2$ 
    using antisym not_less_eq_eq by fastforce
    then show ?thesis
  proof cases
    case 1 then show ?thesis
      using connected_convex_1_gen assms False inside_convex by blast
  next
    case 2
    have bounded  $S$ 
    using assms by (meson bounded_inside bounded_subset compact_imp_bounded)
    then have coms: compact  $S$ 
      by (simp add: S compact_eq_bounded_closed)
    then have bst: bounded  $(S \cup T)$ 
      by (simp add: compact_imp_bounded T)
    then obtain  $r$  where  $0 < r$  and  $r$ :  $S \cup T \subseteq \text{ball } 0 \ r$ 
      using bounded_subset_ballD by blast
    have outst:  $\text{outside } S \cap \text{outside } T \neq \{\}$ 
    proof -
      have  $-\text{ball } 0 \ r \subseteq \text{outside } S$ 
      by (meson convex_ball le_supE outside_subset_convex r)
      moreover have  $-\text{ball } 0 \ r \subseteq \text{outside } T$ 
      by (meson convex_ball le_supE outside_subset_convex r)
      ultimately show ?thesis
      by (metis Compl_subset_Compl_iff Int_subset_iff bounded_ball inf.orderE
        outside_bounded_nonempty outside_no_overlap)
    qed
    have  $S \cap T = \{\}$  using assms
      by (metis disjoint_iff_not_equal inside_no_overlap subsetCE)
    moreover have  $\text{outside } S \cap \text{inside } T \neq \{\}$ 
      by (meson False assms(4) compact_eq_bounded_closed coms open_inside
        outside_compact_in_open T)
    ultimately have  $\text{inside } S \cap T = \{\}$ 
      using inside_outside_intersect_connected [OF  $\langle \text{connected } T \rangle$ , of  $S$ ]
      by (metis 2 compact_eq_bounded_closed coms connected_outside inf.commute
        inside_outside_intersect_connected outst)
    then show ?thesis
      using inside_inside [OF  $\langle S \subseteq \text{inside } T \rangle$ ] by blast
  qed
qed

```

lemma *connected_with_inside*:

```

  fixes  $S :: 'a :: \text{real\_normed\_vector set}$ 
  assumes  $S$ : closed  $S$  and cons: connected  $S$ 

```

```

      shows connected( $S \cup \text{inside } S$ )
    proof (cases  $S \cup \text{inside } S = \text{UNIV}$ )
      case True with assms show ?thesis by auto
    next
      case False
      then obtain b where  $b \notin S$   $b \notin \text{inside } S$  by blast
      have *:  $\exists y T. y \in S \wedge \text{connected } T \wedge a \in T \wedge y \in T \wedge T \subseteq (S \cup \text{inside } S)$ 
        if  $a \in S \cup \text{inside } S$  for a
        using that
      proof
        assume  $a \in S$  then show ?thesis
          by (rule_tac  $x=a$  in exI, rule_tac  $x=\{a\}$  in exI, simp)
      next
        assume  $a: a \in \text{inside } S$ 
        then have ain:  $a \in \text{closure } (\text{inside } S)$ 
          by (simp add: closure_def)
        show ?thesis
          apply (rule exists_path_subpath_to_frontier [OF path_linepath [of a b], of
inside S])
          apply (simp_all add: ain b)
          subgoal for h
            apply (rule_tac  $x=\text{pathfinish } h$  in exI)
            apply (simp add: subsetD [OF frontier_inside_subset [OF S]])
            apply (rule_tac  $x=\text{path\_image } h$  in exI)
            apply (simp add: pathfinish_in_path_image connected_path_image, auto)
            by (metis Diff_single_insert S frontier_inside_subset insert_iff interior_subset subsetD)
          done
        qed
      show ?thesis
        apply (simp add: connected_iff_connected_component)
        apply (clarsimp simp add: connected_component_def dest!: *)
        subgoal for  $x y u u' T t'$ 
          by (rule_tac  $x=(S \cup T \cup t')$  in exI) (auto intro!: connected_Un cons)
        done
      qed
    qed
  qed

```

The proof is virtually the same as that above.

```

lemma connected_with_outside:
  fixes  $S :: 'a :: \text{real\_normed\_vector\_set}$ 
  assumes  $S: \text{closed } S$  and  $\text{cons: connected } S$ 
  shows connected( $S \cup \text{outside } S$ )
proof (cases  $S \cup \text{outside } S = \text{UNIV}$ )
  case True with assms show ?thesis by auto
next
  case False
  then obtain b where  $b \notin S$   $b \notin \text{outside } S$  by blast
  have *:  $\exists y T. y \in S \wedge \text{connected } T \wedge a \in T \wedge y \in T \wedge T \subseteq (S \cup \text{outside } S)$ 
    if  $a \in (S \cup \text{outside } S)$  for a

```

```

using that proof
  assume  $a \in S$  then show ?thesis
    by (rule_tac  $x=a$  in exI, rule_tac  $x=\{a\}$  in exI, simp)
  next
    assume  $a: a \in \text{outside } S$ 
    then have  $\text{ain}: a \in \text{closure } (\text{outside } S)$ 
      by (simp add: closure_def)
    show ?thesis
      apply (rule exists_path_subpath_to_frontier [OF path_linepath [of a b], of
        outside S])
      apply (simp_all add: ain b)
      subgoal for  $h$ 
      apply (rule_tac  $x=\text{pathfinish } h$  in exI)
      apply (simp add: subsetD [OF frontier_outside_subset [OF S]])
      apply (rule_tac  $x=\text{path\_image } h$  in exI)
      apply (simp add: pathfinish_in_path_image connected_path_image, auto)
      by (metis (no_types, lifting) frontier_outside_subset insertE insert_Diff
        interior_eq open_outside pathfinish_in_path_image S subsetCE)
      done
    qed
  show ?thesis
    apply (simp add: connected_iff_connected_component)
    apply (clarsimp simp add: connected_component_def dest!: *)
    subgoal for  $x\ y\ u\ u'\ T\ t'$ 
      by (rule_tac  $x=(S \cup T \cup t')$  in exI) (auto intro!: connected_Un cons)
    done
  qed

lemma inside_inside_eq_empty [simp]:
  fixes  $S :: 'a :: \{\text{real\_normed\_vector}, \text{perfect\_space}\}$  set
  assumes  $S: \text{closed } S$  and  $\text{cons: connected } S$ 
  shows  $\text{inside } (\text{inside } S) = \{\}$ 
  by (metis (no_types) unbounded_outside connected_with_outside [OF assms]
    bounded_Un
    inside_complement_unbounded_connected_empty unbounded_outside
    union_with_outside)

lemma inside_in_components:
   $\text{inside } S \in \text{components } (\neg S) \longleftrightarrow \text{connected}(\text{inside } S) \wedge \text{inside } S \neq \{\}$  (is
?lhs = ?rhs)
proof
  assume  $R: ?rhs$ 
  then have  $\bigwedge x. \llbracket x \in S; x \in \text{inside } S \rrbracket \implies \neg \text{connected } (\text{inside } S)$ 
    by (simp add: inside_outside)
  with  $R$  show ?lhs
    unfolding in_components_maximal
    by (auto intro: inside_same_component connected_componentI)
qed (simp add: in_components_maximal)

```

The proof is like that above.

lemma *outside_in_components*:

outside S $\in$ *components* $(- S) \longleftrightarrow \text{connected}(\text{outside } S) \wedge \text{outside } S \neq \{\}$ (is ?lhs = ?rhs)

proof

assume *R*: ?rhs

then have $\bigwedge x. \llbracket x \in S; x \in \text{outside } S \rrbracket \implies \neg \text{connected } (\text{outside } S)$

by (*meson disjoint_iff outside_no_overlap*)

with *R* **show** ?lhs

unfolding *in_components_maximal*

by (*auto intro: outside_same_component connected_componentI*)

qed (*simp add: in_components_maximal*)

lemma *bounded_unique_outside*:

fixes *S* :: 'a :: euclidean_space set

assumes *bounded S DIM('a) ≥ 2*

shows $(c \in \text{components } (- S) \wedge \neg \text{bounded } c) \longleftrightarrow c = \text{outside } S$

using *assms*

by (*metis cobounded_unique_unbounded_components connected_outside double_compl outside_bounded_nonempty outside_in_components unbounded_outside*)

5.5.25 Condition for an open map's image to contain a ball

proposition *ball_subset_open_map_image*:

fixes *f* :: 'a :: heine_borel $\Rightarrow$ 'b :: {real_normed_vector, heine_borel}

assumes *conf*: *continuous_on* (*closure S*) *f*

and *oint*: *open* (*f* ' *interior S*)

and *le_no*: $\bigwedge z. z \in \text{frontier } S \implies r \leq \text{norm}(f z - f a)$

and *bounded S a* $\in S$ $0 < r$

shows *ball* (*f a*) *r* $\subseteq f$ ' *S*

proof (*cases f* ' *S* = *UNIV*)

case *True* **then show** ?thesis **by** *simp*

next

case *False*

then have *closed* (*frontier* (*f* ' *S*)) *frontier* (*f* ' *S*) $\neq \{\}$

using $\langle a \in S \rangle$ **by** (*auto simp: frontier_eq_empty*)

then obtain *w* **where** *w*: *w* $\in$ *frontier* (*f* ' *S*)

and *dw_le*: $\bigwedge y. y \in \text{frontier } (f \text{ ' } S) \implies \text{norm } (f a - w) \leq \text{norm } (f a - y)$

by (*auto simp add: dist_norm intro: distance_attains_inf [of frontier(f ' S) f a]*)

then obtain ξ **where** ξ : $\bigwedge n. \xi n \in f \text{ ' } S$ **and** *tendsw*: $\xi \longrightarrow w$

by (*metis Diff_iff frontier_def closure_sequential*)

then have $\bigwedge n. \exists x \in S. \xi n = f x$ **by** *force*

then obtain *z* **where** *zs*: $\bigwedge n. z n \in S$ **and** *fz*: $\bigwedge n. \xi n = f (z n)$

by *metis*

then obtain *y* *K* **where** *y*: *y* $\in$ *closure S* **and** *strict_mono* (*K* :: *nat* $\Rightarrow$ *nat*)

and *Klim*: $(z \circ K) \longrightarrow y$

using $\langle \text{bounded } S \rangle$

unfolding *compact_closure [symmetric] compact_def* **by** (*meson closure_subset subset_iff*)

```

then have ftendsw: ((λn. f (z n)) ∘ K) ⟶ w
  by (metis LIMSEQ_subseq_LIMSEQ fun.map_cong0 fz tendsw)
have zKs: ⋀n. (z ∘ K) n ∈ S by (simp add: zs)
have fz: f ∘ z = ξ (λn. f (z n)) = ξ
  using fz by auto
then have (ξ ∘ K) ⟶ f y
  by (metis (no_types) Klim zKs y contf_comp_assoc continuous_on_closure_sequentially)
with fz have wy: w = f y using fz LIMSEQ_unique ftendsw by auto
have rle: r ≤ norm (f y - f a)
proof (rule le_no)
  show y ∈ frontier S
  using w wy oint by (force simp: imageI image_mono interiorI interior_subset
frontier_def y)
qed
have **: (b ∩ (- S) ≠ {} ∧ b - (- S) ≠ {} ⟹ b ∩ f ≠ {})
  ⟹ (b ∩ S ≠ {}) ⟹ b ∩ f = {} ⟹ b ⊆ S
  for b f and S :: 'b set
  by blast
have §: ⋀y. [norm (f a - y) < r; y ∈ frontier (f ' S)] ⟹ False
  by (metis dw_le norm_minus_commute not_less order_trans rle wy)
show ?thesis
apply (rule ** [OF connected_Int_frontier [where t = f'S, OF connected_ball]])

  using ⟨a ∈ S⟩ ⟨0 < r⟩ by (auto simp: disjoint_iff_not_equal dist_norm dest:
§)
qed

```

Special characterizations of classes of functions into and out of $\mathbb{R}$.

lemma *Hausdorff_space_euclidean* [simp]: *Hausdorff_space* (euclidean :: 'a::metric_space topology)

proof -

have $\exists U V. \text{open } U \wedge \text{open } V \wedge x \in U \wedge y \in V \wedge \text{disjnt } U V$

if $x \neq y$

for $x y :: 'a$

proof (intro exI conjI)

let $?r = \text{dist } x y / 2$

have [simp]: $?r > 0$

by (simp add: that)

show $\text{open } (\text{ball } x ?r) \text{ open } (\text{ball } y ?r) \ x \in (\text{ball } x ?r) \ y \in (\text{ball } y ?r)$

by (auto simp add: that)

show $\text{disjnt } (\text{ball } x ?r) (\text{ball } y ?r)$

unfolding *disjnt_def* by (simp add: disjoint_ballI)

qed

then show ?thesis

by (simp add: Hausdorff_space_def)

qed

proposition *embedding_map_into_euclideanreal*:

```

assumes path_connected_space X
shows embedding_map X euclideanreal f  $\longleftrightarrow$ 
      continuous_map X euclideanreal f  $\wedge$  inj_on f (topspace X)
proof safe
show continuous_map X euclideanreal f
  if embedding_map X euclideanreal f
  using continuous_map_in_subtopology homeomorphic_imp_continuous_map
that
  unfolding embedding_map_def by blast
show inj_on f (topspace X)
  if embedding_map X euclideanreal f
  using that homeomorphic_imp_injective_map
  unfolding embedding_map_def by blast
show embedding_map X euclideanreal f
  if cont: continuous_map X euclideanreal f and inj: inj_on f (topspace X)
proof -
obtain g where gf:  $\bigwedge x. x \in \text{topspace } X \implies g(f\ x) = x$ 
  using inv_into_f_f [OF inj] by auto
show ?thesis
unfolding embedding_map_def homeomorphic_map_maps homeomorphic_maps_def
proof (intro exI conjI)
show continuous_map X (top_of_set (f ` topspace X)) f
  by (simp add: cont_continuous_map_in_subtopology)
let ?S = f ` topspace X
have eq:  $\{x \in ?S. g\ x \in U\} = f\ `\ U$  if openin X U for U
  using openin_subset [OF that] by (auto simp: gf)
have 1:  $g\ `\ ?S \subseteq \text{topspace } X$ 
  using eq by blast
have openin (top_of_set ?S)  $\{x \in ?S. g\ x \in T\}$ 
  if openin X T for T
proof -
have  $T \subseteq \text{topspace } X$ 
  by (simp add: openin_subset that)
have RR:  $\forall x \in ?S \cap g\ `\ T. \exists d > 0. \forall x' \in ?S \cap \text{ball } x\ d. g\ x' \in T$ 
proof (clarsimp simp add: gf)
have pcS: path_connectedin euclidean ?S
  using assms cont path_connectedin_continuous_map_image path_connectedin_topspace
by blast
show  $\exists d > 0. \forall x' \in f\ `\ \text{topspace } X \cap \text{ball } (f\ x)\ d. g\ x' \in T$ 
  if  $x \in T$  for x
proof -
have  $x: x \in \text{topspace } X$ 
  using  $\langle T \subseteq \text{topspace } X \rangle \langle x \in T \rangle$  by blast
obtain u v d where  $0 < d$   $u \in \text{topspace } X$   $v \in \text{topspace } X$ 
  and sub_fuv:  $?S \cap \{f\ x - d .. f\ x + d\} \subseteq \{f\ u..f\ v\}$ 
proof (cases  $\exists u \in \text{topspace } X. f\ u < f\ x$ )
case True
  then obtain u where  $u: u \in \text{topspace } X$   $f\ u < f\ x ..$ 
  show ?thesis

```

```

proof (cases  $\exists v \in \text{topspace } X. f\ x < f\ v$ )
  case True
  then obtain v where v:  $v \in \text{topspace } X\ f\ x < f\ v$  ..
  show ?thesis
  proof
    let ?d =  $\min (f\ x - f\ u) (f\ v - f\ x)$ 
    show  $0 < ?d$ 
    by (simp add:  $\langle f\ u < f\ x \rangle \langle f\ x < f\ v \rangle$ )
    show  $f\ ' \text{topspace } X \cap \{f\ x - ?d..f\ x + ?d\} \subseteq \{f\ u..f\ v\}$ 
    by fastforce
  qed (auto simp: u v)
next
  case False
  show ?thesis
  proof
    let ?d =  $f\ x - f\ u$ 
    show  $0 < ?d$ 
    by (simp add: u)
    show  $f\ ' \text{topspace } X \cap \{f\ x - ?d..f\ x + ?d\} \subseteq \{f\ u..f\ x\}$ 
    using x u False by auto
  qed (auto simp: x u)
qed
next
  case False
  note no_u = False
  show ?thesis
  proof (cases  $\exists v \in \text{topspace } X. f\ x < f\ v$ )
    case True
    then obtain v where v:  $v \in \text{topspace } X\ f\ x < f\ v$  ..
    show ?thesis
    proof
      let ?d =  $f\ v - f\ x$ 
      show  $0 < ?d$ 
      by (simp add: v)
      show  $f\ ' \text{topspace } X \cap \{f\ x - ?d..f\ x + ?d\} \subseteq \{f\ x..f\ v\}$ 
      using False by auto
    qed (auto simp: x v)
    next
    case False
    show ?thesis
    proof
      show  $f\ ' \text{topspace } X \cap \{f\ x - 1..f\ x + 1\} \subseteq \{f\ x..f\ x\}$ 
      using False no_u by fastforce
    qed (auto simp: x)
  qed
qed
then obtain h where pathin X h h 0 = u h 1 = v
  using assms unfolding path_connected_space_def by blast
obtain C where compactin X C connectedin X C u  $\in C$  v  $\in C$ 

```

```

proof
  show compactin X (h ‘ {0..1})
    using that by (simp add: ⟨pathin X h⟩ compactin_path_image)
  show connectedin X (h ‘ {0..1})
    using ⟨pathin X h⟩ connectedin_path_image by blast
  qed (use ⟨h 0 = u⟩ ⟨h 1 = v⟩ in auto)
  have continuous_map (subtopology euclideanreal (?S ∩ {f x - d .. f x +
d})) (subtopology X C) g
  proof (rule continuous_inverse_map)
    show compact_space (subtopology X C)
      using ⟨compactin X C⟩ compactin_subspace by blast
    show continuous_map (subtopology X C) euclideanreal f
      by (simp add: cont continuous_map_from_subtopology)
    have {f u .. f v} ⊆ f ‘ topspace (subtopology X C)
    proof (rule connected_contains_Icc)
      show connected (f ‘ topspace (subtopology X C))
        using connectedin_continuous_map_image [OF cont]
        by (simp add: ⟨compactin X C⟩ ⟨connectedin X C⟩ com-
pactin_subset_topspace inf_absorb2)
      show f u ∈ f ‘ topspace (subtopology X C)
        by (simp add: ⟨u ∈ C⟩ ⟨u ∈ topspace X⟩)
      show f v ∈ f ‘ topspace (subtopology X C)
        by (simp add: ⟨v ∈ C⟩ ⟨v ∈ topspace X⟩)
    qed
  then show f ‘ topspace X ∩ {f x - d..f x + d} ⊆ f ‘ topspace (subtopology
X C)
    using sub_fuv by blast
  qed (auto simp: gf)
  then have contg: continuous_map (subtopology euclideanreal (?S ∩ {f x
- d .. f x + d})) X g
    using continuous_map_in_subtopology by blast
  have ∃ e > 0. ∀ x ∈ ?S ∩ {f x - d .. f x + d} ∩ ball (f x) e. g x ∈ T
    using openin_continuous_map_preimage [OF contg ⟨openin X T⟩] x
⟨x ∈ T⟩ ⟨0 < d⟩
    unfolding openin_euclidean_subtopology_iff
    by (force simp: gf dist_commute)
  then obtain e where e > 0 ∧ (∀ x ∈ f ‘ topspace X ∩ {f x - d..f x +
d} ∩ ball (f x) e. g x ∈ T)
    by metis
  with ⟨0 < d⟩ have min d e > 0 ∀ u. u ∈ topspace X ⟶ |f x - f u| <
min d e ⟶ u ∈ T
    using dist_real_def gf by force+
  then show ?thesis
    by (metis (full_types) Int_iff dist_real_def image_iff mem_ball gf)
  qed
qed
then obtain d where d: ∧ r. r ∈ ?S ∩ g - ‘ T ⟹
  d r > 0 ∧ (∀ x ∈ ?S ∩ ball r (d r). g x ∈ T)
  by metis

```

```

    show ?thesis
      unfolding openin_subtopology
    proof (intro exI conjI)
      show  $\{x \in ?S. g\ x \in T\} = (\bigcup r \in ?S \cap g - ' T. \text{ball } r\ (d\ r)) \cap f - ' \text{topspace}$ 
X
      using d by (auto simp: gf)
    qed auto
  qed
then show continuous_map (top_of_set ?S) X g
  by (simp add: continuous_map_def gf)
qed (auto simp: gf)
qed
qed

```

An injective function into $\mathbb{R}$ is a homeomorphism and so an open map.

```

lemma injective_into_1d_eq_homeomorphism:
  fixes f :: 'a::topological_space  $\Rightarrow$  real
  assumes f: continuous_on S f and S: path_connected S
  shows inj_on f S  $\longleftrightarrow$  ( $\exists g. \text{homeomorphism } S\ (f - ' S)\ f\ g$ )
proof
  show  $\exists g. \text{homeomorphism } S\ (f - ' S)\ f\ g$ 
    if inj_on f S
  proof -
    have embedding_map (top_of_set S) euclideanreal f
      using that embedding_map_into_euclideanreal [of top_of_set S f] assms by
auto
    then show ?thesis
      by (simp add: embedding_map_def) (metis all_closedin_homeomorphic_image
f homeomorphism_injective_closed_map that)
    qed
  qed (metis homeomorphism_def inj_onI)

lemma injective_into_1d_imp_open_map:
  fixes f :: 'a::topological_space  $\Rightarrow$  real
  assumes continuous_on S f path_connected S inj_on f S openin (subtopology
euclidean S) T
  shows openin (subtopology euclidean (f - ' S)) (f - ' T)
  using assms homeomorphism_imp_open_map injective_into_1d_eq_homeomorphism
by blast

lemma homeomorphism_into_1d:
  fixes f :: 'a::topological_space  $\Rightarrow$  real
  assumes path_connected S continuous_on S f f - ' S = T inj_on f S
  shows  $\exists g. \text{homeomorphism } S\ T\ f\ g$ 
  using assms injective_into_1d_eq_homeomorphism by blast

```

5.5.26 Rectangular paths

definition *rectpath* **where**

$\text{rectpath } a1 \ a3 = (\text{let } a2 = \text{Complex } (\text{Re } a3) (\text{Im } a1); a4 = \text{Complex } (\text{Re } a1) (\text{Im } a3)$
 $\text{in } \text{linepath } a1 \ a2 \ ++ \ \text{linepath } a2 \ a3 \ ++ \ \text{linepath } a3 \ a4 \ ++ \ \text{linepath } a4 \ a1)$

lemma *path_rectpath* [*simp*, *intro*]: $\text{path } (\text{rectpath } a \ b)$
by (*simp* *add*: *Let_def* *rectpath_def*)

lemma *pathstart_rectpath* [*simp*]: $\text{pathstart } (\text{rectpath } a1 \ a3) = a1$
by (*simp* *add*: *rectpath_def* *Let_def*)

lemma *pathfinish_rectpath* [*simp*]: $\text{pathfinish } (\text{rectpath } a1 \ a3) = a1$
by (*simp* *add*: *rectpath_def* *Let_def*)

lemma *simple_path_rectpath* [*simp*, *intro*]:
assumes $\text{Re } a1 \neq \text{Re } a3 \ \text{Im } a1 \neq \text{Im } a3$
shows $\text{simple\_path } (\text{rectpath } a1 \ a3)$
unfolding *rectpath_def* *Let_def* **using** *assms*
by (*intro* *simple_path_join_loop* *arc_join* *arc_linepath*)
(auto simp: complex_eq_iff path_image_join closed_segment_same_Re closed_segment_same_Im)

lemma *path_image_rectpath*:
assumes $\text{Re } a1 \leq \text{Re } a3 \ \text{Im } a1 \leq \text{Im } a3$
shows $\text{path\_image } (\text{rectpath } a1 \ a3) =$
 $\{z. \text{Re } z \in \{\text{Re } a1, \text{Re } a3\} \wedge \text{Im } z \in \{\text{Im } a1.. \text{Im } a3\}\} \cup$
 $\{z. \text{Im } z \in \{\text{Im } a1, \text{Im } a3\} \wedge \text{Re } z \in \{\text{Re } a1.. \text{Re } a3\}\} \text{ (is ?lhs = ?rhs)}$
proof –
define $a2 \ a4$ **where** $a2 = \text{Complex } (\text{Re } a3) (\text{Im } a1)$ **and** $a4 = \text{Complex } (\text{Re } a1) (\text{Im } a3)$
have $?lhs = \text{closed\_segment } a1 \ a2 \cup \text{closed\_segment } a2 \ a3 \cup$
 $\text{closed\_segment } a4 \ a3 \cup \text{closed\_segment } a1 \ a4$
by (*simp_all* *add*: *rectpath_def* *Let_def* *path_image_join* *closed_segment_commute*
a2_def *a4_def* *Un_assoc*)
also have $\dots = ?rhs$ **using** *assms*
by (*auto simp: rectpath_def Let_def path_image_join a2_def a4_def*
closed_segment_same_Re closed_segment_same_Im closed_segment_eq_real_ivl)
finally show *?thesis* .
qed

lemma *path_image_rectpath_subset_cbox*:
assumes $\text{Re } a \leq \text{Re } b \ \text{Im } a \leq \text{Im } b$
shows $\text{path\_image } (\text{rectpath } a \ b) \subseteq \text{cbox } a \ b$
using *assms* **by** (*auto simp: path_image_rectpath in_cbox_complex_iff*)

lemma *path_image_rectpath_inter_box*:
assumes $\text{Re } a \leq \text{Re } b \ \text{Im } a \leq \text{Im } b$
shows $\text{path\_image } (\text{rectpath } a \ b) \cap \text{box } a \ b = \{\}$

using *assms* **by** (*auto simp: path_image_rectpath in_box_complex_iff*)

lemma *path_image_rectpath_cbox_minus_box*:

assumes $\text{Re } a \leq \text{Re } b$ $\text{Im } a \leq \text{Im } b$

shows $\text{path\_image } (\text{rectpath } a \ b) = \text{cbox } a \ b - \text{box } a \ b$

using *assms* **by** (*auto simp: path_image_rectpath in_cbox_complex_iff in_box_complex_iff*)

end

5.6 Bernstein-Weierstrass and Stone-Weierstrass

By L C Paulson (2015)

theory *Weierstrass_Theorems*

imports *Uniform_Limit Path_Connected Derivative*

begin

5.6.1 Bernstein polynomials

definition *Bernstein* :: $[\text{nat}, \text{nat}, \text{real}] \Rightarrow \text{real}$ **where**

$\text{Bernstein } n \ k \ x \equiv \text{of\_nat } (n \text{ choose } k) * x^k * (1 - x)^{(n - k)}$

lemma *Bernstein_nonneg*: $\llbracket 0 \leq x; x \leq 1 \rrbracket \Longrightarrow 0 \leq \text{Bernstein } n \ k \ x$

by (*simp add: Bernstein_def*)

lemma *Bernstein_pos*: $\llbracket 0 < x; x < 1; k \leq n \rrbracket \Longrightarrow 0 < \text{Bernstein } n \ k \ x$

by (*simp add: Bernstein_def*)

lemma *sum_Bernstein* [*simp*]: $(\sum k \leq n. \text{Bernstein } n \ k \ x) = 1$

using *binomial_ring* [*of x 1-x n*]

by (*simp add: Bernstein_def*)

lemma *binomial_deriv1*:

$(\sum k \leq n. (\text{of\_nat } k * \text{of\_nat } (n \text{ choose } k)) * a^{k-1} * b^{n-k}) = \text{real\_of\_nat } n * (a+b)^{n-1}$

apply (*rule DERIV_unique* [**where** $f = \lambda a. (a+b)^n$ **and** $x=a$])

apply (*subst binomial_ring*)

apply (*rule derivative_eq_intros sum.cong* | *simp add: atMost_atLeast0*) +

done

lemma *binomial_deriv2*:

$(\sum k \leq n. (\text{of\_nat } k * \text{of\_nat } (k-1) * \text{of\_nat } (n \text{ choose } k)) * a^{k-2} * b^{n-k}) =$

$\text{of\_nat } n * \text{of\_nat } (n-1) * (a+b::\text{real})^{n-2}$

apply (*rule DERIV_unique* [**where** $f = \lambda a. \text{of\_nat } n * (a+b::\text{real})^{n-1}$ **and** $x=a$])

apply (*subst binomial_deriv1* [*symmetric*])

apply (*rule derivative_eq_intros sum.cong* | *simp add: Num.numeral_2_eq_2*) +

done

lemma *sum_k_Bernstein* [simp]: $(\sum k \leq n. \text{real } k * \text{Bernstein } n \ k \ x) = \text{of\_nat } n * x$
apply (subst binomial_deriv1 [of n x 1-x, simplified, symmetric])
apply (simp add: sum_distrib_right)
apply (auto simp: Bernstein_def algebra_simps power_eq_if intro!: sum.cong)
done

lemma *sum_kk_Bernstein* [simp]: $(\sum k \leq n. \text{real } k * (\text{real } k - 1) * \text{Bernstein } n \ k \ x) = \text{real } n * (\text{real } n - 1) * x^2$

proof -

have $(\sum k \leq n. \text{real } k * (\text{real } k - 1) * \text{Bernstein } n \ k \ x) =$
 $(\sum k \leq n. \text{real } k * \text{real } (k - \text{Suc } 0) * \text{real } (n \text{ choose } k) * x^{k-2} * (1 - x)^{n-k} * x^2)$

proof (rule sum.cong [OF refl], simp)

fix k

assume $k \leq n$

then consider $k = 0 \mid k = 1 \mid k'$ **where** $k = \text{Suc } (\text{Suc } k')$

by (metis One_nat_def not0_implies_Suc)

then show $k = 0 \vee$

$(\text{real } k - 1) * \text{Bernstein } n \ k \ x =$

$\text{real } (k - \text{Suc } 0) *$

$(\text{real } (n \text{ choose } k) * x^{k-2} * ((1 - x)^{n-k} * x^2))$

by cases (auto simp add: Bernstein_def power2_eq_square algebra_simps)

qed

also have $\dots = \text{real\_of\_nat } n * \text{real\_of\_nat } (n - \text{Suc } 0) * x^2$

by (subst binomial_deriv2 [of n x 1-x, simplified, symmetric]) (simp add: sum_distrib_right)

also have $\dots = n * (n - 1) * x^2$

by auto

finally show ?thesis

by auto

qed

5.6.2 Explicit Bernstein version of the 1D Weierstrass approximation theorem

theorem *Bernstein_Weierstrass*:

fixes $f :: \text{real} \Rightarrow \text{real}$

assumes *contf*: continuous_on {0..1} f **and** $e: 0 < e$

shows $\exists N. \forall n \ x. N \leq n \wedge x \in \{0..1\}$

$\longrightarrow |f \ x - (\sum k \leq n. f(k/n) * \text{Bernstein } n \ k \ x)| < e$

proof -

have bounded (f ‘ {0..1})

using compact_continuous_image compact_imp_bounded *contf* **by blast**

then obtain M **where** $M: \bigwedge x. 0 \leq x \implies x \leq 1 \implies |f \ x| \leq M$

by (force simp add: bounded_iff)

then have $0 \leq M$ **by force**

```

have ucontf: uniformly_continuous_on {0..1} f
  using compact_uniformly_continuous_contf by blast
then obtain d where d: d > 0 ∧ x x'. [| x ∈ {0..1}; x' ∈ {0..1}; |x' - x| < d |]
⇒ |f x' - f x| < e/2
  apply (rule uniformly_continuous_onE [where e = e/2])
  using e by (auto simp: dist_norm)
{ fix n::nat and x::real
  assume n: Suc (nat ⌈4*M/(e*d²)⌉) ≤ n and x: 0 ≤ x ≤ 1
  have 0 < n using n by simp
  have ed0: - (e * d²) < 0
    using e ⟨0 < d⟩ by simp
  also have ... ≤ M * 4
    using ⟨0 ≤ M⟩ by simp
  finally have [simp]: real_of_int (nat ⌈4 * M / (e * d²)⌉) = real_of_int ⌈4 *
M / (e * d²)⌉
    using ⟨0 ≤ M⟩ e ⟨0 < d⟩
    by (simp add: field_simps)
  have 4*M/(e*d²) + 1 ≤ real (Suc (nat ⌈4*M/(e*d²)⌉))
    by (simp add: real_nat_ceiling_ge)
  also have ... ≤ real n
    using n by (simp add: field_simps)
  finally have nbig: 4*M/(e*d²) + 1 ≤ real n .
  have sum_bern: (∑ k ≤ n. (x - k/n)² * Bernstein n k x) = x * (1 - x) / n
  proof -
    have *: ∧ a b x::real. (a - b)² * x = a * (a - 1) * x + (1 - 2 * b) * a * x
    + b * b * x
    by (simp add: algebra_simps power2_eq_square)
    have (∑ k ≤ n. (k - n * x)² * Bernstein n k x) = n * x * (1 - x)
    apply (simp add: * sum.distrib)
    apply (simp flip: sum_distrib_left add: mult.assoc)
    apply (simp add: algebra_simps power2_eq_square)
    done
    then have (∑ k ≤ n. (k - n * x)² * Bernstein n k x) / n² = x * (1 - x) / n
    by (simp add: power2_eq_square)
    then show ?thesis
    using n by (simp add: sum_divide_distrib field_split_simps power2_commute)
  qed
{ fix k
  assume k: k ≤ n
  then have kn: 0 ≤ k / n < 1
    by (auto simp: field_split_simps)
  consider (lessd) |x - k / n| < d | (ged) d ≤ |x - k / n|
    by linarith
  then have |(f x - f (k/n))| ≤ e/2 + 2 * M / d² * (x - k/n)²
  proof cases
    case lessd
    then have |(f x - f (k/n))| < e/2
      using d x kn by (simp add: abs_minus_commute)
    also have ... ≤ (e/2 + 2 * M / d² * (x - k/n)²)

```

```

    using ‹ $M \geq 0$ › d by simp
    finally show ?thesis by simp
next
case ged
then have dle:  $d^2 \leq (x - k/n)^2$ 
  by (metis d(1) less_eq_real_def power2_abs power_mono)
have §:  $1 \leq (x - \text{real } k / \text{real } n)^2 / d^2$ 
  using dle ‹ $d > 0$ › by auto
have |(f x - f (k/n))| ≤ |f x| + |f (k/n)|
  by (rule abs_triangle_ineq4)
also have ... ≤ M + M
  by (meson M add_mono_thms_linordered_semiring(1) kn x)
also have ... ≤ 2 * M * ((x - k/n)2 / d2)
  using § ‹ $M \geq 0$ › mult_left_mono by fastforce
also have ... ≤ e/2 + 2 * M / d2 * (x - k/n)2
  using e by simp
finally show ?thesis .
qed
} note * = this
have |f x - (∑ k ≤ n. f(k / n) * Bernstein n k x)| ≤ |∑ k ≤ n. (f x - f(k / n))
* Bernstein n k x|
  by (simp add: sum_subtractf sum_distrib_left [symmetric] algebra_simps)
also have ... ≤ (∑ k ≤ n. |(f x - f(k / n)) * Bernstein n k x|)
  by (rule sum_abs)
also have ... ≤ (∑ k ≤ n. (e/2 + (2 * M / d2) * (x - k / n)2) * Bernstein n
k x)
  using *
  by (force simp add: abs_mult Bernstein_nonneg x mult_right_mono intro:
sum_mono)
also have ... ≤ e/2 + (2 * M) / (d2 * n)
  unfolding sum.distrib Rings.semiring_class.distrib_right sum_distrib_left
[symmetric] mult.assoc sum_bern
  using ‹ $d > 0$ › x by (simp add: divide_simps ‹ $M \geq 0$ › mult_le_one mult_left_le)
also have ... < e
  using ‹ $d > 0$ › nbig e ‹ $n > 0$ ›
  apply (simp add: field_split_simps)
  using ed0 by linarith
finally have |f x - (∑ k ≤ n. f (real k / real n) * Bernstein n k x)| < e .
}
then show ?thesis
  by auto
qed

```

5.6.3 General Stone-Weierstrass theorem

Source: Bruno Brosowski and Frank Deutsch. An Elementary Proof of the Stone-Weierstrass Theorem. Proceedings of the American Mathematical Society Volume 81, Number 1, January 1981. DOI: 10.2307/2043993 <https://www.jstor.org/stable/2043993>

```

locale function_ring_on =
  fixes  $R :: ('a::t2\_space \Rightarrow real) \text{ set}$  and  $S :: 'a \text{ set}$ 
  assumes compact:  $compact\ S$ 
  assumes continuous:  $f \in R \Longrightarrow continuous\_on\ S\ f$ 
  assumes add:  $f \in R \Longrightarrow g \in R \Longrightarrow (\lambda x. f\ x + g\ x) \in R$ 
  assumes mult:  $f \in R \Longrightarrow g \in R \Longrightarrow (\lambda x. f\ x * g\ x) \in R$ 
  assumes const:  $(\lambda \_. c) \in R$ 
  assumes separable:  $x \in S \Longrightarrow y \in S \Longrightarrow x \neq y \Longrightarrow \exists f \in R. f\ x \neq f\ y$ 

begin
  lemma minus:  $f \in R \Longrightarrow (\lambda x. - f\ x) \in R$ 
    by (frule mult [OF const [of -1]] simp)

  lemma diff:  $f \in R \Longrightarrow g \in R \Longrightarrow (\lambda x. f\ x - g\ x) \in R$ 
    unfolding diff_conv_add_uminus by (metis add minus)

  lemma power:  $f \in R \Longrightarrow (\lambda x. f\ x^n) \in R$ 
    by (induct n (auto simp: const mult))

  lemma sum:  $\llbracket finite\ I; \bigwedge i. i \in I \Longrightarrow f\ i \in R \rrbracket \Longrightarrow (\lambda x. \sum i \in I. f\ i\ x) \in R$ 
    by (induct I rule: finite_induct; simp add: const add)

  lemma prod:  $\llbracket finite\ I; \bigwedge i. i \in I \Longrightarrow f\ i \in R \rrbracket \Longrightarrow (\lambda x. \prod i \in I. f\ i\ x) \in R$ 
    by (induct I rule: finite_induct; simp add: const mult)

  definition normf ::  $('a::t2\_space \Rightarrow real) \Rightarrow real$ 
    where  $normf\ f \equiv SUP\ x \in S. |f\ x|$ 

  lemma normf_upper:
    assumes continuous_on  $S\ f\ x \in S$  shows  $|f\ x| \leq normf\ f$ 
  proof -
    have bdd_above  $((\lambda x. |f\ x|) \text{ ` } S)$ 
    by (simp add: assms(1) bounded_imp_bdd_above compact compact_continuous_image
      compact_imp_bounded continuous_on_rabs)
    then show ?thesis
      using assms cSUP_upper normf_def by fastforce
  qed

  lemma normf_least:  $S \neq \{\} \Longrightarrow (\bigwedge x. x \in S \Longrightarrow |f\ x| \leq M) \Longrightarrow normf\ f \leq M$ 
    by (simp add: normf_def cSUP_least)

end

lemma (in function_ring_on) one:
  assumes  $U$ : open  $U$  and  $t0$ :  $t0 \in S$   $t0 \in U$  and  $t1$ :  $t1 \in S - U$ 
  shows  $\exists V. open\ V \wedge t0 \in V \wedge S \cap V \subseteq U \wedge$ 
     $(\forall e > 0. \exists f \in R. f \text{ ` } S \subseteq \{0..1\} \wedge (\forall t \in S \cap V. f\ t < e) \wedge (\forall t \in S$ 
     $- U. f\ t > 1 - e))$ 
  proof -

```

```

have  $\exists pt \in R. pt \ t0 = 0 \wedge pt \ t > 0 \wedge pt \ ' \ S \subseteq \{0..1\}$  if  $t: t \in S - U$  for  $t$ 
proof -
  have  $t \neq t0$  using  $t \ t0$  by auto
  then obtain  $g$  where  $g: g \in R \ g \ t \neq g \ t0$ 
    using separable  $t0$  by (metis Diff_subset subset_eq  $t$ )
  define  $h$  where [abs_def]:  $h \ x = g \ x - g \ t0$  for  $x$ 
  have  $h \in R$ 
    unfolding  $h\_def$  by (fast intro:  $g \ const \ diff$ )
  then have  $hsq: (\lambda w. (h \ w)^2) \in R$ 
    by (simp add: power2_eq_square mult)
  have  $h \ t \neq h \ t0$ 
    by (simp add:  $h\_def \ g$ )
  then have  $h \ t \neq 0$ 
    by (simp add:  $h\_def$ )
  then have  $ht2: 0 < (h \ t)^2$ 
    by simp
  also have  $\dots \leq normf \ (\lambda w. (h \ w)^2)$ 
    using  $t \ normf\_upper$  [where  $x=t$ ] continuous [OF  $hsq$ ] by force
  finally have  $nfp: 0 < normf \ (\lambda w. (h \ w)^2)$  .
  define  $p$  where [abs_def]:  $p \ x = (1 / normf \ (\lambda w. (h \ w)^2)) * (h \ x)^2$  for  $x$ 
  have  $p \in R$ 
    unfolding  $p\_def$  by (fast intro:  $hsq \ const \ mult$ )
  moreover have  $p \ t0 = 0$ 
    by (simp add:  $p\_def \ h\_def$ )
  moreover have  $p \ t > 0$ 
    using  $nfp \ ht2$  by (simp add:  $p\_def$ )
  moreover have  $\bigwedge x. x \in S \implies p \ x \in \{0..1\}$ 
    using  $nfp \ normf\_upper$  [OF continuous [OF  $hsq$ ]] by (auto simp:  $p\_def$ )
  ultimately show  $\exists pt \in R. pt \ t0 = 0 \wedge pt \ t > 0 \wedge pt \ ' \ S \subseteq \{0..1\}$ 
    by auto
qed
then obtain  $pf$  where  $pf: \bigwedge t. t \in S - U \implies pf \ t \in R \wedge pf \ t \ t0 = 0 \wedge pf \ t \ t > 0$ 
and  $pf01: \bigwedge t. t \in S - U \implies pf \ t \ ' \ S \subseteq \{0..1\}$ 
  by metis
have  $com\_sU: compact \ (S - U)$ 
  using compact closed_Int_compact  $U$  by (simp add: Diff_eq compact_Int_closed
open_closed)
have  $\bigwedge t. t \in S - U \implies \exists A. open \ A \wedge A \cap S = \{x \in S. 0 < pf \ t \ x\}$ 
  apply (rule open_Collect_positive)
  by (metis  $pf \ continuous$ )
then obtain  $Uf$  where  $Uf: \bigwedge t. t \in S - U \implies open \ (Uf \ t) \wedge (Uf \ t) \cap S = \{x \in S. 0 < pf \ t \ x\}$ 
  by metis
then have  $open\_Uf: \bigwedge t. t \in S - U \implies open \ (Uf \ t)$ 
  by blast
have  $tUf: \bigwedge t. t \in S - U \implies t \in Uf \ t$ 
  using  $pf \ Uf$  by blast
then have *:  $S - U \subseteq (\bigcup x \in S - U. Uf \ x)$ 

```

```

    by blast
  obtain subU where subU: subU  $\subseteq$  S - U finite subU S - U  $\subseteq$  ( $\bigcup x \in$  subU.
  Uf x)
    by (blast intro: that compactE_image [OF com_sU open_Uf *])
  then have [simp]: subU  $\neq$  {}
    using t1 by auto
  then have cardp: card subU > 0 using subU
    by (simp add: card_gt_0_iff)
  define p where [abs_def]: p x = (1 / card subU) * ( $\sum t \in$  subU. pf t x) for x
  have pR: p  $\in$  R
    unfolding p_def using subU pf by (fast intro: pf const mult sum)
  have pt0 [simp]: p t0 = 0
    using subU pf by (auto simp: p_def intro: sum.neutral)
  have pt_pos: p t > 0 if t: t  $\in$  S - U for t
  proof -
    obtain i where i: i  $\in$  subU t  $\in$  Uf i using subU t by blast
    show ?thesis
      using subU i t
      apply (clarify simp: p_def field_split_simps)
      apply (rule sum_pos2 [OF <finite subU>])
      using Uf t pf01 apply auto
      apply (force elim!: subsetCE)
      done
  qed
  have p01: p x  $\in$  {0..1} if t: x  $\in$  S for x
  proof -
    have 0  $\leq$  p x
      using subU cardp t pf01
      by (fastforce simp add: p_def field_split_simps intro: sum_nonneg)
    moreover have p x  $\leq$  1
      using subU cardp t
      apply (simp add: p_def field_split_simps)
      apply (rule sum_bounded_above [where 'a=real and K=1, simplified])
      using pf01 by force
    ultimately show ?thesis
      by auto
  qed
  have compact (p ' (S - U))
    by (meson Diff_subset com_sU compact_continuous_image continuous con-
    tinuous_on_subset pR)
  then have open ( $\neg$  (p ' (S - U)))
    by (simp add: compact_imp_closed open_Compl)
  moreover have 0  $\in$   $\neg$  (p ' (S - U))
    by (metis (no_types) ComplI image_iff not_less_iff_gr_or_eq pt_pos)
  ultimately obtain delta0 where delta0: delta0 > 0 ball 0 delta0  $\subseteq$   $\neg$  (p '
  (S - U))
    by (auto simp: elim!: openE)
  then have pt_delta:  $\bigwedge x. x \in$  S - U  $\implies$  p x  $\geq$  delta0
    by (force simp: ball_def dist_norm dest: p01)

```

```

define  $\delta$  where  $\delta = \text{delta0}/2$ 
have  $\text{delta0} \leq 1$  using  $\text{delta0 p01 [of t1] t1}$ 
  by (force simp: ball_def dist_norm dest: p01)
with  $\text{delta0}$  have  $\delta 01: 0 < \delta \ \delta < 1$ 
  by (auto simp:  $\delta\_def$ )
have  $\text{pt\_}\delta: \bigwedge x. x \in S - U \implies p \ x \geq \delta$ 
  using  $\text{pt\_delta delta0}$  by (force simp:  $\delta\_def$ )
have  $\exists A. \text{open } A \wedge A \cap S = \{x \in S. p \ x < \delta/2\}$ 
  by (rule open_Collect_less_Int [OF continuous [OF pR] continuous_on_const])
then obtain  $V$  where  $V: \text{open } V \ V \cap S = \{x \in S. p \ x < \delta/2\}$ 
  by blast
define  $k$  where  $k = \text{nat}[1/\delta] + 1$ 
have  $k > 0$  by (simp add:  $k\_def$ )
have  $k - 1 \leq 1/\delta$ 
  using  $\delta 01$  by (simp add:  $k\_def$ )
with  $\delta 01$  have  $k \leq (1 + \delta)/\delta$ 
  by (auto simp: algebra_simps add_divide_distrib)
also have  $\dots < 2/\delta$ 
  using  $\delta 01$  by (auto simp: field_split_simps)
finally have  $k 2\delta: k < 2/\delta$  .
have  $1/\delta < k$ 
  using  $\delta 01$  unfolding  $k\_def$  by linarith
with  $\delta 01 \ k 2\delta$  have  $k\delta: 1 < k * \delta \ k * \delta < 2$ 
  by (auto simp: field_split_simps)
define  $q$  where  $[\text{abs\_def}]: q \ n \ t = (1 - p \ t^n)^{\wedge(k^n)}$  for  $n \ t$ 
have  $qR: q \ n \in R$  for  $n$ 
  by (simp add:  $q\_def$  const diff power pR)
have  $q 01: \bigwedge n \ t. t \in S \implies q \ n \ t \in \{0..1\}$ 
  using  $p 01$  by (simp add:  $q\_def$  power_le_one algebra_simps)
have  $q t 0 [\text{simp}]: \bigwedge n. n > 0 \implies q \ n \ t 0 = 1$ 
  using  $t 0$  pf by (simp add:  $q\_def$  power_0_left)
{ fix  $t$  and  $n::\text{nat}$ 
  assume  $t: t \in S \cap V$ 
  with  $\langle k > 0 \rangle \ V$  have  $k * p \ t < k * \delta / 2$ 
    by force
  then have  $1 - (k * \delta / 2)^{\wedge n} \leq 1 - (k * p \ t)^{\wedge n}$ 
    using  $\langle k > 0 \rangle \ p 01 \ t$  by (simp add: power_mono)
  also have  $\dots \leq q \ n \ t$ 
    using Bernoulli_inequality [of  $1 - (p \ t)^{\wedge n} \ k^{\wedge n}$ ]
    apply (simp add:  $q\_def$ )
    by (metis IntE atLeastAtMost_iff p01 power_le_one power_mult_distrib t)
  finally have  $1 - (k * \delta / 2)^{\wedge n} \leq q \ n \ t$  .
}
} note limitV = this
{ fix  $t$  and  $n::\text{nat}$ 
  assume  $t: t \in S - U$ 
  with  $\langle k > 0 \rangle \ U$  have  $k * \delta \leq k * p \ t$ 
    by (simp add:  $\text{pt\_}\delta$ )
  with  $k\delta$  have  $kpt: 1 < k * p \ t$ 
    by (blast intro: less_le_trans)

```

```

have ptn_pos: 0 < p t^n
  using pt_pos [OF t] by simp
have ptn_le: p t^n ≤ 1
  by (meson DiffE atLeastAtMost_iff p01 power_le_one t)
have q n t = (1/(k^n * (p t)^n)) * (1 - p t^n)^(k^n) * k^n * (p t)^n
  using pt_pos [OF t] ⟨k>0 by (simp add: q_def)
also have ... ≤ (1/(k * (p t))^n) * (1 - p t^n)^(k^n) * (1 + k^n * (p t)^n)
  using pt_pos [OF t] ⟨k>0 by
  by (simp add: divide_simps mult_left_mono ptn_le)
also have ... ≤ (1/(k * (p t))^n) * (1 - p t^n)^(k^n) * (1 + (p t)^n)^(k^n)
proof (rule mult_left_mono [OF Bernoulli_inequality])
  show 0 ≤ 1 / (real k * p t)^n * (1 - p t^n)^k^n
    using ptn_pos ptn_le by (auto simp: power_mult_distrib)
qed (use ptn_pos in auto)
also have ... = (1/(k * (p t))^n) * (1 - p t^(2*n))^(k^n)
  using pt_pos [OF t] ⟨k>0 by
  by (simp add: algebra_simps power_mult power2_eq_square flip: power_mult_distrib)
also have ... ≤ (1/(k * (p t))^n) * 1
  using pt_pos ⟨k>0 p01 power_le_one t
  by (intro mult_left_mono [OF power_le_one]) auto
also have ... ≤ (1 / (k*δ))^n
  using ⟨k>0 δ01 power_mono pt_δ t
  by (fastforce simp: field_simps)
finally have q n t ≤ (1 / (real k * δ))^n .
} note limitNonU = this
define NN
  where NN e = 1 + nat ⌈max (ln e / ln (real k * δ / 2)) (- ln e / ln (real k
* δ))⌉ for e
  have NN: of_nat (NN e) > ln e / ln (real k * δ / 2) of_nat (NN e) > - ln e
/ ln (real k * δ)
    if 0 < e for e
  unfolding NN_def by linarith+
  have NN1: (k * δ / 2)^NN e < e if e>0 for e
  proof -
    have ln ((real k * δ / 2)^NN e) = real (NN e) * ln (real k * δ / 2)
      by (simp add: ⟨δ>0 ⟨0 < k⟩ ln_realpow)
    also have ... < ln e
      using NN kδ that by (force simp add: field_simps)
    finally show ?thesis
      by (simp add: ⟨δ>0 ⟨0 < k⟩ that)
  qed
  have NN0: (1/(k*δ))^NN e < e if e>0 for e
  proof -
    have 0 < ln (real k) + ln δ
      using δ01(1) ⟨0 < k⟩ kδ(1) ln_gt_zero ln_mult by fastforce
    then have real (NN e) * ln (1 / (real k * δ)) < ln e
      using kδ(1) NN(2) [of e] that by (simp add: ln_div divide_simps)
    then have exp (real (NN e) * ln (1 / (real k * δ))) < e
      by (metis exp_less_mono exp_ln that)

```



```

    then show ?thesis
      by (simp add:  $\delta 01(1) \langle 0 < k \rangle \exp\_of\_nat\_mult$ )
  qed
  { fix t and e::real
    assume  $e > 0$ 
    have  $t \in S \cap V \implies 1 - q (NN\ e) \ t < e$   $t \in S - U \implies q (NN\ e) \ t < e$ 
    proof -
      assume  $t: t \in S \cap V$ 
      show  $1 - q (NN\ e) \ t < e$ 
        by (metis add.commute diff_le_eq not_le limitV [OF t] less_le_trans [OF
NN1 [OF  $\langle e > 0 \rangle$ ]])
      next
        assume  $t: t \in S - U$ 
        show  $q (NN\ e) \ t < e$ 
          using limitNonU [OF t] less_le_trans [OF NN0 [OF  $\langle e > 0 \rangle$ ]] not_le by
blast
      qed
    } then have  $\bigwedge e. e > 0 \implies \exists f \in R. f \cdot S \subseteq \{0..1\} \wedge (\forall t \in S \cap V. f \ t < e) \wedge$ 
 $(\forall t \in S - U. 1 - e < f \ t)$ 
    using q01
    by (rule_tac  $x = \lambda x. 1 - q (NN\ e) \ x$  in bexI) (auto simp: algebra_simps intro:
diff const qR)
    moreover have  $t0V: t0 \in V \ S \cap V \subseteq U$ 
    using pt_δ t0 U V δ01 by fastforce+
    ultimately show ?thesis using V t0V
    by blast
  qed

```

Non-trivial case, with A and B both non-empty

lemma (in function_ring_on) two_special:

```

  assumes  $A: \text{closed } A \ A \subseteq S \ a \in A$ 
    and  $B: \text{closed } B \ B \subseteq S \ b \in B$ 
    and disj:  $A \cap B = \{\}$ 
    and  $e: 0 < e \ e < 1$ 
  shows  $\exists f \in R. f \cdot S \subseteq \{0..1\} \wedge (\forall x \in A. f \ x < e) \wedge (\forall x \in B. f \ x > 1 - e)$ 
proof -
  { fix w
    assume  $w \in A$ 
    then have open ( - B)  $b \in S \ w \notin B \ w \in S$ 
    using assms by auto
    then have  $\exists V. \text{open } V \wedge w \in V \wedge S \cap V \subseteq -B \wedge$ 
 $(\forall e > 0. \exists f \in R. f \cdot S \subseteq \{0..1\} \wedge (\forall x \in S \cap V. f \ x < e) \wedge (\forall x \in S$ 
 $\cap B. f \ x > 1 - e))$ 
    using one [of -B w b] assms  $\langle w \in A \rangle$  by simp
  }
  then obtain Vf where Vf:
 $\bigwedge w. w \in A \implies \text{open } (Vf\ w) \wedge w \in Vf\ w \wedge S \cap Vf\ w \subseteq -B \wedge$ 
 $(\forall e > 0. \exists f \in R. f \cdot S \subseteq \{0..1\} \wedge (\forall x \in S \cap Vf\ w. f \ x < e) \wedge$ 
 $(\forall x \in S \cap B. f \ x > 1 - e))$ 

```

```

    by metis
  then have open_Vf:  $\bigwedge w. w \in A \implies \text{open } (Vf\ w)$ 
    by blast
  have tVf:  $\bigwedge w. w \in A \implies w \in Vf\ w$ 
    using Vf by blast
  then have sum_max_0:  $A \subseteq (\bigcup x \in A. Vf\ x)$ 
    by blast
  have com_A: compact A using A
    by (metis compact compact_Int_closed inf.absorb_iff2)
  obtain subA where subA:  $subA \subseteq A$  finite subA  $A \subseteq (\bigcup x \in subA. Vf\ x)$ 
    by (blast intro: that compactE_image [OF com_A open_Vf sum_max_0])
  then have [simp]:  $subA \neq \{\}$ 
    using  $\langle a \in A \rangle$  by auto
  then have cardp:  $\text{card } subA > 0$  using subA
    by (simp add: card_gt_0_iff)
  have  $\bigwedge w. w \in A \implies \exists f \in R. f \text{ ` } S \subseteq \{0..1\} \wedge (\forall x \in S \cap Vf\ w. f\ x < e / \text{card } subA) \wedge (\forall x \in S \cap B. f\ x > 1 - e / \text{card } subA)$ 
    using Vf e cardp by simp
  then obtain ff where ff:
     $\bigwedge w. w \in A \implies ff\ w \in R \wedge ff\ w \text{ ` } S \subseteq \{0..1\} \wedge$ 
     $(\forall x \in S \cap Vf\ w. ff\ w\ x < e / \text{card } subA) \wedge (\forall x \in S \cap B. ff\ w$ 
 $x > 1 - e / \text{card } subA)$ 
    by metis
  define pff where [abs_def]:  $pff\ x = (\prod w \in subA. ff\ w\ x)$  for x
  have pffR:  $pff \in R$ 
    unfolding pff_def using subA ff by (auto simp: intro: prod)
  moreover
  have pff01:  $pff\ x \in \{0..1\}$  if t:  $x \in S$  for x
  proof -
    have  $0 \leq pff\ x$ 
      using subA cardp t ff
    by (fastforce simp: pff_def field_split_simps sum_nonneg intro: prod_nonneg)
  moreover have  $pff\ x \leq 1$ 
    using subA cardp t ff
  by (fastforce simp add: pff_def field_split_simps sum_nonneg intro: prod_mono
[where g =  $\lambda x. 1$ , simplified])
  ultimately show ?thesis
    by auto
qed
moreover
{ fix v x
  assume v:  $v \in subA$  and x:  $x \in Vf\ v \implies x \in S$ 
  from subA v have  $pff\ x = ff\ v\ x * (\prod w \in subA - \{v\}. ff\ w\ x)$ 
    unfolding pff_def by (metis prod.remove)
  also have  $\dots \leq ff\ v\ x * 1$ 
  proof -
    have  $\bigwedge i. i \in subA - \{v\} \implies 0 \leq ff\ i\ x \wedge ff\ i\ x \leq 1$ 
      by (metis Diff_subset atLeastAtMost_iff ff_image_subset_iff subA(1) subsetD
x(2))

```

```

    moreover have  $0 \leq \text{ff } v \ x$ 
      using  $\text{ff } \text{subA}(1) \ v \ x(2)$  by fastforce
    ultimately show ?thesis
      by (metis mult_left_mono prod_mono [where  $g = \lambda x. 1$ , simplified])
  qed
  also have  $\dots < e / \text{card } \text{subA}$ 
    using  $\text{ff } \text{subA}(1) \ v \ x$  by auto
  also have  $\dots \leq e$ 
    using cardp e by (simp add: field_split_simps)
  finally have  $\text{pff } x < e$  .
}
then have  $\bigwedge x. x \in A \implies \text{pff } x < e$ 
  using  $A \ \forall f \ \text{subA}$  by (metis UN_E contra_subsetD)
moreover
{ fix x
  assume  $x: x \in B$ 
  then have  $x \in S$ 
    using B by auto
  have  $1 - e \leq (1 - e / \text{card } \text{subA})^{\text{card } \text{subA}}$ 
    using Bernoulli_inequality [of  $-e / \text{card } \text{subA}$   $\text{card } \text{subA}$ ] e cardp
    by (auto simp: field_simps)
  also have  $\dots = (\prod w \in \text{subA}. 1 - e / \text{card } \text{subA})$ 
    by (simp add: subA(2))
  also have  $\dots < \text{pff } x$ 
  proof -
    have  $\bigwedge i. i \in \text{subA} \implies e / \text{real } (\text{card } \text{subA}) \leq 1 \wedge 1 - e / \text{real } (\text{card } \text{subA})$ 
  <  $\text{ff } i \ x$ 
    using  $e \ \langle B \subseteq S \rangle \ \text{ff } \text{subA}(1) \ x$  by (force simp: field_split_simps)
    then show ?thesis
      using prod_mono_strict [where  $f = \lambda x. 1 - e / \text{card } \text{subA}$ ] subA(2) by
(force simp add: pff_def)
  qed
  finally have  $1 - e < \text{pff } x$  .
}
ultimately show ?thesis by blast
qed

lemma (in function_ring_on) two:
  assumes  $A: \text{closed } A \ A \subseteq S$ 
    and  $B: \text{closed } B \ B \subseteq S$ 
    and  $\text{disj}: A \cap B = \{\}$ 
    and  $e: 0 < e \ e < 1$ 
  shows  $\exists f \in R. f \restriction S \subseteq \{0..1\} \wedge (\forall x \in A. f \ x < e) \wedge (\forall x \in B. f \ x > 1 - e)$ 
proof (cases  $A \neq \{\} \wedge B \neq \{\}$ )
case True then show ?thesis
  using assms
  by (force simp flip: ex_in_conv intro!: two_special)
next
case False

```

```

then consider A={ } | B={ } by force
then show ?thesis
proof cases
  case 1
  with e show ?thesis
    by (rule_tac x= $\lambda x. 1$  in bexI) (auto simp: const)
  next
  case 2
  with e show ?thesis
    by (rule_tac x= $\lambda x. 0$  in bexI) (auto simp: const)
qed
qed

```

The special case where f is non-negative and $e < (1::'a) / (3::'a)$

```

lemma (in function_ring_on) Stone_Weierstrass_special:
  assumes f: continuous_on S f and fpos:  $\bigwedge x. x \in S \implies f x \geq 0$ 
  and e:  $0 < e \wedge e < 1/3$ 
  shows  $\exists g \in R. \forall x \in S. |f x - g x| < 2 * e$ 
proof -
  define n where  $n = 1 + \text{nat } \lceil \text{norm } f / e \rceil$ 
  define A where  $A j = \{x \in S. f x \leq (j - 1/3) * e\}$  for  $j :: \text{nat}$ 
  define B where  $B j = \{x \in S. f x \geq (j + 1/3) * e\}$  for  $j :: \text{nat}$ 
  have ng1:  $(n-1) * e \geq \text{norm } f$ 
  using e pos_divide_le_eq real_nat_ceiling_ge[of norm f / e]
  by (fastforce simp add: divide_simps n_def)
  moreover have n $\geq$ 1
  by (simp_all add: n_def)
  ultimately have ge_fx:  $(n-1) * e \geq f x$  if  $x \in S$  for  $x$ 
  using f normf_upper that by fastforce
  have closed S
  by (simp add: compact_compact_imp_closed)
  { fix j
    have closed (A j) A j  $\subseteq S$ 
    using  $\langle \text{closed } S \rangle$  continuous_on_closed_Collect_le [OF f continuous_on_const]
    by (simp_all add: A_def Collect_restrict)
    moreover have closed (B j) B j  $\subseteq S$ 
    using  $\langle \text{closed } S \rangle$  continuous_on_closed_Collect_le [OF f continuous_on_const]
    by (simp_all add: B_def Collect_restrict)
    moreover have (A j)  $\cap$  (B j) = { }
    using e by (auto simp: A_def B_def field_simps)
    ultimately have  $\exists f \in R. f ' S \subseteq \{0..1\} \wedge (\forall x \in A j. f x < e/n) \wedge (\forall x \in B j. f x > 1 - e/n)$ 
    using e  $\langle 1 \leq n \rangle$  by (auto intro: two)
  }
  then obtain xf where xfR:  $\bigwedge j. xf j \in R$  and xf01:  $\bigwedge j. xf j ' S \subseteq \{0..1\}$ 
  and xfA:  $\bigwedge x j. x \in A j \implies xf j x < e/n$ 
  and xfB:  $\bigwedge x j. x \in B j \implies xf j x > 1 - e/n$ 
  by metis

```

```

define g where [abs_def]:  $g\ x = e * (\sum_{i \leq n}. x\ i\ x)$  for x
have gR:  $g \in R$ 
  unfolding g_def by (fast intro: mult_const_sum xfR)
have gge0:  $\bigwedge x. x \in S \implies g\ x \geq 0$ 
  using e xf01 by (simp add: g_def zero_le_mult_iff image_subset_iff sum_nonneg)
have A0:  $A\ 0 = \{\}$ 
  using fpos e by (fastforce simp: A_def)
have An:  $A\ n = S$ 
  using e ngt <n≥1> f normf_upper by (fastforce simp: A_def field_simps
of_nat_diff)
have Asub:  $A\ j \subseteq A\ i$  if  $i \geq j$  for i j
  using e that by (force simp: A_def intro: order_trans)
{ fix t
  assume t:  $t \in S$ 
  define j where  $j = (LEAST\ j. t \in A\ j)$ 
  have jn:  $j \leq n$ 
    using t An by (simp add: Least_le j_def)
  have Aj:  $t \in A\ j$ 
    using t An by (fastforce simp add: j_def intro: LeastI)
  then have Ai:  $t \in A\ i$  if  $i \geq j$  for i
    using Asub [OF that] by blast
  then have fj1:  $f\ t \leq (j - 1/3)*e$ 
    by (simp add: A_def)
  then have Anj:  $t \notin A\ i$  if  $i < j$  for i
    using Aj <i<j> not_less_Least by (fastforce simp add: j_def)
  have j1:  $1 \leq j$ 
    using A0 Aj j_def not_less_eq_eq by (fastforce simp add: j_def)
  then have Anj:  $t \notin A\ (j-1)$ 
    using Least_le by (fastforce simp add: j_def)
  then have fj2:  $(j - 4/3)*e < f\ t$ 
    using j1 t by (simp add: A_def of_nat_diff)
  have xf_le1:  $\bigwedge i. x\ i\ t \leq 1$ 
    using xf01 t by force
  have g t =  $e * (\sum_{i \leq n}. x\ i\ t)$ 
    by (simp add: g_def flip: distrib_left)
  also have ... =  $e * (\sum i \in \{..<j\} \cup \{j..n\}. x\ i\ t)$ 
    by (simp add: ivl_disj_un_one(4) jn)
  also have ... =  $e * (\sum_{i < j}. x\ i\ t) + e * (\sum_{i = j..n}. x\ i\ t)$ 
    by (simp add: distrib_left ivl_disj_int sum.union_disjoint)
  also have ...  $\leq e*j + e * ((Suc\ n - j)*e/n)$ 
  proof (intro add_mono mult_left_mono)
    show  $(\sum_{i < j}. x\ i\ t) \leq j$ 
      by (rule sum_bounded_above [OF xf_le1, where A = lessThan j, simplified])
    have  $x\ i\ t \leq e/n$  if  $i \geq j$  for i
      using xFA [OF Ai] that by (simp add: less_eq_real_def)
    then show  $(\sum_{i = j..n}. x\ i\ t) \leq \text{real}\ (Suc\ n - j) * e / \text{real}\ n$ 
      using sum_bounded_above [of {j..n}  $\lambda i. x\ i\ t$ ]
        by fastforce
  qed (use e in auto)

```

```

also have ... ≤ j*e + e*(n - j + 1)*e/n
  using ⟨1 ≤ n⟩ e by (simp add: field_simps del: of_nat_Suc)
also have ... ≤ j*e + e*e
  using ⟨1 ≤ n⟩ e j1 by (simp add: field_simps del: of_nat_Suc)
also have ... < (j + 1/3)*e
  using e by (auto simp: field_simps)
finally have gj1: g t < (j + 1 / 3) * e .
have gj2: (j - 4/3)*e < g t
proof (cases 2 ≤ j)
  case False
  then have j=1 using j1 by simp
  with t gge0 e show ?thesis by force
next
  case True
  then have (j - 4/3)*e < (j-1)*e - e^2
    using e by (auto simp: of_nat_diff algebra_simps power2_eq_square)
  also have ... < (j-1)*e - ((j - 1)/n) * e^2
    using e True jn by (simp add: power2_eq_square field_simps)
  also have ... = e * (j-1) * (1 - e/n)
    by (simp add: power2_eq_square field_simps)
  also have ... ≤ e * (∑ i≤j-2. x f i t)
  proof -
    { fix i
      assume i+2 ≤ j
      then obtain d where i+2+d = j
        using le_Suc_ex that by blast
      then have t ∈ B i
        using Anj e ge_fx [OF t] ⟨1 ≤ n⟩ fpos [OF t] t
        unfolding A_def B_def
        by (auto simp add: field_simps of_nat_diff not_le intro: order_trans
[of _ e * 2 + e * d * 3 + e * i * 3])
      then have x f i t > 1 - e/n
        by (rule x f B)
    }
    moreover have real (j - Suc 0) * (1 - e / real n) ≤ real (card {..j - 2})
      * (1 - e / real n)
      using Suc_diff_le True by fastforce
    ultimately show ?thesis
      using e True by (auto intro: order_trans [OF _ sum_bounded_below [OF
less_imp_le]])
  qed
  also have ... ≤ g t
    using jn e x f 01 t
    by (auto intro!: Groups_Big.sum_mono2 simp add: g_def zero_le_mult_iff
image_subset_iff sum_nonneg)
  finally show ?thesis .
qed
have |f t - g t| < 2 * e
  using fj1 fj2 gj1 gj2 by (simp add: abs_less_iff field_simps)

```

```

}
then show ?thesis
  by (rule_tac x=g in bexI) (auto intro: gR)
qed

```

The “unpretentious” formulation

```

proposition (in function_ring_on) Stone_Weierstrass_basic:
  assumes f: continuous_on S f and e: e > 0
  shows  $\exists g \in R. \forall x \in S. |f x - g x| < e$ 
proof -
  have  $\exists g \in R. \forall x \in S. |(f x + normf f) - g x| < 2 * \min (e/2) (1/4)$ 
proof (rule Stone_Weierstrass_special)
  show continuous_on S ( $\lambda x. f x + normf f$ )
    by (force intro: Limits.continuous_on_add [OF Topological_Spaces.continuous_on_const])
  show  $\bigwedge x. x \in S \implies 0 \leq f x + normf f$ 
    using normf_upper [OF f] by force
qed (use e in auto)
then obtain g where  $g \in R \forall x \in S. |g x - (f x + normf f)| < e$ 
  by force
then show ?thesis
  by (rule_tac x= $\lambda x. g x - normf f$  in bexI) (auto simp: algebra_simps intro:
diff const)
qed

```

```

theorem (in function_ring_on) Stone_Weierstrass:
  assumes f: continuous_on S f
  shows  $\exists F \in UNIV \rightarrow R. LIM n \text{ sequentially. } F n :> \text{uniformly\_on } S f$ 
proof -
  define h where  $h \equiv \lambda n::nat. SOME g. g \in R \wedge (\forall x \in S. |f x - g x| < 1 / (1 + n))$ 
  show ?thesis
proof
  { fix e::real
    assume e:  $0 < e$ 
    then obtain N::nat where  $N: 0 < N \wedge 0 < \text{inverse } N \wedge \text{inverse } N < e$ 
      by (auto simp: real_arch_inverse [of e])
    { fix n :: nat and x :: 'a and g :: 'a  $\Rightarrow$  real
      assume n:  $N \leq n \wedge \forall x \in S. |f x - g x| < 1 / (1 + \text{real } n)$ 
      assume x:  $x \in S$ 
      have  $\neg \text{real } (Suc n) < \text{inverse } e$ 
        using  $\langle N \leq n \rangle N$  using less_imp_inverse_less by force
      then have  $1 / (1 + \text{real } n) \leq e$ 
        using e by (simp add: field_simps)
      then have  $|f x - g x| < e$ 
        using n(2) x by auto
    }
  }
  then have  $\forall_F n \text{ in sequentially. } \forall x \in S. |f x - h n x| < e$ 
    unfolding h_def

```

```

      by (force intro: someI2_bex [OF Stone_Weierstrass_basic [OF f]] eventu-
ally_sequentiallyI [of N])
    }
  then show uniform_limit S h f sequentially
  unfolding uniform_limit_iff by (auto simp: dist_norm abs_minus_commute)
  show h ∈ UNIV → R
  unfolding h_def by (force intro: someI2_bex [OF Stone_Weierstrass_basic
[OF f]])
qed
qed

```

A HOL Light formulation

```

corollary Stone_Weierstrass_HOL:
  fixes R :: ('a::t2_space ⇒ real) set and S :: 'a set
  assumes compact S ∧ c. P(λx. c::real)
    ∧ f. P f ⇒ continuous_on S f
    ∧ f g. P(f) ∧ P(g) ⇒ P(λx. f x + g x) ∧ f g. P(f) ∧ P(g) ⇒ P(λx. f
x * g x)
    ∧ x y. x ∈ S ∧ y ∈ S ∧ x ≠ y ⇒ ∃ f. P(f) ∧ f x ≠ f y
    continuous_on S f
    0 < e
  shows ∃ g. P(g) ∧ (∀ x ∈ S. |f x - g x| < e)
proof -
  interpret PR: function_ring_on Collect P
  by unfold_locales (use assms in auto)
  show ?thesis
  using PR.Stone_Weierstrass_basic [OF ⟨continuous_on S f⟩ ⟨0 < e⟩]
  by blast
qed

```

5.6.4 Polynomial functions

```

inductive real_polynomial_function :: ('a::real_normed_vector ⇒ real) ⇒ bool
where

```

```

  linear: bounded_linear f ⇒ real_polynomial_function f
  | const: real_polynomial_function (λx. c)
  | add: [[real_polynomial_function f; real_polynomial_function g]] ⇒ real_polynomial_function
(λx. f x + g x)
  | mult: [[real_polynomial_function f; real_polynomial_function g]] ⇒ real_polynomial_function
(λx. f x * g x)

```

```

declare real_polynomial_function.intros [intro]

```

```

definition polynomial_function :: ('a::real_normed_vector ⇒ 'b::real_normed_vector)
⇒ bool
where
  polynomial_function p ≡ (∀ f. bounded_linear f ⟶ real_polynomial_function
(f o p))

```


lemma *real_polynomial_function_eq*: *real_polynomial_function* *p* = *polynomial_function* *p*

unfolding *polynomial_function_def*

proof

assume *real_polynomial_function* *p*

then show $\forall f. \text{bounded\_linear } f \longrightarrow \text{real\_polynomial\_function } (f \circ p)$

proof (*induction* *p* *rule*: *real_polynomial_function.induct*)

case (*linear* *h*) **then show** ?*case*

by (*auto simp*: *bounded_linear_compose* *real_polynomial_function.linear*)

next

case (*const* *h*) **then show** ?*case*

by (*simp add*: *real_polynomial_function.const*)

next

case (*add* *h*) **then show** ?*case*

by (*force simp add*: *bounded_linear_def* *linear_add* *real_polynomial_function.add*)

next

case (*mult* *h*) **then show** ?*case*

by (*force simp add*: *real_bounded_linear* *const* *real_polynomial_function.mult*)

qed

next

assume [*rule_format*, *OF* *bounded_linear_ident*]: $\forall f. \text{bounded\_linear } f \longrightarrow \text{real\_polynomial\_function } (f \circ p)$

then show *real_polynomial_function* *p*

by (*simp add*: *o_def*)

qed

lemma *polynomial_function_const* [*iff*]: *polynomial_function* ($\lambda x. c$)

by (*simp add*: *polynomial_function_def* *o_def* *const*)

lemma *polynomial_function_bounded_linear*:

bounded_linear *f* $\implies$ *polynomial_function* *f*

by (*simp add*: *polynomial_function_def* *o_def* *bounded_linear_compose* *real_polynomial_function.linear*)

lemma *polynomial_function_id* [*iff*]: *polynomial_function* ($\lambda x. x$)

by (*simp add*: *polynomial_function_bounded_linear*)

lemma *polynomial_function_add* [*intro*]:

$\llbracket \text{polynomial\_function } f; \text{polynomial\_function } g \rrbracket \implies \text{polynomial\_function } (\lambda x. f\ x + g\ x)$

by (*auto simp*: *polynomial_function_def* *bounded_linear_def* *linear_add* *real_polynomial_function.add* *o_def*)

lemma *polynomial_function_mult* [*intro*]:

assumes *f*: *polynomial_function* *f* **and** *g*: *polynomial_function* *g*

shows *polynomial_function* ($\lambda x. f\ x *_{\mathbb{R}} g\ x$)

proof –

have *real_polynomial_function* ($\lambda x. h\ (g\ x)$) **if** *bounded_linear* *h* **for** *h*

using *g* **that** **unfolding** *polynomial_function_def* *o_def* *bounded_linear_def*

by (*auto simp*: *real_polynomial_function_eq*)

```

moreover have real_polynomial_function f
  by (simp add: f real_polynomial_function_eq)
ultimately show ?thesis
  unfolding polynomial_function_def bounded_linear_def o_def
  by (auto simp: linear.scaleR)
qed

```

```

lemma polynomial_function_cmul [intro]:
  assumes f: polynomial_function f
  shows polynomial_function ( $\lambda x. c *_{\mathbb{R}} f x$ )
  by (rule polynomial_function_mult [OF polynomial_function_const f])

```

```

lemma polynomial_function_minus [intro]:
  assumes f: polynomial_function f
  shows polynomial_function ( $\lambda x. - f x$ )
  using polynomial_function_cmul [OF f, of  $-1$ ] by simp

```

```

lemma polynomial_function_diff [intro]:
   $\llbracket \text{polynomial\_function } f; \text{polynomial\_function } g \rrbracket \implies \text{polynomial\_function } (\lambda x. f x - g x)$ 
  unfolding add_uminus_conv_diff [symmetric]
  by (metis polynomial_function_add polynomial_function_minus)

```

```

lemma polynomial_function_sum [intro]:
   $\llbracket \text{finite } I; \bigwedge i. i \in I \implies \text{polynomial\_function } (\lambda x. f x i) \rrbracket \implies \text{polynomial\_function } (\lambda x. \text{sum } (f x) I)$ 
  by (induct I rule: finite_induct) auto

```

```

lemma real_polynomial_function_minus [intro]:
   $\text{real\_polynomial\_function } f \implies \text{real\_polynomial\_function } (\lambda x. - f x)$ 
  using polynomial_function_minus [of f]
  by (simp add: real_polynomial_function_eq)

```

```

lemma real_polynomial_function_diff [intro]:
   $\llbracket \text{real\_polynomial\_function } f; \text{real\_polynomial\_function } g \rrbracket \implies \text{real\_polynomial\_function } (\lambda x. f x - g x)$ 
  using polynomial_function_diff [of f]
  by (simp add: real_polynomial_function_eq)

```

```

lemma real_polynomial_function_sum [intro]:
   $\llbracket \text{finite } I; \bigwedge i. i \in I \implies \text{real\_polynomial\_function } (\lambda x. f x i) \rrbracket \implies \text{real\_polynomial\_function } (\lambda x. \text{sum } (f x) I)$ 
  using polynomial_function_sum [of I f]
  by (simp add: real_polynomial_function_eq)

```

```

lemma real_polynomial_function_power [intro]:
   $\text{real\_polynomial\_function } f \implies \text{real\_polynomial\_function } (\lambda x. f x ^ n)$ 
  by (induct n) (simp_all add: const_mult)

```

```

lemma real_polynomial_function_compose [intro]:
  assumes f: polynomial_function f and g: real_polynomial_function g
  shows real_polynomial_function (g o f)
  using g
proof (induction g rule: real_polynomial_function.induct)
  case (linear f)
  then show ?case
    using f polynomial_function_def by blast
next
  case (add f g)
  then show ?case
    using f add by (auto simp: polynomial_function_def)
next
  case (mult f g)
  then show ?case
    using f mult by (auto simp: polynomial_function_def)
qed auto

```

```

lemma polynomial_function_compose [intro]:
  assumes f: polynomial_function f and g: polynomial_function g
  shows polynomial_function (g o f)
  using g real_polynomial_function_compose [OF f]
  by (auto simp: polynomial_function_def o_def)

```

```

lemma sum_max_0:
  fixes x::real
  shows  $(\sum_{i \leq \max m n} x^i * (\text{if } i \leq m \text{ then } a \ i \text{ else } 0)) = (\sum_{i \leq m} x^i * a \ i)$ 
proof -
  have  $(\sum_{i \leq \max m n} x^i * (\text{if } i \leq m \text{ then } a \ i \text{ else } 0)) = (\sum_{i \leq \max m n} (\text{if } i \leq m \text{ then } x^i * a \ i \text{ else } 0))$ 
  by (auto simp: algebra_simps intro: sum.cong)
  also have ... =  $(\sum_{i \leq m} (\text{if } i \leq m \text{ then } x^i * a \ i \text{ else } 0))$ 
  by (rule sum.mono_neutral_right) auto
  also have ... =  $(\sum_{i \leq m} x^i * a \ i)$ 
  by (auto simp: algebra_simps intro: sum.cong)
  finally show ?thesis .
qed

```

```

lemma real_polynomial_function_imp_sum:
  assumes real_polynomial_function f
  shows  $\exists a \ n::\text{nat}. f = (\lambda x. \sum_{i \leq n} a \ i * x^i)$ 
using assms
proof (induct f)
  case (linear f)
  then obtain c where f: f =  $(\lambda x. x * c)$ 
  by (auto simp add: real_bounded_linear)
  have x * c =  $(\sum_{i \leq 1} (\text{if } i = 0 \text{ then } 0 \text{ else } c) * x^i)$  for x
  by (simp add: mult_ac)
  with f show ?case

```

```

    by fastforce
next
  case (const c)
  have c = ( $\sum i \leq 0. c * x^i$ ) for x
  by auto
  then show ?case
  by fastforce
  case (add f1 f2)
  then obtain a1 n1 a2 n2 where
    f1 = ( $\lambda x. \sum i \leq n1. a1 i * x^i$ ) f2 = ( $\lambda x. \sum i \leq n2. a2 i * x^i$ )
  by auto
  then have f1 x + f2 x = ( $\sum i \leq \max n1 n2. ((\text{if } i \leq n1 \text{ then } a1 i \text{ else } 0) + (\text{if } i \leq n2 \text{ then } a2 i \text{ else } 0)) * x^i$ )
  for x
  using sum_max_0 [where m=n1 and n=n2] sum_max_0 [where m=n2 and n=n1]
  by (simp add: sum.distrib algebra_simps max.commute)
  then show ?case
  by force
  case (mult f1 f2)
  then obtain a1 n1 a2 n2 where
    f1 = ( $\lambda x. \sum i \leq n1. a1 i * x^i$ ) f2 = ( $\lambda x. \sum i \leq n2. a2 i * x^i$ )
  by auto
  then obtain b1 b2 where
    f1 = ( $\lambda x. \sum i \leq n1. b1 i * x^i$ ) f2 = ( $\lambda x. \sum i \leq n2. b2 i * x^i$ )
    b1 = ( $\lambda i. \text{if } i \leq n1 \text{ then } a1 i \text{ else } 0$ ) b2 = ( $\lambda i. \text{if } i \leq n2 \text{ then } a2 i \text{ else } 0$ )
  by auto
  then have f1 x * f2 x = ( $\sum i \leq n1 + n2. (\sum k \leq i. b1 k * b2 (i - k)) * x^i$ )
  for x
  using polynomial_product [of n1 b1 n2 b2] by (simp add: Set_Interval.atLeast0AtMost)
  then show ?case
  by force
qed

```

lemma *real_polynomial_function_iff_sum:*

real_polynomial_function f $\longleftrightarrow$ ($\exists a n. f = (\lambda x. \sum i \leq n. a i * x^i)$) (is ?lhs = ?rhs)

proof

assume ?lhs then show ?rhs

by (metis real_polynomial_function_imp_sum)

next

assume ?rhs then show ?lhs

by (auto simp: linear_mult_const real_polynomial_function_power real_polynomial_function_sum)

qed

lemma *polynomial_function_iff_Basis_inner:*

fixes f :: 'a::real_normed_vector $\Rightarrow$ 'b::euclidean_space

shows *polynomial_function* f $\longleftrightarrow$ ($\forall b \in \text{Basis}. \text{real\_polynomial\_function } (\lambda x. \text{inner } (f x) b)$)

```

      (is ?lhs = ?rhs)
unfolding polynomial_function_def
proof (intro iffI allI impI)
  assume  $\forall h. \text{bounded\_linear } h \longrightarrow \text{real\_polynomial\_function } (h \circ f)$ 
  then show ?rhs
    by (force simp add: bounded_linear_inner_left o_def)
next
  fix  $h :: 'b \Rightarrow \text{real}$ 
  assume rp:  $\forall b \in \text{Basis}. \text{real\_polynomial\_function } (\lambda x. f x \cdot b)$  and  $h: \text{bounded\_linear } h$ 
  have  $\text{real\_polynomial\_function } (h \circ (\lambda x. \sum b \in \text{Basis}. (f x \cdot b) *_{\mathbb{R}} b))$ 
    using rp
    by (force simp: real_polynomial_function_eq polynomial_function_mult
      intro!: real_polynomial_function_compose [OF _ linear [OF h]])
  then show  $\text{real\_polynomial\_function } (h \circ f)$ 
    by (simp add: euclidean_representation_sum_fun)
qed

```

5.6.5 Stone-Weierstrass theorem for polynomial functions

First, we need to show that they are continuous, differentiable and separable.

lemma *continuous_real_polynomial_function*:

```

  assumes  $\text{real\_polynomial\_function } f$ 
  shows  $\text{continuous } (\text{at } x) f$ 
using assms
by (induct f) (auto simp: linear_continuous_at)

```

lemma *continuous_polynomial_function*:

```

  fixes  $f :: 'a :: \text{real\_normed\_vector} \Rightarrow 'b :: \text{euclidean\_space}$ 
  assumes  $\text{polynomial\_function } f$ 
  shows  $\text{continuous } (\text{at } x) f$ 
proof (rule euclidean_isCont)
  show  $\bigwedge b. b \in \text{Basis} \implies \text{isCont } (\lambda x. (f x \cdot b) *_{\mathbb{R}} b) x$ 
    using assms continuous_real_polynomial_function
    by (force simp: polynomial_function_iff_Basis_inner intro: isCont_scaleR)
qed

```

lemma *continuous_on_polynomial_function*:

```

  fixes  $f :: 'a :: \text{real\_normed\_vector} \Rightarrow 'b :: \text{euclidean\_space}$ 
  assumes  $\text{polynomial\_function } f$ 
  shows  $\text{continuous\_on } S f$ 
using continuous_polynomial_function [OF assms] continuous_at_imp_continuous_on
by blast

```

lemma *has_real_derivative_polynomial_function*:

```

  assumes  $\text{real\_polynomial\_function } p$ 
  shows  $\exists p'. \text{real\_polynomial\_function } p' \wedge$ 
     $(\forall x. (p \text{ has\_real\_derivative } (p' x)) (\text{at } x))$ 
using assms

```

```

proof (induct p)
  case (linear p)
  then show ?case
    by (force simp: real_bounded_linear const intro!: derivative_eq_intros)
next
  case (const c)
  show ?case
    by (rule_tac x= $\lambda x. 0$  in exI) auto
  case (add f1 f2)
  then obtain p1 p2 where
    real_polynomial_function p1  $\wedge x. (f1 \text{ has\_real\_derivative } p1 \ x) \ (at \ x)$ 
    real_polynomial_function p2  $\wedge x. (f2 \text{ has\_real\_derivative } p2 \ x) \ (at \ x)$ 
    by auto
  then show ?case
    by (rule_tac x= $\lambda x. p1 \ x + p2 \ x$  in exI) (auto intro!: derivative_eq_intros)
  case (mult f1 f2)
  then obtain p1 p2 where
    real_polynomial_function p1  $\wedge x. (f1 \text{ has\_real\_derivative } p1 \ x) \ (at \ x)$ 
    real_polynomial_function p2  $\wedge x. (f2 \text{ has\_real\_derivative } p2 \ x) \ (at \ x)$ 
    by auto
  then show ?case
    using mult
    by (rule_tac x= $\lambda x. f1 \ x * p2 \ x + f2 \ x * p1 \ x$  in exI) (auto intro!: derivative_eq_intros)
qed

```

lemma has_vector_derivative_polynomial_function:

```

  fixes p :: real  $\Rightarrow$  'a::euclidean_space
  assumes polynomial_function p
  obtains p' where polynomial_function p'  $\wedge x. (p \text{ has\_vector\_derivative } (p' \ x)) \ (at \ x)$ 
proof -
  { fix b :: 'a
    assume b  $\in$  Basis
    then
      obtain p' where p': real_polynomial_function p' and pd:  $\wedge x. ((\lambda x. p \ x \cdot b) \text{ has\_real\_derivative } p' \ x) \ (at \ x)$ 
      using assms [unfolded polynomial_function_iff_Basis_inner] has_real_derivative_polynomial_function
      by blast
      have polynomial_function ( $\lambda x. p' \ x *_{\mathbb{R}} b$ )
      using  $\langle b \in \text{Basis} \rangle$  p' const [where 'a=real and c=0]
      by (simp add: polynomial_function_iff_Basis_inner inner_Basis)
      then have  $\exists q. \text{polynomial\_function } q \wedge (\forall x. ((\lambda u. (p \ u \cdot b) *_{\mathbb{R}} b) \text{ has\_vector\_derivative } q \ x) \ (at \ x))$ 
      by (fastforce intro: derivative_eq_intros pd)
    }
  then obtain qf where qf:
     $\wedge b. b \in \text{Basis} \implies \text{polynomial\_function } (qf \ b)$ 
     $\wedge b \ x. b \in \text{Basis} \implies ((\lambda u. (p \ u \cdot b) *_{\mathbb{R}} b) \text{ has\_vector\_derivative } qf \ b \ x) \ (at \ x)$ 

```

```

    by metis
  show ?thesis
proof
  show  $\bigwedge x. (p \text{ has\_vector\_derivative } (\sum b \in \text{Basis}. qf \ b \ x)) \ (at \ x)$ 
    apply (subst euclidean_representation_sum_fun [of p, symmetric])
    by (auto intro: has_vector_derivative_sum qf)
  qed (force intro: qf)
qed

lemma real_polynomial_function_separable:
  fixes x :: 'a::euclidean_space
  assumes x  $\neq$  y shows  $\exists f. \text{real\_polynomial\_function } f \wedge f \ x \neq f \ y$ 
proof -
  have real_polynomial_function  $(\lambda u. \sum b \in \text{Basis}. (\text{inner } (x-u) \ b)^2)$ 
  proof (rule real_polynomial_function_sum)
    show  $\bigwedge i. i \in \text{Basis} \implies \text{real\_polynomial\_function } (\lambda u. ((x-u) \cdot i)^2)$ 
      by (auto simp: algebra_simps real_polynomial_function_diff const linear
        bounded_linear_inner_left)
    qed auto
    moreover have  $(\sum b \in \text{Basis}. ((x-y) \cdot b)^2) \neq 0$ 
      using assms by (force simp add: euclidean_eq_iff [of x y] sum_nonneg_eq_0_iff
        algebra_simps)
    ultimately show ?thesis
      by auto
  qed

lemma Stone_Weierstrass_real_polynomial_function:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes compact S continuous_on S f 0 < e
  obtains g where real_polynomial_function g  $\bigwedge x. x \in S \implies |f \ x - g \ x| < e$ 
proof -
  interpret PR: function_ring_on Collect real_polynomial_function
  proof unfold_locales
    qed (use assms continuous_on_polynomial_function real_polynomial_function_eq

      in  $\langle \text{auto intro: real\_polynomial\_function\_separable} \rangle$ )
    show ?thesis
      using PR.Stone_Weierstrass_basic [OF  $\langle \text{continuous\_on } S \ f \ \langle 0 < e \rangle$  ] that by
        blast
    qed

theorem Stone_Weierstrass_polynomial_function:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes S: compact S
    and f: continuous_on S f
    and e: 0 < e
  shows  $\exists g. \text{polynomial\_function } g \wedge (\forall x \in S. \text{norm}(f \ x - g \ x) < e)$ 
proof -
  { fix b :: 'b

```

```

    assume  $b \in \text{Basis}$ 
    have  $\exists p. \text{real\_polynomial\_function } p \wedge (\forall x \in S. |f x \cdot b - p x| < e / \text{DIM}('b))$ 
    proof (rule Stone_Weierstrass_real_polynomial_function [OF  $S \_$ , of  $\lambda x. f x \cdot b$  e / card  $\text{Basis}$ ])
      show continuous_on  $S$  ( $\lambda x. f x \cdot b$ )
      using  $f$  by (auto intro: continuous_intros)
    qed (use  $e$  in auto)
  }
  then obtain  $pf$  where  $pf$ :
     $\bigwedge b. b \in \text{Basis} \implies \text{real\_polynomial\_function } (pf\ b) \wedge (\forall x \in S. |f x \cdot b - pf\ b\ x| < e / \text{DIM}('b))$ 
  by metis
  let  $?g = \lambda x. \sum_{b \in \text{Basis}} pf\ b\ x *_{\mathbb{R}} b$ 
  { fix  $x$ 
    assume  $x \in S$ 
    have  $\text{norm } (\sum_{b \in \text{Basis}} (f x \cdot b) *_{\mathbb{R}} b - pf\ b\ x *_{\mathbb{R}} b) \leq (\sum_{b \in \text{Basis}} \text{norm } ((f x \cdot b) *_{\mathbb{R}} b - pf\ b\ x *_{\mathbb{R}} b))$ 
    by (rule norm_sum)
    also have  $\dots < \text{of\_nat } \text{DIM}('b) * (e / \text{DIM}('b))$ 
    proof (rule sum_bounded_above_strict)
      show  $\bigwedge i. i \in \text{Basis} \implies \text{norm } ((f x \cdot i) *_{\mathbb{R}} i - pf\ i\ x *_{\mathbb{R}} i) < e / \text{real\_DIM}('b)$ 
      by (simp add: Real_Vector_Spaces.scaleR_diff_left [symmetric]  $pf\ \langle x \in S \rangle$ )
    qed (rule DIM_positive)
    also have  $\dots = e$ 
    by (simp add: field_simps)
    finally have  $\text{norm } (\sum_{b \in \text{Basis}} (f x \cdot b) *_{\mathbb{R}} b - pf\ b\ x *_{\mathbb{R}} b) < e$  .
  }
  then have  $\forall x \in S. \text{norm } ((\sum_{b \in \text{Basis}} (f x \cdot b) *_{\mathbb{R}} b) - ?g\ x) < e$ 
  by (auto simp flip: sum_subtractf)
  moreover
  have polynomial_function  $?g$ 
  using  $pf$  by (simp add: polynomial_function_sum polynomial_function_mult real_polynomial_function_eq)
  ultimately show  $?thesis$ 
  using euclidean_representation_sum_fun [of  $f$ ] by (metis (no_types, lifting))
qed

```

proposition *Stone_Weierstrass_uniform_limit*:

```

  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes  $S$ : compact  $S$ 
  and  $f$ : continuous_on  $S$   $f$ 
  obtains  $g$  where uniform_limit  $S$   $g$   $f$  sequentially  $\bigwedge n. \text{polynomial\_function } (g\ n)$ 
proof -
  have  $\text{pos: } \text{inverse } (\text{Suc } n) > 0$  for  $n$  by auto
  obtain  $g$  where  $g$ :  $\bigwedge n. \text{polynomial\_function } (g\ n) \wedge x\ n. x \in S \implies \text{norm}(f\ x - g\ n\ x) < \text{inverse } (\text{Suc } n)$ 
  using Stone_Weierstrass_polynomial_function[OF  $S\ f\ \text{pos}$ ]
  by metis

```



```

have uniform_limit S g f sequentially
proof (rule uniform_limitI)
  fix e::real assume 0 < e
  with LIMSEQ_inverse_real_of_nat have  $\forall_F n$  in sequentially. inverse (Suc
n) < e
    by (rule order_tendstoD)
  moreover have  $\forall_F n$  in sequentially.  $\forall x \in S. \text{dist } (g \ n \ x) \ (f \ x) < \text{inverse } (Suc$ 
n)
    using g by (simp add: dist_norm norm_minus_commute)
  ultimately show  $\forall_F n$  in sequentially.  $\forall x \in S. \text{dist } (g \ n \ x) \ (f \ x) < e$ 
    by (eventually_elim) auto
qed
then show ?thesis using g(1) ..
qed

```

5.6.6 Polynomial functions as paths

One application is to pick a smooth approximation to a path, or just pick a smooth path anyway in an open connected set

```

lemma path_polynomial_function:
  fixes g :: real  $\Rightarrow$  'b::euclidean_space
  shows polynomial_function g  $\implies$  path g
  by (simp add: path_def continuous_on_polynomial_function)

lemma path_approx_polynomial_function:
  fixes g :: real  $\Rightarrow$  'b::euclidean_space
  assumes path g 0 < e
  obtains p where polynomial_function p pathstart p = pathstart g pathfinish p
= pathfinish g
 $\bigwedge t. t \in \{0..1\} \implies \text{norm}(p \ t - g \ t) < e$ 
proof -
  obtain q where poq: polynomial_function q and noq:  $\bigwedge x. x \in \{0..1\} \implies \text{norm}$ 
(g x - q x) < e/4
    using Stone_Weierstrass_polynomial_function [of {0..1} g e/4] assms
    by (auto simp: path_def)
  define pf where pf  $\equiv \lambda t. q \ t + (g \ 0 - q \ 0) + t *_R (g \ 1 - q \ 1 - (g \ 0 - q \ 0))$ 
  show thesis
  proof
    show polynomial_function pf
      by (force simp add: poq pf_def)
    show norm (pf t - g t) < e
      if t  $\in \{0..1\}$  for t
    proof -
      have *: norm (((q t - g t) + (g 0 - q 0)) + (t *_R (g 1 - q 1) + t *_R (q 0
- g 0))) < (e/4 + e/4) + (e/4+e/4)
      proof (intro Real_Vector_Spaces.norm_add_less)
        show norm (q t - g t) < e / 4
          by (metis noq norm_minus_commute that)
        show norm (t *_R (g 1 - q 1)) < e / 4

```

```

      using noq that le_less_trans [OF mult_left_le_one_le noq]
    by auto
  show norm (t *R (q 0 - g 0)) < e / 4
    using noq that le_less_trans [OF mult_left_le_one_le noq]
    by simp (metis norm_minus_commute order_refl zero_le_one)
  qed (use noq norm_minus_commute that in auto)
  then show ?thesis
    by (auto simp add: algebra_simps pf_def)
  qed
  qed (auto simp add: path_defs pf_def)
qed

proposition connected_open_polynomial_connected:
  fixes S :: 'a::euclidean_space set
  assumes S: open S connected S
    and x ∈ S y ∈ S
  shows ∃ g. polynomial_function g ∧ path_image g ⊆ S ∧ pathstart g = x ∧
pathfinish g = y
proof -
  have path_connected S using assms
    by (simp add: connected_open_path_connected)
  with ⟨x ∈ S⟩ ⟨y ∈ S⟩ obtain p where p: path p path_image p ⊆ S pathstart p
= x pathfinish p = y
    by (force simp: path_connected_def)
  have ∃ e. 0 < e ∧ (∀ x ∈ path_image p. ball x e ⊆ S)
proof (cases S = UNIV)
    case True then show ?thesis
      by (simp add: gt_ex)
  next
    case False
    show ?thesis
proof (intro exI conjI ballI)
    show ∧x. x ∈ path_image p ⇒ ball x (setdist (path_image p) (-S)) ⊆ S
      using setdist_le_dist [of _ path_image p _ -S] by fastforce
    show 0 < setdist (path_image p) (- S)
      using S p False
      by (fastforce simp add: setdist_gt_0_compact_closed compact_path_image
open_closed)
    qed
    qed
  then obtain e where 0 < e and eb: ∧x. x ∈ path_image p ⇒ ball x e ⊆ S
    by auto
  obtain pf where polynomial_function pf and pf: pathstart pf = pathstart p
pathfinish pf = pathfinish p
    and pf_e: ∧t. t ∈ {0..1} ⇒ norm(pf t - p t) < e
    using path_approx_polynomial_function [OF ⟨path p⟩ ⟨0 < e⟩] by blast
  show ?thesis
proof (intro exI conjI)
  show polynomial_function pf

```

```

    by fact
  show pathstart pf = x pathfinish pf = y
    by (simp_all add: p pf)
  show path_image pf  $\subseteq$  S
    unfolding path_image_def
  proof clarsimp
    fix x'::real
    assume 0  $\leq$  x' x'  $\leq$  1
    then have dist (p x') (pf x') < e
      by (metis atLeastAtMost_iff dist_commute dist_norm pf_e)
    then show pf x'  $\in$  S
      by (metis 0  $\leq$  x' x'  $\leq$  1 atLeastAtMost_iff eb_imageI mem_ball
path_image_def subset_iff)
    qed
  qed
qed

```

```

lemma differentiable_componentwise_within:
  f differentiable (at a within S)  $\longleftrightarrow$ 
  ( $\forall i \in$  Basis.  $(\lambda x. f x \cdot i)$  differentiable at a within S)
proof -
  { assume  $\forall i \in$  Basis.  $\exists D. ((\lambda x. f x \cdot i)$  has_derivative D) (at a within S)
    then obtain f' where f':
       $\bigwedge i. i \in$  Basis  $\implies ((\lambda x. f x \cdot i)$  has_derivative f' i) (at a within S)
    by metis
    have eq:  $(\lambda x. (\sum j \in$  Basis. f' j x *R j)  $\cdot$  i) = f' i if i  $\in$  Basis for i
      using that by (simp add: inner_add_left inner_add_right)
    have  $\exists D. \forall i \in$  Basis.  $((\lambda x. f x \cdot i)$  has_derivative  $(\lambda x. D x \cdot i))$  (at a within S)
      apply (rule_tac x= $\lambda x::'a. (\sum j \in$  Basis. f' j x *R j) :: 'b in exI)
      apply (simp add: eq f')
      done
    }
  then show ?thesis
    apply (simp add: differentiable_def)
    using has_derivative_componentwise_within
    by blast
qed

```

```

lemma polynomial_function_inner [intro]:
  fixes i :: 'a::euclidean_space
  shows polynomial_function g  $\implies$  polynomial_function  $(\lambda x. g x \cdot i)$ 
  apply (subst euclidean_representation [where x=i, symmetric])
  apply (force simp: inner_sum_right polynomial_function_iff_Basis_inner polynomial_function_sum)
  done

```

Differentiability of real and vector polynomial functions.

```

lemma differentiable_at_real_polynomial_function:
  real_polynomial_function f  $\implies$  f differentiable (at a within S)

```

by (induction f rule: real_polynomial_function.induct)
 (simp_all add: bounded_linear_imp_differentiable)

lemma differentiable_on_real_polynomial_function:

real_polynomial_function p $\implies$ p differentiable_on S

by (simp add: differentiable_at_imp_differentiable_on differentiable_at_real_polynomial_function)

lemma differentiable_at_polynomial_function:

fixes f :: $\_ \Rightarrow 'a::\text{euclidean\_space}$

shows polynomial_function f $\implies$ f differentiable (at a within S)

by (metis differentiable_at_real_polynomial_function polynomial_function_iff_Basis_inner
 differentiable_componentwise_within)

lemma differentiable_on_polynomial_function:

fixes f :: $\_ \Rightarrow 'a::\text{euclidean\_space}$

shows polynomial_function f $\implies$ f differentiable_on S

by (simp add: differentiable_at_polynomial_function differentiable_on_def)

lemma vector_eq_dot_span:

assumes $x \in \text{span } B$ $y \in \text{span } B$ and $i: \bigwedge i. i \in B \implies i \cdot x = i \cdot y$

shows $x = y$

proof –

have $\bigwedge i. i \in B \implies \text{orthogonal } (x - y) \ i$

by (simp add: i inner_commute inner_diff_right orthogonal_def)

moreover have $x - y \in \text{span } B$

by (simp add: assms span_diff)

ultimately have $x - y = 0$

using orthogonal_to_span_orthogonal_self by blast

then show ?thesis by simp

qed

lemma orthonormal_basis_expand:

assumes B: pairwise_orthogonal B

and 1: $\bigwedge i. i \in B \implies \text{norm } i = 1$

and $x \in \text{span } B$

and finite B

shows $(\sum_{i \in B}. (x \cdot i) *_{\mathbb{R}} i) = x$

proof (rule vector_eq_dot_span [OF _ $\langle x \in \text{span } B \rangle$])

show $(\sum_{i \in B}. (x \cdot i) *_{\mathbb{R}} i) \in \text{span } B$

by (simp add: span_clauses span_sum)

show $i \cdot (\sum_{i \in B}. (x \cdot i) *_{\mathbb{R}} i) = i \cdot x$ if $i \in B$ for i

proof –

have [simp]: $i \cdot j = (\text{if } j = i \text{ then } 1 \text{ else } 0)$ if $j \in B$ for j

using B 1 that $\langle i \in B \rangle$

by (force simp: norm_eq_1 orthogonal_def pairwise_def)

have $i \cdot (\sum_{i \in B}. (x \cdot i) *_{\mathbb{R}} i) = (\sum_{j \in B}. x \cdot j * (i \cdot j))$

by (simp add: inner_sum_right)

also have $\dots = (\sum_{j \in B}. \text{if } j = i \text{ then } x \cdot i \text{ else } 0)$

by (rule sum.cong; simp)

```

    also have ... =  $i \cdot x$ 
      by (simp add: ‹finite B› that inner_commute)
    finally show ?thesis .
  qed
qed

```

theorem *Stone_Weierstrass_polynomial_function_subspace:*

```

  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes compact S
    and conf: continuous_on S f
    and 0 < e
    and subspace T f ' S  $\subseteq$  T
  obtains g where polynomial_function g g ' S  $\subseteq$  T
     $\bigwedge x. x \in S \implies \text{norm}(f\ x - g\ x) < e$ 

proof -
  obtain B where B  $\subseteq$  T and orthB: pairwise orthogonal B
    and B1:  $\bigwedge x. x \in B \implies \text{norm}\ x = 1$ 
    and independent B and cardB: card B = dim T
    and spanB: span B = T
  using orthonormal_basis_subspace ‹subspace T› by metis
  then have finite B
    by (simp add: independent_imp_finite)
  then obtain n::nat and b where B = b ' {i. i < n} inj_on b {i. i < n}
    using finite_imp_nat_seg_image_inj_on by metis
  with cardB have n = card B dim T = n
    by (auto simp: card_image)
  have fx:  $(\sum_{i \in B}. (f\ x \cdot i) *_R i) = f\ x$  if  $x \in S$  for x
    by (metis (no_types, lifting) B1 ‹finite B› assms(5) image_subset_iff orthB
    orthonormal_basis_expand spanB sum.cong that)
  have cont: continuous_on S  $(\lambda x. \sum_{i \in B}. (f\ x \cdot i) *_R i)$ 
    by (intro continuous_intros conf)
  obtain g where polynomial_function g
    and g:  $\bigwedge x. x \in S \implies \text{norm}((\sum_{i \in B}. (f\ x \cdot i) *_R i) - g\ x) < e / (n+2)$ 
    using Stone_Weierstrass_polynomial_function [OF ‹compact S› cont, of e /
    real (n + 2)] ‹0 < e›
    by auto
  with fx have g:  $\bigwedge x. x \in S \implies \text{norm}(f\ x - g\ x) < e / (n+2)$ 
    by auto
  show ?thesis
proof
  show polynomial_function  $(\lambda x. \sum_{i \in B}. (g\ x \cdot i) *_R i)$ 
    using ‹polynomial_function g› by (force intro: ‹finite B›)
  show  $(\lambda x. \sum_{i \in B}. (g\ x \cdot i) *_R i) ' S \subseteq T$ 
    using ‹B  $\subseteq$  T›
    by (blast intro: subspace_sum subspace_mul ‹subspace T›)
  show norm  $(f\ x - (\sum_{i \in B}. (g\ x \cdot i) *_R i)) < e$  if  $x \in S$  for x
proof -
  have orth': pairwise  $(\lambda i\ j. \text{orthogonal}((f\ x \cdot i) *_R i - (g\ x \cdot i) *_R i))$ 

```

```

                                ((f x • j) *R j - (g x • j) *R j)) B
  by (auto simp: orthogonal_def inner_diff_right inner_diff_left intro: pairwise_mono [OF orthB])
  then have (norm (∑ i∈B. (f x • i) *R i - (g x • i) *R i))2 =
    (∑ i∈B. (norm ((f x • i) *R i - (g x • i) *R i))2)
    by (simp add: norm_sum_Pythagorean [OF ⟨finite B⟩ orth])
  also have ... = (∑ i∈B. (norm (((f x - g x) • i) *R i))2)
    by (simp add: algebra_simps)
  also have ... ≤ (∑ i∈B. (norm (f x - g x))2)
  proof -
    have ∧i. i ∈ B ⇒ ((f x - g x) • i)2 ≤ (norm (f x - g x))2
      by (metis B1 Cauchy_Schwarz_ineq inner_commute mult.left_neutral norm_eq_1 power2_norm_eq_inner)
    then show ?thesis
      by (intro sum_mono) (simp add: sum_mono B1)
  qed
  also have ... = n * norm (f x - g x)2
    by (simp add: ⟨n = card B⟩)
  also have ... ≤ n * (e / (n+2))2
  proof (rule mult_left_mono)
    show (norm (f x - g x))2 ≤ (e / real (n + 2))2
      by (meson dual_order.order_iff_strict g norm_ge_zero power_mono that)
  qed auto
  also have ... ≤ e2 / (n+2)
    using ⟨0 < e⟩ by (simp add: divide_simps power2_eq_square)
  also have ... < e2
    using ⟨0 < e⟩ by (simp add: divide_simps)
  finally have (norm (∑ i∈B. (f x • i) *R i - (g x • i) *R i))2 < e2 .
  then have (norm (∑ i∈B. (f x • i) *R i - (g x • i) *R i)) < e
    by (simp add: ⟨0 < e⟩ norm_lt_square power2_norm_eq_inner)
  then show ?thesis
    using fx that by (simp add: sum_subtractf)
  qed
qed
qed
qed

```

hide_fact linear add mult const

end

Chapter 6

Measure and Integration Theory

```
theory Sigma_Algebra
imports
  Complex_Main
  HOL-Library.Countable_Set
  HOL-Library.FuncSet
  HOL-Library.Indicator_Function
  HOL-Library.Extended_Nonnegative_Real
  HOL-Library.Disjoint_Sets
begin
```

6.1 Sigma Algebra

Sigma algebras are an elementary concept in measure theory. To measure — that is to integrate — functions, we first have to measure sets. Unfortunately, when dealing with a large universe, it is often not possible to consistently assign a measure to every subset. Therefore it is necessary to define the set of measurable subsets of the universe. A sigma algebra is such a set that has three very natural and desirable properties.

6.1.1 Families of sets

```
locale subset_class =
  fixes  $\Omega :: 'a\ set$  and  $M :: 'a\ set\ set$ 
  assumes space_closed:  $M \subseteq Pow\ \Omega$ 

lemma (in subset_class) sets_into_space:  $x \in M \implies x \subseteq \Omega$ 
  by (metis PowD contra_subsetD space_closed)
```

Semiring of sets

```
locale semiring_of_sets = subset_class +
```

assumes *empty_sets*[*iff*]: $\{\} \in M$
assumes *Int*[*intro*]: $\bigwedge a\ b. a \in M \implies b \in M \implies a \cap b \in M$
assumes *Diff_cover*:
 $\bigwedge a\ b. a \in M \implies b \in M \implies \exists C \subseteq M. \text{finite } C \wedge \text{disjoint } C \wedge a - b = \bigcup C$

lemma (**in** *semiring_of_sets*) *finite_INT*[*intro*]:
assumes *finite I I* $\neq \{\}$ $\bigwedge i. i \in I \implies A\ i \in M$
shows $(\bigcap i \in I. A\ i) \in M$
using *assms* **by** (*induct rule: finite_ne_induct*) *auto*

lemma (**in** *semiring_of_sets*) *Int_space_eq1* [*simp*]: $x \in M \implies \Omega \cap x = x$
by (*metis Int_absorb1 sets_into_space*)

lemma (**in** *semiring_of_sets*) *Int_space_eq2* [*simp*]: $x \in M \implies x \cap \Omega = x$
by (*metis Int_absorb2 sets_into_space*)

lemma (**in** *semiring_of_sets*) *sets_Collect_conj*:
assumes $\{x \in \Omega. P\ x\} \in M$ $\{x \in \Omega. Q\ x\} \in M$
shows $\{x \in \Omega. Q\ x \wedge P\ x\} \in M$
proof –
have $\{x \in \Omega. Q\ x \wedge P\ x\} = \{x \in \Omega. Q\ x\} \cap \{x \in \Omega. P\ x\}$
by *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

lemma (**in** *semiring_of_sets*) *sets_Collect_finite_All'*:
assumes $\bigwedge i. i \in S \implies \{x \in \Omega. P\ i\ x\} \in M$ *finite S S* $\neq \{\}$
shows $\{x \in \Omega. \forall i \in S. P\ i\ x\} \in M$
proof –
have $\{x \in \Omega. \forall i \in S. P\ i\ x\} = (\bigcap i \in S. \{x \in \Omega. P\ i\ x\})$
using $\langle S \neq \{\} \rangle$ **by** *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

Ring of sets

locale *ring_of_sets* = *semiring_of_sets* +
assumes *Un* [*intro*]: $\bigwedge a\ b. a \in M \implies b \in M \implies a \cup b \in M$

lemma (**in** *ring_of_sets*) *finite_Union* [*intro*]:
finite X $\implies X \subseteq M \implies \bigcup X \in M$
by (*induct set: finite*) (*auto simp add: Un*)

lemma (**in** *ring_of_sets*) *finite_UN*[*intro*]:
assumes *finite I* **and** $\bigwedge i. i \in I \implies A\ i \in M$
shows $(\bigcup i \in I. A\ i) \in M$
using *assms* **by** *induct auto*

lemma (**in** *ring_of_sets*) *Diff* [*intro*]:

assumes $a \in M$ $b \in M$ shows $a - b \in M$
 using *Diff_cover*[*OF assms*] by *auto*

lemma *ring_of_setsI*:

assumes *space_closed*: $M \subseteq \text{Pow } \Omega$
 assumes *empty_sets_iff*: $\{\} \in M$
 assumes *Un_intro*: $\bigwedge a b. a \in M \implies b \in M \implies a \cup b \in M$
 assumes *Diff_intro*: $\bigwedge a b. a \in M \implies b \in M \implies a - b \in M$
 shows *ring_of_sets* Ω M

proof

fix a b assume *ab*: $a \in M$ $b \in M$
 from *ab* show $\exists C \subseteq M. \text{finite } C \wedge \text{disjoint } C \wedge a - b = \bigcup C$
 by (*intro exI*[*of* $\{a - b\}$]) (*auto simp: disjoint_def*)
 have $a \cap b = a - (a - b)$ by *auto*
 also have $\dots \in M$ using *ab* by *auto*
 finally show $a \cap b \in M$.

qed *fact+*

lemma *ring_of_sets_iff*: $\text{ring\_of\_sets } \Omega \ M \longleftrightarrow M \subseteq \text{Pow } \Omega \wedge \{\} \in M \wedge$
 $(\forall a \in M. \forall b \in M. a \cup b \in M) \wedge (\forall a \in M. \forall b \in M. a - b \in M)$

proof

assume *ring_of_sets* Ω M
 then interpret *ring_of_sets* Ω M .
 show $M \subseteq \text{Pow } \Omega \wedge \{\} \in M \wedge (\forall a \in M. \forall b \in M. a \cup b \in M) \wedge (\forall a \in M. \forall b \in M.$
 $a - b \in M)$
 using *space_closed* by *auto*

qed (*auto intro!*: *ring_of_setsI*)

lemma (in *ring_of_sets*) *insert_in_sets*:

assumes $\{x\} \in M$ $A \in M$ shows $\text{insert } x \ A \in M$

proof -

have $\{x\} \cup A \in M$ using *assms* by (*rule Un*)
 thus ?thesis by *auto*

qed

lemma (in *ring_of_sets*) *sets_Collect_disj*:

assumes $\{x \in \Omega. P \ x\} \in M$ $\{x \in \Omega. Q \ x\} \in M$
 shows $\{x \in \Omega. Q \ x \vee P \ x\} \in M$

proof -

have $\{x \in \Omega. Q \ x \vee P \ x\} = \{x \in \Omega. Q \ x\} \cup \{x \in \Omega. P \ x\}$
 by *auto*
 with *assms* show ?thesis by *auto*

qed

lemma (in *ring_of_sets*) *sets_Collect_finite_Ex*:

assumes $\bigwedge i. i \in S \implies \{x \in \Omega. P \ i \ x\} \in M$ *finite* S
 shows $\{x \in \Omega. \exists i \in S. P \ i \ x\} \in M$

proof -

have $\{x \in \Omega. \exists i \in S. P \ i \ x\} = (\bigcup i \in S. \{x \in \Omega. P \ i \ x\})$

by *auto*
 with *assms* show *?thesis* by *auto*
 qed

Algebra of sets

locale *algebra* = *ring_of_sets* +
 assumes *top* [*iff*]: $\Omega - a \in M$

lemma (in *algebra*) *compl_sets* [*intro*]:
 $a \in M \implies \Omega - a \in M$
 by *auto*

proposition *algebra_iff_Un*:
 $algebra\ \Omega\ M \longleftrightarrow$
 $M \subseteq Pow\ \Omega \wedge$
 $\{\} \in M \wedge$
 $(\forall a \in M. \Omega - a \in M) \wedge$
 $(\forall a \in M. \forall b \in M. a \cup b \in M) \text{ (is\_ } \_ \longleftrightarrow ?Un)$

proof
 assume *algebra* $\Omega\ M$
 then interpret *algebra* $\Omega\ M$.
 show *?Un* using *sets_into_space* by *auto*
 next

assume *?Un*
 then have $\Omega \in M$ by *auto*
 interpret *ring_of_sets* $\Omega\ M$
 proof (rule *ring_of_setsI*)
 show $\Omega: M \subseteq Pow\ \Omega\ \{\} \in M$
 using $\langle ?Un \rangle$ by *auto*
 fix *a b* assume *a*: $a \in M$ and *b*: $b \in M$
 then show $a \cup b \in M$ using $\langle ?Un \rangle$ by *auto*
 have $a - b = \Omega - ((\Omega - a) \cup b)$
 using $\Omega\ a\ b$ by *auto*
 then show $a - b \in M$
 using *a b* $\langle ?Un \rangle$ by *auto*
 qed
 show *algebra* $\Omega\ M$ proof qed fact
 qed

proposition *algebra_iff_Int*:
 $algebra\ \Omega\ M \longleftrightarrow$
 $M \subseteq Pow\ \Omega \ \& \ \{\} \in M \ \&$
 $(\forall a \in M. \Omega - a \in M) \ \&$
 $(\forall a \in M. \forall b \in M. a \cap b \in M) \text{ (is\_ } \_ \longleftrightarrow ?Int)$
 proof
 assume *algebra* $\Omega\ M$
 then interpret *algebra* $\Omega\ M$.
 show *?Int* using *sets_into_space* by *auto*

```

next
  assume ?Int
  show algebra  $\Omega$   $M$ 
  proof (unfold algebra_iff_Un, intro conjI ballI)
    show  $\Omega: M \subseteq \text{Pow } \Omega \ \{\}$   $\in M$ 
    using <?Int> by auto
    from <?Int> show  $\bigwedge a. a \in M \implies \Omega - a \in M$  by auto
    fix  $a \ b$  assume  $M: a \in M \ b \in M$ 
    hence  $a \cup b = \Omega - ((\Omega - a) \cap (\Omega - b))$ 
    using  $\Omega$  by blast
    also have  $\dots \in M$ 
    using  $M$  <?Int> by auto
    finally show  $a \cup b \in M$  .
  qed
qed

```

```

lemma (in algebra) sets_Collect_neg:
  assumes  $\{x \in \Omega. P \ x\} \in M$ 
  shows  $\{x \in \Omega. \neg P \ x\} \in M$ 
proof -
  have  $\{x \in \Omega. \neg P \ x\} = \Omega - \{x \in \Omega. P \ x\}$  by auto
  with assms show ?thesis by auto
qed

```

```

lemma (in algebra) sets_Collect_imp:
   $\{x \in \Omega. P \ x\} \in M \implies \{x \in \Omega. Q \ x\} \in M \implies \{x \in \Omega. Q \ x \longrightarrow P \ x\} \in M$ 
  unfolding imp_conv_disj by (intro sets_Collect_disj sets_Collect_neg)

```

```

lemma (in algebra) sets_Collect_const:
   $\{x \in \Omega. P\} \in M$ 
  by (cases  $P$ ) auto

```

```

lemma algebra_single_set:
   $X \subseteq S \implies \text{algebra } S \ \{\ \{\}, X, S - X, S \}$ 
  by (auto simp: algebra_iff_Int)

```

Restricted algebras

```

abbreviation (in algebra)
  restricted_space  $A \equiv ((\cap) \ A) \ ' \ M$ 

```

```

lemma (in algebra) restricted_algebra:
  assumes  $A \in M$  shows algebra  $A$  (restricted_space  $A$ )
  using assms by (auto simp: algebra_iff_Int)

```

Sigma Algebras

```

locale sigma_algebra = algebra +
  assumes countable_nat_UN [intro]:  $\bigwedge A. \text{range } A \subseteq M \implies (\bigcup i::\text{nat}. A \ i) \in M$ 

```

lemma (in *algebra*) *is_sigma_algebra*:

assumes *finite M*

shows *sigma_algebra* Ω *M*

proof

fix *A* :: *nat* $\Rightarrow$ 'a *set* **assume** *range A* $\subseteq$ *M*

then have $(\bigcup i. A\ i) = (\bigcup s \in M \cap \text{range } A. s)$

by *auto*

also have $(\bigcup s \in M \cap \text{range } A. s) \in M$

using $\langle \text{finite } M \rangle$ **by** *auto*

finally show $(\bigcup i. A\ i) \in M$.

qed

lemma *countable_UN_eq*:

fixes *A* :: 'i::countable $\Rightarrow$ 'a *set*

shows $(\text{range } A \subseteq M \longrightarrow (\bigcup i. A\ i) \in M) \longleftrightarrow$

$(\text{range } (A \circ \text{from\_nat}) \subseteq M \longrightarrow (\bigcup i. (A \circ \text{from\_nat})\ i) \in M)$

proof –

let $?A' = A \circ \text{from\_nat}$

have *: $(\bigcup i. ?A'\ i) = (\bigcup i. A\ i)$ (**is** $?l = ?r$)

proof *safe*

fix *x i* **assume** *x* $\in A\ i$ **thus** *x* $\in ?l$

by (*auto intro!*: *exI*[*of _ to_nat i*])

next

fix *x i* **assume** *x* $\in ?A'\ i$ **thus** *x* $\in ?r$

by (*auto intro!*: *exI*[*of _ from_nat i*])

qed

have *A* ' *range from_nat* = *range A*

using *surj_from_nat* **by** *simp*

then have **: *range ?A'* = *range A*

by (*simp only*: *image_comp* [*symmetric*])

show *?thesis* **unfolding** * ** ..

qed

lemma (in *sigma_algebra*) *countable_Union* [*intro*]:

assumes *countable X* *X* $\subseteq M$ **shows** $\bigcup X \in M$

proof *cases*

assume *X* $\neq \{\}$

hence $\bigcup X = (\bigcup n. \text{from\_nat\_into } X\ n)$

using *assms* **by** (*auto cong del*: *SUP_cong*)

also have ... $\in M$ **using** *assms*

by (*auto intro!*: *countable_nat_UN*) (*metis* $\langle X \neq \{\} \rangle$ *from_nat_into_subsetD*)

finally show *?thesis* .

qed *simp*

lemma (in *sigma_algebra*) *countable_UN* [*intro*]:

fixes *A* :: 'i::countable $\Rightarrow$ 'a *set*

assumes *A* ' *X* $\subseteq M$

shows $(\bigcup x \in X. A\ x) \in M$

proof –

```

let ?A =  $\lambda i.$  if  $i \in X$  then  $A\ i$  else {}
from assms have  $\text{range } ?A \subseteq M$  by auto
with countable_nat_UN[of ?A  $\circ$  from_nat] countable_UN_eq[of ?A M]
have  $(\bigcup x. ?A\ x) \in M$  by auto
moreover have  $(\bigcup x. ?A\ x) = (\bigcup_{x \in X}. A\ x)$  by (auto split: if_split_asm)
ultimately show ?thesis by simp
qed

```

```

lemma (in sigma_algebra) countable_UN':
  fixes  $A :: 'i \Rightarrow 'a\ \text{set}$ 
  assumes  $X:$  countable  $X$ 
  assumes  $A:$   $A\ X \subseteq M$ 
  shows  $(\bigcup_{x \in X}. A\ x) \in M$ 
proof -
  have  $(\bigcup_{x \in X}. A\ x) = (\bigcup_{i \in \text{to\_nat\_on } X\ 'X}. A\ (\text{from\_nat\_into } X\ i))$ 
    using  $X$  by auto
  also have  $\dots \in M$ 
    using  $A\ X$ 
    by (intro countable_UN) auto
  finally show ?thesis .
qed

```

```

lemma (in sigma_algebra) countable_UN'':
   $\llbracket \text{countable } X; \bigwedge x\ y. x \in X \implies A\ x \in M \rrbracket \implies (\bigcup_{x \in X}. A\ x) \in M$ 
by(erule countable_UN')(auto)

```

```

lemma (in sigma_algebra) countable_INT [intro]:
  fixes  $A :: 'i::\text{countable} \Rightarrow 'a\ \text{set}$ 
  assumes  $A:$   $A\ X \subseteq M$   $X \neq \{\}$ 
  shows  $(\bigcap_{i \in X}. A\ i) \in M$ 
proof -
  from  $A$  have  $\forall i \in X. A\ i \in M$  by fast
  hence  $\Omega - (\bigcup_{i \in X}. \Omega - A\ i) \in M$  by blast
  moreover
  have  $(\bigcap_{i \in X}. A\ i) = \Omega - (\bigcup_{i \in X}. \Omega - A\ i)$  using space_closed  $A$ 
    by blast
  ultimately show ?thesis by metis
qed

```

```

lemma (in sigma_algebra) countable_INT':
  fixes  $A :: 'i \Rightarrow 'a\ \text{set}$ 
  assumes  $X:$  countable  $X$   $X \neq \{\}$ 
  assumes  $A:$   $A\ X \subseteq M$ 
  shows  $(\bigcap_{x \in X}. A\ x) \in M$ 
proof -
  have  $(\bigcap_{x \in X}. A\ x) = (\bigcap_{i \in \text{to\_nat\_on } X\ 'X}. A\ (\text{from\_nat\_into } X\ i))$ 
    using  $X$  by auto
  also have  $\dots \in M$ 
    using  $A\ X$ 

```

by (intro countable_INT) auto
 finally show ?thesis .
 qed

lemma (in sigma_algebra) countable_INT'':
 $UNIV \in M \implies \text{countable } I \implies (\bigwedge i. i \in I \implies F i \in M) \implies (\bigcap i \in I. F i) \in M$
 by (cases I = {}) (auto intro: countable_INT')

lemma (in sigma_algebra) countable:
 assumes $\bigwedge a. a \in A \implies \{a\} \in M$ countable A
 shows $A \in M$
 proof -
 have $(\bigcup a \in A. \{a\}) \in M$
 using assms by (intro countable_UN') auto
 also have $(\bigcup a \in A. \{a\}) = A$ by auto
 finally show ?thesis by auto
 qed

lemma ring_of_sets_Pow: ring_of_sets sp (Pow sp)
 by (auto simp: ring_of_sets_iff)

lemma algebra_Pow: algebra sp (Pow sp)
 by (auto simp: algebra_iff_Un)

lemma sigma_algebra_iff:
 $\text{sigma\_algebra } \Omega \ M \longleftrightarrow$
 $\text{algebra } \Omega \ M \wedge (\forall A. \text{range } A \subseteq M \longrightarrow (\bigcup i::\text{nat}. A i) \in M)$
 by (simp add: sigma_algebra_def sigma_algebra_axioms_def)

lemma sigma_algebra_Pow: sigma_algebra sp (Pow sp)
 by (auto simp: sigma_algebra_iff algebra_iff_Int)

lemma (in sigma_algebra) sets_Collect_countable_All:
 assumes $\bigwedge i. \{x \in \Omega. P i x\} \in M$
 shows $\{x \in \Omega. \forall i::'i::\text{countable}. P i x\} \in M$
 proof -
 have $\{x \in \Omega. \forall i::'i::\text{countable}. P i x\} = (\bigcap i. \{x \in \Omega. P i x\})$ by auto
 with assms show ?thesis by auto
 qed

lemma (in sigma_algebra) sets_Collect_countable_Ex:
 assumes $\bigwedge i. \{x \in \Omega. P i x\} \in M$
 shows $\{x \in \Omega. \exists i::'i::\text{countable}. P i x\} \in M$
 proof -
 have $\{x \in \Omega. \exists i::'i::\text{countable}. P i x\} = (\bigcup i. \{x \in \Omega. P i x\})$ by auto
 with assms show ?thesis by auto
 qed

lemma (in sigma_algebra) sets_Collect_countable_Ex':

```

    assumes  $\bigwedge i. i \in I \implies \{x \in \Omega. P\ i\ x\} \in M$ 
    assumes countable  $I$ 
    shows  $\{x \in \Omega. \exists i \in I. P\ i\ x\} \in M$ 
  proof -
    have  $\{x \in \Omega. \exists i \in I. P\ i\ x\} = (\bigcup i \in I. \{x \in \Omega. P\ i\ x\})$  by auto
    with assms show ?thesis
      by (auto intro!: countable_UN')
  qed

lemma (in sigma_algebra) sets_Collect_countable_All':
  assumes  $\bigwedge i. i \in I \implies \{x \in \Omega. P\ i\ x\} \in M$ 
  assumes countable  $I$ 
  shows  $\{x \in \Omega. \forall i \in I. P\ i\ x\} \in M$ 
proof -
  have  $\{x \in \Omega. \forall i \in I. P\ i\ x\} = (\bigcap i \in I. \{x \in \Omega. P\ i\ x\}) \cap \Omega$  by auto
  with assms show ?thesis
    by (cases  $I = \{\}$ ) (auto intro!: countable_INT')
qed

lemma (in sigma_algebra) sets_Collect_countable_Ex1':
  assumes  $\bigwedge i. i \in I \implies \{x \in \Omega. P\ i\ x\} \in M$ 
  assumes countable  $I$ 
  shows  $\{x \in \Omega. \exists ! i \in I. P\ i\ x\} \in M$ 
proof -
  have  $\{x \in \Omega. \exists ! i \in I. P\ i\ x\} = \{x \in \Omega. \exists i \in I. P\ i\ x \wedge (\forall j \in I. P\ j\ x \longrightarrow i = j)\}$ 
    by auto
  with assms show ?thesis
    by (auto intro!: sets_Collect_countable_All' sets_Collect_countable_Ex' sets_Collect_conj
      sets_Collect_imp sets_Collect_const)
qed

lemmas (in sigma_algebra) sets_Collect =
  sets_Collect_imp sets_Collect_disj sets_Collect_conj sets_Collect_neg sets_Collect_const
  sets_Collect_countable_All sets_Collect_countable_Ex sets_Collect_countable_All

lemma (in sigma_algebra) sets_Collect_countable_Ball:
  assumes  $\bigwedge i. \{x \in \Omega. P\ i\ x\} \in M$ 
  shows  $\{x \in \Omega. \forall i :: 'i :: countable \in X. P\ i\ x\} \in M$ 
  unfolding Ball_def by (intro sets_Collect assms)

lemma (in sigma_algebra) sets_Collect_countable_Bex:
  assumes  $\bigwedge i. \{x \in \Omega. P\ i\ x\} \in M$ 
  shows  $\{x \in \Omega. \exists i :: 'i :: countable \in X. P\ i\ x\} \in M$ 
  unfolding Bex_def by (intro sets_Collect assms)

lemma sigma_algebra_single_set:
  assumes  $X \subseteq S$ 
  shows sigma_algebra  $S\ \{\{\}, X, S - X, S\}$ 
  using algebra.is_sigma_algebra[OF algebra_single_set[OF  $X \subseteq S$ ]] by simp

```

Binary Unions

definition *binary* :: 'a $\Rightarrow$ 'a $\Rightarrow$ nat $\Rightarrow$ 'a
 where *binary* a b = ($\lambda x. b$)(0 := a)

lemma *range_binary_eq*: $\text{range}(\text{binary } a \ b) = \{a, b\}$
 by (auto simp add: *binary_def*)

lemma *Un_range_binary*: $a \cup b = (\bigcup i::\text{nat}. \text{binary } a \ b \ i)$
 by (simp add: *range_binary_eq* cong del: *SUP_cong_simp*)

lemma *Int_range_binary*: $a \cap b = (\bigcap i::\text{nat}. \text{binary } a \ b \ i)$
 by (simp add: *range_binary_eq* cong del: *INF_cong_simp*)

lemma *sigma_algebra_iff2*:
 $\text{sigma\_algebra } \Omega \ M \longleftrightarrow$
 $M \subseteq \text{Pow } \Omega \wedge \{\} \in M \wedge (\forall s \in M. \Omega - s \in M)$
 $\wedge (\forall A. \text{range } A \subseteq M \longrightarrow (\bigcup i::\text{nat}. A \ i) \in M)$ (is ?P $\longleftrightarrow$?R $\wedge$?S $\wedge$?V $\wedge$?W)

proof

assume ?P

then interpret *sigma_algebra* $\Omega \ M$.

from *space_closed* show ?R $\wedge$?S $\wedge$?V $\wedge$?W

by auto

next

assume ?R $\wedge$?S $\wedge$?V $\wedge$?W

then have ?R ?S ?V ?W

by *simp_all*

show ?P

proof (rule *sigma_algebra.intro*)

show *sigma_algebra_axioms* M

by standard (use <?W> in *simp*)

from <?W> have *: $\text{range}(\text{binary } a \ b) \subseteq M \implies \bigcup (\text{range}(\text{binary } a \ b)) \in M$

for a b

by auto

show *algebra* $\Omega \ M$

unfolding *algebra_iff_Un* using <?R> <?S> <?V> *

by (auto simp add: *range_binary_eq*)

qed

qed

Initial Sigma Algebra

Sigma algebras can naturally be created as the closure of any set of M with regard to the properties just postulated.

inductive_set *sigma_sets* :: 'a set $\Rightarrow$ 'a set set $\Rightarrow$ 'a set set
 for sp :: 'a set and A :: 'a set set
 where
 Basic[*intro, simp*]: $a \in A \implies a \in \text{sigma\_sets } sp \ A$


```

| Empty:  $\{\} \in \text{sigma\_sets } sp \ A$ 
| Compl:  $a \in \text{sigma\_sets } sp \ A \implies sp - a \in \text{sigma\_sets } sp \ A$ 
| Union:  $(\bigwedge i::nat. a \ i \in \text{sigma\_sets } sp \ A) \implies (\bigcup i. a \ i) \in \text{sigma\_sets } sp \ A$ 

```

```

lemma (in sigma_algebra) sigma_sets_subset:
  assumes  $a: a \subseteq M$ 
  shows  $\text{sigma\_sets } \Omega \ a \subseteq M$ 
proof
  fix  $x$ 
  assume  $x \in \text{sigma\_sets } \Omega \ a$ 
  from this show  $x \in M$ 
  by (induct rule: sigma_sets.induct, auto) (metis a subsetD)
qed

```

```

lemma sigma_sets_into_sp:  $A \subseteq \text{Pow } sp \implies x \in \text{sigma\_sets } sp \ A \implies x \subseteq sp$ 
by (erule sigma_sets.induct, auto)

```

```

lemma sigma_algebra_sigma_sets:
   $a \subseteq \text{Pow } \Omega \implies \text{sigma\_algebra } \Omega \ (\text{sigma\_sets } \Omega \ a)$ 
by (auto simp add: sigma_algebra_iff2 dest: sigma_sets_into_sp
  intro!: sigma_sets.Union sigma_sets.Empty sigma_sets.Compl)

```

```

lemma sigma_sets_least_sigma_algebra:
  assumes  $A \subseteq \text{Pow } S$ 
  shows  $\text{sigma\_sets } S \ A = \bigcap \{B. A \subseteq B \wedge \text{sigma\_algebra } S \ B\}$ 
proof safe
  fix  $B \ X$  assume  $A \subseteq B$  and  $sa: \text{sigma\_algebra } S \ B$ 
  and  $X: X \in \text{sigma\_sets } S \ A$ 
  from sigma_algebra.sigma_sets_subset[OF sa, simplified, OF  $\langle A \subseteq B \rangle$ ]  $X$ 
  show  $X \in B$  by auto
next
  fix  $X$  assume  $X \in \bigcap \{B. A \subseteq B \wedge \text{sigma\_algebra } S \ B\}$ 
  then have [intro!]:  $\bigwedge B. A \subseteq B \implies \text{sigma\_algebra } S \ B \implies X \in B$ 
  by simp
  have  $A \subseteq \text{sigma\_sets } S \ A$  using assms by auto
  moreover have  $\text{sigma\_algebra } S \ (\text{sigma\_sets } S \ A)$ 
  using assms by (intro sigma_algebra_sigma_sets[of A]) auto
  ultimately show  $X \in \text{sigma\_sets } S \ A$  by auto
qed

```

```

lemma sigma_sets_top:  $sp \in \text{sigma\_sets } sp \ A$ 
by (metis Diff_empty sigma_sets.Compl sigma_sets.Empty)

```

```

lemma binary_in_sigma_sets:
   $\text{binary } a \ b \ i \in \text{sigma\_sets } sp \ A$  if  $a \in \text{sigma\_sets } sp \ A$  and  $b \in \text{sigma\_sets } sp \ A$ 
  using that by (simp add: binary_def)

```

```

lemma sigma_sets_Un:
   $a \cup b \in \text{sigma\_sets } sp \ A$  if  $a \in \text{sigma\_sets } sp \ A$  and  $b \in \text{sigma\_sets } sp \ A$ 

```

using *that* by (simp add: Un_range_binary binary_in_sigma_sets Union)

lemma *sigma_sets_Inter*:

assumes *Asb*: $A \subseteq \text{Pow } sp$

shows $(\bigwedge i::nat. a\ i \in \text{sigma\_sets } sp\ A) \implies (\bigcap i. a\ i) \in \text{sigma\_sets } sp\ A$

proof –

assume *ai*: $\bigwedge i::nat. a\ i \in \text{sigma\_sets } sp\ A$

hence $\bigwedge i::nat. sp-(a\ i) \in \text{sigma\_sets } sp\ A$

by (rule *sigma_sets.Compl*)

hence $(\bigcup i. sp-(a\ i)) \in \text{sigma\_sets } sp\ A$

by (rule *sigma_sets.Union*)

hence $sp-(\bigcup i. sp-(a\ i)) \in \text{sigma\_sets } sp\ A$

by (rule *sigma_sets.Compl*)

also have $sp-(\bigcup i. sp-(a\ i)) = sp\ Int\ (\bigcap i. a\ i)$

by *auto*

also have $\dots = (\bigcap i. a\ i)$ using *ai*

by (blast dest: *sigma_sets_into_sp* [OF *Asb*])

finally show ?thesis .

qed

lemma *sigma_sets_INTER*:

assumes *Asb*: $A \subseteq \text{Pow } sp$

and *ai*: $\bigwedge i::nat. i \in S \implies a\ i \in \text{sigma\_sets } sp\ A$ and *non*: $S \neq \{\}$

shows $(\bigcap i \in S. a\ i) \in \text{sigma\_sets } sp\ A$

proof –

from *ai* have $\bigwedge i. (if\ i \in S\ then\ a\ i\ else\ sp) \in \text{sigma\_sets } sp\ A$

by (simp add: *sigma_sets.intros*(2–) *sigma_sets_top*)

hence $(\bigcap i. (if\ i \in S\ then\ a\ i\ else\ sp)) \in \text{sigma\_sets } sp\ A$

by (rule *sigma_sets_Inter* [OF *Asb*])

also have $(\bigcap i. (if\ i \in S\ then\ a\ i\ else\ sp)) = (\bigcap i \in S. a\ i)$

by *auto* (*metis ai non sigma_sets_into_sp subset_empty subset_iff Asb*) +

finally show ?thesis .

qed

lemma *sigma_sets_UNION*:

countable *B* $\implies (\bigwedge b. b \in B \implies b \in \text{sigma\_sets } X\ A) \implies \bigcup B \in \text{sigma\_sets } X\ A$

using *from_nat_into* [of *B*] *range_from_nat_into* [of *B*] *sigma_sets.Union* [of *from_nat_into B X A*]

by (cases $B = \{\}$) (simp_all add: *sigma_sets.Empty cong del: SUP_cong*)

lemma (in *sigma_algebra*) *sigma_sets_eq*:

sigma_sets $\Omega\ M = M$

proof

show $M \subseteq \text{sigma\_sets } \Omega\ M$

by (*metis Set.subsetI sigma_sets.Basic*)

next

show *sigma_sets* $\Omega\ M \subseteq M$

by (*metis sigma_sets_subset subset_refl*)

qed

lemma *sigma_sets_eqI*:

assumes $A: \bigwedge a. a \in A \implies a \in \text{sigma\_sets } M \ B$

assumes $B: \bigwedge b. b \in B \implies b \in \text{sigma\_sets } M \ A$

shows $\text{sigma\_sets } M \ A = \text{sigma\_sets } M \ B$

proof (intro set_eqI iffI)

fix a assume $a \in \text{sigma\_sets } M \ A$

from this A show $a \in \text{sigma\_sets } M \ B$

by induct (auto intro!: sigma_sets.intros(2-) del: sigma_sets.Basic)

next

fix b assume $b \in \text{sigma\_sets } M \ B$

from this B show $b \in \text{sigma\_sets } M \ A$

by induct (auto intro!: sigma_sets.intros(2-) del: sigma_sets.Basic)

qed

lemma *sigma_sets_subseteq*: assumes $A \subseteq B$ shows $\text{sigma\_sets } X \ A \subseteq \text{sigma\_sets } X \ B$

proof

fix x assume $x \in \text{sigma\_sets } X \ A$ then show $x \in \text{sigma\_sets } X \ B$

by induct (insert $\langle A \subseteq B \rangle$, auto intro: sigma_sets.intros(2-))

qed

lemma *sigma_sets_mono*: assumes $A \subseteq \text{sigma\_sets } X \ B$ shows $\text{sigma\_sets } X \ A \subseteq \text{sigma\_sets } X \ B$

proof

fix x assume $x \in \text{sigma\_sets } X \ A$ then show $x \in \text{sigma\_sets } X \ B$

by induct (insert $\langle A \subseteq \text{sigma\_sets } X \ B \rangle$, auto intro: sigma_sets.intros(2-))

qed

lemma *sigma_sets_mono'*: assumes $A \subseteq B$ shows $\text{sigma\_sets } X \ A \subseteq \text{sigma\_sets } X \ B$

proof

fix x assume $x \in \text{sigma\_sets } X \ A$ then show $x \in \text{sigma\_sets } X \ B$

by induct (insert $\langle A \subseteq B \rangle$, auto intro: sigma_sets.intros(2-))

qed

lemma *sigma_sets_superset_generator*: $A \subseteq \text{sigma\_sets } X \ A$

by (auto intro: sigma_sets.Basic)

lemma (in sigma_algebra) *restriction_in_sets*:

fixes $A :: \text{nat} \Rightarrow 'a \text{ set}$

assumes $S \in M$

and $*$: $\text{range } A \subseteq (\lambda A. S \cap A) \text{ ' } M \text{ (is } \_ \subseteq ?r)$

shows $\text{range } A \subseteq M \ (\bigcup i. A \ i) \in (\lambda A. S \cap A) \text{ ' } M$

proof -

{ fix i have $A \ i \in ?r$ using $*$ by auto

hence $\exists B. A \ i = B \cap S \wedge B \in M$ by auto

hence $A \ i \subseteq S \wedge A \ i \in M$ using $\langle S \in M \rangle$ by auto }

```

    thus range A ⊆ M (⋃ i. A i) ∈ (λA. S ∩ A) ‘ M
    by (auto intro!: image_eqI[of _ _ (⋃ i. A i)])
qed

```

```

lemma (in sigma_algebra) restricted_sigma_algebra:
  assumes S ∈ M
  shows sigma_algebra S (restricted_space S)
  unfolding sigma_algebra_def sigma_algebra_axioms_def
proof safe
  show algebra S (restricted_space S) using restricted_algebra[OF assms] .
next
  fix A :: nat ⇒ 'a set assume range A ⊆ restricted_space S
  from restriction_in_sets[OF assms this[simplified]]
  show (⋃ i. A i) ∈ restricted_space S by simp
qed

```

```

lemma sigma_sets_Int:
  assumes A ∈ sigma_sets sp st A ⊆ sp
  shows (⋂) A ‘ sigma_sets sp st = sigma_sets A ((⋂) A ‘ st)
proof (intro equalityI subsetI)
  fix x assume x ∈ (⋂) A ‘ sigma_sets sp st
  then obtain y where y ∈ sigma_sets sp st x = y ∩ A by auto
  then have x ∈ sigma_sets (A ∩ sp) ((⋂) A ‘ st)
  proof (induct arbitrary: x)
    case (Compl a)
    then show ?case
      by (force intro!: sigma_sets.Compl simp: Diff_Int_distrib ac_simps)
  next
    case (Union a)
    then show ?case
      by (auto intro!: sigma_sets.Union
        simp add: UN_extend_simps simp del: UN_simps)
  qed (auto intro!: sigma_sets.intros(2-))
  then show x ∈ sigma_sets A ((⋂) A ‘ st)
  using ⟨A ⊆ sp⟩ by (simp add: Int_absorb2)
next
  fix x assume x ∈ sigma_sets A ((⋂) A ‘ st)
  then show x ∈ (⋂) A ‘ sigma_sets sp st
  proof induct
    case (Compl a)
    then obtain x where a = A ∩ x x ∈ sigma_sets sp st by auto
    then show ?case using ⟨A ⊆ sp⟩
      by (force simp add: image_iff intro!: bexI[of _ sp - x] sigma_sets.Compl)
  next
    case (Union a)
    then have ∀ i. ∃ x. x ∈ sigma_sets sp st ∧ a i = A ∩ x
      by (auto simp: image_iff Bex_def)
    then obtain f where ∀ x. f x ∈ sigma_sets sp st ∧ a x = A ∩ f x
      by metis

```

```

    then show ?case
      by (auto intro!: bexI[of _ ( $\bigcup x. f x$ )] sigma_sets.Union
          simp add: image_iff)
  qed (auto intro!: sigma_sets.intros(2-))
qed

```

```

lemma sigma_sets_empty_eq: sigma_sets A {} = {{}, A}
proof (intro set_eqI iffI)
  fix a assume a ∈ sigma_sets A {} then show a ∈ {{}, A}
    by induct blast+
qed (auto intro: sigma_sets.Empty sigma_sets_top)

```

```

lemma sigma_sets_single[simp]: sigma_sets A {A} = {{}, A}
proof (intro set_eqI iffI)
  fix x assume x ∈ sigma_sets A {A}
  then show x ∈ {{}, A}
    by induct blast+
next
  fix x assume x ∈ {{}, A}
  then show x ∈ sigma_sets A {A}
    by (auto intro: sigma_sets.Empty sigma_sets_top)
qed

```

```

lemma sigma_sets_sigma_sets_eq:
   $M \subseteq Pow S \implies sigma\_sets S (sigma\_sets S M) = sigma\_sets S M$ 
  by (rule sigma_algebra.sigma_sets_eq[OF sigma_algebra_sigma_sets, of M S])
auto

```

```

lemma sigma_sets_singleton:
  assumes  $X \subseteq S$ 
  shows sigma_sets S { X } = { {}, X, S - X, S }
proof -
  interpret sigma_algebra S { {}, X, S - X, S }
  by (rule sigma_algebra_single_set) fact
  have sigma_sets S { X } ⊆ sigma_sets S { {}, X, S - X, S }
    by (rule sigma_sets_subseteq) simp
  moreover have ... = { {}, X, S - X, S }
    using sigma_sets_eq by simp
  moreover
  { fix A assume A ∈ { {}, X, S - X, S }
    then have A ∈ sigma_sets S { X }
      by (auto intro: sigma_sets.intros(2-) sigma_sets_top) }
  ultimately have sigma_sets S { X } = sigma_sets S { {}, X, S - X, S }
    by (intro antisym) auto
  with sigma_sets_eq show ?thesis by simp
qed

```

```

lemma restricted_sigma:
  assumes S:  $S \in sigma\_sets \Omega M$  and M:  $M \subseteq Pow \Omega$ 

```

```

shows algebra.restricted_space (sigma_sets  $\Omega$   $M$ )  $S$  =
  sigma_sets  $S$  (algebra.restricted_space  $M$   $S$ )
proof -
  from  $S$  sigma_sets_into_sp[OF  $M$ ]
  have  $S \in \text{sigma\_sets } \Omega \ M \ S \subseteq \Omega$  by auto
  from sigma_sets_Int[OF this]
  show ?thesis by simp
qed

lemma sigma_sets_vimage_commute:
  assumes  $X: X \in \Omega \rightarrow \Omega'$ 
  shows  $\{X - ' A \cap \Omega \mid A. A \in \text{sigma\_sets } \Omega' \ M'\}$ 
    = sigma_sets  $\Omega$   $\{X - ' A \cap \Omega \mid A. A \in M'\}$  (is ?L = ?R)
proof
  show ?L  $\subseteq$  ?R
  proof clarify
    fix  $A$  assume  $A \in \text{sigma\_sets } \Omega' \ M'$ 
    then show  $X - ' A \cap \Omega \in ?R$ 
    proof induct
      case Empty then show ?case
        by (auto intro!: sigma_sets.Empty)
    next
      case (Compl  $B$ )
      have [simp]:  $X - ' (\Omega' - B) \cap \Omega = \Omega - (X - ' B \cap \Omega)$ 
        by (auto simp add: funcset_mem [OF  $X$ ])
      with Compl show ?case
        by (auto intro!: sigma_sets.Compl)
    next
      case (Union  $F$ )
      then show ?case
        by (auto simp add: vimage_UN UN_extend_simps(4) simp del: UN_simps
            intro!: sigma_sets.Union)
    qed auto
  qed
  show ?R  $\subseteq$  ?L
  proof clarify
    fix  $A$  assume  $A \in ?R$ 
    then show  $\exists B. A = X - ' B \cap \Omega \wedge B \in \text{sigma\_sets } \Omega' \ M'$ 
    proof induct
      case (Basic  $B$ ) then show ?case by auto
    next
      case Empty then show ?case
        by (auto intro!: sigma_sets.Empty exI[of _ {}])
    next
      case (Compl  $B$ )
      then obtain  $A$  where  $A: B = X - ' A \cap \Omega \ A \in \text{sigma\_sets } \Omega' \ M'$  by auto
      then have [simp]:  $\Omega - B = X - ' (\Omega' - A) \cap \Omega$ 
        by (auto simp add: funcset_mem [OF  $X$ ])
      with A(2) show ?case

```

```

      by (auto intro: sigma_sets.Compl)
    next
      case (Union F)
      then have  $\forall i. \exists B. F\ i = X - ' B \cap \Omega \wedge B \in \text{sigma\_sets } \Omega' M'$  by auto
      then obtain A where  $\forall x. F\ x = X - ' A \cap \Omega \wedge A \in \text{sigma\_sets } \Omega' M'$ 
        by metis
      then show ?case
        by (auto simp: vimage_UN[symmetric] intro: sigma_sets.Union)
    qed
  qed
qed

```

```

lemma (in ring_of_sets) UNION_in_sets:
  fixes A:: nat  $\Rightarrow$  'a set
  assumes A: range A  $\subseteq$  M
  shows  $(\bigcup i \in \{0..<n\}. A\ i) \in M$ 
proof (induct n)
  case 0 show ?case by simp
next
  case (Suc n)
  thus ?case
    by (simp add: atLeastLessThanSuc) (metis A Un UNIV_I image_subset_iff)
qed

```

```

lemma (in ring_of_sets) range_disjointed_sets:
  assumes A: range A  $\subseteq$  M
  shows range (disjointed A)  $\subseteq$  M
proof (auto simp add: disjointed_def)
  fix n
  show A n -  $(\bigcup i \in \{0..<n\}. A\ i) \in M$  using UNION_in_sets
  by (metis A Diff UNIV_I image_subset_iff)
qed

```

```

lemma (in algebra) range_disjointed_sets':
  range A  $\subseteq$  M  $\implies$  range (disjointed A)  $\subseteq$  M
  using range_disjointed_sets .

```

```

lemma sigma_algebra_disjoint_iff:
  sigma_algebra  $\Omega$  M  $\longleftrightarrow$  algebra  $\Omega$  M  $\wedge$ 
   $(\forall A. \text{range } A \subseteq M \longrightarrow \text{disjoint\_family } A \longrightarrow (\bigcup i::\text{nat}. A\ i) \in M)$ 
proof (auto simp add: sigma_algebra_iff)
  fix A :: nat  $\Rightarrow$  'a set
  assume M: algebra  $\Omega$  M
  and A: range A  $\subseteq$  M
  and UnA:  $\forall A. \text{range } A \subseteq M \longrightarrow \text{disjoint\_family } A \longrightarrow (\bigcup i::\text{nat}. A\ i) \in M$ 
  hence range (disjointed A)  $\subseteq$  M  $\longrightarrow$ 
    disjoint_family (disjointed A)  $\longrightarrow$ 
     $(\bigcup i. \text{disjointed } A\ i) \in M$  by blast
  hence  $(\bigcup i. \text{disjointed } A\ i) \in M$ 

```

```

    by (simp add: algebra.range_disjointed_sets'[of  $\Omega$ ]  $M$   $A$  disjoint_family_disjointed)
    thus  $(\bigcup_{i::\text{nat. } A} i) \in M$  by (simp add: UN_disjointed_eq)
qed

```

Ring generated by a semiring

definition (in *semiring_of_sets*) *generated_ring* :: 'a set set **where**
generated_ring = $\{ \bigcup C \mid C. C \subseteq M \wedge \text{finite } C \wedge \text{disjoint } C \}$

lemma (in *semiring_of_sets*) *generated_ringE*[*elim?*]:
assumes $a \in \text{generated\_ring}$
obtains C **where** *finite* C *disjoint* C $C \subseteq M$ $a = \bigcup C$
using *assms* **unfolding** *generated_ring_def* **by** *auto*

lemma (in *semiring_of_sets*) *generated_ringI*[*intro?*]:
assumes *finite* C *disjoint* C $C \subseteq M$ $a = \bigcup C$
shows $a \in \text{generated\_ring}$
using *assms* **unfolding** *generated_ring_def* **by** *auto*

lemma (in *semiring_of_sets*) *generated_ringI_Basic*:
 $A \in M \implies A \in \text{generated\_ring}$
by (rule *generated_ringI*[of $\{A\}$]) (auto simp: *disjoint_def*)

lemma (in *semiring_of_sets*) *generated_ring_disjoint_Un*[*intro?*]:
assumes $a: a \in \text{generated\_ring}$ **and** $b: b \in \text{generated\_ring}$
and $a \cap b = \{\}$
shows $a \cup b \in \text{generated\_ring}$

proof –

from a b **obtain** C_a C_b
where C_a : *finite* C_a *disjoint* C_a $C_a \subseteq M$ $a = \bigcup C_a$
and C_b : *finite* C_b *disjoint* C_b $C_b \subseteq M$ $b = \bigcup C_b$
using *generated_ringE* **by** *metis*

show *?thesis*

proof

from $\langle a \cap b = \{\} \rangle$ C_a C_b **show** *disjoint* $(C_a \cup C_b)$
by (auto *intro!*: *disjoint_union*)

qed (use C_a C_b **in** *auto*)

qed

lemma (in *semiring_of_sets*) *generated_ring_empty*: $\{\} \in \text{generated\_ring}$
by (auto simp: *generated_ring_def* *disjoint_def*)

lemma (in *semiring_of_sets*) *generated_ring_disjoint_Union*:
assumes *finite* A **shows** $A \subseteq \text{generated\_ring} \implies \text{disjoint } A \implies \bigcup A \in \text{generated\_ring}$
using *assms* **by** (induct A) (auto simp: *disjoint_def* *intro!*: *generated_ring_disjoint_Un* *generated_ring_empty*)

lemma (in *semiring_of_sets*) *generated_ring_disjoint_UNION*:


```

finite I  $\implies$  disjoint (A ' I)  $\implies$  ( $\bigwedge i. i \in I \implies A\ i \in$  generated_ring)  $\implies \bigcup (A$ 
' I)  $\in$  generated_ring
by (intro generated_ring_disjoint_Union) auto

```

```

lemma (in semiring_of_sets) generated_ring_Int:
  assumes a: a  $\in$  generated_ring and b: b  $\in$  generated_ring
  shows a  $\cap$  b  $\in$  generated_ring
proof -
  from a b obtain Ca Cb
  where Ca: finite Ca disjoint Ca Ca  $\subseteq$  M a =  $\bigcup$  Ca
    and Cb: finite Cb disjoint Cb Cb  $\subseteq$  M b =  $\bigcup$  Cb
  using generated_ringE by metis
  define C where C = ( $\lambda(a,b). a \cap b$ )' (Ca  $\times$  Cb)
  show ?thesis
  proof
    show disjoint C
  proof (simp add: disjoint_def C_def, intro ballI impI)
    fix a1 b1 a2 b2 assume sets: a1  $\in$  Ca b1  $\in$  Cb a2  $\in$  Ca b2  $\in$  Cb
    assume a1  $\cap$  b1  $\neq$  a2  $\cap$  b2
    then have a1  $\neq$  a2  $\vee$  b1  $\neq$  b2 by auto
    then show (a1  $\cap$  b1)  $\cap$  (a2  $\cap$  b2) = {}
  proof
    assume a1  $\neq$  a2
    with sets Ca have a1  $\cap$  a2 = {}
    by (auto simp: disjoint_def)
    then show ?thesis by auto
  next
    assume b1  $\neq$  b2
    with sets Cb have b1  $\cap$  b2 = {}
    by (auto simp: disjoint_def)
    then show ?thesis by auto
  qed
  qed
qed (use Ca Cb in  $\langle$ auto simp: C_def $\rangle$ )
qed

```

```

lemma (in semiring_of_sets) generated_ring_Inter:
  assumes finite A A  $\neq$  {} shows A  $\subseteq$  generated_ring  $\implies \bigcap A \in$  generated_ring
  using assms by (induct A rule: finite_ne_induct) (auto intro: generated_ring_Int)

```

```

lemma (in semiring_of_sets) generated_ring_INTER:
  finite I  $\implies I \neq$  {}  $\implies$  ( $\bigwedge i. i \in I \implies A\ i \in$  generated_ring)  $\implies \bigcap (A\ ' I) \in$ 
generated_ring
by (intro generated_ring_Inter) auto

```

```

lemma (in semiring_of_sets) generating_ring:
  ring_of_sets  $\Omega$  generated_ring
proof (rule ring_of_setsI)
  let ?R = generated_ring

```

```

show ?R ⊆ Pow Ω
using sets_into_space by (auto simp: generated_ring_def generated_ring_empty)
show {} ∈ ?R by (rule generated_ring_empty)

```

```

{
  fix a b assume a ∈ ?R b ∈ ?R
  then obtain Ca Cb
    where Ca: finite Ca disjoint Ca Ca ⊆ M a = ⋃ Ca
      and Cb: finite Cb disjoint Cb Cb ⊆ M b = ⋃ Cb
    using generated_ringE by metis
  show a - b ∈ ?R
  proof cases
    assume Cb = {} with Cb ⟨a ∈ ?R⟩ show ?thesis
      by simp
    next
      assume Cb ≠ {}
      with Ca Cb have a - b = (⋃ a' ∈ Ca. ⋂ b' ∈ Cb. a' - b') by auto
      also have ... ∈ ?R
      proof (intro generated_ring_INTER generated_ring_disjoint_UNION)
        fix a b assume a ∈ Ca b ∈ Cb
        with Ca Cb Diff_cover[of a b] show a - b ∈ ?R
          by (auto simp add: generated_ring_def)
          (metis DiffI Diff_eq_empty_iff empty_iff)
      next
        show disjoint ((λa'. ⋂ b' ∈ Cb. a' - b') `Ca)
          using Ca by (auto simp add: disjoint_def ⟨Cb ≠ {}⟩)
      next
        show finite Ca finite Cb Cb ≠ {} by fact+
      qed
      finally show a - b ∈ ?R .
    qed
  }
note Diff = this

```

```

fix a b assume sets: a ∈ ?R b ∈ ?R
have a ∪ b = (a - b) ∪ (a ∩ b) ∪ (b - a) by auto
also have ... ∈ ?R
  by (intro sets generated_ring_disjoint_Un generated_ring_Int Diff) auto
finally show a ∪ b ∈ ?R .
qed

```

lemma (in semiring_of_sets) sigma_sets_generated_ring_eq: sigma_sets Ω generated_ring = sigma_sets Ω M

```

proof
  interpret M: sigma_algebra Ω sigma_sets Ω M
  using space_closed by (rule sigma_algebra_sigma_sets)
  show sigma_sets Ω generated_ring ⊆ sigma_sets Ω M
    by (blast intro!: sigma_sets_mono elim: generated_ringE)
  qed (auto intro!: generated_ringI_Basic sigma_sets_mono)

```

A Two-Element Series

definition *binaryset* :: 'a set $\Rightarrow$ 'a set $\Rightarrow$ nat $\Rightarrow$ 'a set
 where *binaryset* A B = ($\lambda x. \{\}$)(0 := A, Suc 0 := B)

lemma *range_binaryset_eq*: $\text{range}(\text{binaryset } A \ B) = \{A, B, \{\}\}$
apply (*simp add: binaryset_def*)
apply (*rule set_eqI*)
apply (*auto simp add: image_iff*)
done

lemma *UN_binaryset_eq*: $(\bigcup i. \text{binaryset } A \ B \ i) = A \cup B$
by (*simp add: range_binaryset_eq cong del: SUP_cong_simp*)

Closed CDI

definition *closed_cdi* :: 'a set $\Rightarrow$ 'a set set $\Rightarrow$ bool **where**
closed_cdi $\Omega \ M \longleftrightarrow$
 $M \subseteq \text{Pow } \Omega \ \&$
 $(\forall s \in M. \Omega - s \in M) \ \&$
 $(\forall A. (\text{range } A \subseteq M) \ \& \ (A \ 0 = \{\}) \ \& \ (\forall n. A \ n \subseteq A \ (\text{Suc } n)) \longrightarrow$
 $(\bigcup i. A \ i) \in M) \ \&$
 $(\forall A. (\text{range } A \subseteq M) \ \& \ \text{disjoint\_family } A \longrightarrow (\bigcup i::\text{nat}. A \ i) \in M)$

inductive_set

smallest_ccdi_sets :: 'a set $\Rightarrow$ 'a set set $\Rightarrow$ 'a set set
for $\Omega \ M$
where
Basic [intro]:
 $a \in M \implies a \in \text{smallest\_ccdi\_sets } \Omega \ M$
| *Compl* [intro]:
 $a \in \text{smallest\_ccdi\_sets } \Omega \ M \implies \Omega - a \in \text{smallest\_ccdi\_sets } \Omega \ M$
| *Inc*:
 $\text{range } A \in \text{Pow}(\text{smallest\_ccdi\_sets } \Omega \ M) \implies A \ 0 = \{\} \implies (\bigwedge n. A \ n \subseteq A \ (\text{Suc } n))$
 $\implies (\bigcup i. A \ i) \in \text{smallest\_ccdi\_sets } \Omega \ M$
| *Disj*:
 $\text{range } A \in \text{Pow}(\text{smallest\_ccdi\_sets } \Omega \ M) \implies \text{disjoint\_family } A$
 $\implies (\bigcup i::\text{nat}. A \ i) \in \text{smallest\_ccdi\_sets } \Omega \ M$

lemma (*in subset_class*) *smallest_closed_cdi1*: $M \subseteq \text{smallest\_ccdi\_sets } \Omega \ M$
by *auto*

lemma (*in subset_class*) *smallest_ccdi_sets*: $\text{smallest\_ccdi\_sets } \Omega \ M \subseteq \text{Pow } \Omega$
apply (*rule subsetI*)
apply (*erule smallest_ccdi_sets.induct*)
apply (*auto intro: range_subsetD dest: sets_into_space*)
done

lemma (*in subset_class*) *smallest_closed_cdi2*: *closed_cdi* $\Omega \ (\text{smallest\_ccdi\_sets}$

```

Ω M)
  apply (auto simp add: closed_cdi_def smallest_ccdi_sets)
  apply (blast intro: smallest_ccdi_sets.Inc smallest_ccdi_sets.Disj) +
done

```

```

lemma closed_cdi_subset: closed_cdi Ω M  $\implies$  M  $\subseteq$  Pow Ω
  by (simp add: closed_cdi_def)

```

```

lemma closed_cdi_Compl: closed_cdi Ω M  $\implies$  s  $\in$  M  $\implies$  Ω - s  $\in$  M
  by (simp add: closed_cdi_def)

```

```

lemma closed_cdi_Inc:
  closed_cdi Ω M  $\implies$  range A  $\subseteq$  M  $\implies$  A 0 = {}  $\implies$  (!n. A n  $\subseteq$  A (Suc n))
 $\implies$  ( $\bigcup i. A i$ )  $\in$  M
  by (simp add: closed_cdi_def)

```

```

lemma closed_cdi_Disj:
  closed_cdi Ω M  $\implies$  range A  $\subseteq$  M  $\implies$  disjoint_family A  $\implies$  ( $\bigcup i::nat. A i$ )  $\in$ 
M
  by (simp add: closed_cdi_def)

```

```

lemma closed_cdi_Un:
  assumes cdi: closed_cdi Ω M and empty: {}  $\in$  M
  and A: A  $\in$  M and B: B  $\in$  M
  and disj: A  $\cap$  B = {}
  shows A  $\cup$  B  $\in$  M

```

```

proof -
  have ra: range (binaryset A B)  $\subseteq$  M
  by (simp add: range_binaryset_eq empty A B)
  have di: disjoint_family (binaryset A B) using disj
  by (simp add: disjoint_family_on_def binaryset_def Int_commute)
  from closed_cdi_Disj [OF cdi ra di]
  show ?thesis
  by (simp add: UN_binaryset_eq)
qed

```

```

lemma (in algebra) smallest_ccdi_sets_Un:
  assumes A: A  $\in$  smallest_ccdi_sets Ω M and B: B  $\in$  smallest_ccdi_sets Ω M
  and disj: A  $\cap$  B = {}
  shows A  $\cup$  B  $\in$  smallest_ccdi_sets Ω M

```

```

proof -
  have ra: range (binaryset A B)  $\in$  Pow (smallest_ccdi_sets Ω M)
  by (simp add: range_binaryset_eq A B smallest_ccdi_sets.Basic)
  have di: disjoint_family (binaryset A B) using disj
  by (simp add: disjoint_family_on_def binaryset_def Int_commute)
  from Disj [OF ra di]
  show ?thesis
  by (simp add: UN_binaryset_eq)
qed

```

```

lemma (in algebra) smallest_ccdi_sets_Int1:
  assumes  $a: a \in M$ 
  shows  $b \in \text{smallest\_ccdi\_sets } \Omega \ M \implies a \cap b \in \text{smallest\_ccdi\_sets } \Omega \ M$ 
proof (induct rule: smallest_ccdi_sets.induct)
  case (Basic x)
  thus ?case
    by (metis a Int smallest_ccdi_sets.Basic)
next
  case (Compl x)
  have  $a \cap (\Omega - x) = \Omega - ((\Omega - a) \cup (a \cap x))$ 
    by blast
  also have  $\dots \in \text{smallest\_ccdi\_sets } \Omega \ M$ 
    by (metis smallest_ccdi_sets.Compl a Compl(2) Diff_Int2 Diff_Int_distrib2
      Diff_disjoint Int_Diff Int_empty_right smallest_ccdi_sets_Un
      smallest_ccdi_sets.Basic smallest_ccdi_sets.Compl)
  finally show ?case .
next
  case (Inc A)
  have 1:  $(\bigcup i. (\lambda i. a \cap A \ i) \ i) = a \cap (\bigcup i. A \ i)$ 
    by blast
  have  $\text{range } (\lambda i. a \cap A \ i) \in \text{Pow}(\text{smallest\_ccdi\_sets } \Omega \ M)$  using Inc
    by blast
  moreover have  $(\lambda i. a \cap A \ i) \ 0 = \{\}$ 
    by (simp add: Inc)
  moreover have  $!!n. (\lambda i. a \cap A \ i) \ n \subseteq (\lambda i. a \cap A \ i) \ (\text{Suc } n)$  using Inc
    by blast
  ultimately have 2:  $(\bigcup i. (\lambda i. a \cap A \ i) \ i) \in \text{smallest\_ccdi\_sets } \Omega \ M$ 
    by (rule smallest_ccdi_sets.Inc)
  show ?case
    by (metis 1 2)
next
  case (Disj A)
  have 1:  $(\bigcup i. (\lambda i. a \cap A \ i) \ i) = a \cap (\bigcup i. A \ i)$ 
    by blast
  have  $\text{range } (\lambda i. a \cap A \ i) \in \text{Pow}(\text{smallest\_ccdi\_sets } \Omega \ M)$  using Disj
    by blast
  moreover have disjoint_family  $(\lambda i. a \cap A \ i)$  using Disj
    by (auto simp add: disjoint_family_on_def)
  ultimately have 2:  $(\bigcup i. (\lambda i. a \cap A \ i) \ i) \in \text{smallest\_ccdi\_sets } \Omega \ M$ 
    by (rule smallest_ccdi_sets.Disj)
  show ?case
    by (metis 1 2)
qed

```

```

lemma (in algebra) smallest_ccdi_sets_Int:
  assumes  $b: b \in \text{smallest\_ccdi\_sets } \Omega \ M$ 
  shows  $a \in \text{smallest\_ccdi\_sets } \Omega \ M \implies a \cap b \in \text{smallest\_ccdi\_sets } \Omega \ M$ 

```

```

proof (induct rule: smallest_ccdi_sets.induct)
  case (Basic  $x$ )
    thus ?case
      by (metis  $b$  smallest_ccdi_sets_Int1)
next
  case (Compl  $x$ )
    have  $(\Omega - x) \cap b = \Omega - (x \cap b \cup (\Omega - b))$ 
      by blast
    also have  $\dots \in \text{smallest\_ccdi\_sets } \Omega \ M$ 
      by (metis Compl(2) Diff_disjoint Int_Diff Int_commute Int_empty_right  $b$ 
        smallest_ccdi_sets.Compl smallest_ccdi_sets_Un)
    finally show ?case .
next
  case (Inc  $A$ )
    have 1:  $(\bigcup i. (\lambda i. A \ i \cap b) \ i) = (\bigcup i. A \ i) \cap b$ 
      by blast
    have range  $(\lambda i. A \ i \cap b) \in \text{Pow}(\text{smallest\_ccdi\_sets } \Omega \ M)$  using Inc
      by blast
    moreover have  $(\lambda i. A \ i \cap b) \ 0 = \{\}$ 
      by (simp add: Inc)
    moreover have !! $n. (\lambda i. A \ i \cap b) \ n \subseteq (\lambda i. A \ i \cap b) \ (\text{Suc } n)$  using Inc
      by blast
    ultimately have 2:  $(\bigcup i. (\lambda i. A \ i \cap b) \ i) \in \text{smallest\_ccdi\_sets } \Omega \ M$ 
      by (rule smallest_ccdi_sets.Inc)
    show ?case
      by (metis 1 2)
next
  case (Disj  $A$ )
    have 1:  $(\bigcup i. (\lambda i. A \ i \cap b) \ i) = (\bigcup i. A \ i) \cap b$ 
      by blast
    have range  $(\lambda i. A \ i \cap b) \in \text{Pow}(\text{smallest\_ccdi\_sets } \Omega \ M)$  using Disj
      by blast
    moreover have disjoint_family  $(\lambda i. A \ i \cap b)$  using Disj
      by (auto simp add: disjoint_family_on_def)
    ultimately have 2:  $(\bigcup i. (\lambda i. A \ i \cap b) \ i) \in \text{smallest\_ccdi\_sets } \Omega \ M$ 
      by (rule smallest_ccdi_sets.Disj)
    show ?case
      by (metis 1 2)
qed

lemma (in algebra) sigma_property_disjoint_lemma:
  assumes  $sbC: M \subseteq C$ 
    and ccdi: closed_cdi  $\Omega \ C$ 
  shows sigma_sets  $\Omega \ M \subseteq C$ 
proof -
  have smallest_ccdi_sets  $\Omega \ M \in \{B . M \subseteq B \wedge \text{sigma\_algebra } \Omega \ B\}$ 
    apply (auto simp add: sigma_algebra_disjoint_iff algebra_iff_Int
      smallest_ccdi_sets_Int)
    apply (metis Union_Pow_eq Union_upper subsetD smallest_ccdi_sets)

```

```

    apply (blast intro: smallest_ccdi_sets.Disj)
  done
  hence sigma_sets (Ω) (M) ⊆ smallest_ccdi_sets Ω M
    by clarsimp
    (drule sigma_algebra.sigma_sets_subset [where a=M], auto)
  also have ... ⊆ C
  proof
    fix x
    assume x: x ∈ smallest_ccdi_sets Ω M
    thus x ∈ C
    proof (induct rule: smallest_ccdi_sets.induct)
      case (Basic x)
      thus ?case
        by (metis Basic subsetD sbC)
    next
      case (Compl x)
      thus ?case
        by (blast intro: closed_ccdi_Compl [OF ccdi, simplified])
    next
      case (Inc A)
      thus ?case
        by (auto intro: closed_ccdi_Inc [OF ccdi, simplified])
    next
      case (Disj A)
      thus ?case
        by (auto intro: closed_ccdi_Disj [OF ccdi, simplified])
    qed
  qed
  finally show ?thesis .
qed

lemma (in algebra) sigma_property_disjoint:
  assumes sbC: M ⊆ C
    and compl: !!s. s ∈ C ∩ sigma_sets (Ω) (M) ⟹ Ω - s ∈ C
    and inc: !!A. range A ⊆ C ∩ sigma_sets (Ω) (M)
      ⟹ A 0 = {} ⟹ (!!n. A n ⊆ A (Suc n))
      ⟹ (⋃ i. A i) ∈ C
    and disj: !!A. range A ⊆ C ∩ sigma_sets (Ω) (M)
      ⟹ disjoint_family A ⟹ (⋃ i::nat. A i) ∈ C
  shows sigma_sets (Ω) (M) ⊆ C
proof -
  have sigma_sets (Ω) (M) ⊆ C ∩ sigma_sets (Ω) (M)
  proof (rule sigma_property_disjoint_lemma)
    show M ⊆ C ∩ sigma_sets (Ω) (M)
      by (metis Int_greatest Set.subsetI sbC sigma_sets.Basic)
  next
    show closed_ccdi Ω (C ∩ sigma_sets (Ω) (M))
      by (simp add: closed_ccdi_def compl inc disj)
      (metis PowI Set.subsetI le_infI2 sigma_sets_into_sp space_closed)
  qed

```

```

      IntE sigma_sets.Compl range_subsetD sigma_sets.Union)
    qed
  thus ?thesis
    by blast
  qed

```

Dynkin systems

```

locale Dynkin_system = subset_class +
  assumes space:  $\Omega \in M$ 
  and compl[intro!]:  $\bigwedge A. A \in M \implies \Omega - A \in M$ 
  and UN[intro!]:  $\bigwedge A. \text{disjoint\_family } A \implies \text{range } A \subseteq M$ 
     $\implies (\bigcup i::\text{nat}. A\ i) \in M$ 

```

```

lemma (in Dynkin_system) empty[intro, simp]:  $\{\} \in M$ 
  using space compl[of  $\Omega$ ] by simp

```

```

lemma (in Dynkin_system) diff:
  assumes sets:  $D \in M$   $E \in M$  and  $D \subseteq E$ 
  shows  $E - D \in M$ 

```

```

proof -
  let ?f =  $\lambda x. \text{if } x = 0 \text{ then } D \text{ else if } x = \text{Suc } 0 \text{ then } \Omega - E \text{ else } \{\}$ 
  have range ?f =  $\{D, \Omega - E, \{\}\}$ 
    by (auto simp: image_iff)
  moreover have  $D \cup (\Omega - E) = (\bigcup i. ?f\ i)$ 
    by (auto simp: image_iff split: if_split_asm)
  moreover
  have disjoint_family ?f unfolding disjoint_family_on_def
    using  $\langle D \in M \rangle [THEN\ \text{sets\_into\_space}] \langle D \subseteq E \rangle$  by auto
  ultimately have  $\Omega - (D \cup (\Omega - E)) \in M$ 
    using sets UN by auto fastforce
  also have  $\Omega - (D \cup (\Omega - E)) = E - D$ 
    using assms sets_into_space by auto
  finally show ?thesis .
qed

```

```

lemma Dynkin_systemI:
  assumes  $\bigwedge A. A \in M \implies A \subseteq \Omega$   $\Omega \in M$ 
  assumes  $\bigwedge A. A \in M \implies \Omega - A \in M$ 
  assumes  $\bigwedge A. \text{disjoint\_family } A \implies \text{range } A \subseteq M$ 
     $\implies (\bigcup i::\text{nat}. A\ i) \in M$ 
  shows Dynkin_system  $\Omega\ M$ 
  using assms by (auto simp: Dynkin_system_def Dynkin_system_axioms_def
    subset_class_def)

```

```

lemma Dynkin_systemI':
  assumes 1:  $\bigwedge A. A \in M \implies A \subseteq \Omega$ 
  assumes empty:  $\{\} \in M$ 
  assumes Diff:  $\bigwedge A. A \in M \implies \Omega - A \in M$ 

```



```

    assumes 2:  $\bigwedge A. \text{disjoint\_family } A \implies \text{range } A \subseteq M$ 
       $\implies (\bigcup i::\text{nat}. A\ i) \in M$ 
    shows Dynkin_system  $\Omega$   $M$ 
  proof -
    from Diff[OF empty] have  $\Omega \in M$  by auto
    from 1 this Diff 2 show ?thesis
      by (intro Dynkin_systemI) auto
  qed

lemma Dynkin_system_trivial:
  shows Dynkin_system  $A$  (Pow  $A$ )
  by (rule Dynkin_systemI) auto

lemma sigma_algebra_imp_Dynkin_system:
  assumes sigma_algebra  $\Omega$   $M$  shows Dynkin_system  $\Omega$   $M$ 
  proof -
    interpret sigma_algebra  $\Omega$   $M$  by fact
    show ?thesis using sets_into_space by (fastforce intro!: Dynkin_systemI)
  qed

```

Intersection sets systems

```

definition Int_stable :: 'a set set  $\Rightarrow$  bool where
  Int_stable  $M \iff (\forall a \in M. \forall b \in M. a \cap b \in M)$ 

lemma (in algebra) Int_stable: Int_stable  $M$ 
  unfolding Int_stable_def by auto

lemma Int_stableI_image:
   $(\bigwedge i\ j. i \in I \implies j \in I \implies \exists k \in I. A\ i \cap A\ j = A\ k) \implies \text{Int\_stable } (A\ ` I)$ 
  by (auto simp: Int_stable_def image_def)

lemma Int_stableI:
   $(\bigwedge a\ b. a \in A \implies b \in A \implies a \cap b \in A) \implies \text{Int\_stable } A$ 
  unfolding Int_stable_def by auto

lemma Int_stableD:
   $\text{Int\_stable } M \implies a \in M \implies b \in M \implies a \cap b \in M$ 
  unfolding Int_stable_def by auto

lemma (in Dynkin_system) sigma_algebra_eq_Int_stable:
   $\text{sigma\_algebra } \Omega\ M \iff \text{Int\_stable } M$ 
  proof
    assume sigma_algebra  $\Omega$   $M$  then show Int_stable  $M$ 
      unfolding sigma_algebra_def using algebra.Int_stable by auto
  next
    assume Int_stable  $M$ 
    show sigma_algebra  $\Omega$   $M$ 
      unfolding sigma_algebra_disjoint_iff algebra_iff_Un

```

```

proof (intro conjI ballI allI impI)
  show  $M \subseteq \text{Pow } (\Omega)$  using sets_into_space by auto
next
  fix  $A\ B$  assume  $A \in M\ B \in M$ 
  then have  $A \cup B = \Omega - ((\Omega - A) \cap (\Omega - B))$ 
     $\Omega - A \in M\ \Omega - B \in M$ 
    using sets_into_space by auto
  then show  $A \cup B \in M$ 
    using  $\langle \text{Int\_stable } M \rangle$  unfolding Int_stable_def by auto
qed auto
qed

```

Smallest Dynkin systems

definition *Dynkin* :: 'a set $\Rightarrow$ 'a set set $\Rightarrow$ 'a set set **where**
 $\text{Dynkin } \Omega\ M = (\bigcap \{D. \text{Dynkin\_system } \Omega\ D \wedge M \subseteq D\})$

```

lemma Dynkin_system_Dynkin:
  assumes  $M \subseteq \text{Pow } (\Omega)$ 
  shows Dynkin_system  $\Omega$  (Dynkin  $\Omega\ M$ )
proof (rule Dynkin_systemI)
  fix  $A$  assume  $A \in \text{Dynkin } \Omega\ M$ 
  moreover
    { fix  $D$  assume  $A \in D$  and  $d: \text{Dynkin\_system } \Omega\ D$ 
      then have  $A \subseteq \Omega$  by (auto simp: Dynkin_system_def subset_class_def) }
  moreover have  $\{D. \text{Dynkin\_system } \Omega\ D \wedge M \subseteq D\} \neq \{\}$ 
    using assms Dynkin_system_trivial by fastforce
  ultimately show  $A \subseteq \Omega$ 
    unfolding Dynkin_def using assms
    by auto
next
  show  $\Omega \in \text{Dynkin } \Omega\ M$ 
    unfolding Dynkin_def using Dynkin_system.space by fastforce
next
  fix  $A$  assume  $A \in \text{Dynkin } \Omega\ M$ 
  then show  $\Omega - A \in \text{Dynkin } \Omega\ M$ 
    unfolding Dynkin_def using Dynkin_system.compl by force
next
  fix  $A :: \text{nat} \Rightarrow 'a\ \text{set}$ 
  assume  $A: \text{disjoint\_family } A\ \text{range } A \subseteq \text{Dynkin } \Omega\ M$ 
  show  $(\bigcup i. A\ i) \in \text{Dynkin } \Omega\ M$  unfolding Dynkin_def
  proof (simp, safe)
    fix  $D$  assume Dynkin_system  $\Omega\ D\ M \subseteq D$ 
    with  $A$  have  $(\bigcup i. A\ i) \in D$ 
      by (intro Dynkin_system.UN) (auto simp: Dynkin_def)
    then show  $(\bigcup i. A\ i) \in D$  by auto
  qed
qed

```

```

lemma Dynkin_Basic[intro]:  $A \in M \implies A \in \text{Dynkin } \Omega \ M$ 
  unfolding Dynkin_def by auto

lemma (in Dynkin_system) restricted_Dynkin_system:
  assumes  $D \in M$ 
  shows  $\text{Dynkin\_system } \Omega \ \{Q. Q \subseteq \Omega \wedge Q \cap D \in M\}$ 
proof (rule Dynkin_systemI, simp_all)
  have  $\Omega \cap D = D$ 
    using  $\langle D \in M \rangle$  sets_into_space by auto
  then show  $\Omega \cap D \in M$ 
    using  $\langle D \in M \rangle$  by auto
next
  fix A assume  $A \subseteq \Omega \wedge A \cap D \in M$ 
  moreover have  $(\Omega - A) \cap D = (\Omega - (A \cap D)) - (\Omega - D)$ 
    by auto
  ultimately show  $(\Omega - A) \cap D \in M$ 
    using  $\langle D \in M \rangle$  by (auto intro: diff)
next
  fix A :: nat  $\Rightarrow$  'a set
  assume disjoint_family A range  $A \subseteq \{Q. Q \subseteq \Omega \wedge Q \cap D \in M\}$ 
  then have  $\bigwedge i. A \ i \subseteq \Omega$  disjoint_family  $(\lambda i. A \ i \cap D)$ 
    range  $(\lambda i. A \ i \cap D) \subseteq M$   $(\bigcup x. A \ x) \cap D = (\bigcup x. A \ x \cap D)$ 
    by ((fastforce simp: disjoint_family_on_def)+)
  then show  $(\bigcup x. A \ x) \subseteq \Omega \wedge (\bigcup x. A \ x) \cap D \in M$ 
    by (auto simp del: UN_simps)
qed

lemma (in Dynkin_system) Dynkin_subset:
  assumes  $N \subseteq M$ 
  shows  $\text{Dynkin } \Omega \ N \subseteq M$ 
proof -
  have  $\text{Dynkin\_system } \Omega \ M ..$ 
  then have  $\text{Dynkin\_system } \Omega \ M$ 
    using assms unfolding Dynkin_system_def Dynkin_system_axioms_def subset_class_def by simp
  with  $\langle N \subseteq M \rangle$  show ?thesis by (auto simp add: Dynkin_def)
qed

lemma sigma_eq_Dynkin:
  assumes sets:  $M \subseteq \text{Pow } \Omega$ 
  assumes Int_stable M
  shows  $\text{sigma\_sets } \Omega \ M = \text{Dynkin } \Omega \ M$ 
proof -
  have  $\text{Dynkin } \Omega \ M \subseteq \text{sigma\_sets } (\Omega) \ (M)$ 
    using sigma_algebra_imp_Dynkin_system
  unfolding Dynkin_def sigma_sets_least_sigma_algebra[OF sets] by auto
  moreover
  interpret Dynkin_system  $\Omega \ \text{Dynkin } \Omega \ M$ 
    using Dynkin_system_Dynkin[OF sets] .

```

```

have sigma_algebra  $\Omega$  (Dynkin  $\Omega$   $M$ )
  unfolding sigma_algebra_eq_Int_stable Int_stable_def
proof (intro ballI)
  fix  $A$   $B$  assume  $A \in \text{Dynkin } \Omega$   $M$   $B \in \text{Dynkin } \Omega$   $M$ 
  let  $?D = \lambda E. \{Q. Q \subseteq \Omega \wedge Q \cap E \in \text{Dynkin } \Omega$   $M\}$ 
  have  $M \subseteq ?D$   $B$ 
  proof
    fix  $E$  assume  $E \in M$ 
    then have  $M \subseteq ?D$   $E$   $E \in \text{Dynkin } \Omega$   $M$ 
      using sets_into_space  $\langle \text{Int\_stable } M \rangle$  by (auto simp: Int_stable_def)
    then have  $\text{Dynkin } \Omega$   $M \subseteq ?D$   $E$ 
      using restricted_Dynkin_system  $\langle E \in \text{Dynkin } \Omega$   $M \rangle$ 
      by (intro Dynkin_system.Dynkin_subset) simp_all
    then have  $B \in ?D$   $E$ 
      using  $\langle B \in \text{Dynkin } \Omega$   $M \rangle$  by auto
    then have  $E \cap B \in \text{Dynkin } \Omega$   $M$ 
      by (subst Int_commute) simp
    then show  $E \in ?D$   $B$ 
      using sets  $\langle E \in M \rangle$  by auto
  qed
  then have  $\text{Dynkin } \Omega$   $M \subseteq ?D$   $B$ 
    using restricted_Dynkin_system  $\langle B \in \text{Dynkin } \Omega$   $M \rangle$ 
    by (intro Dynkin_system.Dynkin_subset) simp_all
  then show  $A \cap B \in \text{Dynkin } \Omega$   $M$ 
    using  $\langle A \in \text{Dynkin } \Omega$   $M \rangle$  sets_into_space by auto
  qed
from sigma_algebra.sigma_sets_subset[OF this, of  $M$ ]
have sigma_sets  $(\Omega)$   $(M) \subseteq \text{Dynkin } \Omega$   $M$  by auto
ultimately have sigma_sets  $(\Omega)$   $(M) = \text{Dynkin } \Omega$   $M$  by auto
then show ?thesis
  by (auto simp: Dynkin_def)
qed

lemma (in Dynkin_system) Dynkin_idem:
  Dynkin  $\Omega$   $M = M$ 
proof -
  have Dynkin  $\Omega$   $M = M$ 
  proof
    show  $M \subseteq \text{Dynkin } \Omega$   $M$ 
      using Dynkin_Basic by auto
    show Dynkin  $\Omega$   $M \subseteq M$ 
      by (intro Dynkin_subset) auto
  qed
  then show ?thesis
    by (auto simp: Dynkin_def)
qed

lemma (in Dynkin_system) Dynkin_lemma:
  assumes Int_stable  $E$ 

```

```

and E:  $E \subseteq M$   $M \subseteq \text{sigma\_sets } \Omega$  E
shows  $\text{sigma\_sets } \Omega$  E = M
proof -
  have  $E \subseteq \text{Pow } \Omega$ 
  using E sets_into_space by force
  then have *:  $\text{sigma\_sets } \Omega$  E =  $\text{Dynkin } \Omega$  E
  using  $\langle \text{Int\_stable } E \rangle$  by (rule sigma_eq_Dynkin)
  then have  $\text{Dynkin } \Omega$  E = M
  using assms Dynkin_subset[OF E(1)] by simp
  with * show ?thesis
  using assms by (auto simp: Dynkin_def)
qed

```

Induction rule for intersection-stable generators

The reason to introduce Dynkin-systems is the following induction rules for σ -algebras generated by a generator closed under intersection.

proposition *sigma_sets_induct_disjoint[consumes 3, case_names basic empty compl union]:*

```

assumes Int_stable G
and closed:  $G \subseteq \text{Pow } \Omega$ 
and A:  $A \in \text{sigma\_sets } \Omega$  G
assumes basic:  $\bigwedge A. A \in G \implies P$  A
and empty:  $P \ \{\}$ 
and compl:  $\bigwedge A. A \in \text{sigma\_sets } \Omega$  G  $\implies P$  A  $\implies P$  ( $\Omega - A$ )
and union:  $\bigwedge A. \text{disjoint\_family } A \implies \text{range } A \subseteq \text{sigma\_sets } \Omega$  G  $\implies (\bigwedge i. P$ 
 $P$  ( $A \ i$ ))  $\implies P$  ( $\bigcup i::\text{nat}. A \ i$ )
shows  $P$  A
proof -
  let ?D = {  $A \in \text{sigma\_sets } \Omega$  G.  $P$  A }
  interpret sigma_algebra  $\Omega$  sigma_sets  $\Omega$  G
  using closed by (rule sigma_algebra_sigma_sets)
  from compl[OF _ empty] closed have space:  $P$   $\Omega$  by simp
  interpret Dynkin_system  $\Omega$  ?D
  by standard (auto dest: sets_into_space intro!: space compl union)
  have sigma_sets  $\Omega$  G = ?D
  by (rule Dynkin_lemma) (auto simp: basic  $\langle \text{Int\_stable } G \rangle$ )
  with A show ?thesis by auto
qed

```

6.1.2 Measure type

definition *positive* :: $'a \text{ set set} \Rightarrow ('a \text{ set} \Rightarrow \text{ennreal}) \Rightarrow \text{bool}$ **where**
positive M $\mu \longleftrightarrow \mu \ \{\} = 0$

definition *countably_additive* :: $'a \text{ set set} \Rightarrow ('a \text{ set} \Rightarrow \text{ennreal}) \Rightarrow \text{bool}$ **where**
countably_additive M $f \longleftrightarrow$
 $(\forall A. \text{range } A \subseteq M \longrightarrow \text{disjoint\_family } A \longrightarrow (\bigcup i. A \ i) \in M \longrightarrow$
 $(\sum i. f \ (A \ i)) = f \ (\bigcup i. A \ i))$

definition *measure_space* :: 'a set $\Rightarrow$ 'a set set $\Rightarrow$ ('a set $\Rightarrow$ ennreal) $\Rightarrow$ bool
where

measure_space Ω A $\mu \longleftrightarrow$

sigma_algebra Ω $A \wedge$ *positive* A $\mu \wedge$ *countably_additive* A μ

typedef 'a *measure* =

$\{(\Omega :: 'a \text{ set}, A, \mu). (\forall a \in -A. \mu \ a = 0) \wedge \text{measure\_space } \Omega \ A \ \mu \}$

proof

have *sigma_algebra* UNIV $\{\{\}, \text{UNIV}\}$

by (auto simp: *sigma_algebra_iff2*)

then show (UNIV, $\{\{\}, \text{UNIV}\}$, $\lambda A. 0$) $\in \{(\Omega, A, \mu). (\forall a \in -A. \mu \ a = 0) \wedge \text{measure\_space } \Omega \ A \ \mu\}$

by (auto simp: *measure_space_def* *positive_def* *countably_additive_def*)

qed

definition *space* :: 'a *measure* $\Rightarrow$ 'a set **where**

space $M = \text{fst } (\text{Rep\_measure } M)$

definition *sets* :: 'a *measure* $\Rightarrow$ 'a set set **where**

sets $M = \text{fst } (\text{snd } (\text{Rep\_measure } M))$

definition *emeasure* :: 'a *measure* $\Rightarrow$ 'a set $\Rightarrow$ ennreal **where**

emeasure $M = \text{snd } (\text{snd } (\text{Rep\_measure } M))$

definition *measure* :: 'a *measure* $\Rightarrow$ 'a set $\Rightarrow$ real **where**

measure $M \ A = \text{enn2real } (\text{emeasure } M \ A)$

declare [[*coercion* *sets*]]

declare [[*coercion* *measure*]]

declare [[*coercion* *emeasure*]]

lemma *measure_space*: *measure_space* (*space* M) (*sets* M) (*emeasure* M)

by (cases M) (auto simp: *space_def* *sets_def* *emeasure_def* *Abs_measure_inverse*)

interpretation *sets*: *sigma_algebra* *space* M *sets* M **for** $M :: 'a \text{ measure}$

using *measure_space*[of M] **by** (auto simp: *measure_space_def*)

definition *measure_of* :: 'a set $\Rightarrow$ 'a set set $\Rightarrow$ ('a set $\Rightarrow$ ennreal) $\Rightarrow$ 'a *measure*
where

measure_of Ω A $\mu =$

Abs_measure (Ω , if $A \subseteq \text{Pow } \Omega$ then *sigma_sets* Ω A else $\{\{\}, \Omega\}$,

$\lambda a. \text{if } a \in \text{sigma\_sets } \Omega \ A \wedge \text{measure\_space } \Omega \ (\text{sigma\_sets } \Omega \ A) \ \mu \text{ then } \mu \ a$
 else 0)

abbreviation *sigma* Ω $A \equiv \text{measure\_of } \Omega \ A \ (\lambda x. 0)$

```

lemma measure_space_0:  $A \subseteq \text{Pow } \Omega \implies \text{measure\_space } \Omega (\text{sigma\_sets } \Omega A)$ 
( $\lambda x. 0$ )
  unfolding measure_space_def
  by (auto intro!: sigma_algebra_sigma_sets simp: positive_def countably_additive_def)

lemma sigma_algebra_trivial: sigma_algebra  $\Omega \{\{\}, \Omega\}$ 
by unfold_locales(fastforce intro: exI[where x={}] exI[where x={\Omega}])+

lemma measure_space_0': measure_space  $\Omega \{\{\}, \Omega\}$  ( $\lambda x. 0$ )
by(simp add: measure_space_def positive_def countably_additive_def sigma_algebra_trivial)

lemma measure_space_closed:
  assumes measure_space  $\Omega M \mu$ 
  shows  $M \subseteq \text{Pow } \Omega$ 
proof -
  interpret sigma_algebra  $\Omega M$  using assms by(simp add: measure_space_def)
  show ?thesis by(rule space_closed)
qed

lemma (in ring_of_sets) positive_cong_eq:
  ( $\bigwedge a. a \in M \implies \mu' a = \mu a$ )  $\implies \text{positive } M \mu' = \text{positive } M \mu$ 
  by (auto simp add: positive_def)

lemma (in sigma_algebra) countably_additive_eq:
  ( $\bigwedge a. a \in M \implies \mu' a = \mu a$ )  $\implies \text{countably\_additive } M \mu' = \text{countably\_additive } M \mu$ 
  unfolding countably_additive_def
  by (intro arg_cong[where f=All] ext) (auto simp add: countably_additive_def subset_eq)

lemma measure_space_eq:
  assumes closed:  $A \subseteq \text{Pow } \Omega$  and eq:  $\bigwedge a. a \in \text{sigma\_sets } \Omega A \implies \mu a = \mu' a$ 
  shows measure_space  $\Omega (\text{sigma\_sets } \Omega A) \mu = \text{measure\_space } \Omega (\text{sigma\_sets } \Omega A) \mu'$ 
proof -
  interpret sigma_algebra  $\Omega \text{sigma\_sets } \Omega A$  using closed by (rule sigma_algebra_sigma_sets)
  from positive_cong_eq[OF eq, of \lambda i. i] countably_additive_eq[OF eq, of \lambda i. i]
  show ?thesis
  by (auto simp: measure_space_def)
qed

lemma measure_of_eq:
  assumes closed:  $A \subseteq \text{Pow } \Omega$  and eq:  $(\bigwedge a. a \in \text{sigma\_sets } \Omega A \implies \mu a = \mu' a)$ 
  shows measure_of  $\Omega A \mu = \text{measure\_of } \Omega A \mu'$ 
proof -
  have measure_space  $\Omega (\text{sigma\_sets } \Omega A) \mu = \text{measure\_space } \Omega (\text{sigma\_sets } \Omega A) \mu'$ 
  using assms by (rule measure_space_eq)
  with eq show ?thesis

```

by (*auto simp add: measure_of_def intro!: arg_cong[where f=Abs_measure]*)
qed

lemma

shows *space_measure_of_conv*: $\text{space}(\text{measure\_of } \Omega \ A \ \mu) = \Omega$ (**is** *?space*)
and *sets_measure_of_conv*:
 $\text{sets}(\text{measure\_of } \Omega \ A \ \mu) = (\text{if } A \subseteq \text{Pow } \Omega \text{ then } \text{sigma\_sets } \Omega \ A \text{ else } \{\{\}, \Omega\})$
(**is** *?sets*)
and *emeasure_measure_of_conv*:
 $\text{emeasure}(\text{measure\_of } \Omega \ A \ \mu) =$
 $(\lambda B. \text{if } B \in \text{sigma\_sets } \Omega \ A \wedge \text{measure\_space } \Omega (\text{sigma\_sets } \Omega \ A) \ \mu \text{ then } \mu B$
 $\text{else } 0)$ (**is** *?emeasure*)
proof –
have *?space* $\wedge$ *?sets* $\wedge$ *?emeasure*
proof(*cases measure_space* Ω (*sigma_sets* $\Omega \ A$) μ)
case *True*
from *measure_space_closed*[*OF this*] *sigma_sets_superset_generator*[*of A* Ω]
have $A \subseteq \text{Pow } \Omega$ **by** *simp*
hence $\text{measure\_space } \Omega (\text{sigma\_sets } \Omega \ A) \ \mu = \text{measure\_space } \Omega (\text{sigma\_sets}$
 $\Omega \ A)$
 $(\lambda a. \text{if } a \in \text{sigma\_sets } \Omega \ A \text{ then } \mu a \text{ else } 0)$
by(*rule measure_space_eq*) *auto*
with *True* $\langle A \subseteq \text{Pow } \Omega \rangle$ **show** *?thesis*
by(*simp add: measure_of_def space_def sets_def emeasure_def Abs_measure_inverse*)
next
case *False* **thus** *?thesis*
by(*cases A* $\subseteq \text{Pow } \Omega$)(*simp_all add: Abs_measure_inverse measure_of_def*
 $\text{sets\_def space\_def emeasure\_def measure\_space\_0 measure\_space\_0'}$)
qed
thus *?space* *?sets* *?emeasure* **by** *simp_all*
qed

lemma [*simp*]:

assumes $A: A \subseteq \text{Pow } \Omega$
shows *sets_measure_of*: $\text{sets}(\text{measure\_of } \Omega \ A \ \mu) = \text{sigma\_sets } \Omega \ A$
and *space_measure_of*: $\text{space}(\text{measure\_of } \Omega \ A \ \mu) = \Omega$
using *assms*
by(*simp_all add: sets_measure_of_conv space_measure_of_conv*)

lemma *space_in_measure_of*[*simp*]: $\Omega \in \text{sets}(\text{measure\_of } \Omega \ M \ \mu)$
by (*subst sets_measure_of_conv*) (*auto simp: sigma_sets_top*)

lemma (**in** *sigma_algebra*) *sets_measure_of_eq*[*simp*]: $\text{sets}(\text{measure\_of } \Omega \ M \ \mu)$
 $= M$
using *space_closed* **by** (*auto intro!: sigma_sets_eq*)

lemma (**in** *sigma_algebra*) *space_measure_of_eq*[*simp*]: $\text{space}(\text{measure\_of } \Omega \ M$
 $\mu) = \Omega$
by (*rule space_measure_of_conv*)

lemma *measure_of_subset*: $M \subseteq \text{Pow } \Omega \implies M' \subseteq M \implies \text{sets } (\text{measure\_of } \Omega M' \mu) \subseteq \text{sets } (\text{measure\_of } \Omega M \mu)$
by (auto intro!: *sigma_sets_subseteq*)

lemma *emeasure_sigma*: $\text{emeasure } (\text{sigma } \Omega A) = (\lambda x. 0)$
unfolding *measure_of_def* *emeasure_def*
by (*subst Abs_measure_inverse*)
 (*auto simp: measure_space_def positive_def countably_additive_def*
intro!: sigma_algebra_sigma_sets sigma_algebra_trivial)

lemma *sigma_sets_mono''*:
assumes $A \in \text{sigma\_sets } C D$
assumes $B \subseteq D$
assumes $D \subseteq \text{Pow } C$
shows $\text{sigma\_sets } A B \subseteq \text{sigma\_sets } C D$
proof
fix x **assume** $x \in \text{sigma\_sets } A B$
thus $x \in \text{sigma\_sets } C D$
proof *induct*
case (*Basic a*) **with** *assms* **have** $a \in D$ **by** *auto*
thus *?case* **..**
next
case *Empty* **show** *?case* **by** (*rule sigma_sets.Empty*)
next
from *assms* **have** $A \in \text{sets } (\text{sigma } C D)$ **by** (*subst sets_measure_of[OF ‹D ⊆ Pow C›]*)
moreover **case** (*Compl a*) **hence** $a \in \text{sets } (\text{sigma } C D)$ **by** (*subst sets_measure_of[OF ‹D ⊆ Pow C›]*)
ultimately **have** $A - a \in \text{sets } (\text{sigma } C D)$ **..**
thus *?case* **by** (*subst (asm) sets_measure_of[OF ‹D ⊆ Pow C›]*)
next
case (*Union a*)
thus *?case* **by** (*intro sigma_sets.Union*)
qed
qed

lemma *in_measure_of[intro, simp]*: $M \subseteq \text{Pow } \Omega \implies A \in M \implies A \in \text{sets } (\text{measure\_of } \Omega M \mu)$
by *auto*

lemma *space_empty_iff*: $\text{space } N = \{\} \longleftrightarrow \text{sets } N = \{\{\}\}$
by (*metis Pow_empty Sup_bot_conv(1) cSup_singleton empty_iff*
sets.sigma_sets_eq sets.space_closed sigma_sets_top subset_singletonD)

Constructing simple 'a measure

proposition *emeasure_measure_of*:
assumes $M: M = \text{measure\_of } \Omega A \mu$

```

assumes ms:  $A \subseteq \text{Pow } \Omega$  positive (sets M)  $\mu$  countably_additive (sets M)  $\mu$ 
assumes X:  $X \in \text{sets } M$ 
shows  $\text{emeasure } M \ X = \mu \ X$ 
proof –
  interpret  $\text{sigma\_algebra } \Omega \ \text{sigma\_sets } \Omega \ A$  by (rule sigma_algebra_sigma_sets)
fact
  have  $\text{measure\_space } \Omega \ (\text{sigma\_sets } \Omega \ A) \ \mu$ 
  using ms M by (simp add: measure_space_def sigma_algebra_sigma_sets)
  thus ?thesis using X ms
  by(simp add: M emeasure_measure_of_conv sets_measure_of_conv)
qed

```

```

lemma emeasure_measure_of_sigma:
assumes ms:  $\text{sigma\_algebra } \Omega \ M$  positive M  $\mu$  countably_additive M  $\mu$ 
assumes A:  $A \in M$ 
shows  $\text{emeasure } (\text{measure\_of } \Omega \ M \ \mu) \ A = \mu \ A$ 
proof –
  interpret  $\text{sigma\_algebra } \Omega \ M$  by fact
  have  $\text{measure\_space } \Omega \ (\text{sigma\_sets } \Omega \ M) \ \mu$ 
  using ms sigma_sets_eq by (simp add: measure_space_def)
  thus ?thesis by(simp add: emeasure_measure_of_conv A)
qed

```

```

lemma measure_cases[cases type: measure]:
obtains  $(\text{measure}) \ \Omega \ A \ \mu$  where  $x = \text{Abs\_measure } (\Omega, \ A, \ \mu) \ \forall a \in -A. \ \mu \ a = 0$ 
 $\text{measure\_space } \Omega \ A \ \mu$ 
by atomize_elim (cases x, auto)

```

```

lemma sets_le_imp_space_le:  $\text{sets } A \subseteq \text{sets } B \implies \text{space } A \subseteq \text{space } B$ 
by (auto dest: sets.sets_into_space)

```

```

lemma sets_eq_imp_space_eq:  $\text{sets } M = \text{sets } M' \implies \text{space } M = \text{space } M'$ 
by (auto intro!: antisym sets_le_imp_space_le)

```

```

lemma emeasure_notin_sets:  $A \notin \text{sets } M \implies \text{emeasure } M \ A = 0$ 
by (cases M) (auto simp: sets_def emeasure_def Abs_measure_inverse measure_space_def)

```

```

lemma emeasure_neq_0_sets:  $\text{emeasure } M \ A \neq 0 \implies A \in \text{sets } M$ 
using emeasure_notin_sets[of A M] by blast

```

```

lemma measure_notin_sets:  $A \notin \text{sets } M \implies \text{measure } M \ A = 0$ 
by (simp add: measure_def emeasure_notin_sets zero_ennreal.rep_eq)

```

```

lemma measure_eqI:
fixes M N :: 'a measure
assumes  $\text{sets } M = \text{sets } N$  and eq:  $\bigwedge A. A \in \text{sets } M \implies \text{emeasure } M \ A = \text{emeasure } N \ A$ 
shows  $M = N$ 

```

```

proof (cases M N rule: measure_cases[case_product measure_cases])
  case (measure_measure  $\Omega$  A  $\mu$   $\Omega'$  A'  $\mu'$ )
  interpret M: sigma_algebra  $\Omega$  A using measure_measure by (auto simp: measure_space_def)
  interpret N: sigma_algebra  $\Omega'$  A' using measure_measure by (auto simp: measure_space_def)
  have A = sets M A' = sets N
  using measure_measure by (simp_all add: sets_def Abs_measure_inverse)
  with  $\langle \text{sets } M = \text{sets } N \rangle$  have AA': A = A' by simp
  moreover from M.top N.top M.space_closed N.space_closed AA' have  $\Omega = \Omega'$ 
by auto
  moreover { fix B have  $\mu B = \mu' B$ 
    proof cases
      assume  $B \in A$ 
      with  $\langle A = \text{sets } M \rangle$  have  $\text{emeasure } M B = \text{emeasure } N B$  by simp
      with measure_measure show  $\mu B = \mu' B$ 
      by (simp add: emeasure_def Abs_measure_inverse)
    next
      assume  $B \notin A$ 
      with  $\langle A = \text{sets } M \rangle \langle A' = \text{sets } N \rangle \langle A = A' \rangle$  have  $B \notin \text{sets } M B \notin \text{sets } N$ 
      by auto
      then have  $\text{emeasure } M B = 0 \text{ emeasure } N B = 0$ 
      by (simp_all add: emeasure_notin_sets)
      with measure_measure show  $\mu B = \mu' B$ 
      by (simp add: emeasure_def Abs_measure_inverse)
    qed }
  then have  $\mu = \mu'$  by auto
  ultimately show M = N
  by (simp add: measure_measure)
qed

```

```

lemma sigma_eqI:
  assumes [simp]:  $M \subseteq \text{Pow } \Omega \ N \subseteq \text{Pow } \Omega \ \text{sigma\_sets } \Omega \ M = \text{sigma\_sets } \Omega \ N$ 
  shows  $\text{sigma } \Omega \ M = \text{sigma } \Omega \ N$ 
  by (rule measure_eqI) (simp_all add: emeasure_sigma)

```

Measurable functions

```

definition measurable :: 'a measure  $\Rightarrow$  'b measure  $\Rightarrow$  ('a  $\Rightarrow$  'b) set
  (infixr  $\rightarrow_M$  60) where
  measurable A B = {f  $\in$  space A  $\rightarrow$  space B.  $\forall y \in \text{sets } B. f - 'y \cap \text{space } A \in \text{sets } A$ }

```

```

lemma measurableI:
  ( $\bigwedge x. x \in \text{space } M \implies f x \in \text{space } N$ )  $\implies$  ( $\bigwedge A. A \in \text{sets } N \implies f - 'A \cap \text{space } M \in \text{sets } M$ )  $\implies$ 
  f  $\in$  measurable M N
  by (auto simp: measurable_def)

```

lemma *measurable_space*:

$f \in \text{measurable } M \ A \implies x \in \text{space } M \implies f \ x \in \text{space } A$
unfolding *measurable_def* **by** *auto*

lemma *measurable_sets*:

$f \in \text{measurable } M \ A \implies S \in \text{sets } A \implies f -' S \cap \text{space } M \in \text{sets } M$
unfolding *measurable_def* **by** *auto*

lemma *measurable_sets_Collect*:

assumes $f: f \in \text{measurable } M \ N$ **and** $P: \{x \in \text{space } N. P \ x\} \in \text{sets } N$ **shows**
 $\{x \in \text{space } M. P \ (f \ x)\} \in \text{sets } M$
proof –
have $f -' \{x \in \text{space } N. P \ x\} \cap \text{space } M = \{x \in \text{space } M. P \ (f \ x)\}$
using *measurable_space[OF f]* **by** *auto*
with *measurable_sets[OF f P]* **show** *?thesis*
by *simp*
qed

lemma *measurable_sigma_sets*:

assumes $B: \text{sets } N = \text{sigma\_sets } \Omega \ A \ A \subseteq \text{Pow } \Omega$
and $f: f \in \text{space } M \rightarrow \Omega$
and $ba: \bigwedge y. y \in A \implies (f -' y) \cap \text{space } M \in \text{sets } M$
shows $f \in \text{measurable } M \ N$
proof –
interpret $A: \text{sigma\_algebra } \Omega \ \text{sigma\_sets } \Omega \ A$ **using** $B(2)$ **by** *(rule sigma_algebra_sigma_sets)*
from $B \ \text{sets.top[of } N] \ A.\text{top} \ \text{sets.space\_closed[of } N] \ A.\text{space\_closed}$ **have** $\Omega: \Omega$
 $= \text{space } N$ **by** *force*

{ fix X **assume** $X \in \text{sigma\_sets } \Omega \ A$
then **have** $f -' X \cap \text{space } M \in \text{sets } M \wedge X \subseteq \Omega$
proof *induct*
case *(Basic a)* **then** **show** *?case*
by *(auto simp add: ba) (metis B(2) subsetD PowD)*
next
case *(Compl a)*
have $[simp]: f -' \Omega \cap \text{space } M = \text{space } M$
by *(auto simp add: funcset_mem [OF f])*
then **show** *?case*
by *(auto simp add: vimage_Diff Diff_Int_distrib2 sets.compl_sets Compl)*
next
case *(Union a)*
then **show** *?case*
by *(simp add: vimage_UN, simp only: UN_extend_simps(4)) blast*
qed auto }
with f **show** *?thesis*
by *(auto simp add: measurable_def B \Omega)*
qed

lemma *measurable_measure_of*:

```

assumes B:  $N \subseteq \text{Pow } \Omega$ 
and f:  $f \in \text{space } M \rightarrow \Omega$ 
and ba:  $\bigwedge y. y \in N \implies (f -' y) \cap \text{space } M \in \text{sets } M$ 
shows  $f \in \text{measurable } M (\text{measure\_of } \Omega N \mu)$ 
proof -
  have  $\text{sets } (\text{measure\_of } \Omega N \mu) = \text{sigma\_sets } \Omega N$ 
  using B by (rule sets_measure_of)
  from this assms show ?thesis by (rule measurable_sigma_sets)
qed

lemma measurable_iff_measure_of:
  assumes  $N \subseteq \text{Pow } \Omega$   $f \in \text{space } M \rightarrow \Omega$ 
  shows  $f \in \text{measurable } M (\text{measure\_of } \Omega N \mu) \longleftrightarrow (\forall A \in N. f -' A \cap \text{space } M \in \text{sets } M)$ 
  by (metis assms in_measure_of measurable_measure_of assms measurable_sets)

lemma measurable_cong_sets:
  assumes sets:  $\text{sets } M = \text{sets } M' \text{ sets } N = \text{sets } N'$ 
  shows  $\text{measurable } M N = \text{measurable } M' N'$ 
  using sets[THEN sets_eq_imp_space_eq] sets by (simp add: measurable_def)

lemma measurable_cong:
  assumes  $\bigwedge w. w \in \text{space } M \implies f w = g w$ 
  shows  $f \in \text{measurable } M M' \longleftrightarrow g \in \text{measurable } M M'$ 
  unfolding measurable_def using assms
  by (simp cong: vimage_inter_cong Pi_cong)

lemma measurable_cong':
  assumes  $\bigwedge w. w \in \text{space } M \implies f w = g w$ 
  shows  $f \in \text{measurable } M M' \longleftrightarrow g \in \text{measurable } M M'$ 
  unfolding measurable_def using assms
  by (simp cong: vimage_inter_cong Pi_cong add: simp_implies_def)

lemma measurable_cong_simp:
   $M = N \implies M' = N' \implies (\bigwedge w. w \in \text{space } M \implies f w = g w) \implies$ 
   $f \in \text{measurable } M M' \longleftrightarrow g \in \text{measurable } N N'$ 
  by (metis measurable_cong)

lemma measurable_compose:
  assumes f:  $f \in \text{measurable } M N$  and g:  $g \in \text{measurable } N L$ 
  shows  $(\lambda x. g (f x)) \in \text{measurable } M L$ 
proof -
  have  $\bigwedge A. (\lambda x. g (f x)) -' A \cap \text{space } M = f -' (g -' A \cap \text{space } N) \cap \text{space } M$ 
  using measurable_space[OF f] by auto
  with measurable_space[OF g] show ?thesis
  by (auto intro: measurable_sets[OF f] measurable_sets[OF g]
    simp del: vimage_Int simp add: measurable_def)
qed

```

lemma *measurable_comp*:

$f \in \text{measurable } M \ N \implies g \in \text{measurable } N \ L \implies g \circ f \in \text{measurable } M \ L$
using *measurable_compose*[*of f M N g L*] **by** (*simp add: comp_def*)

lemma *measurable_const*:

$c \in \text{space } M' \implies (\lambda x. c) \in \text{measurable } M \ M'$
by (*auto simp add: measurable_def*)

lemma *measurable_ident*: $\text{id} \in \text{measurable } M \ M$

by (*auto simp add: measurable_def*)

lemma *measurable_id*: $(\lambda x. x) \in \text{measurable } M \ M$

by (*simp add: measurable_def*)

lemma *measurable_ident_sets*:

assumes *eq*: $\text{sets } M = \text{sets } M'$ **shows** $(\lambda x. x) \in \text{measurable } M \ M'$
using *measurable_ident*[*of M*]
unfolding *id_def measurable_def eq sets_eq_imp_space_eq*[*OF eq*] .

lemma *sets_Least*:

assumes *meas*: $\bigwedge i::\text{nat}. \{x \in \text{space } M. P \ i \ x\} \in M$
shows $(\lambda x. \text{LEAST } j. P \ j \ x) - 'A \cap \text{space } M \in \text{sets } M$

proof –

{ **fix** *i* **have** $(\lambda x. \text{LEAST } j. P \ j \ x) - '\{i\} \cap \text{space } M \in \text{sets } M$

proof *cases*

assume *i*: $(\text{LEAST } j. \text{False}) = i$

have $(\lambda x. \text{LEAST } j. P \ j \ x) - '\{i\} \cap \text{space } M =$

$\{x \in \text{space } M. P \ i \ x\} \cap (\text{space } M - (\bigcup_{j < i}. \{x \in \text{space } M. P \ j \ x\})) \cup (\text{space } M - (\bigcup_{i. \{x \in \text{space } M. P \ i \ x\}}))$

by (*simp add: set_eq_iff, safe*)

(*insert i, auto dest: Least_le intro: LeastI intro!: Least_equality*)

with *meas* **show** *?thesis*

by (*auto intro!: sets.Int*)

next

assume *i*: $(\text{LEAST } j. \text{False}) \neq i$

then **have** $(\lambda x. \text{LEAST } j. P \ j \ x) - '\{i\} \cap \text{space } M =$

$\{x \in \text{space } M. P \ i \ x\} \cap (\text{space } M - (\bigcup_{j < i}. \{x \in \text{space } M. P \ j \ x\}))$

proof (*simp add: set_eq_iff, safe*)

fix *x* **assume** *neq*: $(\text{LEAST } j. \text{False}) \neq (\text{LEAST } j. P \ j \ x)$

have $\exists j. P \ j \ x$

by (*rule ccontr*) (*insert neq, auto*)

then **show** $P \ (\text{LEAST } j. P \ j \ x) \ x$ **by** (*rule LeastI_ex*)

qed (*auto dest: Least_le intro!: Least_equality*)

with *meas* **show** *?thesis*

by *auto*

qed }

then **have** $(\bigcup_{i \in A}. (\lambda x. \text{LEAST } j. P \ j \ x) - '\{i\} \cap \text{space } M) \in \text{sets } M$

by (*intro sets.countable_UN*) *auto*

moreover **have** $(\bigcup_{i \in A}. (\lambda x. \text{LEAST } j. P \ j \ x) - '\{i\} \cap \text{space } M) =$

```

  (λx. LEAST j. P j x) - ' A ∩ space M by auto
  ultimately show ?thesis by auto
qed

```

```

lemma measurable_mono1:
  M' ⊆ Pow Ω ⇒ M ⊆ M' ⇒
    measurable (measure_of Ω M μ) N ⊆ measurable (measure_of Ω M' μ') N
  using measure_of_subset[of M' Ω M] by (auto simp add: measurable_def)

```

Counting space

```

definition count_space :: 'a set ⇒ 'a measure where
  count_space Ω = measure_of Ω (Pow Ω) (λA. if finite A then of_nat (card A)
  else ∞)

```

```

lemma
  shows space_count_space[simp]: space (count_space Ω) = Ω
  and sets_count_space[simp]: sets (count_space Ω) = Pow Ω
  using sigma_sets_into_sp[of Pow Ω Ω]
  by (auto simp: count_space_def)

```

```

lemma measurable_count_space_eq1[simp]:
  f ∈ measurable (count_space A) M ⟷ f ∈ A → space M
  unfolding measurable_def by simp

```

```

lemma measurable_compose_countable':
  assumes f: ⋀i. i ∈ I ⇒ (λx. f i x) ∈ measurable M N
  and g: g ∈ measurable M (count_space I) and I: countable I
  shows (λx. f (g x) x) ∈ measurable M N
  unfolding measurable_def

```

```

proof safe
  fix x assume x ∈ space M then show f (g x) x ∈ space N
    using measurable_space[OF f] g[THEN measurable_space] by auto
next
  fix A assume A: A ∈ sets N
  have (λx. f (g x) x) - ' A ∩ space M = (⋃ i∈I. (g - ' {i} ∩ space M) ∩ (f i - '
  A ∩ space M))
    using measurable_space[OF g] by auto
  also have ... ∈ sets M
    using f[THEN measurable_sets, OF _ A] g[THEN measurable_sets]
    by (auto intro!: sets.countable_UN' I intro: sets.Int[OF measurable_sets measurable_sets])
  finally show (λx. f (g x) x) - ' A ∩ space M ∈ sets M .
qed

```

```

lemma measurable_count_space_eq_countable:
  assumes countable A
  shows f ∈ measurable M (count_space A) ⟷ (f ∈ space M → A ∧ (∀ a∈A. f
  - ' {a} ∩ space M ∈ sets M))

```

proof –

```

{ fix  $X$  assume  $X \subseteq A$   $f \in \text{space } M \rightarrow A$ 
  with  $\langle \text{countable } A \rangle$  have  $f - 'X \cap \text{space } M = (\bigcup_{a \in X}. f - ' \{a\} \cap \text{space } M)$ 
countable  $X$ 
  by (auto dest: countable_subset)
  moreover assume  $\forall a \in A. f - ' \{a\} \cap \text{space } M \in \text{sets } M$ 
  ultimately have  $f - 'X \cap \text{space } M \in \text{sets } M$ 
  using  $\langle X \subseteq A \rangle$  by (auto intro!: sets.countable_UN' simp del: UN_simps) }
then show ?thesis
  unfolding measurable_def by auto
qed

```

lemma measurable_count_space_eq2:

```

finite  $A \implies f \in \text{measurable } M (\text{count\_space } A) \longleftrightarrow (f \in \text{space } M \rightarrow A \wedge$ 
 $(\forall a \in A. f - ' \{a\} \cap \text{space } M \in \text{sets } M))$ 
by (intro measurable_count_space_eq_countable countable_finite)

```

lemma measurable_count_space_eq2_countable:

```

fixes  $f :: 'a \Rightarrow 'c::\text{countable}$ 
shows  $f \in \text{measurable } M (\text{count\_space } A) \longleftrightarrow (f \in \text{space } M \rightarrow A \wedge (\forall a \in A. f$ 
 $- ' \{a\} \cap \text{space } M \in \text{sets } M))$ 
by (intro measurable_count_space_eq_countable countableI_type)

```

lemma measurable_compose_countable:

```

assumes  $f: \bigwedge i::'i::\text{countable}. (\lambda x. f\ i\ x) \in \text{measurable } M\ N$  and  $g: g \in \text{measurable}$ 
 $M (\text{count\_space } \text{UNIV})$ 
shows  $(\lambda x. f\ (g\ x)\ x) \in \text{measurable } M\ N$ 
by (rule measurable_compose_countable'[OF assms]) auto

```

lemma measurable_count_space_const:

```

 $(\lambda x. c) \in \text{measurable } M (\text{count\_space } \text{UNIV})$ 
by (simp add: measurable_const)

```

lemma measurable_count_space:

```

 $f \in \text{measurable } (\text{count\_space } A) (\text{count\_space } \text{UNIV})$ 
by simp

```

lemma measurable_compose_rev:

```

assumes  $f: f \in \text{measurable } L\ N$  and  $g: g \in \text{measurable } M\ L$ 
shows  $(\lambda x. f\ (g\ x)) \in \text{measurable } M\ N$ 
using measurable_compose[OF g f] .

```

lemma measurable_empty_iff:

```

 $\text{space } N = \{\} \implies f \in \text{measurable } M\ N \longleftrightarrow \text{space } M = \{\}$ 
by (auto simp add: measurable_def Pi_iff)

```


Extend measure

definition *extend_measure* :: 'a set $\Rightarrow$ 'b set $\Rightarrow$ ('b $\Rightarrow$ 'a set) $\Rightarrow$ ('b $\Rightarrow$ ennreal)
 $\Rightarrow$ 'a measure

where

extend_measure Ω I G μ =
 (if $(\exists \mu'. (\forall i \in I. \mu' (G i) = \mu i) \wedge \text{measure\_space } \Omega (\text{sigma\_sets } \Omega (G'I)) \mu')$
 $\wedge \neg (\forall i \in I. \mu i = 0)$
 then *measure_of* $\Omega (G'I)$ (SOME $\mu'. (\forall i \in I. \mu' (G i) = \mu i) \wedge \text{measure\_space}$
 $\Omega (\text{sigma\_sets } \Omega (G'I)) \mu')$
 else *measure_of* $\Omega (G'I)$ ($\lambda_. 0$))

lemma *space_extend_measure*: $G \text{ ' } I \subseteq \text{Pow } \Omega \Longrightarrow \text{space } (\text{extend\_measure } \Omega \text{ } I \text{ } G \text{ } \mu) = \Omega$

unfolding *extend_measure_def* **by** *simp*

lemma *sets_extend_measure*: $G \text{ ' } I \subseteq \text{Pow } \Omega \Longrightarrow \text{sets } (\text{extend\_measure } \Omega \text{ } I \text{ } G \text{ } \mu) = \text{sigma\_sets } \Omega (G'I)$

unfolding *extend_measure_def* **by** *simp*

lemma *emeasure_extend_measure*:

assumes $M: M = \text{extend\_measure } \Omega \text{ } I \text{ } G \text{ } \mu$

and $\text{eq}: \bigwedge i. i \in I \Longrightarrow \mu' (G i) = \mu i$

and $\text{ms}: G \text{ ' } I \subseteq \text{Pow } \Omega \text{ positive } (\text{sets } M) \mu' \text{ countably\_additive } (\text{sets } M) \mu'$

and $i \in I$

shows $\text{emeasure } M (G i) = \mu i$

proof *cases*

assume *: $(\forall i \in I. \mu i = 0)$

with M **have** $M\_eq: M = \text{measure\_of } \Omega (G'I) (\lambda_. 0)$

by (*simp add: extend_measure_def*)

from *measure_space_0[OF ms(1)] ms <i ∈ I>*

have $\text{emeasure } M (G i) = 0$

by (*intro emeasure_measure_of[OF M_eq]*) (*auto simp add: M measure_space_def sets_extend_measure*)

with $\langle i \in I \rangle$ * **show** ?thesis

by *simp*

next

define P **where** $P \mu' \longleftrightarrow (\forall i \in I. \mu' (G i) = \mu i) \wedge \text{measure\_space } \Omega (\text{sigma\_sets } \Omega (G'I)) \mu'$

assume $\neg (\forall i \in I. \mu i = 0)$

moreover

have $\text{measure\_space } (\text{space } M) (\text{sets } M) \mu'$

using ms **unfolding** *measure_space_def* **by** *auto standard*

with ms eq **have** $\exists \mu'. P \mu'$

unfolding P_def

by (*intro exI[of _ μ']*) (*auto simp add: M space_extend_measure sets_extend_measure*)

ultimately have $M\_eq: M = \text{measure\_of } \Omega (G'I) (Eps P)$

by (*simp add: M extend_measure_def P_def[symmetric]*)

from $\langle \exists \mu'. P \mu' \rangle$ **have** $P: P (Eps P)$ **by** (*rule someI_ex*)

```

show emeasure M (G i) =  $\mu$  i
proof (subst emeasure_measure_of [OF M_eq])
  have sets_M: sets M = sigma_sets  $\Omega$  (G'I)
    using M_eq ms by (auto simp: sets_extend_measure)
  then show G i  $\in$  sets M using  $\langle i \in I \rangle$  by auto
  show positive (sets M) (Eps P) countably_additive (sets M) (Eps P) Eps P
(G i) =  $\mu$  i
    using P  $\langle i \in I \rangle$  by (auto simp add: sets_M measure_space_def P_def)
qed fact
qed

```

```

lemma emeasure_extend_measure_Pair:
assumes M: M = extend_measure  $\Omega$   $\{(i, j). I\ i\ j\}$  ( $\lambda(i, j). G\ i\ j$ ) ( $\lambda(i, j). \mu\ i\ j$ )
and eq:  $\bigwedge i\ j. I\ i\ j \implies \mu' (G\ i\ j) = \mu\ i\ j$ 
and ms:  $\bigwedge i\ j. I\ i\ j \implies G\ i\ j \in Pow\ \Omega$  positive (sets M)  $\mu'$  countably_additive
(sets M)  $\mu'$ 
and I\ i\ j
shows emeasure M (G i\ j) =  $\mu\ i\ j$ 
using emeasure_extend_measure [OF M __ ms(2,3), of (i,j)] eq ms(1)  $\langle I\ i\ j \rangle$ 
by (auto simp: subset_eq)

```

6.1.3 The smallest σ -algebra regarding a function

definition *vimage_algebra* :: '*a* *set* $\Rightarrow$ (*a* $\Rightarrow$ '*b*) $\Rightarrow$ '*b* *measure* $\Rightarrow$ '*a* *measure* **where**

vimage_algebra *X* *f* *M* = *sigma* *X* $\{f - 'A \cap X \mid A. A \in \text{sets } M\}$

```

lemma space_vimage_algebra [simp]: space (vimage_algebra X f M) = X
unfolding vimage_algebra_def by (rule space_measure_of) auto

```

```

lemma sets_vimage_algebra: sets (vimage_algebra X f M) = sigma_sets X  $\{f - 'A \cap X \mid A. A \in \text{sets } M\}$ 
unfolding vimage_algebra_def by (rule sets_measure_of) auto

```

```

lemma sets_vimage_algebra2:
  f  $\in$  X  $\rightarrow$  space M  $\implies$  sets (vimage_algebra X f M) =  $\{f - 'A \cap X \mid A. A \in \text{sets } M\}$ 
using sigma_sets_vimage_commute [of f X space M sets M]
unfolding sets_vimage_algebra sets.sigma_sets_eq by simp

```

```

lemma sets_vimage_algebra_cong: sets M = sets N  $\implies$  sets (vimage_algebra X
f M) = sets (vimage_algebra X f N)
by (simp add: sets_vimage_algebra)

```

```

lemma vimage_algebra_cong:
assumes X = Y
assumes  $\bigwedge x. x \in Y \implies f\ x = g\ x$ 
assumes sets M = sets N
shows vimage_algebra X f M = vimage_algebra Y g N

```

by (auto simp: vimage_algebra_def assms intro!: arg_cong2[**where** $f = \text{sigma}$])

lemma in_vimage_algebra: $A \in \text{sets } M \implies f - ' A \cap X \in \text{sets } (\text{vimage\_algebra } X f M)$
by (auto simp: vimage_algebra_def)

lemma sets_image_in_sets:
assumes $N: \text{space } N = X$
assumes $f: f \in \text{measurable } N M$
shows $\text{sets } (\text{vimage\_algebra } X f M) \subseteq \text{sets } N$
unfolding sets_vimage_algebra $N[\text{symmetric}]$
by (rule sets.sigma_sets_subset) (auto intro!: measurable_sets f)

lemma measurable_vimage_algebra1: $f \in X \rightarrow \text{space } M \implies f \in \text{measurable } (\text{vimage\_algebra } X f M) M$
unfolding measurable_def **by** (auto intro: in_vimage_algebra)

lemma measurable_vimage_algebra2:
assumes $g: g \in \text{space } N \rightarrow X$ **and** $f: (\lambda x. f (g x)) \in \text{measurable } N M$
shows $g \in \text{measurable } N (\text{vimage\_algebra } X f M)$
unfolding vimage_algebra_def
proof (rule measurable_measure_of)
fix A **assume** $A \in \{f - ' A \cap X \mid A. A \in \text{sets } M\}$
then obtain Y **where** $Y: Y \in \text{sets } M$ **and** $A: A = f - ' Y \cap X$
by auto
then have $g - ' A \cap \text{space } N = (\lambda x. f (g x)) - ' Y \cap \text{space } N$
using g **by** auto
also have $\dots \in \text{sets } N$
using f Y **by** (rule measurable_sets)
finally show $g - ' A \cap \text{space } N \in \text{sets } N$.
qed (insert g , auto)

lemma vimage_algebra_sigma:
assumes $X: X \subseteq \text{Pow } \Omega'$ **and** $f: f \in \Omega \rightarrow \Omega'$
shows $\text{vimage\_algebra } \Omega f (\text{sigma } \Omega' X) = \text{sigma } \Omega \{f - ' A \cap \Omega \mid A. A \in X\}$
(is ?V = ?S)
proof (rule measure_eqI)
have $\Omega: \{f - ' A \cap \Omega \mid A. A \in X\} \subseteq \text{Pow } \Omega$ **by** auto
show $\text{sets } ?V = \text{sets } ?S$
using sigma_sets_vimage_commute[OF f , of X]
by (simp add: space_measure_of_conv f sets_vimage_algebra2 ΩX)
qed (simp add: vimage_algebra_def emeasure_sigma)

lemma vimage_algebra_vimage_algebra_eq:
assumes $*: f \in X \rightarrow Y$ $g \in Y \rightarrow \text{space } M$
shows $\text{vimage\_algebra } X f (\text{vimage\_algebra } Y g M) = \text{vimage\_algebra } X (\lambda x. g (f x)) M$
(is ?VV = ?V)
proof (rule measure_eqI)

```

have (λx. g (f x)) ∈ X → space M ∧ A. A ∩ f -‘ Y ∩ X = A ∩ X
using * by auto
with * show sets ?VV = sets ?V
by (simp add: sets_vimage_algebra2 vimage_comp comp_def flip: ex_simps)
qed (simp add: vimage_algebra_def emeasure_sigma)

```

Restricted Space Sigma Algebra

definition *restrict_space* :: 'a measure ⇒ 'a set ⇒ 'a measure **where**
restrict_space M Ω = *measure_of* (Ω ∩ space M) (((∩) Ω) -‘ sets M) (*emeasure* M)

lemma *space_restrict_space*: *space* (*restrict_space* M Ω) = Ω ∩ space M
using *sets.sets_into_space* **unfolding** *restrict_space_def* **by** (*subst space_measure_of*) *auto*

lemma *space_restrict_space2* [*simp*]: Ω ∈ sets M ⇒ *space* (*restrict_space* M Ω) = Ω
by (*simp add: space_restrict_space sets.sets_into_space*)

lemma *sets_restrict_space*: sets (*restrict_space* M Ω) = ((∩) Ω) -‘ sets M
unfolding *restrict_space_def*

proof (*subst sets_measure_of*)

show (∩) Ω -‘ sets M ⊆ Pow (Ω ∩ space M)

by (*auto dest: sets.sets_into_space*)

have *sigma_sets* (Ω ∩ space M) {((λx. x) -‘ X) ∩ (Ω ∩ space M) | X. X ∈ sets M} =

(λX. X ∩ (Ω ∩ space M)) -‘ sets M

by (*subst sigma_sets_vimage_commute*[*symmetric*, **where** Ω' = space M])

(*auto simp add: sets.sigma_sets_eq*)

moreover have {((λx. x) -‘ X) ∩ (Ω ∩ space M) | X. X ∈ sets M} = (λX. X ∩ (Ω ∩ space M)) -‘ sets M

by *auto*

moreover have (λX. X ∩ (Ω ∩ space M)) -‘ sets M = ((∩) Ω) -‘ sets M

by (*intro image_cong*) (*auto dest: sets.sets_into_space*)

ultimately show *sigma_sets* (Ω ∩ space M) ((∩) Ω -‘ sets M) = (∩) Ω -‘ sets M

by *simp*

qed

lemma *restrict_space_sets_cong*:

A = B ⇒ sets M = sets N ⇒ sets (*restrict_space* M A) = sets (*restrict_space* N B)

by (*auto simp: sets_restrict_space*)

lemma *sets_restrict_space_count_space* :

sets (*restrict_space* (*count_space* A) B) = sets (*count_space* (A ∩ B))

by(*auto simp add: sets_restrict_space*)

lemma *sets_restrict_UNIV*[*simp*]: sets (*restrict_space* M UNIV) = sets M

by (auto simp add: sets_restrict_space)

lemma sets_restrict_restrict_space:

sets (restrict_space (restrict_space M A) B) = sets (restrict_space M (A ∩ B))
 unfolding sets_restrict_space_image_comp by (intro image_cong) auto

lemma sets_restrict_space_iff:

$\Omega \cap \text{space } M \in \text{sets } M \implies A \in \text{sets } (\text{restrict\_space } M \ \Omega) \iff (A \subseteq \Omega \wedge A \in \text{sets } M)$

proof (subst sets_restrict_space, safe)

fix A assume $\Omega \cap \text{space } M \in \text{sets } M$ and A: $A \in \text{sets } M$

then have $(\Omega \cap \text{space } M) \cap A \in \text{sets } M$

by rule

also have $(\Omega \cap \text{space } M) \cap A = \Omega \cap A$

using sets_sets_into_space[OF A] by auto

finally show $\Omega \cap A \in \text{sets } M$

by auto

qed auto

lemma sets_restrict_space_cong: $\text{sets } M = \text{sets } N \implies \text{sets } (\text{restrict\_space } M \ \Omega)$

$= \text{sets } (\text{restrict\_space } N \ \Omega)$

by (simp add: sets_restrict_space)

lemma restrict_space_eq_vimage_algebra:

$\Omega \subseteq \text{space } M \implies \text{sets } (\text{restrict\_space } M \ \Omega) = \text{sets } (\text{vimage\_algebra } \Omega \ (\lambda x. x)$

$M)$

unfolding restrict_space_def

apply (subst sets_measure_of)

apply (auto simp add: image_subset_iff dest: sets_sets_into_space) []

apply (auto simp add: sets_vimage_algebra intro!: arg_cong2[where f=sigma_sets])
 done

lemma sets_Collect_restrict_space_iff:

assumes $S \in \text{sets } M$

shows $\{x \in \text{space } (\text{restrict\_space } M \ S). P \ x\} \in \text{sets } (\text{restrict\_space } M \ S) \iff$

$\{x \in \text{space } M. x \in S \wedge P \ x\} \in \text{sets } M$

proof –

have $\{x \in S. P \ x\} = \{x \in \text{space } M. x \in S \wedge P \ x\}$

using sets_sets_into_space[OF assms] by auto

then show ?thesis

by (subst sets_restrict_space_iff) (auto simp add: space_restrict_space assms)

qed

lemma measurable_restrict_space1:

assumes $f: f \in \text{measurable } M \ N$

shows $f \in \text{measurable } (\text{restrict\_space } M \ \Omega) \ N$

unfolding measurable_def

proof (intro CollectI conjI ballI)

show $sp: f \in \text{space } (\text{restrict\_space } M \ \Omega) \rightarrow \text{space } N$

```

    using measurable_space[OF f] by (auto simp: space_restrict_space)

  fix A assume A ∈ sets N
  have f - ' A ∩ space (restrict_space M Ω) = (f - ' A ∩ space M) ∩ (Ω ∩ space
M)
    by (auto simp: space_restrict_space)
  also have ... ∈ sets (restrict_space M Ω)
    unfolding sets_restrict_space
    using measurable_sets[OF f ‹A ∈ sets N›] by blast
  finally show f - ' A ∩ space (restrict_space M Ω) ∈ sets (restrict_space M Ω)
  .
qed

lemma measurable_restrict_space2_iff:
  f ∈ measurable M (restrict_space N Ω) ⟷ (f ∈ measurable M N ∧ f ∈ space
M → Ω)
proof -
  have ∧A. f ∈ space M → Ω ⟹ f - ' Ω ∩ f - ' A ∩ space M = f - ' A ∩ space
M
    by auto
  then show ?thesis
    by (auto simp: measurable_def space_restrict_space Pi_Int[symmetric] sets_restrict_space)
qed

lemma measurable_restrict_space2:
  f ∈ space M → Ω ⟹ f ∈ measurable M N ⟹ f ∈ measurable M (restrict_space
N Ω)
  by (simp add: measurable_restrict_space2_iff)

lemma measurable_piecewise_restrict:
  assumes I: countable C
    and X: ∧Ω. Ω ∈ C ⟹ Ω ∩ space M ∈ sets M space M ⊆ ⋃ C
    and f: ∧Ω. Ω ∈ C ⟹ f ∈ measurable (restrict_space M Ω) N
  shows f ∈ measurable M N
proof (rule measurableI)
  fix x assume x ∈ space M
  with X obtain Ω where Ω ∈ C x ∈ Ω x ∈ space M by auto
  then show f x ∈ space N
    by (auto simp: space_restrict_space intro: f measurable_space)
next
  fix A assume A: A ∈ sets N
  have f - ' A ∩ space M = (⋃ Ω ∈ C. (f - ' A ∩ (Ω ∩ space M)))
    using X by (auto simp: subset_eq)
  also have ... ∈ sets M
    using measurable_sets[OF f A] X I
    by (intro sets.countable_UN') (auto simp: sets_restrict_space_iff space_restrict_space)
  finally show f - ' A ∩ space M ∈ sets M .
qed

```

lemma *measurable_piecewise_restrict_iff*:

countable C $\implies (\bigwedge \Omega. \Omega \in C \implies \Omega \cap \text{space } M \in \text{sets } M) \implies \text{space } M \subseteq (\bigcup C)$
 $\implies$
 $f \in \text{measurable } M \ N \longleftrightarrow (\forall \Omega \in C. f \in \text{measurable } (\text{restrict\_space } M \ \Omega) \ N)$
by (*auto intro: measurable_piecewise_restrict measurable_restrict_space1*)

lemma *measurable_If_restrict_space_iff*:

$\{x \in \text{space } M. P \ x\} \in \text{sets } M \implies$
 $(\lambda x. \text{if } P \ x \text{ then } f \ x \text{ else } g \ x) \in \text{measurable } M \ N \longleftrightarrow$
 $(f \in \text{measurable } (\text{restrict\_space } M \ \{x. P \ x\}) \ N \wedge g \in \text{measurable } (\text{restrict\_space } M \ \{x. \neg P \ x\}) \ N)$
by (*subst measurable_piecewise_restrict_iff[where C={x. P x}, {x. ¬ P x}]*)
(auto simp: Int_def sets.sets_Collect_neg space_restrict_space conj_commute[of
_ x \in space M for x]
cong: measurable_cong')

lemma *measurable_If*:

$f \in \text{measurable } M \ M' \implies g \in \text{measurable } M \ M' \implies \{x \in \text{space } M. P \ x\} \in \text{sets } M \implies$
 $(\lambda x. \text{if } P \ x \text{ then } f \ x \text{ else } g \ x) \in \text{measurable } M \ M'$
unfolding *measurable_If_restrict_space_iff* **by** (*auto intro: measurable_restrict_space1*)

lemma *measurable_If_set*:

assumes *measure*: $f \in \text{measurable } M \ M' \ g \in \text{measurable } M \ M'$
assumes *P*: $A \cap \text{space } M \in \text{sets } M$
shows $(\lambda x. \text{if } x \in A \text{ then } f \ x \text{ else } g \ x) \in \text{measurable } M \ M'$
proof (*rule measurable_If[OF measure]*)
have $\{x \in \text{space } M. x \in A\} = A \cap \text{space } M$ **by** *auto*
thus $\{x \in \text{space } M. x \in A\} \in \text{sets } M$ **using** $\langle A \cap \text{space } M \in \text{sets } M \rangle$ **by** *auto*
qed

lemma *measurable_restrict_space_iff*:

$\Omega \cap \text{space } M \in \text{sets } M \implies c \in \text{space } N \implies$
 $f \in \text{measurable } (\text{restrict\_space } M \ \Omega) \ N \longleftrightarrow (\lambda x. \text{if } x \in \Omega \text{ then } f \ x \text{ else } c) \in \text{measurable } M \ N$
by (*subst measurable_If_restrict_space_iff*)
(simp_all add: Int_def conj_commute measurable_const)

lemma *restrict_space_singleton*: $\{x\} \in \text{sets } M \implies \text{sets } (\text{restrict\_space } M \ \{x\})$

$= \text{sets } (\text{count\_space } \{x\})$
using *sets_restrict_space_iff[of {x} M]*
by (*auto simp add: sets_restrict_space_iff dest!: subset_singletonD*)

lemma *measurable_restrict_countable*:

assumes *X[intro]*: *countable X*
assumes *sets[simp]*: $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$
assumes *space[simp]*: $\bigwedge x. x \in X \implies f \ x \in \text{space } N$
assumes *f*: $f \in \text{measurable } (\text{restrict\_space } M \ (- \ X)) \ N$
shows $f \in \text{measurable } M \ N$

```

using f sets.countable[OF sets X]
by (intro measurable_piecewise_restrict[where M=M and C={- X}  $\cup$  (( $\lambda x.$ 
{x}) ' X))])
  (auto simp: Diff_Int_distrib2 Compl_eq_Diff_UNIV Int_insert_left sets.Diff
restrict_space_singleton
simp del: sets_count_space cong: measurable_cong_sets)

```

```

lemma measurable_discrete_difference:
  assumes f:  $f \in \text{measurable } M \ N$ 
  assumes X: countable X  $\wedge x. x \in X \implies \{x\} \in \text{sets } M \wedge x. x \in X \implies g \ x \in$ 
space N
  assumes eq:  $\wedge x. x \in \text{space } M \implies x \notin X \implies f \ x = g \ x$ 
  shows  $g \in \text{measurable } M \ N$ 
  by (rule measurable_restrict_countable[OF X])
    (auto simp: eq[symmetric] space_restrict_space cong: measurable_cong' intro:
f measurable_restrict_space1)

```

```

lemma measurable_count_space_extend:  $A \subseteq B \implies f \in \text{space } M \rightarrow A \implies f \in$ 
 $M \rightarrow_M \text{count\_space } B \implies f \in M \rightarrow_M \text{count\_space } A$ 
  by (auto simp: measurable_def)

```

end

6.2 Measurability Prover

```

theory Measurable
imports
  Sigma_Algebra
  HOL-Library.Order_Continuity
begin

```

```

lemma (in algebra) sets_Collect_finite_All:
  assumes  $\wedge i. i \in S \implies \{x \in \Omega. P \ i \ x\} \in M \text{ finite } S$ 
  shows  $\{x \in \Omega. \forall i \in S. P \ i \ x\} \in M$ 
proof -
  have  $\{x \in \Omega. \forall i \in S. P \ i \ x\} = (\text{if } S = \{\} \text{ then } \Omega \text{ else } \bigcap i \in S. \{x \in \Omega. P \ i \ x\})$ 
  by auto
  with assms show ?thesis by (auto intro!: sets_Collect_finite_All')
qed

```

```

abbreviation pred M P  $\equiv P \in \text{measurable } M \ (\text{count\_space } (\text{UNIV}::\text{bool set}))$ 

```

```

lemma pred_def:  $\text{pred } M \ P \longleftrightarrow \{x \in \text{space } M. P \ x\} \in \text{sets } M$ 
proof
  assume pred M P
  then have  $P - \{ \text{True} \} \cap \text{space } M \in \text{sets } M$ 
  by (auto simp: measurable_count_space_eq2)
  also have  $P - \{ \text{True} \} \cap \text{space } M = \{x \in \text{space } M. P \ x\}$  by auto

```



```

    finally show  $\{x \in \text{space } M. P\ x\} \in \text{sets } M$  .
next
  assume  $P: \{x \in \text{space } M. P\ x\} \in \text{sets } M$ 
  moreover
  { fix  $X$ 
    have  $X \in \text{Pow } (\text{UNIV} :: \text{bool set})$  by simp
    then have  $P - 'X \cap \text{space } M = \{x \in \text{space } M. ((X = \{\text{True}\} \longrightarrow P\ x) \wedge (X = \{\text{False}\} \longrightarrow \neg P\ x) \wedge X \neq \{\})\}$ 
    =  $\{False\} \longrightarrow \neg P\ x) \wedge X \neq \{\}$ 
    unfolding UNIV_bool Pow_insert Pow_empty by auto
    then have  $P - 'X \cap \text{space } M \in \text{sets } M$ 
    by (auto intro!: sets.sets_Collect_neg sets.sets_Collect_imp sets.sets_Collect_conj
sets.sets_Collect_const P) }
    then show  $\text{pred } M\ P$ 
    by (auto simp: measurable_def)
qed

```

```

lemma pred_sets1:  $\{x \in \text{space } M. P\ x\} \in \text{sets } M \implies f \in \text{measurable } N\ M \implies$ 
 $\text{pred } N\ (\lambda x. P\ (f\ x))$ 
  by (rule measurable_compose[where  $f=f$  and  $N=M$ ]) (auto simp: pred_def)

```

```

lemma pred_sets2:  $A \in \text{sets } N \implies f \in \text{measurable } M\ N \implies \text{pred } M\ (\lambda x. f\ x \in A)$ 
  by (rule measurable_compose[where  $f=f$  and  $N=N$ ]) (auto simp: pred_def
Int_def[symmetric])

```

ML_file $\langle \text{measurable.ML} \rangle$

```

attribute_setup measurable = <
  Scan.lift (
    (Args.add >> K true || Args.del >> K false || Scan.succeed true) --
    Scan.optional (Args.parens (
      Scan.optional (Args.$$$ raw >> K true) false --
      Scan.optional (Args.$$$ generic >> K Measurable.Generic) Measurable.Concrete))
    (false, Measurable.Concrete) >>
    Measurable.measurable_thm_attr)
> declaration of measurability theorems

```

```

attribute_setup measurable_dest = Measurable.dest_thm_attr
  add dest rule to measurability prover

```

```

attribute_setup measurable_cong = Measurable.cong_thm_attr
  add congruence rules to measurability prover

```

```

method_setup measurable = < Scan.lift (Scan.succeed (METHOD o Measurable.measurable_tac)) >
  measurability prover

```

```

simproc_setup measurable ( $A \in \text{sets } M \mid f \in \text{measurable } M\ N$ ) = <K Measurable.simproc>

```

```

setup <
  Global_Theory.add_thms_dynamic (binding <measurable>, Measurable.get_all)
>

```

```

declare
  pred_sets1[measurable_dest]
  pred_sets2[measurable_dest]
  sets.sets_into_space[measurable_dest]

```

```

declare
  sets.top[measurable]
  sets.empty_sets[measurable (raw)]
  sets.Un[measurable (raw)]
  sets.Diff[measurable (raw)]

```

```

declare
  measurable_count_space[measurable (raw)]
  measurable_ident[measurable (raw)]
  measurable_id[measurable (raw)]
  measurable_const[measurable (raw)]
  measurable_If[measurable (raw)]
  measurable_comp[measurable (raw)]
  measurable_sets[measurable (raw)]

```

```

declare measurable_cong_sets[measurable_cong]
declare sets_restrict_space_cong[measurable_cong]
declare sets_restrict_UNIV[measurable_cong]

```

```

lemma predE[measurable (raw)]:
  pred M P  $\implies \{x \in \text{space } M. P\ x\} \in \text{sets } M$ 
unfolding pred_def .

```

```

lemma pred_intros_imp'[measurable (raw)]:
  (K  $\implies \text{pred } M (\lambda x. P\ x)$ )  $\implies \text{pred } M (\lambda x. K \longrightarrow P\ x)$ 
by (cases K) auto

```

```

lemma pred_intros_conj1'[measurable (raw)]:
  (K  $\implies \text{pred } M (\lambda x. P\ x)$ )  $\implies \text{pred } M (\lambda x. K \wedge P\ x)$ 
by (cases K) auto

```

```

lemma pred_intros_conj2'[measurable (raw)]:
  (K  $\implies \text{pred } M (\lambda x. P\ x)$ )  $\implies \text{pred } M (\lambda x. P\ x \wedge K)$ 
by (cases K) auto

```

```

lemma pred_intros_disj1'[measurable (raw)]:
  ( $\neg K \implies \text{pred } M (\lambda x. P\ x)$ )  $\implies \text{pred } M (\lambda x. K \vee P\ x)$ 
by (cases K) auto

```

lemma *pred_intros_disj2* [measurable (raw)]:
 $(\neg K \implies \text{pred } M (\lambda x. P x)) \implies \text{pred } M (\lambda x. P x \vee K)$
by (cases K) auto

lemma *pred_intros_logic* [measurable (raw)]:
 $\text{pred } M (\lambda x. x \in \text{space } M)$
 $\text{pred } M (\lambda x. P x) \implies \text{pred } M (\lambda x. \neg P x)$
 $\text{pred } M (\lambda x. Q x) \implies \text{pred } M (\lambda x. P x) \implies \text{pred } M (\lambda x. Q x \wedge P x)$
 $\text{pred } M (\lambda x. Q x) \implies \text{pred } M (\lambda x. P x) \implies \text{pred } M (\lambda x. Q x \longrightarrow P x)$
 $\text{pred } M (\lambda x. Q x) \implies \text{pred } M (\lambda x. P x) \implies \text{pred } M (\lambda x. Q x \vee P x)$
 $\text{pred } M (\lambda x. Q x) \implies \text{pred } M (\lambda x. P x) \implies \text{pred } M (\lambda x. Q x = P x)$
 $\text{pred } M (\lambda x. f x \in \text{UNIV})$
 $\text{pred } M (\lambda x. f x \in \{\})$
 $\text{pred } M (\lambda x. P' (f x) x) \implies \text{pred } M (\lambda x. f x \in \{y. P' y x\})$
 $\text{pred } M (\lambda x. f x \in (B x)) \implies \text{pred } M (\lambda x. f x \in - (B x))$
 $\text{pred } M (\lambda x. f x \in (A x)) \implies \text{pred } M (\lambda x. f x \in (B x)) \implies \text{pred } M (\lambda x. f x \in (A x) - (B x))$
 $\text{pred } M (\lambda x. f x \in (A x)) \implies \text{pred } M (\lambda x. f x \in (B x)) \implies \text{pred } M (\lambda x. f x \in (A x) \cap (B x))$
 $\text{pred } M (\lambda x. f x \in (A x)) \implies \text{pred } M (\lambda x. f x \in (B x)) \implies \text{pred } M (\lambda x. f x \in (A x) \cup (B x))$
 $\text{pred } M (\lambda x. g (f x) \in (X x)) \implies \text{pred } M (\lambda x. f x \in (g x) - ' (X x))$
by (auto simp: iff_conv_conj_imp pred_def)

lemma *pred_intros_countable* [measurable (raw)]:
fixes $P :: 'a \Rightarrow 'i :: \text{countable} \Rightarrow \text{bool}$
shows
 $(\bigwedge i. \text{pred } M (\lambda x. P x i)) \implies \text{pred } M (\lambda x. \forall i. P x i)$
 $(\bigwedge i. \text{pred } M (\lambda x. P x i)) \implies \text{pred } M (\lambda x. \exists i. P x i)$
by (auto intro!: sets.sets_Collect_countable_All sets.sets_Collect_countable_Ex simp: pred_def)

lemma *pred_intros_countable_bounded* [measurable (raw)]:
fixes $X :: 'i :: \text{countable set}$
shows
 $(\bigwedge i. i \in X \implies \text{pred } M (\lambda x. x \in N x i)) \implies \text{pred } M (\lambda x. x \in (\bigcap_{i \in X}. N x i))$
 $(\bigwedge i. i \in X \implies \text{pred } M (\lambda x. x \in N x i)) \implies \text{pred } M (\lambda x. x \in (\bigcup_{i \in X}. N x i))$
 $(\bigwedge i. i \in X \implies \text{pred } M (\lambda x. P x i)) \implies \text{pred } M (\lambda x. \forall i \in X. P x i)$
 $(\bigwedge i. i \in X \implies \text{pred } M (\lambda x. P x i)) \implies \text{pred } M (\lambda x. \exists i \in X. P x i)$
by simp_all (auto simp: Bex_def Ball_def)

lemma *pred_intros_finite* [measurable (raw)]:
 $\text{finite } I \implies (\bigwedge i. i \in I \implies \text{pred } M (\lambda x. x \in N x i)) \implies \text{pred } M (\lambda x. x \in (\bigcap_{i \in I}. N x i))$
 $\text{finite } I \implies (\bigwedge i. i \in I \implies \text{pred } M (\lambda x. x \in N x i)) \implies \text{pred } M (\lambda x. x \in (\bigcup_{i \in I}. N x i))$
 $\text{finite } I \implies (\bigwedge i. i \in I \implies \text{pred } M (\lambda x. P x i)) \implies \text{pred } M (\lambda x. \forall i \in I. P x i)$
 $\text{finite } I \implies (\bigwedge i. i \in I \implies \text{pred } M (\lambda x. P x i)) \implies \text{pred } M (\lambda x. \exists i \in I. P x i)$
by (auto intro!: sets.sets_Collect_finite_Ex sets.sets_Collect_finite_All simp:

iff_conv_conj_imp pred_def)

lemma *countable_Un_Int*[*measurable (raw)*]:

$(\bigwedge i :: 'i :: \text{countable}. i \in I \implies N\ i \in \text{sets } M) \implies (\bigcup_{i \in I}. N\ i) \in \text{sets } M$
 $I \neq \{\} \implies (\bigwedge i :: 'i :: \text{countable}. i \in I \implies N\ i \in \text{sets } M) \implies (\bigcap_{i \in I}. N\ i) \in \text{sets } M$
by *auto*

declare

finite_UN[*measurable (raw)*]
finite_INT[*measurable (raw)*]

lemma *sets_Int_pred*[*measurable (raw)*]:

assumes *space*: $A \cap B \subseteq \text{space } M$ **and** [*measurable*]: *pred* *M* $(\lambda x. x \in A)$ *pred* *M* $(\lambda x. x \in B)$
shows $A \cap B \in \text{sets } M$
proof –
have $\{x \in \text{space } M. x \in A \cap B\} \in \text{sets } M$ **by** *auto*
also have $\{x \in \text{space } M. x \in A \cap B\} = A \cap B$
using *space* **by** *auto*
finally show *?thesis* .
qed

lemma [*measurable (raw generic)*]:

assumes *f*: $f \in \text{measurable } M\ N$ **and** *c*: $c \in \text{space } N \implies \{c\} \in \text{sets } N$
shows *pred_eq_const1*: *pred* *M* $(\lambda x. f\ x = c)$
and *pred_eq_const2*: *pred* *M* $(\lambda x. c = f\ x)$
proof –
show *pred* *M* $(\lambda x. f\ x = c)$
proof *cases*
assume $c \in \text{space } N$
with *measurable_sets*[*OF* *f* *c*] **show** *?thesis*
by (*auto simp: Int_def conj_commute pred_def*)
next
assume $c \notin \text{space } N$
with *f*[*THEN measurable_space*] **have** $\{x \in \text{space } M. f\ x = c\} = \{\}$ **by** *auto*
then show *?thesis* **by** (*auto simp: pred_def cong: conj_cong*)
qed
then show *pred* *M* $(\lambda x. c = f\ x)$
by (*simp add: eq_commute*)
qed

lemma *pred_count_space_const1*[*measurable (raw)*]:

$f \in \text{measurable } M\ (\text{count\_space } \text{UNIV}) \implies \text{Measurable.pred } M\ (\lambda x. f\ x = c)$
by (*intro pred_eq_const1*[**where** $N = \text{count\_space } \text{UNIV}$]) (*auto*)

lemma *pred_count_space_const2*[*measurable (raw)*]:

$f \in \text{measurable } M\ (\text{count\_space } \text{UNIV}) \implies \text{Measurable.pred } M\ (\lambda x. c = f\ x)$
by (*intro pred_eq_const2*[**where** $N = \text{count\_space } \text{UNIV}$]) (*auto*)

lemma *pred_le_const*[*measurable (raw generic)*]:
assumes $f: f \in \text{measurable } M \ N$ **and** $c: \{.. \ c\} \in \text{sets } N$ **shows** $\text{pred } M \ (\lambda x. f \ x \leq c)$
using *measurable_sets*[*OF f c*]
by (*auto simp: Int_def conj_commute eq_commute pred_def*)

lemma *pred_const_le*[*measurable (raw generic)*]:
assumes $f: f \in \text{measurable } M \ N$ **and** $c: \{c \ ..\} \in \text{sets } N$ **shows** $\text{pred } M \ (\lambda x. c \leq f \ x)$
using *measurable_sets*[*OF f c*]
by (*auto simp: Int_def conj_commute eq_commute pred_def*)

lemma *pred_less_const*[*measurable (raw generic)*]:
assumes $f: f \in \text{measurable } M \ N$ **and** $c: \{.. < c\} \in \text{sets } N$ **shows** $\text{pred } M \ (\lambda x. f \ x < c)$
using *measurable_sets*[*OF f c*]
by (*auto simp: Int_def conj_commute eq_commute pred_def*)

lemma *pred_const_less*[*measurable (raw generic)*]:
assumes $f: f \in \text{measurable } M \ N$ **and** $c: \{c < ..\} \in \text{sets } N$ **shows** $\text{pred } M \ (\lambda x. c < f \ x)$
using *measurable_sets*[*OF f c*]
by (*auto simp: Int_def conj_commute eq_commute pred_def*)

declare
sets.Int[*measurable (raw)*]

lemma *pred_in_If*[*measurable (raw)*]:
 $(P \implies \text{pred } M \ (\lambda x. x \in A \ x)) \implies (\neg P \implies \text{pred } M \ (\lambda x. x \in B \ x)) \implies$
 $\text{pred } M \ (\lambda x. x \in (\text{if } P \text{ then } A \ x \text{ else } B \ x))$
by *auto*

lemma *sets_range*[*measurable_dest*]:
 $A \ ' \ I \subseteq \text{sets } M \implies i \in I \implies A \ i \in \text{sets } M$
by *auto*

lemma *pred_sets_range*[*measurable_dest*]:
 $A \ ' \ I \subseteq \text{sets } N \implies i \in I \implies f \in \text{measurable } M \ N \implies \text{pred } M \ (\lambda x. f \ x \in A \ i)$
using *pred_sets2*[*OF sets_range*] **by** *auto*

lemma *sets_All*[*measurable_dest*]:
 $\forall i. A \ i \in \text{sets } (M \ i) \implies A \ i \in \text{sets } (M \ i)$
by *auto*

lemma *pred_sets_All*[*measurable_dest*]:
 $\forall i. A \ i \in \text{sets } (N \ i) \implies f \in \text{measurable } M \ (N \ i) \implies \text{pred } M \ (\lambda x. f \ x \in A \ i)$
using *pred_sets2*[*OF sets_All, of A N f*] **by** *auto*

lemma *sets_Ball*[*measurable_dest*]:

$\forall i \in I. A\ i \in \text{sets}\ (M\ i) \implies i \in I \implies A\ i \in \text{sets}\ (M\ i)$
by *auto*

lemma *pred_sets_Ball*[*measurable_dest*]:

$\forall i \in I. A\ i \in \text{sets}\ (N\ i) \implies i \in I \implies f \in \text{measurable}\ M\ (N\ i) \implies \text{pred}\ M\ (\lambda x. f\ x \in A\ i)$
using *pred_sets2*[*OF sets_Ball, of _ _ _ f*] **by** *auto*

lemma *measurable_finite*[*measurable (raw)*]:

fixes *S* :: 'a $\Rightarrow$ nat set
assumes [*measurable*]: $\bigwedge i. \{x \in \text{space}\ M. i \in S\ x\} \in \text{sets}\ M$
shows $\text{pred}\ M\ (\lambda x. \text{finite}\ (S\ x))$
unfolding *finite_nat_set_iff_bounded* **by** (*simp add: Ball_def*)

lemma *measurable_Least*[*measurable*]:

assumes [*measurable*]: $(\bigwedge i :: \text{nat}. (\lambda x. P\ i\ x) \in \text{measurable}\ M\ (\text{count\_space}\ UNIV))$
shows $(\lambda x. \text{LEAST}\ i. P\ i\ x) \in \text{measurable}\ M\ (\text{count\_space}\ UNIV)$
unfolding *measurable_def* **by** (*safe intro!: sets_Least simp_all*)

lemma *measurable_Max_nat*[*measurable (raw)*]:

fixes *P* :: nat $\Rightarrow$ 'a $\Rightarrow$ bool
assumes [*measurable*]: $\bigwedge i. \text{Measurable.pred}\ M\ (P\ i)$
shows $(\lambda x. \text{Max}\ \{i. P\ i\ x\}) \in \text{measurable}\ M\ (\text{count\_space}\ UNIV)$
unfolding *measurable_count_space_eq2_countable*
proof *safe*
fix *n*

{ **fix** *x* **assume** $\forall i. \exists n \geq i. P\ n\ x$
then have *infinite* $\{i. P\ i\ x\}$
unfolding *infinite_nat_iff_unbounded_le* **by** *auto*
then have $\text{Max}\ \{i. P\ i\ x\} = \text{the}\ \text{None}$
by (*rule Max.infinite*) }
note *1 = this*

{ **fix** *x i j* **assume** $P\ i\ x \forall n \geq j. \neg P\ n\ x$
then have *finite* $\{i. P\ i\ x\}$
by (*auto simp: subset_eq not_le[symmetric] finite_nat_iff_bounded*)
with $\langle P\ i\ x \rangle$ **have** $P\ (\text{Max}\ \{i. P\ i\ x\})\ x \leq \text{Max}\ \{i. P\ i\ x\}$ *finite* $\{i. P\ i\ x\}$
using *Max_in[of {i. P i x}]* **by** *auto* }
note *2 = this*

have $(\lambda x. \text{Max}\ \{i. P\ i\ x\}) - ' \{n\} \cap \text{space}\ M = \{x \in \text{space}\ M. \text{Max}\ \{i. P\ i\ x\} = n\}$
by *auto*
also have ... =
 $\{x \in \text{space}\ M. \text{if } (\forall i. \exists n \geq i. P\ n\ x) \text{ then the None} = n \text{ else if } (\exists i. P\ i\ x) \text{ then } P\ n\ x \wedge (\forall i > n. \neg P\ i\ x)\}$

```

      else Max {} = n}
    by (intro arg_cong[where f=Collect] ext conj_cong)
      (auto simp add: 1 2 not_le[symmetric] intro!: Max_eqI)
  also have ... ∈ sets M
    by measurable
  finally show (λx. Max {i. P i x}) - ' {n} ∩ space M ∈ sets M .
qed simp

lemma measurable_Min_nat[measurable (raw)]:
  fixes P :: nat ⇒ 'a ⇒ bool
  assumes [measurable]: ∧i. Measurable.pred M (P i)
  shows (λx. Min {i. P i x}) ∈ measurable M (count_space UNIV)
  unfolding measurable_count_space_eq2_countable
proof safe
  fix n

  { fix x assume ∀i. ∃n≥i. P n x
    then have infinite {i. P i x}
      unfolding infinite_nat_iff_unbounded_le by auto
    then have Min {i. P i x} = the None
      by (rule Min.infinite) }
  note 1 = this

  { fix x i j assume P i x ∀n≥j. ¬ P n x
    then have finite {i. P i x}
      by (auto simp: subset_eq not_le[symmetric] finite_nat_iff_bounded)
    with ⟨P i x⟩ have P (Min {i. P i x}) x Min {i. P i x} ≤ i finite {i. P i x}
      using Min_in[of {i. P i x}] by auto }
  note 2 = this

  have (λx. Min {i. P i x}) - ' {n} ∩ space M = {x∈space M. Min {i. P i x} =
n}
    by auto
  also have ... =
    {x∈space M. if (∀i. ∃n≥i. P n x) then the None = n else
      if (∃i. P i x) then P n x ∧ (∀i<n. ¬ P i x)
      else Min {} = n}
    by (intro arg_cong[where f=Collect] ext conj_cong)
      (auto simp add: 1 2 not_le[symmetric] intro!: Min_eqI)
  also have ... ∈ sets M
    by measurable
  finally show (λx. Min {i. P i x}) - ' {n} ∩ space M ∈ sets M .
qed simp

lemma measurable_count_space_insert[measurable (raw)]:
  s ∈ S ⇒ A ∈ sets (count_space S) ⇒ insert s A ∈ sets (count_space S)
  by simp

lemma sets_UNIV [measurable (raw)]: A ∈ sets (count_space UNIV)

```

by *simp*

```

lemma measurable_card[measurable]:
  fixes  $S :: 'a \Rightarrow \text{nat set}$ 
  assumes [measurable]:  $\bigwedge i. \{x \in \text{space } M. i \in S\ x\} \in \text{sets } M$ 
  shows  $(\lambda x. \text{card } (S\ x)) \in \text{measurable } M$  (count_space UNIV)
  unfolding measurable_count_space_eq2_countable
proof safe
  fix  $n$  show  $(\lambda x. \text{card } (S\ x)) - ' \{n\} \cap \text{space } M \in \text{sets } M$ 
  proof (cases  $n$ )
  case 0
  then have  $(\lambda x. \text{card } (S\ x)) - ' \{n\} \cap \text{space } M = \{x \in \text{space } M. \text{infinite } (S\ x) \vee$ 
 $(\forall i. i \notin S\ x)\}$ 
  by auto
  also have  $\dots \in \text{sets } M$ 
  by measurable
  finally show ?thesis .
next
  case (Suc  $i$ )
  then have  $(\lambda x. \text{card } (S\ x)) - ' \{n\} \cap \text{space } M =$ 
 $(\bigcup F \in \{A \in \{A. \text{finite } A\}. \text{card } A = n\}. \{x \in \text{space } M. (\forall i. i \in S\ x \longleftrightarrow i \in$ 
 $F)\})$ 
  unfolding set_eq_iff[symmetric] Collect_bex_eq[symmetric] by (auto intro:
  card_ge_0_finite)
  also have  $\dots \in \text{sets } M$ 
  by (intro sets.countable_UN' countable_Collect countable_Collect_finite)
  auto
  finally show ?thesis .
qed
qed rule

```

```

lemma measurable_pred_countable[measurable (raw)]:
  assumes countable  $X$ 
  shows
     $(\bigwedge i. i \in X \implies \text{Measurable.pred } M (\lambda x. P\ x\ i)) \implies \text{Measurable.pred } M (\lambda x.$ 
 $\forall i \in X. P\ x\ i)$ 
     $(\bigwedge i. i \in X \implies \text{Measurable.pred } M (\lambda x. P\ x\ i)) \implies \text{Measurable.pred } M (\lambda x.$ 
 $\exists i \in X. P\ x\ i)$ 
  unfolding pred_def
  by (auto intro!: sets.sets_Collect_countable_All' sets.sets_Collect_countable_Ex'
  assms)

```

6.2.1 Measurability for (co)inductive predicates

```

lemma measurable_bot[measurable]:  $\text{bot} \in \text{measurable } M$  (count_space UNIV)
  by (simp add: bot_fun_def)

```

```

lemma measurable_top[measurable]:  $\text{top} \in \text{measurable } M$  (count_space UNIV)
  by (simp add: top_fun_def)

```



```

lemma measurable_SUP[measurable]:
  fixes  $F :: 'i \Rightarrow 'a \Rightarrow 'b :: \{\text{complete\_lattice, countable}\}$ 
  assumes [simp]: countable  $I$ 
  assumes [measurable]:  $\bigwedge i. i \in I \implies F\ i \in \text{measurable } M\ (\text{count\_space } UNIV)$ 
  shows  $(\lambda x. \text{SUP } i \in I. F\ i\ x) \in \text{measurable } M\ (\text{count\_space } UNIV)$ 
  unfolding measurable_count_space_eq2_countable
proof (safe intro!: UNIV_I)
  fix  $a$ 
  have  $(\lambda x. \text{SUP } i \in I. F\ i\ x) - ' \{a\} \cap \text{space } M =$ 
     $\{x \in \text{space } M. (\forall i \in I. F\ i\ x \leq a) \wedge (\forall b. (\forall i \in I. F\ i\ x \leq b) \longrightarrow a \leq b)\}$ 
  unfolding SUP_le_iff[symmetric] by auto
  also have  $\dots \in \text{sets } M$ 
    by measurable
  finally show  $(\lambda x. \text{SUP } i \in I. F\ i\ x) - ' \{a\} \cap \text{space } M \in \text{sets } M$  .
qed

```

```

lemma measurable_INF[measurable]:
  fixes  $F :: 'i \Rightarrow 'a \Rightarrow 'b :: \{\text{complete\_lattice, countable}\}$ 
  assumes [simp]: countable  $I$ 
  assumes [measurable]:  $\bigwedge i. i \in I \implies F\ i \in \text{measurable } M\ (\text{count\_space } UNIV)$ 
  shows  $(\lambda x. \text{INF } i \in I. F\ i\ x) \in \text{measurable } M\ (\text{count\_space } UNIV)$ 
  unfolding measurable_count_space_eq2_countable
proof (safe intro!: UNIV_I)
  fix  $a$ 
  have  $(\lambda x. \text{INF } i \in I. F\ i\ x) - ' \{a\} \cap \text{space } M =$ 
     $\{x \in \text{space } M. (\forall i \in I. a \leq F\ i\ x) \wedge (\forall b. (\forall i \in I. b \leq F\ i\ x) \longrightarrow b \leq a)\}$ 
  unfolding le_INF_iff[symmetric] by auto
  also have  $\dots \in \text{sets } M$ 
    by measurable
  finally show  $(\lambda x. \text{INF } i \in I. F\ i\ x) - ' \{a\} \cap \text{space } M \in \text{sets } M$  .
qed

```

```

lemma measurable_lfp_coinduct[consumes 1, case_names continuity step]:
  fixes  $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b :: \{\text{complete\_lattice, countable}\})$ 
  assumes  $P\ M$ 
  assumes  $F$ : sup_continuous  $F$ 
  assumes *:  $\bigwedge M\ A. P\ M \implies (\bigwedge N. P\ N \implies A \in \text{measurable } N\ (\text{count\_space } UNIV)) \implies F\ A \in \text{measurable } M\ (\text{count\_space } UNIV)$ 
  shows  $\text{lfp } F \in \text{measurable } M\ (\text{count\_space } UNIV)$ 
proof -
  { fix  $i$  from  $\langle P\ M \rangle$  have  $((F \rightsquigarrow i) \text{ bot}) \in \text{measurable } M\ (\text{count\_space } UNIV)$ 
    by (induct  $i$  arbitrary:  $M$ ) (auto intro!: *) }
  then have  $(\lambda x. \text{SUP } i. (F \rightsquigarrow i) \text{ bot } x) \in \text{measurable } M\ (\text{count\_space } UNIV)$ 
    by measurable
  also have  $(\lambda x. \text{SUP } i. (F \rightsquigarrow i) \text{ bot } x) = \text{lfp } F$ 
    by (subst sup_continuous_lfp) (auto intro:  $F$  simp: image_comp)
  finally show ?thesis .
qed

```

lemma *measurable_lfp*:
fixes $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b::\{\text{complete\_lattice}, \text{countable}\})$
assumes $F: \text{sup\_continuous } F$
assumes $*$: $\bigwedge A. A \in \text{measurable } M (\text{count\_space } UNIV) \Rightarrow F A \in \text{measurable } M (\text{count\_space } UNIV)$
shows $\text{lfp } F \in \text{measurable } M (\text{count\_space } UNIV)$
by (*coinduction rule: measurable_lfp_coinduct[OF _ F]*) (*blast intro: **)

lemma *measurable_gfp_coinduct*[*consumes 1, case_names continuity step*]:
fixes $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b::\{\text{complete\_lattice}, \text{countable}\})$
assumes $P M$
assumes $F: \text{inf\_continuous } F$
assumes $*$: $\bigwedge M A. P M \Rightarrow (\bigwedge N. P N \Rightarrow A \in \text{measurable } N (\text{count\_space } UNIV)) \Rightarrow F A \in \text{measurable } M (\text{count\_space } UNIV)$
shows $\text{gfp } F \in \text{measurable } M (\text{count\_space } UNIV)$
proof –
{ fix i **from** $\langle P M \rangle$ **have** $((F \rightsquigarrow i) \text{ top}) \in \text{measurable } M (\text{count\_space } UNIV)$
by (*induct i arbitrary: M*) (*auto intro!: **) **}**
then have $(\lambda x. \text{INF } i. (F \rightsquigarrow i) \text{ top } x) \in \text{measurable } M (\text{count\_space } UNIV)$
by *measurable*
also have $(\lambda x. \text{INF } i. (F \rightsquigarrow i) \text{ top } x) = \text{gfp } F$
by (*subst inf_continuous_gfp*) (*auto intro: F simp: image_comp*)
finally show *?thesis* .
qed

lemma *measurable_gfp*:
fixes $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b::\{\text{complete\_lattice}, \text{countable}\})$
assumes $F: \text{inf\_continuous } F$
assumes $*$: $\bigwedge A. A \in \text{measurable } M (\text{count\_space } UNIV) \Rightarrow F A \in \text{measurable } M (\text{count\_space } UNIV)$
shows $\text{gfp } F \in \text{measurable } M (\text{count\_space } UNIV)$
by (*coinduction rule: measurable_gfp_coinduct[OF _ F]*) (*blast intro: **)

lemma *measurable_lfp2_coinduct*[*consumes 1, case_names continuity step*]:
fixes $F :: ('a \Rightarrow 'c \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c \Rightarrow 'b::\{\text{complete\_lattice}, \text{countable}\})$
assumes $P M s$
assumes $F: \text{sup\_continuous } F$
assumes $*$: $\bigwedge M A s. P M s \Rightarrow (\bigwedge N t. P N t \Rightarrow A t \in \text{measurable } N (\text{count\_space } UNIV)) \Rightarrow F A s \in \text{measurable } M (\text{count\_space } UNIV)$
shows $\text{lfp } F s \in \text{measurable } M (\text{count\_space } UNIV)$
proof –
{ fix i **from** $\langle P M s \rangle$ **have** $(\lambda x. (F \rightsquigarrow i) \text{ bot } s x) \in \text{measurable } M (\text{count\_space } UNIV)$
by (*induct i arbitrary: M s*) (*auto intro!: **) **}**
then have $(\lambda x. \text{SUP } i. (F \rightsquigarrow i) \text{ bot } s x) \in \text{measurable } M (\text{count\_space } UNIV)$
by *measurable*
also have $(\lambda x. \text{SUP } i. (F \rightsquigarrow i) \text{ bot } s x) = \text{lfp } F s$
by (*subst sup_continuous_lfp*) (*auto simp: F simp: image_comp*)

finally show ?thesis .
qed

lemma measurable_gfp2_coinduct[consumes 1, case_names continuity step]:
fixes $F :: ('a \Rightarrow 'c \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c \Rightarrow 'b :: \{\text{complete\_lattice, countable}\})$
assumes $P \ M \ s$
assumes $F: \text{inf\_continuous } F$
assumes $*: \bigwedge M \ A \ s. P \ M \ s \implies (\bigwedge N \ t. P \ N \ t \implies A \ t \in \text{measurable } N \ (\text{count\_space } UNIV)) \implies F \ A \ s \in \text{measurable } M \ (\text{count\_space } UNIV)$
shows $\text{gfp } F \ s \in \text{measurable } M \ (\text{count\_space } UNIV)$
proof –
{ fix i from } $\langle P \ M \ s \rangle$ **have $(\lambda x. (F \ \sim i) \ \text{top } s \ x) \in \text{measurable } M \ (\text{count\_space } UNIV)$
by (induct i arbitrary: $M \ s$) (auto intro!: *) }
then have $(\lambda x. \text{INF } i. (F \ \sim i) \ \text{top } s \ x) \in \text{measurable } M \ (\text{count\_space } UNIV)$
by measurable
also have $(\lambda x. \text{INF } i. (F \ \sim i) \ \text{top } s \ x) = \text{gfp } F \ s$
by (subst inf_continuous_gfp) (auto simp: $F \ \text{simp: image\_comp}$)
finally show ?thesis .
qed**

lemma measurable_enat_coinduct:
fixes $f :: 'a \Rightarrow \text{enat}$
assumes $R \ f$
assumes $*: \bigwedge f. R \ f \implies \exists g \ h \ i \ P. R \ g \wedge f = (\lambda x. \text{if } P \ x \ \text{then } h \ x \ \text{else } e\text{Suc } (g \ (i \ x))) \wedge$
 $\text{Measurable.pred } M \ P \wedge$
 $i \in \text{measurable } M \ M \wedge$
 $h \in \text{measurable } M \ (\text{count\_space } UNIV)$
shows $f \in \text{measurable } M \ (\text{count\_space } UNIV)$
proof (simp add: measurable_count_space_eq2_countable, rule)
fix $a :: \text{enat}$
have $f - \{a\} \cap \text{space } M = \{x \in \text{space } M. f \ x = a\}$
by auto
{ fix i :: nat
from } $\langle R \ f \rangle$ **have $\text{Measurable.pred } M \ (\lambda x. f \ x = \text{enat } i)$
proof (induction i arbitrary: f)
case 0
from $*(OF \ \text{this})$ **obtain** $g \ h \ i \ P$
where $f: f = (\lambda x. \text{if } P \ x \ \text{then } h \ x \ \text{else } e\text{Suc } (g \ (i \ x)))$ **and**
 $[\text{measurable}]: \text{Measurable.pred } M \ P \ i \in \text{measurable } M \ M \ h \in \text{measurable } M \ (\text{count\_space } UNIV)$
by auto
have $\text{Measurable.pred } M \ (\lambda x. P \ x \wedge h \ x = 0)$
by measurable
also have $(\lambda x. P \ x \wedge h \ x = 0) = (\lambda x. f \ x = \text{enat } 0)$
by (auto simp: $f \ \text{zero\_enat\_def[symmetric]}$)
finally show ?case .
next**

```

    case (Suc n)
    from *[OF Suc.prem] obtain g h i P
    where f: f = (λx. if P x then h x else eSuc (g (i x))) and R g and
      M[measurable]: Measurable.pred M P i ∈ measurable M M h ∈ measurable
M (count_space UNIV)
    by auto
    have (λx. f x = enat (Suc n)) =
      (λx. (P x → h x = enat (Suc n)) ∧ (¬ P x → g (i x) = enat n))
    by (auto simp: f zero_enat_def[symmetric] eSuc_enat[symmetric])
    also have Measurable.pred M ...
    by (intro pred_intros_logic measurable_compose[OF M(2)] Suc ‹R g›)
measurable
    finally show ?case .
qed
then have f - ' {enat i} ∩ space M ∈ sets M
    by (simp add: pred_def Int_def conj_commute) }
note fin = this
show f - ' {a} ∩ space M ∈ sets M
proof (cases a)
  case infinity
  then have f - ' {a} ∩ space M = space M - (⋃ n. f - ' {enat n} ∩ space M)
    by auto
  also have ... ∈ sets M
    by (intro sets.Diff sets.top sets.Un sets.countable_UN) (auto intro!: fin)
  finally show ?thesis .
qed (simp add: fin)
qed

```

lemma measurable_THE:

```

fixes P :: 'a ⇒ 'b ⇒ bool
assumes [measurable]: ∧i. Measurable.pred M (P i)
assumes I[simp]: countable I ∧ i x. x ∈ space M ⇒ P i x ⇒ i ∈ I
assumes unique: ∧i i j. x ∈ space M ⇒ P i x ⇒ P j x ⇒ i = j
shows (λx. THE i. P i x) ∈ measurable M (count_space UNIV)
unfolding measurable_def

```

proof safe

```

fix X
define f where f x = (THE i. P i x) for x
define undef where undef = (THE i::'a. False)
{ fix i x assume x ∈ space M P i x then have f x = i
  unfolding f_def using unique by auto }
note f_eq = this
{ fix x assume x ∈ space M ∀ i ∈ I. ¬ P i x
  then have ∧i. ¬ P i x
    using I(2)[of x] by auto
  then have f x = undef
    by (auto simp: undef_def f_def) }
then have f - ' X ∩ space M = (⋃ i ∈ I ∩ X. {x ∈ space M. P i x}) ∪
  (if undef ∈ X then space M - (⋃ i ∈ I. {x ∈ space M. P i x}) else {})

```

```

    by (auto dest: f_eq)
  also have ... ∈ sets M
    by (auto intro!: sets.Diff sets.countable_UN')
  finally show f - ' X ∩ space M ∈ sets M .
qed simp

```

```

lemma measurable_Ex1[measurable (raw)]:
  assumes [simp]: countable I and [measurable]:  $\bigwedge i. i \in I \implies \text{Measurable.pred } M$ 
  (P i)
  shows  $\text{Measurable.pred } M (\lambda x. \exists ! i \in I. P i x)$ 
  unfolding bex1_def by measurable

```

```

lemma measurable_Sup_nat[measurable (raw)]:
  fixes F :: 'a  $\Rightarrow$  nat set
  assumes [measurable]:  $\bigwedge i. \text{Measurable.pred } M (\lambda x. i \in F x)$ 
  shows  $(\lambda x. \text{Sup } (F x)) \in M \rightarrow_M \text{count\_space UNIV}$ 
proof (clarsimp simp add: measurable_count_space_eq2_countable)
  fix a
  have F_empty_iff:  $F x = \{\} \longleftrightarrow (\forall i. i \notin F x)$  for x
  by auto
  have  $\text{Measurable.pred } M (\lambda x. \text{if finite } (F x) \text{ then if } F x = \{\} \text{ then } a = 0$ 
     $\text{else } a \in F x \wedge (\forall j. j \in F x \longrightarrow j \leq a) \text{ else } a = \text{the None})$ 
  unfolding finite_nat_set_iff_bounded Ball_def F_empty_iff by measurable
  moreover have  $(\lambda x. \text{Sup } (F x)) - ' \{a\} \cap \text{space } M =$ 
     $\{x \in \text{space } M. \text{if finite } (F x) \text{ then if } F x = \{\} \text{ then } a = 0$ 
     $\text{else } a \in F x \wedge (\forall j. j \in F x \longrightarrow j \leq a) \text{ else } a = \text{the None}\}$ 
  by (intro set_eqI)
    (auto simp: Sup_nat_def Max.infinite intro!: Max_in Max_eqI)
  ultimately show  $(\lambda x. \text{Sup } (F x)) - ' \{a\} \cap \text{space } M \in \text{sets } M$ 
  by auto
qed

```

```

lemma measurable_if_split[measurable (raw)]:
  (c  $\implies \text{Measurable.pred } M f$ )  $\implies (\neg c \implies \text{Measurable.pred } M g) \implies$ 
   $\text{Measurable.pred } M (\text{if } c \text{ then } f \text{ else } g)$ 
  by simp

```

```

lemma pred_restrict_space:
  assumes S ∈ sets M
  shows  $\text{Measurable.pred } (\text{restrict\_space } M S) P \longleftrightarrow \text{Measurable.pred } M (\lambda x. x$ 
   $\in S \wedge P x)$ 
  unfolding pred_def sets_Collect_restrict_space_iff[OF assms] ..

```

```

lemma measurable_predpow[measurable]:
  assumes  $\text{Measurable.pred } M T$ 
  assumes  $\bigwedge Q. \text{Measurable.pred } M Q \implies \text{Measurable.pred } M (R Q)$ 
  shows  $\text{Measurable.pred } M ((R \rightsquigarrow n) T)$ 
  by (induct n) (auto intro: assms)

```

```

lemma measurable_compose_countable_restrict:
  assumes P: countable {i. P i}
  and f:  $f \in M \rightarrow_M \text{count\_space UNIV}$ 
  and Q:  $\bigwedge i. P\ i \implies \text{pred } M\ (Q\ i)$ 
  shows  $\text{pred } M\ (\lambda x. P\ (f\ x) \wedge Q\ (f\ x)\ x)$ 
proof –
  have P_f:  $\{x \in \text{space } M. P\ (f\ x)\} \in \text{sets } M$ 
  unfolding pred_def[symmetric] by (rule measurable_compose[OF f]) simp
  have pred_restrict_space  $M\ \{x \in \text{space } M. P\ (f\ x)\}\ (\lambda x. Q\ (f\ x)\ x)$ 
  proof (rule measurable_compose_countable'[where g=f, OF _ _ P])
    show  $f \in \text{restrict\_space } M\ \{x \in \text{space } M. P\ (f\ x)\} \rightarrow_M \text{count\_space } \{i. P\ i\}$ 
    by (rule measurable_count_space_extend[OF subset_UNIV])
    (auto simp: space_restrict_space intro!: measurable_restrict_space1 f)
  qed (auto intro!: measurable_restrict_space1 Q)
  then show ?thesis
    unfolding pred_restrict_space[OF P_f] by (simp cong: measurable_cong)
qed

lemma measurable_limsup [measurable (raw)]:
  assumes [measurable]:  $\bigwedge n. A\ n \in \text{sets } M$ 
  shows limsup A  $\in \text{sets } M$ 
by (subst limsup_INF_SUP, auto)

lemma measurable_liminf [measurable (raw)]:
  assumes [measurable]:  $\bigwedge n. A\ n \in \text{sets } M$ 
  shows liminf A  $\in \text{sets } M$ 
by (subst liminf_SUP_INF, auto)

lemma measurable_case_enat[measurable (raw)]:
  assumes f:  $f \in M \rightarrow_M \text{count\_space UNIV}$  and g:  $\bigwedge i. g\ i \in M \rightarrow_M N$  and h:
   $h \in M \rightarrow_M N$ 
  shows  $(\lambda x. \text{case } f\ x\ \text{of } \text{enat } i \Rightarrow g\ i\ x \mid \infty \Rightarrow h\ x) \in M \rightarrow_M N$ 
  apply (rule measurable_compose_countable[OF _ f])
  subgoal for i
    by (cases i) (auto intro: g h)
  done

hide_const (open) pred

end

```

6.3 Measure Spaces

```

theory Measure_Space
imports
  Measurable HOL-Library.Extended_Nonnegative_Real
begin

```

6.3.1 Relate extended reals and the indicator function

```

lemma suminf_cmult_indicator:
  fixes f :: nat  $\Rightarrow$  ennreal
  assumes disjoint_family A x  $\in$  A i
  shows  $(\sum n. f\ n * indicator\ (A\ n)\ x) = f\ i$ 
proof -
  have **:  $\bigwedge n. f\ n * indicator\ (A\ n)\ x = (if\ n = i\ then\ f\ n\ else\ 0 :: ennreal)$ 
    using  $\langle x \in A\ i \rangle$  assms unfolding disjoint_family_on_def indicator_def by
  auto
  then have  $\bigwedge n. (\sum j < n. f\ j * indicator\ (A\ j)\ x) = (if\ i < n\ then\ f\ i\ else\ 0 ::$ 
ennreal)
    by (auto simp: sum.If_cases)
  moreover have  $(SUP\ n. if\ i < n\ then\ f\ i\ else\ 0) = (f\ i :: ennreal)$ 
proof (rule SUP_eqI)
  fix y :: ennreal assume  $\bigwedge n. n \in UNIV \implies (if\ i < n\ then\ f\ i\ else\ 0) \leq y$ 
  from this[of Suc i] show  $f\ i \leq y$  by auto
qed (insert assms, simp)
ultimately show ?thesis using assms
  by (subst suminf_eq_SUP) (auto simp: indicator_def)
qed

```

```

lemma suminf_indicator:
  assumes disjoint_family A
  shows  $(\sum n. indicator\ (A\ n)\ x :: ennreal) = indicator\ (\bigcup i. A\ i)\ x$ 
proof cases
  assume *:  $x \in (\bigcup i. A\ i)$ 
  then obtain i where  $x \in A\ i$  by auto
  from suminf_cmult_indicator[OF assms(1), OF  $\langle x \in A\ i \rangle$ , of  $\lambda k. 1$ ]
  show ?thesis using * by simp
qed simp

```

```

lemma sum_indicator_disjoint_family:
  fixes f :: 'd  $\Rightarrow$  'e::semiring_1
  assumes d: disjoint_family_on A P and  $x \in A\ j$  and finite P and  $j \in P$ 
  shows  $(\sum i \in P. f\ i * indicator\ (A\ i)\ x) = f\ j$ 
proof -
  have  $P \cap \{i. x \in A\ i\} = \{j\}$ 
    using d  $\langle x \in A\ j \rangle \langle j \in P \rangle$  unfolding disjoint_family_on_def
    by auto
  with  $\langle finite\ P \rangle$  show ?thesis
    by (simp add: indicator_def)
qed

```

The type for emeasure spaces is already defined in *HOL–Analysis.Sigma_Algebra*, as it is also used to represent sigma algebras (with an arbitrary emeasure).

6.3.2 Extend binary sets

```

lemma LIMSEQ_binaryset:

```

```

assumes  $f: f \{ \} = 0$ 
shows  $(\lambda n. \sum i < n. f (binaryset\ A\ B\ i)) \longrightarrow f\ A + f\ B$ 
proof -
  have  $(\lambda n. \sum i < Suc\ (Suc\ n). f (binaryset\ A\ B\ i)) = (\lambda n. f\ A + f\ B)$ 
  proof
    fix  $n$ 
    show  $(\sum i < Suc\ (Suc\ n). f (binaryset\ A\ B\ i)) = f\ A + f\ B$ 
    by  $(induct\ n)\ (auto\ simp\ add: binaryset\_def\ f)$ 
  qed
moreover
have  $\dots \longrightarrow f\ A + f\ B$  by  $(rule\ tendsto\_const)$ 
ultimately
have  $(\lambda n. \sum i < Suc\ (Suc\ n). f (binaryset\ A\ B\ i)) \longrightarrow f\ A + f\ B$ 
by  $metis$ 
hence  $(\lambda n. \sum i < n+2. f (binaryset\ A\ B\ i)) \longrightarrow f\ A + f\ B$ 
by  $simp$ 
thus  $?thesis$  by  $(rule\ LIMSEQ\_offset\ [where\ k=2])$ 
qed

```

```

lemma  $binaryset\_sums$ :
assumes  $f: f \{ \} = 0$ 
shows  $(\lambda n. f (binaryset\ A\ B\ n))\ sums\ (f\ A + f\ B)$ 
by  $(simp\ add: sums\_def\ LIMSEQ\_binaryset\ [where\ f=f,\ OF\ f]\ atLeast0LessThan)$ 

```

```

lemma  $suminf\_binaryset\_eq$ :
fixes  $f :: 'a\ set \Rightarrow 'b::\{comm\_monoid\_add,\ t2\_space\}$ 
shows  $f \{ \} = 0 \implies (\sum n. f (binaryset\ A\ B\ n)) = f\ A + f\ B$ 
by  $(metis\ binaryset\_sums\ sums\_unique)$ 

```

6.3.3 Properties of a premeasure μ

The definitions for *positive* and *countably_additive* should be here, by they are necessary to define 'a *measure* in *HOL-Analysis.Sigma_Algebra*.

```

definition  $subadditive$  where
   $subadditive\ M\ f \longleftrightarrow (\forall x \in M. \forall y \in M. x \cap y = \{ \} \longrightarrow f\ (x \cup y) \leq f\ x + f\ y)$ 

```

```

lemma  $subadditiveD$ :  $subadditive\ M\ f \implies x \cap y = \{ \} \implies x \in M \implies y \in M \implies$ 
 $f\ (x \cup y) \leq f\ x + f\ y$ 
by  $(auto\ simp\ add: subadditive\_def)$ 

```

```

definition  $countably\_subadditive$  where
   $countably\_subadditive\ M\ f \longleftrightarrow$ 
 $(\forall A. range\ A \subseteq M \longrightarrow disjoint\_family\ A \longrightarrow (\bigcup i. A\ i) \in M \longrightarrow (f\ (\bigcup i. A\ i) \leq (\sum i. f\ (A\ i))))$ 

```

```

lemma  $(in\ ring\_of\_sets)\ countably\_subadditive\_subadditive$ :
fixes  $f :: 'a\ set \Rightarrow ennreal$ 
assumes  $f$ :  $positive\ M\ f$  and  $cs$ :  $countably\_subadditive\ M\ f$ 
shows  $subadditive\ M\ f$ 

```



```

proof (auto simp add: subadditive_def)
  fix x y
  assume x: x ∈ M and y: y ∈ M and x ∩ y = {}
  hence disjoint_family (binaryset x y)
    by (auto simp add: disjoint_family_on_def binaryset_def)
  hence range (binaryset x y) ⊆ M ⟶
    (⋃ i. binaryset x y i) ∈ M ⟶
    f (⋃ i. binaryset x y i) ≤ (∑ n. f (binaryset x y n))
  using cs by (auto simp add: countably_subadditive_def)
  hence {x,y,{} } ⊆ M ⟶ x ∪ y ∈ M ⟶
    f (x ∪ y) ≤ (∑ n. f (binaryset x y n))
  by (simp add: range_binaryset_eq UN_binaryset_eq)
  thus f (x ∪ y) ≤ f x + f y using f x y
  by (auto simp add: Un_o_def suminf_binaryset_eq positive_def)
qed

```

definition additive **where**

additive M μ ⟷ (∀ x ∈ M. ∀ y ∈ M. x ∩ y = {} ⟶ μ (x ∪ y) = μ x + μ y)

definition increasing **where**

increasing M μ ⟷ (∀ x ∈ M. ∀ y ∈ M. x ⊆ y ⟶ μ x ≤ μ y)

lemma positiveD1: positive M f ⟹ f {} = 0 **by** (auto simp: positive_def)

lemma positiveD_empty:

positive M f ⟹ f {} = 0

by (auto simp add: positive_def)

lemma additiveD:

additive M f ⟹ x ∩ y = {} ⟹ x ∈ M ⟹ y ∈ M ⟹ f (x ∪ y) = f x + f y

by (auto simp add: additive_def)

lemma increasingD:

increasing M f ⟹ x ⊆ y ⟹ x ∈ M ⟹ y ∈ M ⟹ f x ≤ f y

by (auto simp add: increasing_def)

lemma countably_additiveI[case_names countably]:

(⋀ A. range A ⊆ M ⟹ disjoint_family A ⟹ (⋃ i. A i) ∈ M ⟹ (∑ i. f (A i)) = f (⋃ i. A i))

⟹ countably_additive M f

by (simp add: countably_additive_def)

lemma (in ring_of_sets) disjointed_additive:

assumes f: positive M f additive M f **and** A: range A ⊆ M incseq A

shows (∑ i ≤ n. f (disjointed A i)) = f (A n)

proof (induct n)

case (Suc n)

then have (∑ i ≤ Suc n. f (disjointed A i)) = f (A n) + f (disjointed A (Suc n))

by simp

```

also have ... = f (A n  $\cup$  disjointed A (Suc n))
  using A by (subst f(2)[THEN additiveD]) (auto simp: disjointed_mono)
also have A n  $\cup$  disjointed A (Suc n) = A (Suc n)
  using <incseq A> by (auto dest: incseq_SucD simp: disjointed_mono)
finally show ?case .
qed simp

```

```

lemma (in ring_of_sets) additive_sum:
  fixes A:: 'a  $\Rightarrow$  'a set
  assumes f: positive M f and ad: additive M f and finite S
    and A: A `S  $\subseteq$  M
    and disj: disjoint_family_on A S
  shows ( $\sum_{i \in S}. f (A i)$ ) = f ( $\bigcup_{i \in S}. A i$ )
  using <finite S> disj A
proof induct
  case empty show ?case using f by (simp add: positive_def)
next
  case (insert s S)
  then have A s  $\cap$  ( $\bigcup_{i \in S}. A i$ ) = {}
    by (auto simp add: disjoint_family_on_def neq_iff)
  moreover
  have A s  $\in$  M using insert by blast
  moreover have ( $\bigcup_{i \in S}. A i$ )  $\in$  M
    using insert <finite S> by auto
  ultimately have f (A s  $\cup$  ( $\bigcup_{i \in S}. A i$ )) = f (A s) + f( $\bigcup_{i \in S}. A i$ )
    using ad UNION_in_sets A by (auto simp add: additive_def)
  with insert show ?case using ad disjoint_family_on_mono[of S insert s S A]
    by (auto simp add: additive_def subset_insertI)
qed

```

```

lemma (in ring_of_sets) additive_increasing:
  fixes f :: 'a set  $\Rightarrow$  ennreal
  assumes posf: positive M f and addf: additive M f
  shows increasing M f
proof (auto simp add: increasing_def)
  fix x y
  assume xy: x  $\in$  M y  $\in$  M x  $\subseteq$  y
  then have y - x  $\in$  M by auto
  then have f x + 0  $\leq$  f x + f (y-x) by (intro add_left_mono zero_le)
  also have ... = f (x  $\cup$  (y-x)) using addf
    by (auto simp add: additive_def) (metis Diff_disjoint Un_Diff_cancel Diff
xy(1,2))
  also have ... = f y
    by (metis Un_Diff_cancel Un_absorb1 xy(3))
  finally show f x  $\leq$  f y by simp
qed

```

```

lemma (in ring_of_sets) subadditive:
  fixes f :: 'a set  $\Rightarrow$  ennreal

```

```

    assumes  $f$ : positive  $M$   $f$  additive  $M$   $f$  and  $A$ :  $A \cdot S \subseteq M$  and  $S$ : finite  $S$ 
    shows  $f$  ( $\bigcup i \in S. A \ i$ )  $\leq$  ( $\sum i \in S. f \ (A \ i)$ )
  using  $S \ A$ 
  proof (induct  $S$ )
    case empty thus ?case using  $f$  by (auto simp: positive_def)
  next
    case (insert  $x \ F$ )
    hence in_M:  $A \ x \in M$  ( $\bigcup i \in F. A \ i$ )  $\in M$  ( $\bigcup i \in F. A \ i$ )  $- A \ x \in M$  using  $A$ 
  by force+
    have subs: ( $\bigcup i \in F. A \ i$ )  $- A \ x \subseteq$  ( $\bigcup i \in F. A \ i$ ) by auto
    have ( $\bigcup i \in (\text{insert } x \ F). A \ i$ )  $= A \ x \cup ((\bigcup i \in F. A \ i) - A \ x)$  by auto
    hence  $f$  ( $\bigcup i \in (\text{insert } x \ F). A \ i$ )  $= f \ (A \ x \cup ((\bigcup i \in F. A \ i) - A \ x))$ 
      by simp
    also have  $\dots = f \ (A \ x) + f \ ((\bigcup i \in F. A \ i) - A \ x)$ 
      using  $f(2)$  by (rule additiveD) (insert in_M, auto)
    also have  $\dots \leq f \ (A \ x) + f \ (\bigcup i \in F. A \ i)$ 
      using additive_increasing[OF  $f$ ] in_M subs by (auto simp: increasing_def
intro: add_left_mono)
    also have  $\dots \leq f \ (A \ x) + (\sum i \in F. f \ (A \ i))$  using insert by (auto intro:
add_left_mono)
    finally show  $f$  ( $\bigcup i \in (\text{insert } x \ F). A \ i$ )  $\leq (\sum i \in (\text{insert } x \ F). f \ (A \ i))$  using
insert by simp
  qed

```

```

lemma (in ring_of_sets) countably_additive_additive:
  fixes  $f$  :: 'a set  $\Rightarrow$  ennreal
  assumes posf: positive  $M$   $f$  and ca: countably_additive  $M$   $f$ 
  shows additive  $M$   $f$ 
  proof (auto simp add: additive_def)
    fix  $x \ y$ 
    assume  $x$ :  $x \in M$  and  $y$ :  $y \in M$  and  $x \cap y = \{\}$ 
    hence disjoint_family (binaryset  $x \ y$ )
      by (auto simp add: disjoint_family_on_def binaryset_def)
    hence range (binaryset  $x \ y$ )  $\subseteq M \longrightarrow$ 
      ( $\bigcup i. \text{binaryset } x \ y \ i$ )  $\in M \longrightarrow$ 
       $f \ (\bigcup i. \text{binaryset } x \ y \ i) = (\sum n. f \ (\text{binaryset } x \ y \ n))$ 
      using ca
    by (simp add: countably_additive_def)
    hence  $\{x, y, \{\}\} \subseteq M \longrightarrow x \cup y \in M \longrightarrow$ 
       $f \ (x \cup y) = (\sum n. f \ (\text{binaryset } x \ y \ n))$ 
    by (simp add: range_binaryset_eq UN_binaryset_eq)
    thus  $f \ (x \cup y) = f \ x + f \ y$  using posf  $x \ y$ 
    by (auto simp add: Un_suminf_binaryset_eq positive_def)
  qed

```

```

lemma (in algebra) increasing_additive_bound:
  fixes  $A$ :: nat  $\Rightarrow$  'a set and  $f$  :: 'a set  $\Rightarrow$  ennreal
  assumes  $f$ : positive  $M$   $f$  and ad: additive  $M$   $f$ 
    and inc: increasing  $M$   $f$ 

```

```

    and A: range A ⊆ M
    and disj: disjoint_family A
  shows (∑ i. f (A i)) ≤ f Ω
proof (safe intro!: suminf_le_const)
  fix N
  note disj_N = disjoint_family_on_mono[OF _ disj, of {.. $N$ }]
  have (∑ i<N. f (A i)) = f (⋃ i∈{.. $N$ }. A i)
    using A by (intro additive_sum [OF f ad _]) (auto simp: disj_N)
  also have ... ≤ f Ω using space_closed A
    by (intro increasingD[OF inc] finite_UN) auto
  finally show (∑ i<N. f (A i)) ≤ f Ω by simp
qed (insert f A, auto simp: positive_def)

lemma (in ring_of_sets) countably_additiveI_finite:
  fixes μ :: 'a set ⇒ ennreal
  assumes finite Ω positive M μ additive M μ
  shows countably_additive M μ
proof (rule countably_additiveI)
  fix F :: nat ⇒ 'a set assume F: range F ⊆ M (⋃ i. F i) ∈ M and disj:
    disjoint_family F

  have ∀ i∈{i. F i ≠ {}}. ∃ x. x ∈ F i by auto
  from bchoice[OF this] obtain f where f: ⋀ i. F i ≠ {} ⇒ f i ∈ F i by auto

  have inj_f: inj_on f {i. F i ≠ {}}
  proof (rule inj_onI, simp)
    fix i j a b assume *: f i = f j F i ≠ {} F j ≠ {}
    then have f i ∈ F i f j ∈ F j using f by force+
    with disj * show i = j by (auto simp: disjoint_family_on_def)
  qed
  have finite (⋃ i. F i)
    by (metis F(2) asms(1) infinite_super_sets_into_space)

  have F_subset: {i. μ (F i) ≠ 0} ⊆ {i. F i ≠ {}}
    by (auto simp: positiveD_empty[OF ⟨positive M μ⟩])
  moreover have fin_not_empty: finite {i. F i ≠ {}}
  proof (rule finite_imageD)
    from f have f'{i. F i ≠ {}} ⊆ (⋃ i. F i) by auto
    then show finite (f'{i. F i ≠ {}})
      by (rule finite_subset) fact
  qed fact
  ultimately have fin_not_0: finite {i. μ (F i) ≠ 0}
    by (rule finite_subset)

  have disj_not_empty: disjoint_family_on F {i. F i ≠ {}}
    using disj by (auto simp: disjoint_family_on_def)

  from fin_not_0 have (∑ i. μ (F i)) = (∑ i | μ (F i) ≠ 0. μ (F i))
    by (rule suminf_finite) auto

```

```

also have ... = ( $\sum i \mid F i \neq \{\}$ .  $\mu (F i)$ )
  using fin_not_empty F_subset by (rule sum.mono_neutral_left) auto
also have ... =  $\mu (\bigcup i \in \{i. F i \neq \{\}\}. F i)$ 
  using ‹positive M  $\mu$ › ‹additive M  $\mu$ › fin_not_empty disj_not_empty F by
(intro additive_sum) auto
also have ... =  $\mu (\bigcup i. F i)$ 
  by (rule arg_cong[where f= $\mu$ ]) auto
finally show ( $\sum i. \mu (F i)$ ) =  $\mu (\bigcup i. F i)$  .
qed

```

lemma (in ring_of_sets) countably_additive_iff_continuous_from_below:

```

fixes f :: 'a set  $\Rightarrow$  ennreal
assumes f: positive M f additive M f
shows countably_additive M f  $\longleftrightarrow$ 
  ( $\forall A. \text{range } A \subseteq M \longrightarrow \text{incseq } A \longrightarrow (\bigcup i. A i) \in M \longrightarrow (\lambda i. f (A i)) \longrightarrow$ 
 $f (\bigcup i. A i)$ )
  unfolding countably_additive_def
proof safe
  assume count_sum:  $\forall A. \text{range } A \subseteq M \longrightarrow \text{disjoint\_family } A \longrightarrow \bigcup (A \text{ ' UNIV})$ 
 $\in M \longrightarrow (\sum i. f (A i)) = f (\bigcup (A \text{ ' UNIV}))$ 
  fix A :: nat  $\Rightarrow$  'a set assume A:  $\text{range } A \subseteq M \text{ incseq } A (\bigcup i. A i) \in M$ 
  then have dA:  $\text{range } (\text{disjointed } A) \subseteq M$  by (auto simp: range_disjointed_sets)
  with count_sum[THEN spec, of disjointed A] A(3)
  have f_UN:  $(\sum i. f (\text{disjointed } A i)) = f (\bigcup i. A i)$ 
    by (auto simp: UN_disjointed_eq disjoint_family_disjointed)
  moreover have  $(\lambda n. (\sum i < n. f (\text{disjointed } A i))) \longrightarrow (\sum i. f (\text{disjointed } A i))$ 
    using f(1)[unfolded positive_def] dA
    by (auto intro!: summable_LIMSEQ)
  from LIMSEQ_Suc[OF this]
  have  $(\lambda n. (\sum i \leq n. f (\text{disjointed } A i))) \longrightarrow (\sum i. f (\text{disjointed } A i))$ 
    unfolding lessThan_Suc_atMost .
  moreover have  $\bigwedge n. (\sum i \leq n. f (\text{disjointed } A i)) = f (A n)$ 
    using disjointed_additive[OF f A(1,2)] .
  ultimately show  $(\lambda i. f (A i)) \longrightarrow f (\bigcup i. A i)$  by simp
next
  assume cont:  $\forall A. \text{range } A \subseteq M \longrightarrow \text{incseq } A \longrightarrow (\bigcup i. A i) \in M \longrightarrow (\lambda i. f (A i)) \longrightarrow f (\bigcup i. A i)$ 
  fix A :: nat  $\Rightarrow$  'a set assume A:  $\text{range } A \subseteq M \text{ disjoint\_family } A (\bigcup i. A i) \in M$ 
  have *:  $(\bigcup n. (\bigcup i < n. A i)) = (\bigcup i. A i)$  by auto
  have  $(\lambda n. f (\bigcup i < n. A i)) \longrightarrow f (\bigcup i. A i)$ 
  proof (unfold *[symmetric], intro cont[rule_format])
    show  $\text{range } (\lambda i. \bigcup i < i. A i) \subseteq M (\bigcup i. \bigcup i < i. A i) \in M$ 
      using A * by auto
  qed (force intro!: incseq_SucI)
  moreover have  $\bigwedge n. f (\bigcup i < n. A i) = (\sum i < n. f (A i))$ 
    using A
  by (intro additive_sum[OF f, of _ A, symmetric])
    (auto intro: disjoint_family_on_mono[where B=UNIV])

```

```

ultimately
have (λi. f (A i)) sums f (⋃ i. A i)
  unfolding sums_def by simp
from sums_unique[OF this]
show (∑ i. f (A i)) = f (⋃ i. A i) by simp
qed

```

```

lemma (in ring_of_sets) continuous_from_above_iff_empty_continuous:
  fixes f :: 'a set ⇒ ennreal
  assumes f: positive M f additive M f
  shows (∀ A. range A ⊆ M ⟶ decseq A ⟶ (⋂ i. A i) ∈ M ⟶ (∀ i. f (A i) ≠
  ∞) ⟶ (λi. f (A i)) ⟶ f (⋂ i. A i))
    ⟷ (∀ A. range A ⊆ M ⟶ decseq A ⟶ (⋂ i. A i) = {} ⟶ (∀ i. f (A i) ≠
  ∞) ⟶ (λi. f (A i)) ⟶ 0)
proof safe
  assume cont: (∀ A. range A ⊆ M ⟶ decseq A ⟶ (⋂ i. A i) ∈ M ⟶ (∀ i. f
  (A i) ≠ ∞) ⟶ (λi. f (A i)) ⟶ f (⋂ i. A i))
  fix A :: nat ⇒ 'a set assume A: range A ⊆ M decseq A (⋂ i. A i) = {} ∀ i. f
  (A i) ≠ ∞
  with cont[THEN spec, of A] show (λi. f (A i)) ⟶ 0
  using ⟨positive M f⟩[unfolded positive_def] by auto
next
  assume cont: ∀ A. range A ⊆ M ⟶ decseq A ⟶ (⋂ i. A i) = {} ⟶ (∀ i. f
  (A i) ≠ ∞) ⟶ (λi. f (A i)) ⟶ 0
  fix A :: nat ⇒ 'a set assume A: range A ⊆ M decseq A (⋂ i. A i) ∈ M ∀ i. f
  (A i) ≠ ∞

```

```

have f_mono: ⋀ a b. a ∈ M ⟹ b ∈ M ⟹ a ⊆ b ⟹ f a ≤ f b
  using additive_increasing[OF f] unfolding increasing_def by simp

```

```

have decseq_fA: decseq (λi. f (A i))
  using A by (auto simp: decseq_def intro!: f_mono)
have decseq: decseq (λi. A i - (⋂ i. A i))
  using A by (auto simp: decseq_def)
then have decseq_f: decseq (λi. f (A i - (⋂ i. A i)))
  using A unfolding decseq_def by (auto intro!: f_mono Diff)
have f (⋂ x. A x) ≤ f (A 0)
  using A by (auto intro!: f_mono)
then have f_Int_fin: f (⋂ x. A x) ≠ ∞
  using A by (auto simp: top_unique)
{ fix i
  have f (A i - (⋂ i. A i)) ≤ f (A i) using A by (auto intro!: f_mono)
  then have f (A i - (⋂ i. A i)) ≠ ∞
    using A by (auto simp: top_unique) }
note f_fin = this
have (λi. f (A i - (⋂ i. A i))) ⟶ 0
proof (intro cont[rule_format, OF decseq_f_fin])
  show range (λi. A i - (⋂ i. A i)) ⊆ M (⋂ i. A i - (⋂ i. A i)) = {}
    using A by auto

```

```

qed
from INF_Lim[OF decseq_f this]
have (INF n. f (A n - ( $\bigcap i. A i$ ))) = 0 .
moreover have (INF n. f ( $\bigcap i. A i$ )) = f ( $\bigcap i. A i$ )
  by auto
ultimately have (INF n. f (A n - ( $\bigcap i. A i$ )) + f ( $\bigcap i. A i$ )) = 0 + f ( $\bigcap i. A i$ )
i)
  using A(4) f_fin f_Int_fin
  by (subst INF_ennreal_add_const) (auto simp: decseq_f)
moreover {
  fix n
  have f (A n - ( $\bigcap i. A i$ )) + f ( $\bigcap i. A i$ ) = f ((A n - ( $\bigcap i. A i$ ))  $\cup$  ( $\bigcap i. A i$ ))
    using A by (subst f(2)[THEN additiveD]) auto
  also have (A n - ( $\bigcap i. A i$ ))  $\cup$  ( $\bigcap i. A i$ ) = A n
    by auto
  finally have f (A n - ( $\bigcap i. A i$ )) + f ( $\bigcap i. A i$ ) = f (A n) . }
ultimately have (INF n. f (A n)) = f ( $\bigcap i. A i$ )
  by simp
with LIMSEQ_INF[OF decseq_f A]
show ( $\lambda i. f (A i)$ )  $\longrightarrow$  f ( $\bigcap i. A i$ ) by simp
qed

```

```

lemma (in ring_of_sets) empty_continuous_imp_continuous_from_below:
  fixes f :: 'a set  $\Rightarrow$  ennreal
  assumes f: positive M f additive M f  $\forall A \in M. f A \neq \infty$ 
  assumes cont:  $\forall A. \text{range } A \subseteq M \longrightarrow \text{decseq } A \longrightarrow (\bigcap i. A i) = \{\} \longrightarrow (\lambda i. f (A i)) \longrightarrow 0$ 
  assumes A:  $\text{range } A \subseteq M \text{ incseq } A (\bigcup i. A i) \in M$ 
  shows ( $\lambda i. f (A i)$ )  $\longrightarrow$  f ( $\bigcup i. A i$ )
proof -
  from A have ( $\lambda i. f ((\bigcup i. A i) - A i)$ )  $\longrightarrow$  0
    by (intro cont[rule_format]) (auto simp: decseq_def incseq_def)
  moreover
  { fix i
    have f (( $\bigcup i. A i$ ) - A i  $\cup$  A i) = f (( $\bigcup i. A i$ ) - A i) + f (A i)
      using A by (intro f(2)[THEN additiveD]) auto
    also have (( $\bigcup i. A i$ ) - A i)  $\cup$  A i = ( $\bigcup i. A i$ )
      by auto
    finally have f (( $\bigcup i. A i$ ) - A i) = f ( $\bigcup i. A i$ ) - f (A i)
      using f(3)[rule_format, of A i] A by (auto simp: ennreal_add_diff_cancel subset_eq) }
  moreover have  $\forall_F i \text{ in sequentially. } f (A i) \leq f (\bigcup i. A i)$ 
    using increasingD[OF additive_increasing[OF f(1, 2)], of A  $\bigcup i. A i$ ] A
    by (auto intro!: always_eventually simp: subset_eq)
  ultimately show ( $\lambda i. f (A i)$ )  $\longrightarrow$  f ( $\bigcup i. A i$ )
    by (auto intro: ennreal_tendsto_const_minus)
qed

```

```

lemma (in ring_of_sets) empty_continuous_imp_countably_additive:

```

```

fixes  $f :: 'a \text{ set} \Rightarrow \text{ennreal}$ 
assumes  $f$ : positive  $M$   $f$  additive  $M$   $f$  and  $\text{fin}$ :  $\forall A \in M. f\ A \neq \infty$ 
assumes  $\text{cont}$ :  $\bigwedge A. \text{range } A \subseteq M \implies \text{decseq } A \implies (\bigcap i. A\ i) = \{\} \implies (\lambda i. f\ (A\ i)) \longrightarrow 0$ 
shows countably_additive  $M$   $f$ 
using countably_additive_iff_continuous_from_below[ $OF\ f$ ]
using empty_continuous_imp_continuous_from_below[ $OF\ f\ \text{fin}$ ]  $\text{cont}$ 
by blast

```

6.3.4 Properties of *emeasure*

```

lemma emeasure_positive: positive (sets  $M$ ) (emeasure  $M$ )
by (cases  $M$ ) (auto simp: sets_def emeasure_def Abs_measure_inverse measure_space_def)

```

```

lemma emeasure_empty[simp, intro]: emeasure  $M\ \{\} = 0$ 
using emeasure_positive[of  $M$ ] by (simp add: positive_def)

```

```

lemma emeasure_single_in_space: emeasure  $M\ \{x\} \neq 0 \implies x \in \text{space } M$ 
using emeasure_notin_sets[of  $\{x\}\ M$ ] by (auto dest: sets.sets_into_space zero_less_iff_neq_zero[THEN2])

```

```

lemma emeasure_countably_additive: countably_additive (sets  $M$ ) (emeasure  $M$ )
by (cases  $M$ ) (auto simp: sets_def emeasure_def Abs_measure_inverse measure_space_def)

```

```

lemma suminf_emeasure:
 $\text{range } A \subseteq \text{sets } M \implies \text{disjoint\_family } A \implies (\sum i. \text{emeasure } M\ (A\ i)) = \text{emeasure } M\ (\bigcup i. A\ i)$ 
using sets.countable_UN[of  $A\ \text{UNIV } M$ ] emeasure_countably_additive[of  $M$ ]
by (simp add: countably_additive_def)

```

```

lemma sums_emeasure:
 $\text{disjoint\_family } F \implies (\bigwedge i. F\ i \in \text{sets } M) \implies (\lambda i. \text{emeasure } M\ (F\ i)) \text{ sums } \text{emeasure } M\ (\bigcup i. F\ i)$ 
unfolding sums_iff by (intro conjI suminf_emeasure) auto

```

```

lemma emeasure_additive: additive (sets  $M$ ) (emeasure  $M$ )
by (metis sets.countably_additive_additive emeasure_positive emeasure_countably_additive)

```

```

lemma plus_emeasure:
 $a \in \text{sets } M \implies b \in \text{sets } M \implies a \cap b = \{\} \implies \text{emeasure } M\ a + \text{emeasure } M\ b = \text{emeasure } M\ (a \cup b)$ 
using additiveD[ $OF\ \text{emeasure\_additive}$ ] ..

```

```

lemma emeasure_Un:
 $A \in \text{sets } M \implies B \in \text{sets } M \implies \text{emeasure } M\ (A \cup B) = \text{emeasure } M\ A + \text{emeasure } M\ (B - A)$ 
using plus_emeasure[of  $A\ M\ B - A$ ] by auto

```


lemma *emeasure_Un_Int*:

assumes $A \in \text{sets } M$ $B \in \text{sets } M$

shows $\text{emeasure } M A + \text{emeasure } M B = \text{emeasure } M (A \cup B) + \text{emeasure } M (A \cap B)$

proof –

have $A = (A - B) \cup (A \cap B)$ **by** *auto*

then have $\text{emeasure } M A = \text{emeasure } M (A - B) + \text{emeasure } M (A \cap B)$

by (*metis Diff_Diff_Int Diff_disjoint assms plus_emeasure sets.Diff*)

moreover have $A \cup B = (A - B) \cup B$ **by** *auto*

then have $\text{emeasure } M (A \cup B) = \text{emeasure } M (A - B) + \text{emeasure } M B$

by (*metis Diff_disjoint Int_commute assms plus_emeasure sets.Diff*)

ultimately show *?thesis* **by** (*metis add.assoc add.commute*)

qed

lemma *sum_emeasure*:

$F I \subseteq \text{sets } M \implies \text{disjoint\_family\_on } F I \implies \text{finite } I \implies$

$(\sum_{i \in I} \text{emeasure } M (F i)) = \text{emeasure } M (\bigcup_{i \in I} F i)$

by (*metis sets.additive_sum emeasure_positive emeasure_additive*)

lemma *emeasure_mono*:

$a \subseteq b \implies b \in \text{sets } M \implies \text{emeasure } M a \leq \text{emeasure } M b$

by (*metis zero_le sets.additive_increasing emeasure_additive emeasure_notin_sets emeasure_positive increasingD*)

lemma *emeasure_space*:

$\text{emeasure } M A \leq \text{emeasure } M (\text{space } M)$

by (*metis emeasure_mono emeasure_notin_sets sets.sets_into_space sets.top zero_le*)

lemma *emeasure_Diff*:

assumes *finite*: $\text{emeasure } M B \neq \infty$

and [*measurable*]: $A \in \text{sets } M$ $B \in \text{sets } M$ **and** $B \subseteq A$

shows $\text{emeasure } M (A - B) = \text{emeasure } M A - \text{emeasure } M B$

proof –

have $(A - B) \cup B = A$ **using** $\langle B \subseteq A \rangle$ **by** *auto*

then have $\text{emeasure } M A = \text{emeasure } M ((A - B) \cup B)$ **by** *simp*

also have $\dots = \text{emeasure } M (A - B) + \text{emeasure } M B$

by (*subst plus_emeasure[symmetric]*) *auto*

finally show $\text{emeasure } M (A - B) = \text{emeasure } M A - \text{emeasure } M B$

using *finite* **by** *simp*

qed

lemma *emeasure_compl*:

$s \in \text{sets } M \implies \text{emeasure } M s \neq \infty \implies \text{emeasure } M (\text{space } M - s) = \text{emeasure } M (\text{space } M) - \text{emeasure } M s$

by (*rule emeasure_Diff*) (*auto dest: sets.sets_into_space*)

lemma *Lim_emeasure_incseq*:

$\text{range } A \subseteq \text{sets } M \implies \text{incseq } A \implies (\lambda i. (\text{emeasure } M (A \ i))) \longrightarrow \text{emeasure } M (\bigcup i. A \ i)$
using *emeasure_countably_additive*
by (*auto simp add: sets.countably_additive_iff_continuous_from_below emeasure_positive emeasure_additive*)

lemma *incseq_emeasure*:
assumes $\text{range } B \subseteq \text{sets } M \text{ incseq } B$
shows $\text{incseq } (\lambda i. \text{emeasure } M (B \ i))$
using *assms* **by** (*auto simp: incseq_def intro!: emeasure_mono*)

lemma *SUP_emeasure_incseq*:
assumes $A: \text{range } A \subseteq \text{sets } M \text{ incseq } A$
shows $(\text{SUP } n. \text{emeasure } M (A \ n)) = \text{emeasure } M (\bigcup i. A \ i)$
using *LIMSEQ_SUP[OF incseq_emeasure, OF A] Lim_emeasure_incseq[OF A]*
by (*simp add: LIMSEQ_unique*)

lemma *decseq_emeasure*:
assumes $\text{range } B \subseteq \text{sets } M \text{ decseq } B$
shows $\text{decseq } (\lambda i. \text{emeasure } M (B \ i))$
using *assms* **by** (*auto simp: decseq_def intro!: emeasure_mono*)

lemma *INF_emeasure_decseq*:
assumes $A: \text{range } A \subseteq \text{sets } M$ **and** $\text{decseq } A$
and *finite*: $\bigwedge i. \text{emeasure } M (A \ i) \neq \infty$
shows $(\text{INF } n. \text{emeasure } M (A \ n)) = \text{emeasure } M (\bigcap i. A \ i)$
proof –
have *le_MI*: $\text{emeasure } M (\bigcap i. A \ i) \leq \text{emeasure } M (A \ 0)$
using *A* **by** (*auto intro!: emeasure_mono*)
hence $*$: $\text{emeasure } M (\bigcap i. A \ i) \neq \infty$ **using** *finite[of 0]* **by** (*auto simp: top_unique*)

have $\text{emeasure } M (A \ 0) - (\text{INF } n. \text{emeasure } M (A \ n)) = (\text{SUP } n. \text{emeasure } M (A \ 0) - \text{emeasure } M (A \ n))$
by (*simp add: ennreal_INF_const_minus*)
also **have** $\dots = (\text{SUP } n. \text{emeasure } M (A \ 0 - A \ n))$
using *A finite* $\langle \text{decseq } A \rangle$ [*unfolded decseq_def*] **by** (*subst emeasure_Diff*) *auto*
also **have** $\dots = \text{emeasure } M (\bigcup i. A \ 0 - A \ i)$
proof (*rule SUP_emeasure_incseq*)
show $\text{range } (\lambda n. A \ 0 - A \ n) \subseteq \text{sets } M$
using *A* **by** *auto*
show $\text{incseq } (\lambda n. A \ 0 - A \ n)$
using $\langle \text{decseq } A \rangle$ **by** (*auto simp add: incseq_def decseq_def*)
qed
also **have** $\dots = \text{emeasure } M (A \ 0) - \text{emeasure } M (\bigcap i. A \ i)$
using *A finite* $*$ **by** (*simp, subst emeasure_Diff*) *auto*
finally **show** *?thesis*
by (*rule ennreal_minus_cancel[rotated 3]*)
(insert finite A, auto intro: INF_lower emeasure_mono)

qed

lemma *INF_emeasure_decseq'*:
assumes $A: \bigwedge i. A\ i \in \text{sets } M$ **and** *decseq* A
and *finite*: $\exists i. \text{emeasure } M\ (A\ i) \neq \infty$
shows $(\text{INF } n. \text{emeasure } M\ (A\ n)) = \text{emeasure } M\ (\bigcap i. A\ i)$
proof –
from *finite* **obtain** i **where** $i: \text{emeasure } M\ (A\ i) < \infty$
by (*auto simp: less_top*)
have *fin*: $i \leq j \implies \text{emeasure } M\ (A\ j) < \infty$ **for** j
by (*rule le_less_trans[OF emeasure_mono i]*) (*auto intro!: decseqD[OF <decseq A>] A*)

have $(\text{INF } n. \text{emeasure } M\ (A\ n)) = (\text{INF } n. \text{emeasure } M\ (A\ (n + i)))$
proof (*rule INF_eq*)
show $\exists j \in \text{UNIV}. \text{emeasure } M\ (A\ (j + i)) \leq \text{emeasure } M\ (A\ i')$ **for** i'
by (*intro bexI[of _ i'] emeasure_mono decseqD[OF <decseq A>] A*) *auto*
qed *auto*
also **have** $\dots = \text{emeasure } M\ (\text{INF } n. (A\ (n + i)))$
using $A\ <\text{decseq } A>$ **fin** **by** (*intro INF_emeasure_decseq*) (*auto simp: decseq_def less_top*)
also **have** $(\text{INF } n. (A\ (n + i))) = (\text{INF } n. A\ n)$
by (*meson INF_eq UNIV_I assms(2) decseqD le_add1*)
finally **show** *?thesis* .
qed

lemma *emeasure_INT_decseq_subset*:
fixes $F :: \text{nat} \Rightarrow 'a\ \text{set}$
assumes $I: I \neq \{\}$ **and** $F: \bigwedge i\ j. i \in I \implies j \in I \implies i \leq j \implies F\ j \subseteq F\ i$
assumes $F\_sets[\text{measurable}]: \bigwedge i. i \in I \implies F\ i \in \text{sets } M$
and *fin*: $\bigwedge i. i \in I \implies \text{emeasure } M\ (F\ i) \neq \infty$
shows $\text{emeasure } M\ (\bigcap i \in I. F\ i) = (\text{INF } i \in I. \text{emeasure } M\ (F\ i))$
proof *cases*
assume *finite* I
have $(\bigcap i \in I. F\ i) = F\ (\text{Max } I)$
using $I\ <\text{finite } I>$ **by** (*intro antisym INF_lower INF_greatest F*) *auto*
moreover **have** $(\text{INF } i \in I. \text{emeasure } M\ (F\ i)) = \text{emeasure } M\ (F\ (\text{Max } I))$
using $I\ <\text{finite } I>$ **by** (*intro antisym INF_lower INF_greatest F emeasure_mono*) *auto*
ultimately **show** *?thesis*
by *simp*
next
assume *infinite* I
define L **where** $L\ n = (\text{LEAST } i. i \in I \wedge i \geq n)$ **for** n
have $L: L\ n \in I \wedge n \leq L\ n$ **for** n
unfolding L_def
proof (*rule LeastI_ex*)
show $\exists x. x \in I \wedge n \leq x$
using $<\text{infinite } I>$ *finite_subset*[*of* $I\ \{..< n\}$]

```

    by (rule_tac ccontr) (auto simp: not_le)
  qed
  have L_eq[simp]:  $i \in I \implies L\ i = i$  for  $i$ 
    unfolding L_def by (intro Least_equality) auto
  have L_mono:  $i \leq j \implies L\ i \leq L\ j$  for  $i\ j$ 
    using L[of j] unfolding L_def by (intro Least_le) (auto simp: L_def)

  have emeasure M ( $\bigcap i. F\ (L\ i)$ ) = ( $\text{INF } i. \text{emeasure } M\ (F\ (L\ i))$ )
  proof (intro INF_emeasure_decseq[symmetric])
    show decseq ( $\lambda i. F\ (L\ i)$ )
      using L by (intro antimonoI F L_mono) auto
  qed (insert L fin, auto)
  also have ... = ( $\text{INF } i \in I. \text{emeasure } M\ (F\ i)$ )
  proof (intro antisym INF_greatest)
    show  $i \in I \implies (\text{INF } i. \text{emeasure } M\ (F\ (L\ i))) \leq \text{emeasure } M\ (F\ i)$  for  $i$ 
      by (intro INF_lower2[of i]) auto
  qed (insert L, auto intro: INF_lower)
  also have ( $\bigcap i. F\ (L\ i)$ ) = ( $\bigcap i \in I. F\ i$ )
  proof (intro antisym INF_greatest)
    show  $i \in I \implies (\bigcap i. F\ (L\ i)) \subseteq F\ i$  for  $i$ 
      by (intro INF_lower2[of i]) auto
  qed (insert L, auto)
  finally show ?thesis .
qed

lemma Lim_emeasure_decseq:
  assumes A:  $\text{range } A \subseteq \text{sets } M$  decseq A and fin:  $\bigwedge i. \text{emeasure } M\ (A\ i) \neq \infty$ 
  shows ( $\lambda i. \text{emeasure } M\ (A\ i)$ )  $\longrightarrow \text{emeasure } M\ (\bigcap i. A\ i)$ 
  using LIMSEQ_INF[OF decseq_emeasure, OF A]
  using INF_emeasure_decseq[OF A fin] by simp

lemma emeasure_lfp[consumes 1, case_names cont measurable]:
  assumes P M
  assumes cont:  $\text{sup\_continuous } F$ 
  assumes *:  $\bigwedge M\ A. P\ M \implies (\bigwedge N. P\ N \implies \text{Measurable.pred } N\ A) \implies \text{Measurable.pred } M\ (F\ A)$ 
  shows  $\text{emeasure } M\ \{x \in \text{space } M. \text{lfp } F\ x\} = (\text{SUP } i. \text{emeasure } M\ \{x \in \text{space } M. (F \frown i)\ (\lambda x. \text{False})\ } x)$ 
  proof -
    have  $\text{emeasure } M\ \{x \in \text{space } M. \text{lfp } F\ x\} = \text{emeasure } M\ (\bigcup i. \{x \in \text{space } M. (F \frown i)\ (\lambda x. \text{False})\ } x)$ 
      using sup_continuous_lfp[OF cont] by (auto simp add: bot_fun_def intro!: arg_cong2[where f=emeasure])
    moreover { fix i from  $\langle P\ M \rangle$  have  $\{x \in \text{space } M. (F \frown i)\ (\lambda x. \text{False})\ } x \in \text{sets } M$ 
      by (induct i arbitrary: M) (auto simp add: pred_def[symmetric] intro: *) }
    moreover have incseq ( $\lambda i. \{x \in \text{space } M. (F \frown i)\ (\lambda x. \text{False})\ } x$ )
    proof (rule incseq_SucI)
      fix i

```

```

    have  $(F \rightsquigarrow i) (\lambda x. \text{False}) \leq (F \rightsquigarrow (\text{Suc } i)) (\lambda x. \text{False})$ 
  proof (induct i)
    case 0 show ?case by (simp add: le_fun_def)
  next
    case Suc thus ?case using monoD[OF sup_continuous_mono[OF cont] Suc]
  by auto
  qed
  then show  $\{x \in \text{space } M. (F \rightsquigarrow i) (\lambda x. \text{False}) x\} \subseteq \{x \in \text{space } M. (F \rightsquigarrow \text{Suc } i) (\lambda x. \text{False}) x\}$ 
  by auto
  qed
  ultimately show ?thesis
  by (subst SUP_emeasure_incseq) auto
qed

```

```

lemma emeasure_lfp:
  assumes [simp]:  $\bigwedge s. \text{sets } (M s) = \text{sets } N$ 
  assumes cont:  $\text{sup\_continuous } F \text{ sup\_continuous } f$ 
  assumes meas:  $\bigwedge P. \text{Measurable.pred } N P \implies \text{Measurable.pred } N (F P)$ 
  assumes iter:  $\bigwedge P s. \text{Measurable.pred } N P \implies P \leq \text{lfp } F \implies \text{emeasure } (M s) \{x \in \text{space } N. F P x\} = f (\lambda s. \text{emeasure } (M s) \{x \in \text{space } N. P x\}) s$ 
  shows  $\text{emeasure } (M s) \{x \in \text{space } N. \text{lfp } F x\} = \text{lfp } f s$ 
proof (subst lfp_transfer_bounded[where  $\alpha = \lambda F s. \text{emeasure } (M s) \{x \in \text{space } N. F x\}$  and  $g = f$  and  $f = F$  and  $P = \text{Measurable.pred } N, \text{ symmetric}$ ])
  fix C assume incseq C  $\bigwedge i. \text{Measurable.pred } N (C i)$ 
  then show  $(\lambda s. \text{emeasure } (M s) \{x \in \text{space } N. (\text{SUP } i. C i) x\}) = (\text{SUP } i. (\lambda s. \text{emeasure } (M s) \{x \in \text{space } N. C i x\}))$ 
  unfolding SUP_apply[abs_def]
  by (subst SUP_emeasure_incseq) (auto simp: mono_def fun_eq_iff intro!: arg_cong2[where f = emeasure])
qed (auto simp add: iter le_fun_def SUP_apply[abs_def] intro!: meas cont)

```

```

lemma emeasure_subadditive_finite:
  finite I  $\implies A \text{ ' } I \subseteq \text{sets } M \implies \text{emeasure } M (\bigcup i \in I. A i) \leq (\sum i \in I. \text{emeasure } M (A i))$ 
  by (rule sets.subadditive[OF emeasure_positive emeasure_additive]) auto

```

```

lemma emeasure_subadditive:
   $A \in \text{sets } M \implies B \in \text{sets } M \implies \text{emeasure } M (A \cup B) \leq \text{emeasure } M A + \text{emeasure } M B$ 
  using emeasure_subadditive_finite[of  $\{\text{True}, \text{False}\} \lambda \text{True} \Rightarrow A \mid \text{False} \Rightarrow B M$ ]
  by simp

```

```

lemma emeasure_subadditive_countably:
  assumes range f  $\subseteq \text{sets } M$ 
  shows  $\text{emeasure } M (\bigcup i. f i) \leq (\sum i. \text{emeasure } M (f i))$ 
proof -
  have  $\text{emeasure } M (\bigcup i. f i) = \text{emeasure } M (\bigcup i. \text{disjointed } f i)$ 
  unfolding UN_disjointed_eq ..

```

also have $\dots = (\sum i. \text{emeasure } M (\text{disjointed } f \ i))$
 using `sets.range_disjointed_sets[OF assms] suminf_emeasure[of disjointed f]`
 by `(simp add: disjoint_family_disjointed comp_def)`
 also have $\dots \leq (\sum i. \text{emeasure } M (f \ i))$
 using `sets.range_disjointed_sets[OF assms] assms`
 by `(auto intro!: suminf_le emeasure_mono disjointed_subset)`
 finally show `?thesis` .
 qed

lemma `emeasure_insert`:
 assumes `sets: {x} ∈ sets M A ∈ sets M and x ∉ A`
 shows `emeasure M (insert x A) = emeasure M {x} + emeasure M A`
proof –
 have `{x} ∩ A = {}` using `⟨x ∉ A⟩` by `auto`
 from `plus_emeasure[OF sets this]` show `?thesis` by `simp`
 qed

lemma `emeasure_insert_ne`:
 $A \neq \{x\} \implies \{x\} \in \text{sets } M \implies A \in \text{sets } M \implies x \notin A \implies \text{emeasure } M (\text{insert } x \ A) = \text{emeasure } M \{x\} + \text{emeasure } M A$
 by `(rule emeasure_insert)`

lemma `emeasure_eq_sum_singleton`:
 assumes `finite S ∧ x. x ∈ S ⟹ {x} ∈ sets M`
 shows `emeasure M S = (∑ x ∈ S. emeasure M {x})`
 using `sum_emeasure[of λx. {x} S M] assms`
 by `(auto simp: disjoint_family_on_def subset_eq)`

lemma `sum_emeasure_cover`:
 assumes `finite S and A ∈ sets M and br_in_M: B ‘ S ⊆ sets M`
 assumes `A: A ⊆ (⋃ i ∈ S. B i)`
 assumes `disj: disjoint_family_on B S`
 shows `emeasure M A = (∑ i ∈ S. emeasure M (A ∩ (B i)))`
proof –
 have $(\sum i \in S. \text{emeasure } M (A \cap (B \ i))) = \text{emeasure } M (\bigcup i \in S. A \cap (B \ i))$
proof `(rule sum_emeasure)`
 show `disjoint_family_on (λi. A ∩ B i) S`
 using `⟨disjoint_family_on B S⟩`
 unfolding `disjoint_family_on_def` by `auto`
 qed `(insert assms, auto)`
 also have $(\bigcup i \in S. A \cap (B \ i)) = A$
 using `A` by `auto`
 finally show `?thesis` by `simp`
 qed

lemma `emeasure_eq_0`:
 $N \in \text{sets } M \implies \text{emeasure } M N = 0 \implies K \subseteq N \implies \text{emeasure } M K = 0$
 by `(metis emeasure_mono order_eq_iff zero_le)`

```

lemma emeasure_UN_eq_0:
  assumes  $\bigwedge i::\text{nat}. \text{emeasure } M (N\ i) = 0$  and  $\text{range } N \subseteq \text{sets } M$ 
  shows  $\text{emeasure } M (\bigcup i. N\ i) = 0$ 
proof -
  have  $\text{emeasure } M (\bigcup i. N\ i) \leq 0$ 
    using emeasure_subadditive_countably[OF assms(2)] assms(1) by simp
  then show ?thesis
    by (auto intro: antisym zero_le)
qed

lemma measure_eqI_finite:
  assumes [simp]:  $\text{sets } M = \text{Pow } A$   $\text{sets } N = \text{Pow } A$  and finite A
  assumes eq:  $\bigwedge a. a \in A \implies \text{emeasure } M \{a\} = \text{emeasure } N \{a\}$ 
  shows  $M = N$ 
proof (rule measure_eqI)
  fix X assume  $X \in \text{sets } M$ 
  then have  $X \subseteq A$  by auto
  then have  $\text{emeasure } M\ X = (\sum a \in X. \text{emeasure } M \{a\})$ 
    using  $\langle \text{finite } A \rangle$  by (subst emeasure_eq_sum_singleton) (auto dest: finite_subset)
  also have  $\dots = (\sum a \in X. \text{emeasure } N \{a\})$ 
    using X eq by (auto intro!: sum.cong)
  also have  $\dots = \text{emeasure } N\ X$ 
    using  $X \langle \text{finite } A \rangle$  by (subst emeasure_eq_sum_singleton) (auto dest: finite_subset)
  finally show  $\text{emeasure } M\ X = \text{emeasure } N\ X$  .
qed simp

lemma measure_eqI_generator_eq:
  fixes M N :: 'a measure and E :: 'a set set and A :: nat  $\Rightarrow$  'a set
  assumes Int_stable E  $E \subseteq \text{Pow } \Omega$ 
  and eq:  $\bigwedge X. X \in E \implies \text{emeasure } M\ X = \text{emeasure } N\ X$ 
  and M:  $\text{sets } M = \text{sigma\_sets } \Omega\ E$ 
  and N:  $\text{sets } N = \text{sigma\_sets } \Omega\ E$ 
  and A:  $\text{range } A \subseteq E$   $(\bigcup i. A\ i) = \Omega$   $\bigwedge i. \text{emeasure } M (A\ i) \neq \infty$ 
  shows  $M = N$ 
proof -
  let ? $\mu$  = emeasure M and ? $\nu$  = emeasure N
  interpret S: sigma_algebra  $\Omega$  sigma_sets  $\Omega\ E$  by (rule sigma_algebra_sigma_sets)
  fact
  have space M =  $\Omega$ 
    using sets.top[of M] sets.space_closed[of M] S.top S.space_closed  $\langle \text{sets } M = \text{sigma\_sets } \Omega\ E \rangle$ 
    by blast

  { fix F D assume  $F \in E$  and ? $\mu$  F  $\neq \infty$ 
    then have [intro]:  $F \in \text{sigma\_sets } \Omega\ E$  by auto
    have ? $\nu$  F  $\neq \infty$  using  $\langle ?\mu\ F \neq \infty \rangle \langle F \in E \rangle$  eq by simp
    assume D  $\in \text{sets } M$ 
    with  $\langle \text{Int\_stable } E \rangle \langle E \subseteq \text{Pow } \Omega \rangle$  have  $\text{emeasure } M (F \cap D) = \text{emeasure } N$ 

```

```

(F ∩ D)
  unfolding M
  proof (induct rule: sigma_sets_induct_disjoint)
    case (basic A)
      then have F ∩ A ∈ E using ⟨Int_stable E⟩ ⟨F ∈ E⟩ by (auto simp:
Int_stable_def)
      then show ?case using eq by auto
    next
      case empty then show ?case by simp
    next
      case (compl A)
        then have **: F ∩ (Ω - A) = F - (F ∩ A)
          and [intro]: F ∩ A ∈ sigma_sets Ω E
            using ⟨F ∈ E⟩ S.sets_into_space by (auto simp: M)
          have ?ν (F ∩ A) ≤ ?ν F by (auto intro!: emeasure_mono simp: M N)
          then have ?ν (F ∩ A) ≠ ∞ using ⟨?ν F ≠ ∞⟩ by (auto simp: top_unique)
          have ?μ (F ∩ A) ≤ ?μ F by (auto intro!: emeasure_mono simp: M N)
          then have ?μ (F ∩ A) ≠ ∞ using ⟨?μ F ≠ ∞⟩ by (auto simp: top_unique)
          then have ?μ (F ∩ (Ω - A)) = ?μ F - ?μ (F ∩ A) unfolding **
            using ⟨F ∩ A ∈ sigma_sets Ω E⟩ by (auto intro!: emeasure_Diff simp: M
N)
          also have ... = ?ν F - ?ν (F ∩ A) using eq ⟨F ∈ E⟩ compl by simp
          also have ... = ?ν (F ∩ (Ω - A)) unfolding **
            using ⟨F ∩ A ∈ sigma_sets Ω E⟩ ⟨?ν (F ∩ A) ≠ ∞⟩
              by (auto intro!: emeasure_Diff[symmetric] simp: M N)
          finally show ?case
            using ⟨space M = Ω⟩ by auto
        next
          case (union A)
            then have ?μ (⋃ x. F ∩ A x) = ?ν (⋃ x. F ∩ A x)
              by (subst (1 2) suminf_emeasure[symmetric]) (auto simp: disjoint_family_on_def
subset_eq M N)
            with A show ?case
              by auto
          qed }
    note * = this
  show M = N
  proof (rule measure_eqI)
    show sets M = sets N
      using M N by simp
    have [simp, intro]: ⋀ i. A i ∈ sets M
      using A(1) by (auto simp: subset_eq M)
    fix F assume F ∈ sets M
    let ?D = disjointed (λ i. F ∩ A i)
    from ⟨space M = Ω⟩ have F_eq: F = (⋃ i. ?D i)
      using ⟨F ∈ sets M⟩ [THEN sets.sets_into_space] A(2)[symmetric] by (auto
simp: UN_disjointed_eq)
    have [simp, intro]: ⋀ i. ?D i ∈ sets M
      using sets.range_disjointed_sets[of λ i. F ∩ A i M] ⟨F ∈ sets M⟩

```



```

    by (auto simp: subset_eq)
  have disjoint_family ?D
    by (auto simp: disjoint_family_disjointed)
  moreover
  have  $(\sum i. \text{emeasure } M \text{ } (?D \ i)) = (\sum i. \text{emeasure } N \text{ } (?D \ i))$ 
  proof (intro arg_cong[where f=suminf] ext)
    fix i
    have  $A \ i \cap ?D \ i = ?D \ i$ 
      by (auto simp: disjointed_def)
    then show  $\text{emeasure } M \text{ } (?D \ i) = \text{emeasure } N \text{ } (?D \ i)$ 
      using *[of  $A \ i \ ?D \ i$ ,  $OF \ \_ \ A(\mathfrak{I})$ ]  $A(1)$ ] by auto
  qed
  ultimately show  $\text{emeasure } M \ F = \text{emeasure } N \ F$ 
  by (simp add: image_subset_iff  $\langle \text{sets } M = \text{sets } N \rangle$ [symmetric]  $F\_eq$ [symmetric]
    suminf_emeasure)
  qed
qed

lemma space_empty:  $\text{space } M = \{\} \implies M = \text{count\_space } \{\}$ 
  by (rule measure_eqI) (simp_all add: space_empty_iff)

lemma measure_eqI_generator_eq_countable:
  fixes  $M \ N :: 'a \text{ measure}$  and  $E :: 'a \text{ set set}$  and  $A :: 'a \text{ set set}$ 
  assumes  $E: \text{Int\_stable } E \ E \subseteq \text{Pow } \Omega \ \bigwedge X. X \in E \implies \text{emeasure } M \ X = \text{emeasure } N \ X$ 
    and  $\text{sets: sets } M = \text{sigma\_sets } \Omega \ E \ \text{sets } N = \text{sigma\_sets } \Omega \ E$ 
    and  $A: A \subseteq E \ (\bigcup A) = \Omega \ \text{countable } A \ \bigwedge a. a \in A \implies \text{emeasure } M \ a \neq \infty$ 
  shows  $M = N$ 
proof cases
  assume  $\Omega = \{\}$ 
  have *:  $\text{sigma\_sets } \Omega \ E = \text{sets } (\text{sigma } \Omega \ E)$ 
    using  $E(2)$  by simp
  have  $\text{space } M = \Omega \ \text{space } N = \Omega$ 
    using  $\text{sets } E(2)$  unfolding * by (auto dest: sets_eq_imp_space_eq simp del:
    sets_measure_of)
  then show  $M = N$ 
    unfolding  $\langle \Omega = \{\} \rangle$  by (auto dest: space_empty)
next
  assume  $\Omega \neq \{\}$  with  $\langle \bigcup A = \Omega \rangle$  have  $A \neq \{\}$  by auto
  from this  $\langle \text{countable } A \rangle$  have  $\text{rng: range } (\text{from\_nat\_into } A) = A$ 
    by (rule range_from_nat_into)
  show  $M = N$ 
proof (rule measure_eqI_generator_eq[OF  $E \ \text{sets}$ ])
  show  $\text{range } (\text{from\_nat\_into } A) \subseteq E$ 
    unfolding  $\text{rng}$  using  $\langle A \subseteq E \rangle$  .
  show  $(\bigcup i. \text{from\_nat\_into } A \ i) = \Omega$ 
    unfolding  $\text{rng}$  using  $\langle \bigcup A = \Omega \rangle$  .
  show  $\text{emeasure } M \ (\text{from\_nat\_into } A \ i) \neq \infty$  for  $i$ 
    using  $\text{rng}$  by (intro  $A$ ) auto

```

1318

qed
qed

lemma *measure_of_of_measure*: *measure_of* (*space M*) (*sets M*) (*emeasure M*)
= *M*
proof (*intro measure_eqI emeasure_measure_of_sigma*)
 show *sigma_algebra* (*space M*) (*sets M*) ..
 show *positive* (*sets M*) (*emeasure M*)
 by (*simp add: positive_def*)
 show *countably_additive* (*sets M*) (*emeasure M*)
 by (*simp add: emeasure_countably_additive*)
qed simp_all

6.3.5 μ -null sets

definition *null_sets* :: 'a *measure* $\Rightarrow$ 'a *set set* **where**
 null_sets M = {*N* $\in$ *sets M*. *emeasure M N* = 0}

lemma *null_setsD1*[*dest*]: *A* $\in$ *null_sets M* $\implies$ *emeasure M A* = 0
 by (*simp add: null_sets_def*)

lemma *null_setsD2*[*dest*]: *A* $\in$ *null_sets M* $\implies$ *A* $\in$ *sets M*
 unfolding null_sets_def by simp

lemma *null_setsI*[*intro*]: *emeasure M A* = 0 $\implies$ *A* $\in$ *sets M* $\implies$ *A* $\in$ *null_sets M*
 unfolding null_sets_def by simp

interpretation *null_sets*: *ring_of_sets space M null_sets M for M*

proof (*rule ring_of_setsI*)
 show *null_sets M* $\subseteq$ *Pow* (*space M*)
 using sets.sets_into_space by auto
 show {} $\in$ *null_sets M*
 by auto
 fix A B assume null_sets: *A* $\in$ *null_sets M* *B* $\in$ *null_sets M*
 then have sets: *A* $\in$ *sets M* *B* $\in$ *sets M*
 by auto
 then have *: *emeasure M* (*A* $\cup$ *B*) $\leq$ *emeasure M A* + *emeasure M B*
 emeasure M (*A* $-$ *B*) $\leq$ *emeasure M A*
 by (*auto intro!*: *emeasure_subadditive emeasure_mono*)
 then have *emeasure M B* = 0 *emeasure M A* = 0
 using null_sets by auto
 with sets * show *A* $-$ *B* $\in$ *null_sets M* *A* $\cup$ *B* $\in$ *null_sets M*
 by (*auto intro!*: *antisym zero_le*)
qed

lemma *UN_from_nat_into*:
 assumes *I*: *countable I* *I* $\neq$ {}
 shows ($\bigcup_{i \in I. N\ i}$) = ($\bigcup_{i. N\ (from\_nat\_into\ I\ i)}$)

```

proof –
  have  $(\bigcup_{i \in I}. N\ i) = \bigcup (N \text{ ` } \text{range } (\text{from\_nat\_into } I))$ 
    using  $I$  by simp
  also have  $\dots = (\bigcup i. (N \circ \text{from\_nat\_into } I)\ i)$ 
    by simp
  finally show ?thesis by simp
qed

lemma null_sets_UN':
  assumes countable I
  assumes  $\bigwedge i. i \in I \implies N\ i \in \text{null\_sets } M$ 
  shows  $(\bigcup_{i \in I}. N\ i) \in \text{null\_sets } M$ 
proof cases
  assume  $I = \{\}$  then show ?thesis by simp
next
  assume  $I \neq \{\}$ 
  show ?thesis
  proof (intro conjI CollectI null_setsI)
    show  $(\bigcup_{i \in I}. N\ i) \in \text{sets } M$ 
      using assms by (intro sets.countable_UN') auto
    have  $\text{emeasure } M (\bigcup_{i \in I}. N\ i) \leq (\sum n. \text{emeasure } M (N (\text{from\_nat\_into } I\ n)))$ 
      unfolding UN_from_nat_into [OF  $\langle \text{countable } I \rangle \langle I \neq \{\} \rangle]$ 
      using assms  $\langle I \neq \{\} \rangle$  by (intro emeasure_subadditive_countably) (auto intro: from_nat_into)
    also have  $(\lambda n. \text{emeasure } M (N (\text{from\_nat\_into } I\ n))) = (\lambda \_. 0)$ 
      using assms  $\langle I \neq \{\} \rangle$  by (auto intro: from_nat_into)
    finally show  $\text{emeasure } M (\bigcup_{i \in I}. N\ i) = 0$ 
      by (intro antisym zero_le) simp
  qed
qed

lemma null_sets_UN[intro]:
   $(\bigwedge i::i::\text{countable}. N\ i \in \text{null\_sets } M) \implies (\bigcup i. N\ i) \in \text{null\_sets } M$ 
  by (rule null_sets_UN') auto

lemma null_set_Int1:
  assumes  $B \in \text{null\_sets } M$   $A \in \text{sets } M$  shows  $A \cap B \in \text{null\_sets } M$ 
proof (intro CollectI conjI null_setsI)
  show  $\text{emeasure } M (A \cap B) = 0$  using assms
    by (intro emeasure_eq_0[of B _ A ∩ B]) auto
qed (insert assms, auto)

lemma null_set_Int2:
  assumes  $B \in \text{null\_sets } M$   $A \in \text{sets } M$  shows  $B \cap A \in \text{null\_sets } M$ 
  using assms by (subst Int_commute) (rule null_set_Int1)

lemma emeasure_Diff_null_set:
  assumes  $B \in \text{null\_sets } M$   $A \in \text{sets } M$ 
  shows  $\text{emeasure } M (A - B) = \text{emeasure } M\ A$ 

```

```

proof –
  have *:  $A - B = (A - (A \cap B))$  by auto
  have  $A \cap B \in \text{null\_sets } M$  using assms by (rule null\_set\_Int1)
  then show ?thesis
    unfolding * using assms
    by (subst emeasure\_Diff) auto
qed

lemma null\_set\_Diff:
  assumes  $B \in \text{null\_sets } M$   $A \in \text{sets } M$  shows  $B - A \in \text{null\_sets } M$ 
proof (intro CollectI conjI null\_setsI)
  show  $\text{emeasure } M (B - A) = 0$  using assms by (intro emeasure\_eq\_0[of B -
 $B - A]$ ) auto
qed (insert assms, auto)

lemma emeasure\_Un\_null\_set:
  assumes  $A \in \text{sets } M$   $B \in \text{null\_sets } M$ 
  shows  $\text{emeasure } M (A \cup B) = \text{emeasure } M A$ 
proof –
  have *:  $A \cup B = A \cup (B - A)$  by auto
  have  $B - A \in \text{null\_sets } M$  using assms(2,1) by (rule null\_set\_Diff)
  then show ?thesis
    unfolding * using assms
    by (subst plus\_emeasure[symmetric]) auto
qed

lemma emeasure\_Un':
  assumes  $A \in \text{sets } M$   $B \in \text{sets } M$   $A \cap B \in \text{null\_sets } M$ 
  shows  $\text{emeasure } M (A \cup B) = \text{emeasure } M A + \text{emeasure } M B$ 
proof –
  have  $A \cup B = A \cup (B - A \cap B)$  by blast
  also have  $\text{emeasure } M \dots = \text{emeasure } M A + \text{emeasure } M (B - A \cap B)$ 
    using assms by (subst plus\_emeasure) auto
  also have  $\text{emeasure } M (B - A \cap B) = \text{emeasure } M B$ 
    using assms by (intro emeasure\_Diff\_null\_set) auto
  finally show ?thesis .
qed

```

6.3.6 The almost everywhere filter (i.e. quantifier)

definition *ae_filter* :: 'a measure $\Rightarrow$ 'a filter **where**
 $\text{ae\_filter } M = (\text{INF } N \in \text{null\_sets } M. \text{principal } (\text{space } M - N))$

abbreviation *almost_everywhere* :: 'a measure $\Rightarrow$ ('a $\Rightarrow$ bool) $\Rightarrow$ bool **where**
 $\text{almost\_everywhere } M P \equiv \text{eventually } P (\text{ae\_filter } M)$

syntax
 $\_almost\_everywhere :: \text{pttrn} \Rightarrow 'a \Rightarrow \text{bool} \Rightarrow \text{bool} \text{ (AE } \_ \text{ in } \_ \text{. } \_ [0,0,10] 10)$

translations

$$AE\ x\ in\ M.\ P \equiv CONST\ almost\_everywhere\ M\ (\lambda x.\ P)$$
abbreviation

$$set\_almost\_everywhere\ A\ M\ P \equiv AE\ x\ in\ M.\ x \in A \longrightarrow P\ x$$
syntax

$$\begin{aligned} & \_set\_almost\_everywhere :: pttrn \Rightarrow 'a\ set \Rightarrow 'a \Rightarrow bool \Rightarrow bool \\ & (AE\ \_ \in\ \_ \ in\ \_ / \_ \ [0,0,0,10]\ 10) \end{aligned}$$
translations

$$AE\ x \in A\ in\ M.\ P \equiv CONST\ set\_almost\_everywhere\ A\ M\ (\lambda x.\ P)$$

lemma *eventually_ae_filter*: $eventually\ P\ (ae\_filter\ M) \longleftrightarrow (\exists N \in null\_sets\ M. \{x \in space\ M. \neg P\ x\} \subseteq N)$

unfolding *ae_filter_def* **by** (*subst eventually_INF_base*) (*auto simp: eventually_principal subset_eq*)

lemma *AE_I'*:

$$N \in null\_sets\ M \Longrightarrow \{x \in space\ M. \neg P\ x\} \subseteq N \Longrightarrow (AE\ x\ in\ M.\ P\ x)$$

unfolding *eventually_ae_filter* **by** *auto*

lemma *AE_iff_null*:

assumes $\{x \in space\ M. \neg P\ x\} \in sets\ M$ (**is** $?P \in sets\ M$)

shows $(AE\ x\ in\ M.\ P\ x) \longleftrightarrow \{x \in space\ M. \neg P\ x\} \in null\_sets\ M$

proof

assume $AE\ x\ in\ M.\ P\ x$ **then obtain** N **where** $N: N \in sets\ M\ ?P \subseteq N$ **emeasure** $M\ N = 0$

unfolding *eventually_ae_filter* **by** *auto*

have $emeasure\ M\ ?P \leq emeasure\ M\ N$

using *assms* $N(1,2)$ **by** (*auto intro: emeasure_mono*)

then have $emeasure\ M\ ?P = 0$

unfolding $\langle emeasure\ M\ N = 0 \rangle$ **by** *auto*

then show $?P \in null\_sets\ M$ **using** *assms* **by** *auto*

next

assume $?P \in null\_sets\ M$ **with** *assms* **show** $AE\ x\ in\ M.\ P\ x$ **by** (*auto intro: AE_I'*)

qed

lemma *AE_iff_null_sets*:

$$N \in sets\ M \Longrightarrow N \in null\_sets\ M \longleftrightarrow (AE\ x\ in\ M.\ x \notin N)$$

using *Int_absorb1* [*OF sets.sets_into_space, of N M*]

by (*subst AE_iff_null*) (*auto simp: Int_def[symmetric]*)

lemma *AE_not_in*:

$$N \in null\_sets\ M \Longrightarrow AE\ x\ in\ M.\ x \notin N$$

by (*metis AE_iff_null_sets null_setsD2*)

lemma *AE_iff_measurable*:

1322

$N \in \text{sets } M \implies \{x \in \text{space } M. \neg P x\} = N \implies (AE\ x\ \text{in } M. P\ x) \longleftrightarrow \text{emeasure } M\ N = 0$

using *AE_iff_null[of _ P]* **by** *auto*

lemma *AE_E[consumes 1]*:

assumes *AE x in M. P x*

obtains *N* **where** $\{x \in \text{space } M. \neg P x\} \subseteq N$ *emeasure M N = 0* $N \in \text{sets } M$

using *assms unfolding eventually_ae_filter* **by** *auto*

lemma *AE_E2*:

assumes *AE x in M. P x* $\{x \in \text{space } M. P x\} \in \text{sets } M$

shows *emeasure M {x ∈ space M. ¬ P x} = 0* (**is** *emeasure M ?P = 0*)

proof –

have $\{x \in \text{space } M. \neg P x\} = \text{space } M - \{x \in \text{space } M. P x\}$ **by** *auto*

with *AE_iff_null[of M P]* *assms* **show** *?thesis* **by** *auto*

qed

lemma *AE_E3*:

assumes *AE x in M. P x*

obtains *N* **where** $\bigwedge x. x \in \text{space } M - N \implies P\ x$ $N \in \text{null\_sets } M$

using *assms unfolding eventually_ae_filter* **by** *auto*

lemma *AE_I*:

assumes $\{x \in \text{space } M. \neg P x\} \subseteq N$ *emeasure M N = 0* $N \in \text{sets } M$

shows *AE x in M. P x*

using *assms unfolding eventually_ae_filter* **by** *auto*

lemma *AE_mp[elim!]*:

assumes *AE_P*: *AE x in M. P x* **and** *AE_imp*: *AE x in M. P x $\longrightarrow$ Q x*

shows *AE x in M. Q x*

proof –

from *AE_P* **obtain** *A* **where** $P: \{x \in \text{space } M. \neg P x\} \subseteq A$

and *A*: $A \in \text{sets } M$ *emeasure M A = 0*

by (*auto elim!: AE_E*)

from *AE_imp* **obtain** *B* **where** *imp*: $\{x \in \text{space } M. P x \wedge \neg Q x\} \subseteq B$

and *B*: $B \in \text{sets } M$ *emeasure M B = 0*

by (*auto elim!: AE_E*)

show *?thesis*

proof (*intro AE_I*)

have *emeasure M (A $\cup$ B) $\leq$ 0*

using *emeasure_subadditive[of A M B]* *A B* **by** *auto*

then show $A \cup B \in \text{sets } M$ *emeasure M (A $\cup$ B) = 0*

using *A B* **by** *auto*

show $\{x \in \text{space } M. \neg Q x\} \subseteq A \cup B$

using *P imp* **by** *auto*

qed

qed

The next lemma is convenient to combine with a lemma whose conclusion is of the form $AE\ x\ in\ M.\ P\ x = Q\ x$: for such a lemma, there is no *[symmetric]* variant, but using *AE_symmetric[OF...]* will replace it.

lemma

shows *AE_iffI*: $AE\ x\ in\ M.\ P\ x \implies AE\ x\ in\ M.\ P\ x \longleftrightarrow Q\ x \implies AE\ x\ in\ M.\ Q\ x$

and *AE_disjI1*: $AE\ x\ in\ M.\ P\ x \implies AE\ x\ in\ M.\ P\ x \vee Q\ x$

and *AE_disjI2*: $AE\ x\ in\ M.\ Q\ x \implies AE\ x\ in\ M.\ P\ x \vee Q\ x$

and *AE_conjI*: $AE\ x\ in\ M.\ P\ x \implies AE\ x\ in\ M.\ Q\ x \implies AE\ x\ in\ M.\ P\ x \wedge Q\ x$

and *AE_conj_iff[simp]*: $(AE\ x\ in\ M.\ P\ x \wedge Q\ x) \longleftrightarrow (AE\ x\ in\ M.\ P\ x) \wedge (AE\ x\ in\ M.\ Q\ x)$

by *auto*

lemma *AE_symmetric*:

assumes $AE\ x\ in\ M.\ P\ x = Q\ x$

shows $AE\ x\ in\ M.\ Q\ x = P\ x$

using *assms* **by** *auto*

lemma *AE_impI*:

$(P \implies AE\ x\ in\ M.\ Q\ x) \implies AE\ x\ in\ M.\ P \longrightarrow Q\ x$

by *fastforce*

lemma *AE_measure*:

assumes *AE*: $AE\ x\ in\ M.\ P\ x$ **and** *sets*: $\{x \in space\ M.\ P\ x\} \in sets\ M$ (**is** $?P \in sets\ M$)

shows $emeasure\ M\ \{x \in space\ M.\ P\ x\} = emeasure\ M\ (space\ M)$

proof –

from *AE_E[OF AE]* **obtain** *N*

where $N: \{x \in space\ M.\ \neg P\ x\} \subseteq N$ $emeasure\ M\ N = 0$ $N \in sets\ M$

by *auto*

with *sets* **have** $emeasure\ M\ (space\ M) \leq emeasure\ M\ (?P \cup N)$

by (*intro emeasure_mono*) *auto*

also have $\dots \leq emeasure\ M\ ?P + emeasure\ M\ N$

using *sets N* **by** (*intro emeasure_subadditive*) *auto*

also have $\dots = emeasure\ M\ ?P$ **using** *N* **by** *simp*

finally show $emeasure\ M\ ?P = emeasure\ M\ (space\ M)$

using *emeasure_space[of M ?P]* **by** *auto*

qed

lemma *AE_space*: $AE\ x\ in\ M.\ x \in space\ M$

by (*rule AE_I[where N={}]*) *auto*

lemma *AE_I2[simp, intro]*:

$(\bigwedge x. x \in space\ M \implies P\ x) \implies AE\ x\ in\ M.\ P\ x$

using *AE_space* **by** *force*

lemma *AE_Ball_mp*:

$\forall x \in space\ M.\ P\ x \implies AE\ x\ in\ M.\ P\ x \longrightarrow Q\ x \implies AE\ x\ in\ M.\ Q\ x$

by *auto*

lemma *AE_cong[cong]*:

$(\bigwedge x. x \in \text{space } M \implies P\ x \longleftrightarrow Q\ x) \implies (AE\ x\ \text{in } M. P\ x) \longleftrightarrow (AE\ x\ \text{in } M. Q\ x)$

by *auto*

lemma *AE_cong_simp*: $M = N \implies (\bigwedge x. x \in \text{space } N = \text{simp} \implies P\ x = Q\ x) \implies (AE\ x\ \text{in } M. P\ x) \longleftrightarrow (AE\ x\ \text{in } N. Q\ x)$

by (*auto simp: simp_implies_def*)

lemma *AE_all_countable*:

$(AE\ x\ \text{in } M. \forall i. P\ i\ x) \longleftrightarrow (\forall i::'i::\text{countable}. AE\ x\ \text{in } M. P\ i\ x)$

proof

assume $\forall i. AE\ x\ \text{in } M. P\ i\ x$

from *this[unfolded eventually_ae_filter Bex_def, THEN choice]*

obtain *N* **where** $N: \bigwedge i. N\ i \in \text{null\_sets } M \ \bigwedge i. \{x \in \text{space } M. \neg P\ i\ x\} \subseteq N\ i$

by *auto*

have $\{x \in \text{space } M. \neg (\forall i. P\ i\ x)\} \subseteq (\bigcup i. \{x \in \text{space } M. \neg P\ i\ x\})$ **by** *auto*

also have $\dots \subseteq (\bigcup i. N\ i)$ **using** *N* **by** *auto*

finally have $\{x \in \text{space } M. \neg (\forall i. P\ i\ x)\} \subseteq (\bigcup i. N\ i)$.

moreover from *N* **have** $(\bigcup i. N\ i) \in \text{null\_sets } M$

by (*intro null_sets_UN*) *auto*

ultimately show $AE\ x\ \text{in } M. \forall i. P\ i\ x$

unfolding *eventually_ae_filter* **by** *auto*

qed *auto*

lemma *AE_ball_countable*:

assumes [*intro*]: *countable X*

shows $(AE\ x\ \text{in } M. \forall y \in X. P\ x\ y) \longleftrightarrow (\forall y \in X. AE\ x\ \text{in } M. P\ x\ y)$

proof

assume $\forall y \in X. AE\ x\ \text{in } M. P\ x\ y$

from *this[unfolded eventually_ae_filter Bex_def, THEN bchoice]*

obtain *N* **where** $N: \bigwedge y. y \in X \implies N\ y \in \text{null\_sets } M \ \bigwedge y. y \in X \implies \{x \in \text{space } M. \neg P\ x\ y\} \subseteq N\ y$

by *auto*

have $\{x \in \text{space } M. \neg (\forall y \in X. P\ x\ y)\} \subseteq (\bigcup y \in X. \{x \in \text{space } M. \neg P\ x\ y\})$

by *auto*

also have $\dots \subseteq (\bigcup y \in X. N\ y)$

using *N* **by** *auto*

finally have $\{x \in \text{space } M. \neg (\forall y \in X. P\ x\ y)\} \subseteq (\bigcup y \in X. N\ y)$.

moreover from *N* **have** $(\bigcup y \in X. N\ y) \in \text{null\_sets } M$

by (*intro null_sets_UN*) *auto*

ultimately show $AE\ x\ \text{in } M. \forall y \in X. P\ x\ y$

unfolding *eventually_ae_filter* **by** *auto*

qed *auto*

lemma *AE_ball_countable'*:

$(\bigwedge N. N \in I \implies AE\ x\ \text{in } M. P\ N\ x) \implies \text{countable } I \implies AE\ x\ \text{in } M. \forall N \in I.$

$P \ N \ x$

unfolding $AE\_ball\_countable$ **by** $simp$

lemma $AE\_pairwise$: $countable \ F \implies pairwise \ (\lambda A \ B. AE \ x \ in \ M. R \ x \ A \ B) \ F$
 $\longleftrightarrow (AE \ x \ in \ M. pairwise \ (R \ x) \ F)$

unfolding $pairwise\_alt$ **by** ($simp \ add: AE\_ball\_countable$)

lemma $AE\_discrete\_difference$:

assumes X : $countable \ X$

assumes $null$: $\bigwedge x. x \in X \implies emeasure \ M \ \{x\} = 0$

assumes $sets$: $\bigwedge x. x \in X \implies \{x\} \in sets \ M$

shows $AE \ x \ in \ M. x \notin X$

proof –

have $(\bigcup x \in X. \{x\}) \in null\_sets \ M$

using $assms$ **by** ($intro \ null\_sets\_UN'$) $auto$

from $AE\_not\_in[OF \ this]$ **show** $AE \ x \ in \ M. x \notin X$

by $auto$

qed

lemma AE_finite_all :

assumes f : $finite \ S$ **shows** $(AE \ x \ in \ M. \forall i \in S. P \ i \ x) \longleftrightarrow (\forall i \in S. AE \ x \ in \ M. P \ i \ x)$

using f **by** $induct \ auto$

lemma AE_finite_allI :

assumes $finite \ S$

shows $(\bigwedge s. s \in S \implies AE \ x \ in \ M. Q \ s \ x) \implies AE \ x \ in \ M. \forall s \in S. Q \ s \ x$

using $AE\_finite\_all[OF \ \langle finite \ S \rangle]$ **by** $auto$

lemma $emeasure\_mono\_AE$:

assumes imp : $AE \ x \ in \ M. x \in A \longrightarrow x \in B$

and B : $B \in sets \ M$

shows $emeasure \ M \ A \leq emeasure \ M \ B$

proof $cases$

assume A : $A \in sets \ M$

from imp **obtain** N **where** N : $\{x \in space \ M. \neg (x \in A \longrightarrow x \in B)\} \subseteq N \ N \in null\_sets \ M$

by ($auto \ simp: eventually\_ae\_filter$)

have $emeasure \ M \ A = emeasure \ M \ (A - N)$

using $N \ A$ **by** ($subst \ emeasure\_Diff\_null\_set$) $auto$

also **have** $emeasure \ M \ (A - N) \leq emeasure \ M \ (B - N)$

using $N \ A \ B \ sets.sets\_into\_space$ **by** ($auto \ intro!: emeasure\_mono$)

also **have** $emeasure \ M \ (B - N) = emeasure \ M \ B$

using $N \ B$ **by** ($subst \ emeasure\_Diff\_null\_set$) $auto$

finally **show** $?thesis$.

qed ($simp \ add: emeasure\_notin\_sets$)

lemma $emeasure\_eq\_AE$:

assumes iff : $AE \ x \ in \ M. x \in A \longleftrightarrow x \in B$

assumes $A: A \in \text{sets } M$ **and** $B: B \in \text{sets } M$
shows $\text{emeasure } M A = \text{emeasure } M B$
using *assms* **by** (*safe intro!*: *antisym* *emeasure_mono_AE*) *auto*

lemma *emeasure_Collect_eq_AE*:
 $AE\ x\ \text{in}\ M.\ P\ x \longleftrightarrow Q\ x \implies \text{Measurable.pred } M\ Q \implies \text{Measurable.pred } M\ P$
 $\implies$
 $\text{emeasure } M\ \{x \in \text{space } M.\ P\ x\} = \text{emeasure } M\ \{x \in \text{space } M.\ Q\ x\}$
by (*intro* *emeasure_eq_AE*) *auto*

lemma *emeasure_eq_0_AE*: $AE\ x\ \text{in}\ M.\ \neg P\ x \implies \text{emeasure } M\ \{x \in \text{space } M.\ P\ x\} = 0$
using *AE_iff_measurable*[*OF_refl*, *of M* $\lambda x.\ \neg P\ x$]
by (*cases* $\{x \in \text{space } M.\ P\ x\} \in \text{sets } M$) (*simp_all* *add: emeasure_notin_sets*)

lemma *emeasure_0_AE*:
assumes $\text{emeasure } M\ (\text{space } M) = 0$
shows $AE\ x\ \text{in}\ M.\ P\ x$
using *eventually_ae_filter* *assms* **by** *blast*

lemma *emeasure_add_AE*:
assumes [*measurable*]: $A \in \text{sets } M\ B \in \text{sets } M\ C \in \text{sets } M$
assumes 1: $AE\ x\ \text{in}\ M.\ x \in C \longleftrightarrow x \in A \vee x \in B$
assumes 2: $AE\ x\ \text{in}\ M.\ \neg (x \in A \wedge x \in B)$
shows $\text{emeasure } M\ C = \text{emeasure } M\ A + \text{emeasure } M\ B$
proof –
have $\text{emeasure } M\ C = \text{emeasure } M\ (A \cup B)$
by (*rule* *emeasure_eq_AE*) (*insert* 1, *auto*)
also have $\dots = \text{emeasure } M\ A + \text{emeasure } M\ (B - A)$
by (*subst plus_emeasure*) *auto*
also have $\text{emeasure } M\ (B - A) = \text{emeasure } M\ B$
by (*rule* *emeasure_eq_AE*) (*insert* 2, *auto*)
finally show *?thesis* .
qed

6.3.7 σ -finite Measures

locale *sigma_finite_measure* =
fixes $M :: 'a\ \text{measure}$
assumes *sigma_finite_countable*:
 $\exists A :: 'a\ \text{set}.\ \text{countable } A \wedge A \subseteq \text{sets } M \wedge (\bigcup A) = \text{space } M \wedge (\forall a \in A.\ \text{emeasure } M\ a \neq \infty)$

lemma (*in sigma_finite_measure*) *sigma_finite*:
obtains $A :: \text{nat} \Rightarrow 'a\ \text{set}$
where $\text{range } A \subseteq \text{sets } M\ (\bigcup i.\ A\ i) = \text{space } M\ \bigwedge i.\ \text{emeasure } M\ (A\ i) \neq \infty$
proof –
obtain $A :: 'a\ \text{set set}$ **where**
 $[simp]: \text{countable } A$ **and**

```

  A:  $A \subseteq \text{sets } M$   $(\bigcup A) = \text{space } M \wedge a. a \in A \implies \text{emeasure } M a \neq \infty$ 
  using sigma_finite_countable by metis
show thesis
proof cases
  assume  $A = \{\}$  with  $\langle \bigcup A = \text{space } M \rangle$  show thesis
    by (intro that[of  $\lambda_. \{\}$ ]) auto
next
  assume  $A \neq \{\}$ 
  show thesis
  proof
    show  $\text{range } (\text{from\_nat\_into } A) \subseteq \text{sets } M$ 
      using  $\langle A \neq \{\} \rangle A$  by auto
    have  $(\bigcup i. \text{from\_nat\_into } A i) = \bigcup A$ 
      using  $\text{range\_from\_nat\_into}[OF \langle A \neq \{\} \rangle \langle \text{countable } A \rangle]$  by auto
    with  $A$  show  $(\bigcup i. \text{from\_nat\_into } A i) = \text{space } M$ 
      by auto
    qed (intro  $A$   $\text{from\_nat\_into } \langle A \neq \{\} \rangle$ )
  qed
qed

lemma (in sigma_finite_measure) sigma_finite_disjoint:
  obtains  $A :: \text{nat} \Rightarrow 'a \text{ set}$ 
  where  $\text{range } A \subseteq \text{sets } M$   $(\bigcup i. A i) = \text{space } M \wedge i. \text{emeasure } M (A i) \neq \infty$ 
disjoint_family  $A$ 
proof -
  obtain  $A :: \text{nat} \Rightarrow 'a \text{ set}$  where
    range:  $\text{range } A \subseteq \text{sets } M$  and
    space:  $(\bigcup i. A i) = \text{space } M$  and
    measure:  $\bigwedge i. \text{emeasure } M (A i) \neq \infty$ 
    using sigma_finite by blast
  show thesis
  proof (rule that[of disjointed  $A$ ])
    show  $\text{range } (\text{disjointed } A) \subseteq \text{sets } M$ 
      by (rule sets.range_disjointed_sets[OF range])
    show  $(\bigcup i. \text{disjointed } A i) = \text{space } M$ 
      and disjoint_family (disjointed  $A$ )
      using disjoint_family_disjointed UN_disjointed_eq[of  $A$ ] space range
      by auto
    show  $\text{emeasure } M (\text{disjointed } A i) \neq \infty$  for  $i$ 
  proof -
    have  $\text{emeasure } M (\text{disjointed } A i) \leq \text{emeasure } M (A i)$ 
      using range disjointed_subset[of  $A i$ ] by (auto intro!: emeasure_mono)
    then show ?thesis using measure[of  $i$ ] by (auto simp: top_unique)
  qed
  qed
qed
qed

```

```

lemma (in sigma_finite_measure) sigma_finite_incseq:
  obtains  $A :: \text{nat} \Rightarrow 'a \text{ set}$ 

```

```

where  $\text{range } A \subseteq \text{sets } M \ (\bigcup i. A \ i) = \text{space } M \ \bigwedge i. \text{emeasure } M \ (A \ i) \neq \infty$ 
 $\text{incseq } A$ 
proof –
  obtain  $F :: \text{nat} \Rightarrow 'a \text{ set}$  where
     $F: \text{range } F \subseteq \text{sets } M \ (\bigcup i. F \ i) = \text{space } M \ \bigwedge i. \text{emeasure } M \ (F \ i) \neq \infty$ 
    using  $\text{sigma\_finite}$  by  $\text{blast}$ 
  show  $\text{thesis}$ 
proof ( $\text{rule that[of } \lambda n. \bigcup i \leq n. F \ i]$ )
  show  $\text{range } (\lambda n. \bigcup i \leq n. F \ i) \subseteq \text{sets } M$ 
    using  $F$  by ( $\text{force simp: incseq\_def}$ )
  show  $(\bigcup n. \bigcup i \leq n. F \ i) = \text{space } M$ 
proof –
  from  $F$  have  $\bigwedge x. x \in \text{space } M \implies \exists i. x \in F \ i$  by  $\text{auto}$ 
  with  $F$  show  $?thesis$  by  $\text{fastforce}$ 
qed
show  $\text{emeasure } M \ (\bigcup i \leq n. F \ i) \neq \infty$  for  $n$ 
proof –
  have  $\text{emeasure } M \ (\bigcup i \leq n. F \ i) \leq (\sum i \leq n. \text{emeasure } M \ (F \ i))$ 
    using  $F$  by ( $\text{auto intro!: emeasure\_subadditive\_finite}$ )
  also have  $\dots < \infty$ 
    using  $F$  by ( $\text{auto simp: sum\_Pinfty less\_top}$ )
  finally show  $?thesis$  by  $\text{simp}$ 
qed
show  $\text{incseq } (\lambda n. \bigcup i \leq n. F \ i)$ 
  by ( $\text{force simp: incseq\_def}$ )
qed
qed

lemma (in  $\text{sigma\_finite\_measure}$ )  $\text{approx\_PInf\_emeasure\_with\_finite}$ :
  fixes  $C::\text{real}$ 
  assumes  $W\_meas: W \in \text{sets } M$ 
    and  $W\_inf: \text{emeasure } M \ W = \infty$ 
  obtains  $Z$  where  $Z \in \text{sets } M \ Z \subseteq W \ \text{emeasure } M \ Z < \infty \ \text{emeasure } M \ Z > C$ 
proof –
  obtain  $A :: \text{nat} \Rightarrow 'a \text{ set}$ 
    where  $A: \text{range } A \subseteq \text{sets } M \ (\bigcup i. A \ i) = \text{space } M \ \bigwedge i. \text{emeasure } M \ (A \ i) \neq \infty$ 
 $\text{incseq } A$ 
    using  $\text{sigma\_finite\_incseq}$  by  $\text{blast}$ 
  define  $B$  where  $B = (\lambda i. W \cap A \ i)$ 
  have  $B\_meas: \bigwedge i. B \ i \in \text{sets } M$  using  $W\_meas \langle \text{range } A \subseteq \text{sets } M \rangle B\_def$  by
 $\text{blast}$ 
  have  $b: \bigwedge i. B \ i \subseteq W$  using  $B\_def$  by  $\text{blast}$ 

  { fix  $i$ 
    have  $\text{emeasure } M \ (B \ i) \leq \text{emeasure } M \ (A \ i)$ 
      using  $A$  by ( $\text{intro emeasure\_mono}$ ) ( $\text{auto simp: } B\_def$ )
    also have  $\text{emeasure } M \ (A \ i) < \infty$ 
      using  $\langle \bigwedge i. \text{emeasure } M \ (A \ i) \neq \infty \rangle$  by ( $\text{simp add: less\_top}$ )
    finally have  $\text{emeasure } M \ (B \ i) < \infty . \}$ 

```

```

note  $c = this$ 

have  $W = (\bigcup i. B\ i)$  using  $B\_def \langle (\bigcup i. A\ i) = space\ M \rangle$   $W\_meas$  by auto
moreover have  $incseq\ B$  using  $B\_def \langle incseq\ A \rangle$  by (simp add: incseq_def
subset_eq)
ultimately have  $(\lambda i. emeasure\ M\ (B\ i)) \longrightarrow emeasure\ M\ W$  using  $W\_meas$ 
 $B\_meas$ 
by (simp add: B_meas Lim_emeasure_incseq_image_subset_iff)
then have  $(\lambda i. emeasure\ M\ (B\ i)) \longrightarrow \infty$  using  $W\_inf$  by simp
from order_tendstoD(1)[OF this, of C]
obtain  $i$  where  $d: emeasure\ M\ (B\ i) > C$ 
by (auto simp: eventually_sequentially)
have  $B\ i \in sets\ M$   $B\ i \subseteq W$   $emeasure\ M\ (B\ i) < \infty$   $emeasure\ M\ (B\ i) > C$ 
using  $B\_meas\ b\ c\ d$  by auto
then show ?thesis using that by blast
qed

```

6.3.8 Measure space induced by distribution of $(\rightarrow_M)$ -functions

definition $distr :: 'a\ measure \Rightarrow 'b\ measure \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b\ measure$ **where**
 $distr\ M\ N\ f =$
 $measure\_of\ (space\ N)\ (sets\ N)\ (\lambda A. emeasure\ M\ (f - ' A \cap space\ M))$

lemma

shows $sets\_distr[simp, measurable\_cong]: sets\ (distr\ M\ N\ f) = sets\ N$
and $space\_distr[simp]: space\ (distr\ M\ N\ f) = space\ N$
by (*auto simp: distr_def*)

lemma

shows $measurable\_distr\_eq1[simp]: measurable\ (distr\ Mf\ Nf\ f)\ Mf' = measurable\ Nf\ Mf'$
and $measurable\_distr\_eq2[simp]: measurable\ Mg'\ (distr\ Mg\ Ng\ g) = measurable\ Mg'\ Ng$
by (*auto simp: measurable_def*)

lemma $distr\_cong:$

$M = K \Longrightarrow sets\ N = sets\ L \Longrightarrow (\bigwedge x. x \in space\ M \Longrightarrow f\ x = g\ x) \Longrightarrow distr\ M\ N\ f = distr\ K\ L\ g$
using $sets\_eq\_imp\_space\_eq[of\ N\ L]$ **by** (*simp add: distr_def Int_def cong: rev_conj_cong*)

lemma $emeasure\_distr:$

fixes $f :: 'a \Rightarrow 'b$
assumes $f: f \in measurable\ M\ N$ **and** $A: A \in sets\ N$
shows $emeasure\ (distr\ M\ N\ f)\ A = emeasure\ M\ (f - ' A \cap space\ M)$ (**is** $\_ = ?\mu\ A$)
unfolding $distr\_def$
proof (*rule emeasure_measure_of_sigma*)
show *positive* $(sets\ N)\ ?\mu$

```

    by (auto simp: positive_def)

show countably_additive (sets N) ?μ
proof (intro countably_additiveI)
  fix A :: nat ⇒ 'b set assume range A ⊆ sets N disjoint_family A
  then have A: ∧i. A i ∈ sets N (⋃i. A i) ∈ sets N by auto
  then have *: range (λi. f -' (A i) ∩ space M) ⊆ sets M
    using f by (auto simp: measurable_def)
  moreover have (⋃i. f -' A i ∩ space M) ∈ sets M
    using * by blast
  moreover have **: disjoint_family (λi. f -' A i ∩ space M)
    using ‹disjoint_family A› by (auto simp: disjoint_family_on_def)
  ultimately show (∑i. ?μ (A i)) = ?μ (⋃i. A i)
    using suminf_emeasure[OF_**] A f
    by (auto simp: comp_def vimage_UN)
qed
show sigma_algebra (space N) (sets N) ..
qed fact

lemma emeasure_Collect_distr:
  assumes X[measurable]: X ∈ measurable M N Measurable.pred N P
  shows emeasure (distr M N X) {x∈space N. P x} = emeasure M {x∈space M.
P (X x)}
  by (subst emeasure_distr)
    (auto intro!: arg_cong2[where f=emeasure] X(1)[THEN measurable_space])

lemma emeasure_lfp2[consumes 1, case_names cont f measurable]:
  assumes P M
  assumes cont: sup_continuous F
  assumes f: ∧M. P M ⇒ f ∈ measurable M' M
  assumes *: ∧M A. P M ⇒ (∧N. P N ⇒ Measurable.pred N A) ⇒ Measurable.pred M (F A)
  shows emeasure M' {x∈space M'. lfp F (f x)} = (SUP i. emeasure M' {x∈space
M'. (F ~ i) (λx. False) (f x)})
proof (subst (1 2) emeasure_Collect_distr[symmetric, where X=f])
  show f ∈ measurable M' M f ∈ measurable M' M
    using f[OF ‹P M›] by auto
  { fix i show Measurable.pred M ((F ~ i) (λx. False))
    using ‹P M› by (induction i arbitrary: M) (auto intro!: *) }
  show Measurable.pred M (lfp F)
    using ‹P M› cont * by (rule measurable_lfp_coinduct[of P])

  have emeasure (distr M' M f) {x ∈ space (distr M' M f). lfp F x} =
    (SUP i. emeasure (distr M' M f) {x ∈ space (distr M' M f). (F ~ i) (λx.
False) x})
    using ‹P M›
  proof (coinduction arbitrary: M rule: emeasure_lfp')
    case (measurable A N) then have ∧N. P N ⇒ Measurable.pred (distr M' N
f) A

```

```

    by metis
  then have  $\bigwedge N. P\ N \implies \text{Measurable.pred}\ N\ A$ 
    by simp
  with  $\langle P\ N \rangle [THEN\ *]$  show ?case
    by auto
qed fact
then show  $\text{emeasure}\ (\text{distr}\ M'\ M\ f)\ \{x \in \text{space}\ M. \text{lf}\!p\ F\ x\} =$ 
   $(\text{SUP}\ i. \text{emeasure}\ (\text{distr}\ M'\ M\ f)\ \{x \in \text{space}\ M. (F\ \frown\ i)\ (\lambda x. \text{False})\ x\})$ 
    by simp
qed

lemma distr_id[simp]:  $\text{distr}\ N\ N\ (\lambda x. x) = N$ 
  by (rule measure_eqI) (auto simp: emeasure_distr)

lemma distr_id2:  $\text{sets}\ M = \text{sets}\ N \implies \text{distr}\ N\ M\ (\lambda x. x) = N$ 
  by (rule measure_eqI) (auto simp: emeasure_distr)

lemma measure_distr:
   $f \in \text{measurable}\ M\ N \implies S \in \text{sets}\ N \implies \text{measure}\ (\text{distr}\ M\ N\ f)\ S = \text{measure}\ M$ 
   $(f - ' S \cap \text{space}\ M)$ 
  by (simp add: emeasure_distr measure_def)

lemma distr_cong_AE:
  assumes 1:  $M = K\ \text{sets}\ N = \text{sets}\ L$  and
    2:  $(AE\ x\ \text{in}\ M. f\ x = g\ x)$  and  $f \in \text{measurable}\ M\ N$  and  $g \in \text{measurable}\ K\ L$ 
  shows  $\text{distr}\ M\ N\ f = \text{distr}\ K\ L\ g$ 
proof (rule measure_eqI)
  fix A assume  $A \in \text{sets}\ (\text{distr}\ M\ N\ f)$ 
  with assms show  $\text{emeasure}\ (\text{distr}\ M\ N\ f)\ A = \text{emeasure}\ (\text{distr}\ K\ L\ g)\ A$ 
    by (auto simp add: emeasure_distr intro!: emeasure_eq_AE measurable_sets)
qed (insert 1, simp)

lemma AE_distrD:
  assumes  $f: f \in \text{measurable}\ M\ M'$ 
    and  $AE: AE\ x\ \text{in}\ \text{distr}\ M\ M'\ f. P\ x$ 
  shows  $AE\ x\ \text{in}\ M. P\ (f\ x)$ 
proof -
  from  $AE [THEN\ AE\_E]$  obtain N
    where  $\{x \in \text{space}\ (\text{distr}\ M\ M'\ f). \neg P\ x\} \subseteq N$ 
    and  $\text{emeasure}\ (\text{distr}\ M\ M'\ f)\ N = 0$ 
    and  $N \in \text{sets}\ (\text{distr}\ M\ M'\ f)$ 
  by auto
  with f show ?thesis
    by (simp add: eventually_ae_filter, intro bexI[of  $\_ f - ' N \cap \text{space}\ M$ ])
    (auto simp: emeasure_distr measure_def)
qed

lemma AE_distr_iff:
  assumes  $f[\text{measurable}]: f \in \text{measurable}\ M\ N$  and  $P[\text{measurable}]: \{x \in \text{space}\ N.$ 

```

```

P x} ∈ sets N
shows (AE x in distr M N f. P x) ⟷ (AE x in M. P (f x))
proof (subst (1 2) AE_iff_measurable[OF _ refl])
  have f - ' {x ∈ space N. ¬ P x} ∩ space M = {x ∈ space M. ¬ P (f x)}
    using f[THEN measurable_space] by auto
  then show (emeasure (distr M N f) {x ∈ space (distr M N f). ¬ P x} = 0) =
    (emeasure M {x ∈ space M. ¬ P (f x)} = 0)
    by (simp add: emeasure_distr)
qed auto

```

```

lemma null_sets_distr_iff:
  f ∈ measurable M N ⟹ A ∈ null_sets (distr M N f) ⟷ f - ' A ∩ space M ∈
  null_sets M ∧ A ∈ sets N
  by (auto simp add: null_sets_def emeasure_distr)

```

```

proposition distr_distr:
  g ∈ measurable N L ⟹ f ∈ measurable M N ⟹ distr (distr M N f) L g = distr
  M L (g ∘ f)
  by (auto simp add: emeasure_distr measurable_space
    intro!: arg_cong[where f=emeasure M] measure_eqI)

```

6.3.9 Real measure values

```

lemma ring_of_finite_sets: ring_of_sets (space M) {A ∈ sets M. emeasure M A
  ≠ top}
proof (rule ring_of_setsI)
  show a ∈ {A ∈ sets M. emeasure M A ≠ top} ⟹ b ∈ {A ∈ sets M. emeasure
  M A ≠ top} ⟹
    a ∪ b ∈ {A ∈ sets M. emeasure M A ≠ top} for a b
    using emeasure_subadditive[of a M b] by (auto simp: top_unique)

  show a ∈ {A ∈ sets M. emeasure M A ≠ top} ⟹ b ∈ {A ∈ sets M. emeasure
  M A ≠ top} ⟹
    a - b ∈ {A ∈ sets M. emeasure M A ≠ top} for a b
    using emeasure_mono[of a - b a M] by (auto simp: top_unique)
qed (auto dest: sets_sets_into_space)

```

```

lemma measure_nonneg[simp]: 0 ≤ measure M A
  unfolding measure_def by auto

```

```

lemma measure_nonneg' [simp]: ¬ measure M A < 0
  using measure_nonneg not_le by blast

```

```

lemma zero_less_measure_iff: 0 < measure M A ⟷ measure M A ≠ 0
  using measure_nonneg[of M A] by (auto simp add: le_less)

```

```

lemma measure_le_0_iff: measure M X ≤ 0 ⟷ measure M X = 0
  using measure_nonneg[of M X] by linarith

```


lemma *measure_empty[simp]*: $\text{measure } M \ \{\} = 0$
unfolding *measure_def* **by** (*simp add: zero_enreal.rep_eq*)

lemma *emeasure_eq_enreal_measure*:
 $\text{emeasure } M \ A \neq \text{top} \implies \text{emeasure } M \ A = \text{enreal } (\text{measure } M \ A)$
by (*cases emeasure M A rule: enreal_cases*) (*auto simp: measure_def*)

lemma *measure_zero_top*: $\text{emeasure } M \ A = \text{top} \implies \text{measure } M \ A = 0$
by (*simp add: measure_def*)

lemma *measure_eq_emeasure_eq_enreal*: $0 \leq x \implies \text{emeasure } M \ A = \text{enreal } x \implies \text{measure } M \ A = x$
using *emeasure_eq_enreal_measure[of M A]*
by (*cases A ∈ M*) (*auto simp: measure_notin_sets emeasure_notin_sets*)

lemma *enn2real_plus*: $a < \text{top} \implies b < \text{top} \implies \text{enn2real } (a + b) = \text{enn2real } a + \text{enn2real } b$
by (*simp add: enn2real_def plus_enreal.rep_eq real_of_ereal_add less_top del: real_of_ereal_enn2ereal*)

lemma *enn2real_sum*: $(\bigwedge i. i \in I \implies f \ i < \text{top}) \implies \text{enn2real } (\text{sum } f \ I) = \text{sum } (\text{enn2real } \circ f) \ I$
by (*induction I rule: infinite_finite_induct*) (*auto simp: enn2real_plus*)

lemma *measure_eq_AE*:
assumes *iff*: $AE \ x \text{ in } M. x \in A \longleftrightarrow x \in B$
assumes *A*: $A \in \text{sets } M$ **and** *B*: $B \in \text{sets } M$
shows $\text{measure } M \ A = \text{measure } M \ B$
using *assms emeasure_eq_AE[OF assms]* **by** (*simp add: measure_def*)

lemma *measure_Union*:
 $\text{emeasure } M \ A \neq \infty \implies \text{emeasure } M \ B \neq \infty \implies A \in \text{sets } M \implies B \in \text{sets } M \implies A \cap B = \{\} \implies$
 $\text{measure } M \ (A \cup B) = \text{measure } M \ A + \text{measure } M \ B$
by (*simp add: measure_def plus_emeasure[symmetric] enn2real_plus less_top*)

lemma *measure_finite_Union*:
 $\text{finite } S \implies A' S \subseteq \text{sets } M \implies \text{disjoint\_family\_on } A \ S \implies (\bigwedge i. i \in S \implies \text{emeasure } M \ (A \ i) \neq \infty) \implies$
 $\text{measure } M \ (\bigcup_{i \in S} A \ i) = (\sum_{i \in S} \text{measure } M \ (A \ i))$
by (*induction S rule: finite_induct*)
(auto simp: disjoint_family_on_insert measure_Union sum_emeasure[symmetric] sets.countable_UN'[OF countable_finite])

lemma *measure_Diff*:
assumes *finite*: $\text{emeasure } M \ A \neq \infty$
and *measurable*: $A \in \text{sets } M \ B \in \text{sets } M \ B \subseteq A$
shows $\text{measure } M \ (A - B) = \text{measure } M \ A - \text{measure } M \ B$
proof –

```

have emeasure M (A - B) ≤ emeasure M A emeasure M B ≤ emeasure M A
  using measurable by (auto intro!: emeasure_mono)
hence measure M ((A - B) ∪ B) = measure M (A - B) + measure M B
  using measurable finite by (rule_tac measure_Union) (auto simp: top_unique)
thus ?thesis using ‹B ⊆ A› by (auto simp: Un_absorb2)
qed

```

lemma *measure_UNION*:

```

assumes measurable: range A ⊆ sets M disjoint_family A
assumes finite: emeasure M (⋃ i. A i) ≠ ∞
shows (λ i. measure M (A i)) sums (measure M (⋃ i. A i))
proof -
  have (λ i. emeasure M (A i)) sums (emeasure M (⋃ i. A i))
    unfolding suminf_emeasure[OF measurable, symmetric] by (simp add: summable_sums)
  moreover
  { fix i
    have emeasure M (A i) ≤ emeasure M (⋃ i. A i)
      using measurable by (auto intro!: emeasure_mono)
    then have emeasure M (A i) = ennreal ((measure M (A i)))
      using finite by (intro emeasure_eq_ennreal_measure) (auto simp: top_unique)
  }
  ultimately show ?thesis using finite
    by (subst (asm) (2) emeasure_eq_ennreal_measure) simp_all
qed

```

lemma *measure_subadditive*:

```

assumes measurable: A ∈ sets M B ∈ sets M
and fin: emeasure M A ≠ ∞ emeasure M B ≠ ∞
shows measure M (A ∪ B) ≤ measure M A + measure M B
proof -
  have emeasure M (A ∪ B) ≠ ∞
    using emeasure_subadditive[OF measurable] fin by (auto simp: top_unique)
  then show (measure M (A ∪ B)) ≤ (measure M A) + (measure M B)
    using emeasure_subadditive[OF measurable] fin
    apply simp
    apply (subst (asm) (2 3 4) emeasure_eq_ennreal_measure)
    apply (auto simp flip: ennreal_plus)
    done
qed

```

lemma *measure_subadditive_finite*:

```

assumes A: finite I A'I ⊆ sets M and fin: ⋀ i. i ∈ I ⇒ emeasure M (A i) ≠
∞
shows measure M (⋃ i ∈ I. A i) ≤ (∑ i ∈ I. measure M (A i))
proof -
  { have emeasure M (⋃ i ∈ I. A i) ≤ (∑ i ∈ I. emeasure M (A i))
    using emeasure_subadditive_finite[OF A] .
  }
  also have ... < ∞
    using fin by (simp add: less_top A)

```

```

    finally have  $\text{emeasure } M (\bigcup_{i \in I}. A \ i) \neq \text{top}$  by simp }
  note * = this
  show ?thesis
    using  $\text{emeasure\_subadditive\_finite}[OF A]$  fin
    unfolding  $\text{emeasure\_eq\_ennreal\_measure}[OF *]$ 
    by (simp add: sum_nonneg  $\text{emeasure\_eq\_ennreal\_measure}$ )
qed

lemma  $\text{measure\_subadditive\_countably}$ :
  assumes  $A: \text{range } A \subseteq \text{sets } M$  and fin:  $(\sum i. \text{emeasure } M (A \ i)) \neq \infty$ 
  shows  $\text{measure } M (\bigcup i. A \ i) \leq (\sum i. \text{measure } M (A \ i))$ 
proof -
  from fin have **:  $\bigwedge i. \text{emeasure } M (A \ i) \neq \text{top}$ 
    using  $\text{ennreal\_suminf\_lessD}[of \lambda i. \text{emeasure } M (A \ i)]$  by (simp add: less_top)
  { have  $\text{emeasure } M (\bigcup i. A \ i) \leq (\sum i. \text{emeasure } M (A \ i))$ 
    using  $\text{emeasure\_subadditive\_countably}[OF A]$  .
    also have  $\dots < \infty$ 
      using fin by (simp add: less_top)
    finally have  $\text{emeasure } M (\bigcup i. A \ i) \neq \text{top}$  by simp }
  then have  $\text{ennreal } (\text{measure } M (\bigcup i. A \ i)) = \text{emeasure } M (\bigcup i. A \ i)$ 
    by (rule  $\text{emeasure\_eq\_ennreal\_measure}[symmetric]$ )
  also have  $\dots \leq (\sum i. \text{emeasure } M (A \ i))$ 
    using  $\text{emeasure\_subadditive\_countably}[OF A]$  .
  also have  $\dots = \text{ennreal } (\sum i. \text{measure } M (A \ i))$ 
    using fin unfolding  $\text{emeasure\_eq\_ennreal\_measure}[OF **]$ 
    by (subst  $\text{suminf\_ennreal}$ ) (auto simp: **)
  finally show ?thesis
    apply (rule  $\text{ennreal\_le\_iff}[THEN \text{iffD1}, rotated]$ )
    apply (intro  $\text{suminf\_nonneg allI measure\_nonneg summable\_suminf\_not\_top}$ )
    using fin
    apply (simp add:  $\text{emeasure\_eq\_ennreal\_measure}[OF **]$ )
    done
qed

lemma  $\text{measure\_Un\_null\_set}$ :  $A \in \text{sets } M \implies B \in \text{null\_sets } M \implies \text{measure } M (A \cup B) = \text{measure } M A$ 
  by (simp add:  $\text{measure\_def emeasure\_Un\_null\_set}$ )

lemma  $\text{measure\_Diff\_null\_set}$ :  $A \in \text{sets } M \implies B \in \text{null\_sets } M \implies \text{measure } M (A - B) = \text{measure } M A$ 
  by (simp add:  $\text{measure\_def emeasure\_Diff\_null\_set}$ )

lemma  $\text{measure\_eq\_sum\_singleton}$ :
  finite  $S \implies (\bigwedge x. x \in S \implies \{x\} \in \text{sets } M) \implies (\bigwedge x. x \in S \implies \text{emeasure } M \{x\} \neq \infty) \implies$ 
 $\text{measure } M S = (\sum_{x \in S}. \text{measure } M \{x\})$ 
  using  $\text{emeasure\_eq\_sum\_singleton}[of S M]$ 
  by (intro  $\text{measure\_eq\_emeasure\_eq\_ennreal}$ ) (auto simp: sum_nonneg  $\text{emeasure\_eq\_ennreal\_measure}$ )

```

lemma *Lim_measure_incseq*:
assumes $A: \text{range } A \subseteq \text{sets } M \text{ incseq } A$ **and** $\text{fin}: \text{emeasure } M (\bigcup i. A i) \neq \infty$
shows $(\lambda i. \text{measure } M (A i)) \longrightarrow \text{measure } M (\bigcup i. A i)$
proof (*rule tendsto_ennrealD*)
have $\text{ennreal } (\text{measure } M (\bigcup i. A i)) = \text{emeasure } M (\bigcup i. A i)$
using *fin* **by** (*auto simp: emeasure_eq_ennreal_measure*)
moreover **have** $\text{ennreal } (\text{measure } M (A i)) = \text{emeasure } M (A i)$ **for** i
using *assms emeasure_mono[of A _ $\bigcup i. A i$ M]*
by (*intro emeasure_eq_ennreal_measure[symmetric] (auto simp: less_top UN_upper intro: le_less_trans)*)
ultimately show $(\lambda x. \text{ennreal } (\text{measure } M (A x))) \longrightarrow \text{ennreal } (\text{measure } M (\bigcup i. A i))$
using A **by** (*auto intro!: Lim_emeasure_incseq*)
qed *auto*

lemma *Lim_measure_decseq*:
assumes $A: \text{range } A \subseteq \text{sets } M \text{ decseq } A$ **and** $\text{fin}: \bigwedge i. \text{emeasure } M (A i) \neq \infty$
shows $(\lambda n. \text{measure } M (A n)) \longrightarrow \text{measure } M (\bigcap i. A i)$
proof (*rule tendsto_ennrealD*)
have $\text{ennreal } (\text{measure } M (\bigcap i. A i)) = \text{emeasure } M (\bigcap i. A i)$
using *fin[of 0] A emeasure_mono[of $\bigcap i. A i$ A 0 M]*
by (*auto intro!: emeasure_eq_ennreal_measure[symmetric] simp: INT_lower less_top intro: le_less_trans*)
moreover **have** $\text{ennreal } (\text{measure } M (A i)) = \text{emeasure } M (A i)$ **for** i
using A *fin[of i]* **by** (*intro emeasure_eq_ennreal_measure[symmetric] auto*)
ultimately show $(\lambda x. \text{ennreal } (\text{measure } M (A x))) \longrightarrow \text{ennreal } (\text{measure } M (\bigcap i. A i))$
using *fin* A **by** (*auto intro!: Lim_emeasure_decseq*)
qed *auto*

6.3.10 Set of measurable sets with finite measure

definition *fmeasurable* :: $'a \text{ measure} \Rightarrow 'a \text{ set set}$ **where**
 $\text{fmeasurable } M = \{A \in \text{sets } M. \text{emeasure } M A < \infty\}$

lemma *fmeasurableD[dest, measurable_dest]*: $A \in \text{fmeasurable } M \implies A \in \text{sets } M$
by (*auto simp: fmeasurable_def*)

lemma *fmeasurableD2*: $A \in \text{fmeasurable } M \implies \text{emeasure } M A \neq \text{top}$
by (*auto simp: fmeasurable_def*)

lemma *fmeasurableI*: $A \in \text{sets } M \implies \text{emeasure } M A < \infty \implies A \in \text{fmeasurable } M$
by (*auto simp: fmeasurable_def*)

lemma *fmeasurableI_null_sets*: $A \in \text{null\_sets } M \implies A \in \text{fmeasurable } M$
by (*auto simp: fmeasurable_def*)

lemma *fmeasurableI2*: $A \in \text{fmeasurable } M \implies B \subseteq A \implies B \in \text{sets } M \implies B \in \text{fmeasurable } M$

using *emeasure_mono*[of $B \ A \ M$] **by** (auto simp: *fmeasurable_def*)

lemma *measure_mono_fmeasurable*:

$A \subseteq B \implies A \in \text{sets } M \implies B \in \text{fmeasurable } M \implies \text{measure } M \ A \leq \text{measure } M \ B$

by (auto simp: *measure_def fmeasurable_def* intro!: *emeasure_mono enn2real_mono*)

lemma *emeasure_eq_measure2*: $A \in \text{fmeasurable } M \implies \text{emeasure } M \ A = \text{measure } M \ A$

by (simp add: *emeasure_eq_ennreal_measure fmeasurable_def less_top*)

interpretation *fmeasurable*: *ring_of_sets space M fmeasurable M*

proof (rule *ring_of_setsI*)

show $\text{fmeasurable } M \subseteq \text{Pow } (\text{space } M) \ \{\} \in \text{fmeasurable } M$

by (auto simp: *fmeasurable_def* dest: *sets.sets_into_space*)

fix $a \ b$ **assume** *: $a \in \text{fmeasurable } M \ b \in \text{fmeasurable } M$

then have $\text{emeasure } M \ (a \cup b) \leq \text{emeasure } M \ a + \text{emeasure } M \ b$

by (intro *emeasure_subadditive*) auto

also have $\dots < \text{top}$

using * **by** (auto simp: *fmeasurable_def*)

finally show $a \cup b \in \text{fmeasurable } M$

using * **by** (auto intro: *fmeasurableI*)

show $a - b \in \text{fmeasurable } M$

using *emeasure_mono*[of $a - b \ a \ M$] * **by** (auto simp: *fmeasurable_def*)

qed

6.3.11 Measurable sets formed by unions and intersections

lemma *fmeasurable_Diff*: $A \in \text{fmeasurable } M \implies B \in \text{sets } M \implies A - B \in \text{fmeasurable } M$

using *fmeasurableI2*[of $A \ M \ A - B$] **by** auto

lemma *fmeasurable_Int_fmeasurable*:

$\llbracket S \in \text{fmeasurable } M; T \in \text{sets } M \rrbracket \implies (S \cap T) \in \text{fmeasurable } M$

by (meson *fmeasurableD fmeasurableI2 inf_le1 sets.Int*)

lemma *fmeasurable_UN*:

assumes *countable I* $\bigwedge i. i \in I \implies F \ i \subseteq A \ \bigwedge i. i \in I \implies F \ i \in \text{sets } M \ A \in \text{fmeasurable } M$

shows $(\bigcup i \in I. F \ i) \in \text{fmeasurable } M$

proof (rule *fmeasurableI2*)

show $A \in \text{fmeasurable } M \ (\bigcup i \in I. F \ i) \subseteq A$ **using** *assms* **by** auto

show $(\bigcup i \in I. F \ i) \in \text{sets } M$

using *assms* **by** (intro *sets.countable_UN'*) auto

qed

lemma *fmeasurable_INT*:

```

    assumes countable I i ∈ I ∧ i. i ∈ I ⇒ F i ∈ sets M F i ∈ fmeasurable M
    shows (⋂ i∈I. F i) ∈ fmeasurable M
  proof (rule fmeasurableI2)
    show F i ∈ fmeasurable M (⋂ i∈I. F i) ⊆ F i
      using assms by auto
    show (⋂ i∈I. F i) ∈ sets M
      using assms by (intro sets.countable_INT') auto
  qed

```

```

lemma measurable_measure_Diff:
  assumes A ∈ fmeasurable M B ∈ sets M B ⊆ A
  shows measure M (A - B) = measure M A - measure M B
  by (simp add: assms fmeasurableD fmeasurableD2 measure_Diff)

```

```

lemma measurable_Un_null_set:
  assumes B ∈ null_sets M
  shows (A ∪ B ∈ fmeasurable M ∧ A ∈ sets M) ⟷ A ∈ fmeasurable M
    using assms by (fastforce simp add: fmeasurable.Un fmeasurableI_null_sets
intro: fmeasurableI2)

```

```

lemma measurable_Diff_null_set:
  assumes B ∈ null_sets M
  shows (A - B) ∈ fmeasurable M ∧ A ∈ sets M ⟷ A ∈ fmeasurable M
    using assms
  by (metis Un_Diff_cancel2 fmeasurable.Diff fmeasurableD fmeasurableI_null_sets
measurable_Un_null_set)

```

```

lemma fmeasurable_Diff_D:
  assumes m: T - S ∈ fmeasurable M S ∈ fmeasurable M and sub: S ⊆ T
  shows T ∈ fmeasurable M
  proof -
    have T = S ∪ (T - S)
      using assms by blast
    then show ?thesis
      by (metis m fmeasurable.Un)
  qed

```

```

lemma measure_Un2:
  A ∈ fmeasurable M ⇒ B ∈ fmeasurable M ⇒ measure M (A ∪ B) = measure
M A + measure M (B - A)
  using measure_Union[of M A B - A] by (auto simp: fmeasurableD2 fmeasur-
able.Diff)

```

```

lemma measure_Un3:
  assumes A ∈ fmeasurable M B ∈ fmeasurable M
  shows measure M (A ∪ B) = measure M A + measure M B - measure M (A
∩ B)
  proof -
    have measure M (A ∪ B) = measure M A + measure M (B - A)

```

```

    using assms by (rule measure_Un2)
  also have  $B - A = B - (A \cap B)$ 
    by auto
  also have  $\text{measure } M (B - (A \cap B)) = \text{measure } M B - \text{measure } M (A \cap B)$ 
    using assms by (intro measure_Diff) (auto simp: fmeasurable_def)
  finally show ?thesis
    by simp
qed

```

lemma *measure_Un_AE*:

AE x in M. $x \notin A \vee x \notin B \implies A \in \text{fmeasurable } M \implies B \in \text{fmeasurable } M \implies$
 $\text{measure } M (A \cup B) = \text{measure } M A + \text{measure } M B$
by (subst measure_Un2) (auto intro!: measure_eq_AE)

lemma *measure_UNION_AE*:

assumes *I: finite I*
shows $(\bigwedge i. i \in I \implies F i \in \text{fmeasurable } M) \implies \text{pairwise } (\lambda i j. \text{AE } x \text{ in } M. x \notin$
 $F i \vee x \notin F j) I \implies$
 $\text{measure } M (\bigcup i \in I. F i) = (\sum i \in I. \text{measure } M (F i))$
unfolding *AE_pairwise[OF countable_finite, OF I]*
using *I*
proof (induction I rule: finite_induct)
case (insert x I)
have $\text{measure } M (F x \cup \bigcup (F ` I)) = \text{measure } M (F x) + \text{measure } M (\bigcup (F `$
 $I))$
by (rule measure_Un_AE) (use insert in <auto simp: pairwise_insert>)
with insert **show** ?case
by (simp add: pairwise_insert)
qed simp

lemma *measure_UNION'*:

finite I $\implies (\bigwedge i. i \in I \implies F i \in \text{fmeasurable } M) \implies \text{pairwise } (\lambda i j. \text{disjnt } (F$
 $i) (F j)) I \implies$
 $\text{measure } M (\bigcup i \in I. F i) = (\sum i \in I. \text{measure } M (F i))$
by (intro measure_UNION_AE) (auto simp: disjnt_def elim!: pairwise_mono
intro!: always_eventually)

lemma *measure_Union_AE*:

finite F $\implies (\bigwedge S. S \in F \implies S \in \text{fmeasurable } M) \implies \text{pairwise } (\lambda S T. \text{AE } x \text{ in}$
 $M. x \notin S \vee x \notin T) F \implies$
 $\text{measure } M (\bigcup F) = (\sum S \in F. \text{measure } M S)$
using *measure_UNION_AE[of F $\lambda x. x M$]* **by** simp

lemma *measure_Union'*:

finite F $\implies (\bigwedge S. S \in F \implies S \in \text{fmeasurable } M) \implies \text{pairwise disjnt } F \implies$
 $\text{measure } M (\bigcup F) = (\sum S \in F. \text{measure } M S)$
using *measure_UNION'[of F $\lambda x. x M$]* **by** simp

lemma *measure_Un_le*:

assumes $A \in \text{sets } M \ B \in \text{sets } M$ **shows** $\text{measure } M \ (A \cup B) \leq \text{measure } M \ A + \text{measure } M \ B$
proof *cases*
assume $A \in \text{fmeasurable } M \wedge B \in \text{fmeasurable } M$
with $\text{measure\_subadditive}[\text{of } A \ M \ B]$ *assms* **show** *?thesis*
by (*auto simp: fmeasurableD2*)
next
assume $\neg (A \in \text{fmeasurable } M \wedge B \in \text{fmeasurable } M)$
then have $A \cup B \notin \text{fmeasurable } M$
using $\text{fmeasurableI2}[\text{of } A \cup B \ M \ A]$ $\text{fmeasurableI2}[\text{of } A \cup B \ M \ B]$ *assms* **by** *auto*
with *assms* **show** *?thesis*
by (*auto simp: fmeasurable_def measure_def less_top[symmetric]*)
qed

lemma measure_UNION_le :
 $\text{finite } I \implies (\bigwedge i. i \in I \implies F \ i \in \text{sets } M) \implies \text{measure } M \ (\bigcup_{i \in I}. F \ i) \leq (\sum_{i \in I}. \text{measure } M \ (F \ i))$
proof (*induction I rule: finite_induct*)
case (*insert i I*)
then have $\text{measure } M \ (\bigcup_{i \in \text{insert } i \ I}. F \ i) = \text{measure } M \ (F \ i \cup \bigcup (F \ ' \ I))$
by *simp*
also from *insert* **have** $\text{measure } M \ (F \ i \cup \bigcup (F \ ' \ I)) \leq \text{measure } M \ (F \ i) + \text{measure } M \ (\bigcup (F \ ' \ I))$
by (*intro measure_Un_le sets.finite_Union*) *auto*
also have $\text{measure } M \ (\bigcup_{i \in I}. F \ i) \leq (\sum_{i \in I}. \text{measure } M \ (F \ i))$
using *insert* **by** *auto*
finally show *?case*
using *insert* **by** *simp*
qed *simp*

lemma measure_Union_le :
 $\text{finite } F \implies (\bigwedge S. S \in F \implies S \in \text{sets } M) \implies \text{measure } M \ (\bigcup F) \leq (\sum_{S \in F}. \text{measure } M \ S)$
using $\text{measure\_UNION\_le}[\text{of } F \ \lambda x. x \ M]$ **by** *simp*

Version for indexed union over a countable set

lemma
assumes *countable I* **and** $I: \bigwedge i. i \in I \implies A \ i \in \text{fmeasurable } M$
and *bound*: $\bigwedge I'. I' \subseteq I \implies \text{finite } I' \implies \text{measure } M \ (\bigcup_{i \in I'}. A \ i) \leq B$
shows $\text{fmeasurable\_UN\_bound}$: $(\bigcup_{i \in I}. A \ i) \in \text{fmeasurable } M$ (**is** *?fm*)
and measure_UN_bound : $\text{measure } M \ (\bigcup_{i \in I}. A \ i) \leq B$ (**is** *?m*)
proof –
have $B \geq 0$
using *bound* **by** *force*
have *?fm* $\wedge$ *?m*
proof *cases*
assume $I = \{\}$
with $\langle B \geq 0 \rangle$ **show** *?thesis*


```

    by simp
  next
    assume  $I \neq \{\}$ 
    have  $(\bigcup_{i \in I}. A \ i) = (\bigcup_{i. (\bigcup_{n \leq i}. A \ (from\_nat\_into \ I \ n)))$ 
      by (subst range_from_nat_into[symmetric, OF  $\langle I \neq \{\} \rangle \langle countable \ I \rangle$ ]) auto
    then have  $emeasure \ M \ (\bigcup_{i \in I}. A \ i) = emeasure \ M \ (\bigcup_{i. (\bigcup_{n \leq i}. A \ (from\_nat\_into \ I \ n)))$  by simp
    also have  $\dots = (SUP \ i. emeasure \ M \ (\bigcup_{n \leq i}. A \ (from\_nat\_into \ I \ n)))$ 
      using  $I \neq \{\}$  [THEN from_nat_into] by (intro SUP_emeasure_incseq[symmetric])
    (fastforce simp: incseq_Suc_iff)+
    also have  $\dots \leq B$ 
    proof (intro SUP_least)
      fix  $i :: nat$ 
      have  $emeasure \ M \ (\bigcup_{n \leq i}. A \ (from\_nat\_into \ I \ n)) = measure \ M \ (\bigcup_{n \leq i}. A \ (from\_nat\_into \ I \ n))$ 
      using  $I \neq \{\}$  [THEN from_nat_into] by (intro emeasure_eq_measure2 fmeasurable.finite_UN) auto
      also have  $\dots = measure \ M \ (\bigcup_{n \in from\_nat\_into \ I \ \{\dots i\}. A \ n)$ 
        by simp
      also have  $\dots \leq B$ 
        by (intro ennreal_leI bound) (auto intro: from_nat_into[OF  $\langle I \neq \{\} \rangle$ ])
      finally show  $emeasure \ M \ (\bigcup_{n \leq i}. A \ (from\_nat\_into \ I \ n)) \leq ennreal \ B$  .
    qed
    finally have  $\ast: emeasure \ M \ (\bigcup_{i \in I}. A \ i) \leq B$  .
    then have ?fm
      using  $I \neq \{\}$  by (intro fmeasurableI conjI) (auto simp: less_top[symmetric] top_unique)
    with  $\ast \langle 0 \leq B \rangle$  show ?thesis
      by (simp add: emeasure_eq_measure2)
    qed
    then show ?fm ?m by auto
  qed

```

Version for big union of a countable set

lemma

```

  assumes countable  $\mathcal{D}$ 
  and meas:  $\bigwedge D. D \in \mathcal{D} \implies D \in fmeasurable \ M$ 
  and bound:  $\bigwedge \mathcal{E}. [\mathcal{E} \subseteq \mathcal{D}; \text{finite } \mathcal{E}] \implies measure \ M \ (\bigcup \mathcal{E}) \leq B$ 
  shows fmeasurable_Union_bound:  $\bigcup \mathcal{D} \in fmeasurable \ M$  (is ?fm)
  and measure_Union_bound:  $measure \ M \ (\bigcup \mathcal{D}) \leq B$  (is ?m)
proof -
  have  $B \geq 0$ 
  using bound by force
  have ?fm  $\wedge$  ?m
  proof (cases  $\mathcal{D} = \{\}$ )
    case True
    with  $\langle B \geq 0 \rangle$  show ?thesis
      by auto
  next

```

```

case False
then obtain D :: nat ⇒ 'a set where D: D = range D
  using ⟨countable D⟩ uncountable_def by force
  have 1:  $\bigwedge i. D\ i \in \text{fmeasurable } M$ 
    by (simp add: D meas)
  have 2:  $\bigwedge I'. \text{finite } I' \implies \text{measure } M (\bigcup_{x \in I'} D\ x) \leq B$ 
    by (simp add: D bound_image_subset_iff)
  show ?thesis
    unfolding D
    by (intro conjI fmeasurable_UN_bound [OF _ 1 2]) measure_UN_bound
[OF _ 1 2]) auto
qed
then show ?fm ?m by auto
qed

```

Version for indexed union over the type of naturals

```

lemma
  fixes S :: nat ⇒ 'a set
  assumes S:  $\bigwedge i. S\ i \in \text{fmeasurable } M$  and B:  $\bigwedge n. \text{measure } M (\bigcup_{i \leq n} S\ i) \leq B$ 
  shows fmeasurable_countable_Union:  $(\bigcup i. S\ i) \in \text{fmeasurable } M$ 
    and measure_countable_Union_le:  $\text{measure } M (\bigcup i. S\ i) \leq B$ 
proof -
  have mB:  $\text{measure } M (\bigcup i \in I. S\ i) \leq B$  if finite I for I
  proof -
    have  $(\bigcup i \in I. S\ i) \subseteq (\bigcup i \leq \text{Max } I. S\ i)$ 
    using Max_ge that by force
    then have  $\text{measure } M (\bigcup i \in I. S\ i) \leq \text{measure } M (\bigcup i \leq \text{Max } I. S\ i)$ 
    by (rule measure_mono_fmeasurable) (use S in ⟨blast+⟩)
    then show ?thesis
    using B order_trans by blast
  qed
  show  $(\bigcup i. S\ i) \in \text{fmeasurable } M$ 
  by (auto intro: fmeasurable_UN_bound [OF _ S mB])
  show  $\text{measure } M (\bigcup n. S\ n) \leq B$ 
  by (auto intro: measure_UN_bound [OF _ S mB])
qed

```

```

lemma measure_diff_le_measure_setdiff:
  assumes S ∈ fmeasurable M T ∈ fmeasurable M
  shows  $\text{measure } M\ S - \text{measure } M\ T \leq \text{measure } M\ (S - T)$ 
proof -
  have  $\text{measure } M\ S \leq \text{measure } M\ ((S - T) \cup T)$ 
  by (simp add: assms fmeasurable.Un fmeasurableD measure_mono_fmeasurable)
  also have  $\dots \leq \text{measure } M\ (S - T) + \text{measure } M\ T$ 
  using assms by (blast intro: measure_Un_le)
  finally show ?thesis
  by (simp add: algebra_simps)
qed

```

```

lemma suminf_exist_split2:
  fixes f :: nat  $\Rightarrow$  'a::real_normed_vector
  assumes summable f
  shows  $(\lambda n. (\sum k. f(k+n))) \longrightarrow 0$ 
by (subst lim_sequentially, auto simp add: dist_norm suminf_exist_split[OF _
  assms])

```

```

lemma emeasure_union_summable:
  assumes [measurable]:  $\bigwedge n. A\ n \in \text{sets } M$ 
  and  $\bigwedge n. \text{emeasure } M\ (A\ n) < \infty$  summable  $(\lambda n. \text{measure } M\ (A\ n))$ 
  shows  $\text{emeasure } M\ (\bigcup n. A\ n) < \infty$   $\text{emeasure } M\ (\bigcup n. A\ n) \leq (\sum n. \text{measure } M\ (A\ n))$ 
proof -
  define B where  $B = (\lambda N. (\bigcup n \in \{..<N\}. A\ n))$ 
  have [measurable]:  $B\ N \in \text{sets } M$  for N unfolding B_def by auto
  have  $(\lambda N. \text{emeasure } M\ (B\ N)) \longrightarrow \text{emeasure } M\ (\bigcup N. B\ N)$ 
  apply (rule Lim_emeasure_incseq) unfolding B_def by (auto simp add:
  SUP_subset_mono incseq_def)
  moreover have  $\text{emeasure } M\ (B\ N) \leq \text{ennreal } (\sum n. \text{measure } M\ (A\ n))$  for N
  proof -
    have *:  $(\sum n \in \{..<N\}. \text{measure } M\ (A\ n)) \leq (\sum n. \text{measure } M\ (A\ n))$ 
    using assms(3) measure_nonneg sum_le_suminf by blast

    have  $\text{emeasure } M\ (B\ N) \leq (\sum n \in \{..<N\}. \text{emeasure } M\ (A\ n))$ 
    unfolding B_def by (rule emeasure_subadditive_finite, auto)
    also have ... =  $(\sum n \in \{..<N\}. \text{ennreal}(\text{measure } M\ (A\ n)))$ 
    using assms(2) by (simp add: emeasure_eq_ennreal_measure_less_top)
    also have ... =  $\text{ennreal } (\sum n \in \{..<N\}. \text{measure } M\ (A\ n))$ 
    by auto
    also have ...  $\leq \text{ennreal } (\sum n. \text{measure } M\ (A\ n))$ 
    using * by (auto simp: ennreal_leI)
    finally show ?thesis by simp
  qed
  ultimately have  $\text{emeasure } M\ (\bigcup N. B\ N) \leq \text{ennreal } (\sum n. \text{measure } M\ (A\ n))$ 
  by (simp add: Lim_bounded)
  then show  $\text{emeasure } M\ (\bigcup n. A\ n) \leq (\sum n. \text{measure } M\ (A\ n))$ 
  unfolding B_def by (metis UN_UN_flatten UN_lessThan_UNIV)
  then show  $\text{emeasure } M\ (\bigcup n. A\ n) < \infty$ 
  by (auto simp: less_top[symmetric] top_unique)
qed

```

```

lemma borel_cantelli_limsup1:
  assumes [measurable]:  $\bigwedge n. A\ n \in \text{sets } M$ 
  and  $\bigwedge n. \text{emeasure } M\ (A\ n) < \infty$  summable  $(\lambda n. \text{measure } M\ (A\ n))$ 
  shows  $\text{limsup } A \in \text{null\_sets } M$ 
proof -
  have  $\text{emeasure } M\ (\text{limsup } A) \leq 0$ 
  proof (rule LIMSEQ_le_const)
    have  $(\lambda n. (\sum k. \text{measure } M\ (A\ (k+n)))) \longrightarrow 0$  by (rule suminf_exist_split2[OF

```

```

assms(3)])
  then show ( $\lambda n. \text{ennreal } (\sum k. \text{measure } M (A (k+n)))$ )  $\longrightarrow 0$ 
    unfolding ennreal_0[symmetric] by (intro tendsto_ennrealI)
  have  $\text{emeasure } M (\text{limsup } A) \leq (\sum k. \text{measure } M (A (k+n)))$  for  $n$ 
  proof -
    have  $I: (\bigcup k \in \{n..\}. A k) = (\bigcup k. A (k+n))$  by (auto, metis le_add_diff_inverse2,
fastforce)
    have  $\text{emeasure } M (\text{limsup } A) \leq \text{emeasure } M (\bigcup k \in \{n..\}. A k)$ 
      by (rule emeasure_mono, auto simp add: limsup_INF_SUP)
    also have  $\dots = \text{emeasure } M (\bigcup k. A (k+n))$ 
      using I by auto
    also have  $\dots \leq (\sum k. \text{measure } M (A (k+n)))$ 
      apply (rule emeasure_union_summable)
    using assms summable_ignore_initial_segment[OF assms(3), of n] by auto
    finally show ?thesis by simp
  qed
  then show  $\exists N. \forall n \geq N. \text{emeasure } M (\text{limsup } A) \leq (\sum k. \text{measure } M (A$ 
( $k+n$ )))
    by auto
  qed
  then show ?thesis using assms(1) measurable_limsup by auto
qed

```

lemma borel_cantelli_AE1:

```

assumes [measurable]:  $\bigwedge n. A n \in \text{sets } M$ 
  and  $\bigwedge n. \text{emeasure } M (A n) < \infty$  summable ( $\lambda n. \text{measure } M (A n)$ )
  shows AE  $x$  in  $M. \text{eventually } (\lambda n. x \in \text{space } M - A n) \text{ sequentially}$ 
proof -
  have AE  $x$  in  $M. x \notin \text{limsup } A$ 
    using borel_cantelli_limsup1[OF assms] unfolding eventually_ae_filter by
auto
  moreover
  {
    fix  $x$  assume  $x \notin \text{limsup } A$ 
    then obtain  $N$  where  $x \notin (\bigcup n \in \{N..\}. A n)$  unfolding limsup_INF_SUP
  by blast
    then have  $\text{eventually } (\lambda n. x \notin A n) \text{ sequentially}$  using eventually_sequentially
  by auto
  }
  ultimately show ?thesis by auto
qed

```

6.3.12 Measure spaces with $\text{emeasure } M (\text{space } M) < \infty$

locale finite_measure = sigma_finite_measure M for M +
assumes $\text{finite\_emeasure\_space}: \text{emeasure } M (\text{space } M) \neq \text{top}$

lemma finite_measureI[Pure.intro!]:

$\text{emeasure } M (\text{space } M) \neq \infty \implies \text{finite\_measure } M$

proof qed (*auto intro!*: *exI*[*of* _ {*space* *M*}])

lemma (*in finite_measure*) *emeasure_finite*[*simp*, *intro*]: *emeasure* *M* *A* $\neq$ *top*
using *finite_emeasure_space* *emeasure_space*[*of* *M* *A*] **by** (*auto simp: top_unique*)

lemma (*in finite_measure*) *fmeasurable_eq_sets*: *fmeasurable* *M* = *sets* *M*
by (*auto simp: fmeasurable_def less_top[symmetric]*)

lemma (*in finite_measure*) *emeasure_eq_measure*: *emeasure* *M* *A* = *ennreal* (*measure* *M* *A*)
by (*intro emeasure_eq_ennreal_measure*) *simp*

lemma (*in finite_measure*) *emeasure_real*: $\exists r. 0 \leq r \wedge \text{emeasure } M \ A = \text{ennreal } r$
using *emeasure_finite*[*of* *A*] **by** (*cases emeasure M A rule: ennreal_cases*) *auto*

lemma (*in finite_measure*) *bounded_measure*: *measure* *M* *A* $\leq$ *measure* *M* (*space* *M*)
using *emeasure_space*[*of* *M* *A*] *emeasure_real*[*of* *A*] *emeasure_real*[*of* *space* *M*]
by (*auto simp: measure_def*)

lemma (*in finite_measure*) *finite_measure_Diff*:
assumes *sets*: *A* $\in$ *sets* *M* *B* $\in$ *sets* *M* **and** *B* $\subseteq$ *A*
shows *measure* *M* (*A* - *B*) = *measure* *M* *A* - *measure* *M* *B*
using *measure_Diff*[*OF* _ *assms*] **by** *simp*

lemma (*in finite_measure*) *finite_measure_Union*:
assumes *sets*: *A* $\in$ *sets* *M* *B* $\in$ *sets* *M* **and** *A* $\cap$ *B* = {}
shows *measure* *M* (*A* $\cup$ *B*) = *measure* *M* *A* + *measure* *M* *B*
using *measure_Union*[*OF* _ _ *assms*] **by** *simp*

lemma (*in finite_measure*) *finite_measure_finite_Union*:
assumes *measurable*: *finite* *S* *A*:*S* $\subseteq$ *sets* *M* *disjoint_family_on* *A* *S*
shows *measure* *M* ($\bigcup_{i \in S.} A \ i$) = ($\sum_{i \in S.} \text{measure } M \ (A \ i)$)
using *measure_finite_Union*[*OF* *assms*] **by** *simp*

lemma (*in finite_measure*) *finite_measure_UNION*:
assumes *A*: *range* *A* $\subseteq$ *sets* *M* *disjoint_family* *A*
shows ($\lambda i. \text{measure } M \ (A \ i)$) *sums* (*measure* *M* ($\bigcup i. A \ i$))
using *measure_UNION*[*OF* *A*] **by** *simp*

lemma (*in finite_measure*) *finite_measure_mono*:
assumes *A* $\subseteq$ *B* *B* $\in$ *sets* *M* **shows** *measure* *M* *A* $\leq$ *measure* *M* *B*
using *emeasure_mono*[*OF* *assms*] *emeasure_real*[*of* *A*] *emeasure_real*[*of* *B*] **by**
(*auto simp: measure_def*)

lemma (*in finite_measure*) *finite_measure_subadditive*:
assumes *m*: *A* $\in$ *sets* *M* *B* $\in$ *sets* *M*
shows *measure* *M* (*A* $\cup$ *B*) $\leq$ *measure* *M* *A* + *measure* *M* *B*

using *measure_subadditive*[*OF m*] **by** *simp*

lemma (**in** *finite_measure*) *finite_measure_subadditive_finite*:
assumes *finite I A* *I* $\subseteq$ *sets M* **shows** *measure M* $(\bigcup_{i \in I}. A\ i) \leq (\sum_{i \in I}. \text{measure } M\ (A\ i))$
using *measure_subadditive_finite*[*OF assms*] **by** *simp*

lemma (**in** *finite_measure*) *finite_measure_subadditive_countably*:
 $\text{range } A \subseteq \text{sets } M \implies \text{summable } (\lambda i. \text{measure } M\ (A\ i)) \implies \text{measure } M\ (\bigcup i. A\ i) \leq (\sum i. \text{measure } M\ (A\ i))$
by (*rule measure_subadditive_countably*)
(simp_all add: ennreal_suminf_neq_top emeasure_eq_measure)

lemma (**in** *finite_measure*) *finite_measure_eq_sum_singleton*:
assumes *finite S* **and** $\bigwedge x. x \in S \implies \{x\} \in \text{sets } M$
shows *measure M S* $= (\sum_{x \in S}. \text{measure } M\ \{x\})$
using *measure_eq_sum_singleton*[*OF assms*] **by** *simp*

lemma (**in** *finite_measure*) *finite_Lim_measure_incseq*:
assumes *A*: $\text{range } A \subseteq \text{sets } M$ *incseq A*
shows $(\lambda i. \text{measure } M\ (A\ i)) \longrightarrow \text{measure } M\ (\bigcup i. A\ i)$
using *Lim_measure_incseq*[*OF A*] **by** *simp*

lemma (**in** *finite_measure*) *finite_Lim_measure_decseq*:
assumes *A*: $\text{range } A \subseteq \text{sets } M$ *decseq A*
shows $(\lambda n. \text{measure } M\ (A\ n)) \longrightarrow \text{measure } M\ (\bigcap i. A\ i)$
using *Lim_measure_decseq*[*OF A*] **by** *simp*

lemma (**in** *finite_measure*) *finite_measure_compl*:
assumes *S*: $S \in \text{sets } M$
shows *measure M* $(\text{space } M - S) = \text{measure } M\ (\text{space } M) - \text{measure } M\ S$
using *measure_Diff*[*OF _ sets.top S sets.sets_into_space*] **by** *simp*

lemma (**in** *finite_measure*) *finite_measure_mono_AE*:
assumes *imp*: $AE\ x\ \text{in } M. x \in A \longrightarrow x \in B$ **and** *B*: $B \in \text{sets } M$
shows *measure M A* $\leq \text{measure } M\ B$
using *assms emeasure_mono_AE*[*OF imp B*]
by (*simp add: emeasure_eq_measure*)

lemma (**in** *finite_measure*) *finite_measure_eq_AE*:
assumes *iff*: $AE\ x\ \text{in } M. x \in A \longleftrightarrow x \in B$
assumes *A*: $A \in \text{sets } M$ **and** *B*: $B \in \text{sets } M$
shows *measure M A* $= \text{measure } M\ B$
using *assms emeasure_eq_AE*[*OF assms*] **by** (*simp add: emeasure_eq_measure*)

lemma (**in** *finite_measure*) *measure_increasing*: *increasing M* $(\text{measure } M)$
by (*auto intro!: finite_measure_mono simp: increasing_def*)

lemma (**in** *finite_measure*) *measure_zero_union*:

```

    assumes  $s \in \text{sets } M$   $t \in \text{sets } M$   $\text{measure } M \ t = 0$ 
    shows  $\text{measure } M \ (s \cup t) = \text{measure } M \ s$ 
using assms
proof -
  have  $\text{measure } M \ (s \cup t) \leq \text{measure } M \ s$ 
    using finite_measure_subadditive[of  $s \ t$ ] assms by auto
  moreover have  $\text{measure } M \ (s \cup t) \geq \text{measure } M \ s$ 
    using assms by (blast intro: finite_measure_mono)
  ultimately show ?thesis by simp
qed

lemma (in finite_measure) measure_eq_compl:
  assumes  $s \in \text{sets } M$   $t \in \text{sets } M$ 
  assumes  $\text{measure } M \ (\text{space } M - s) = \text{measure } M \ (\text{space } M - t)$ 
  shows  $\text{measure } M \ s = \text{measure } M \ t$ 
  using assms finite_measure_compl by auto

lemma (in finite_measure) measure_eq_bigunion_image:
  assumes  $\text{range } f \subseteq \text{sets } M$   $\text{range } g \subseteq \text{sets } M$ 
  assumes disjoint_family  $f$  disjoint_family  $g$ 
  assumes  $\bigwedge n :: \text{nat. } \text{measure } M \ (f \ n) = \text{measure } M \ (g \ n)$ 
  shows  $\text{measure } M \ (\bigcup i. f \ i) = \text{measure } M \ (\bigcup i. g \ i)$ 
using assms
proof -
  have  $a: (\lambda i. \text{measure } M \ (f \ i)) \text{ sums } (\text{measure } M \ (\bigcup i. f \ i))$ 
    by (rule finite_measure_UNION[OF assms(1,3)])
  have  $b: (\lambda i. \text{measure } M \ (g \ i)) \text{ sums } (\text{measure } M \ (\bigcup i. g \ i))$ 
    by (rule finite_measure_UNION[OF assms(2,4)])
  show ?thesis using sums_unique[OF  $b$ ] sums_unique[OF  $a$ ] assms by simp
qed

lemma (in finite_measure) measure_countably_zero:
  assumes  $\text{range } c \subseteq \text{sets } M$ 
  assumes  $\bigwedge i. \text{measure } M \ (c \ i) = 0$ 
  shows  $\text{measure } M \ (\bigcup i :: \text{nat. } c \ i) = 0$ 
proof (rule antisym)
  show  $\text{measure } M \ (\bigcup i :: \text{nat. } c \ i) \leq 0$ 
    using finite_measure_subadditive_countably[OF assms(1)] by (simp add: assms(2))
qed simp

lemma (in finite_measure) measure_space_inter:
  assumes events:  $s \in \text{sets } M$   $t \in \text{sets } M$ 
  assumes  $\text{measure } M \ t = \text{measure } M \ (\text{space } M)$ 
  shows  $\text{measure } M \ (s \cap t) = \text{measure } M \ s$ 
proof -
  have  $\text{measure } M \ ((\text{space } M - s) \cup (\text{space } M - t)) = \text{measure } M \ (\text{space } M - s)$ 
    using events assms finite_measure_compl[of  $t$ ] by (auto intro!: measure_zero_union)
  also have  $(\text{space } M - s) \cup (\text{space } M - t) = \text{space } M - (s \cap t)$ 
    by blast

```

finally show $\text{measure } M (s \cap t) = \text{measure } M s$
using events by (*auto intro!*: $\text{measure\_eq\_compl}[of\ s \cap t\ s]$)
qed

lemma (*in finite_measure*) *measure_equiprobable_finite_unions*:
assumes s : $\text{finite } s \wedge x. x \in s \implies \{x\} \in \text{sets } M$
assumes $\bigwedge x\ y. \llbracket x \in s; y \in s \rrbracket \implies \text{measure } M \{x\} = \text{measure } M \{y\}$
shows $\text{measure } M s = \text{real } (\text{card } s) * \text{measure } M \{\text{SOME } x. x \in s\}$
proof cases
assume $s \neq \{\}$
then have $\exists x. x \in s$ **by** *blast*
from *someI_ex[OF this] assms*
have *prob_some*: $\bigwedge x. x \in s \implies \text{measure } M \{x\} = \text{measure } M \{\text{SOME } y. y \in s\}$
by *blast*
have $\text{measure } M s = (\sum x \in s. \text{measure } M \{x\})$
using *finite_measure_eq_sum_singleton[OF s] by simp*
also have $\dots = (\sum x \in s. \text{measure } M \{\text{SOME } y. y \in s\})$ **using** *prob_some by auto*
also have $\dots = \text{real } (\text{card } s) * \text{measure } M \{\text{SOME } x. x \in s\}$
using *sum_constant assms by simp*
finally show *?thesis* **by** *simp*
qed simp

lemma (*in finite_measure*) *measure_real_sum_image_fn*:
assumes $e \in \text{sets } M$
assumes $\bigwedge x. x \in s \implies e \cap f\ x \in \text{sets } M$
assumes $\text{finite } s$
assumes *disjoint*: $\bigwedge x\ y. \llbracket x \in s; y \in s; x \neq y \rrbracket \implies f\ x \cap f\ y = \{\}$
assumes *upper*: $\text{space } M \subseteq (\bigcup i \in s. f\ i)$
shows $\text{measure } M e = (\sum x \in s. \text{measure } M (e \cap f\ x))$
proof –
have $e \subseteq (\bigcup i \in s. f\ i)$
using $\langle e \in \text{sets } M \rangle \text{sets.sets\_into\_space upper}$ **by** *blast*
then have $e = (\bigcup i \in s. e \cap f\ i)$
by *auto*
hence $\text{measure } M e = \text{measure } M (\bigcup i \in s. e \cap f\ i)$ **by** *simp*
also have $\dots = (\sum x \in s. \text{measure } M (e \cap f\ x))$
proof (*rule finite_measure_finite_Union*)
show $\text{finite } s$ **by** *fact*
show $(\lambda i. e \cap f\ i)'s \subseteq \text{sets } M$ **using** *assms(2)* **by** *auto*
show *disjoint_family_on* $(\lambda i. e \cap f\ i)\ s$
using *disjoint* **by** (*auto simp: disjoint_family_on_def*)
qed
finally show *?thesis* .
qed

lemma (*in finite_measure*) *measure_exclude*:
assumes $A \in \text{sets } M$ $B \in \text{sets } M$
assumes $\text{measure } M A = \text{measure } M (\text{space } M) A \cap B = \{\}$


```

shows  $\text{measure } M \ B = 0$ 
using  $\text{measure\_space\_inter}$ [of  $B \ A$ ] assms by ( $\text{auto simp: ac\_sims}$ )
lemma (in  $\text{finite\_measure}$ )  $\text{finite\_measure\_distr}$ :
  assumes  $f: f \in \text{measurable } M \ M'$ 
  shows  $\text{finite\_measure } (\text{distr } M \ M' \ f)$ 
proof ( $\text{rule finite\_measureI}$ )
  have  $f - ' \text{space } M' \cap \text{space } M = \text{space } M$  using  $f$  by ( $\text{auto dest: measurable\_space}$ )
  with  $f$  show  $\text{emeasure } (\text{distr } M \ M' \ f) \ (\text{space } (\text{distr } M \ M' \ f)) \neq \infty$  by ( $\text{auto simp: emeasure\_distr}$ )
qed

```

```

lemma  $\text{emeasure\_gfp}$ [consumes 1, case_names cont measurable]:
  assumes  $\text{sets}[simp]: \bigwedge s. \text{sets } (M \ s) = \text{sets } N$ 
  assumes  $\bigwedge s. \text{finite\_measure } (M \ s)$ 
  assumes  $\text{cont: inf\_continuous } F \ \text{inf\_continuous } f$ 
  assumes  $\text{meas: } \bigwedge P. \text{Measurable.pred } N \ P \implies \text{Measurable.pred } N \ (F \ P)$ 
  assumes  $\text{iter: } \bigwedge P \ s. \text{Measurable.pred } N \ P \implies \text{emeasure } (M \ s) \ \{x \in \text{space } N. F \ P \ x\} = f \ (\lambda s. \text{emeasure } (M \ s) \ \{x \in \text{space } N. P \ x\}) \ s$ 
  assumes  $\text{bound: } \bigwedge P. f \ P \leq f \ (\lambda s. \text{emeasure } (M \ s) \ (\text{space } (M \ s)))$ 
  shows  $\text{emeasure } (M \ s) \ \{x \in \text{space } N. \text{gfp } F \ x\} = \text{gfp } f \ s$ 
proof ( $\text{subst } \text{gfp\_transfer\_bounded}$ [where  $\alpha = \lambda F \ s. \text{emeasure } (M \ s) \ \{x \in \text{space } N. F \ x\}$  and  $g = f$  and  $f = F$  and  $P = \text{Measurable.pred } N, \text{ symmetric}$ ])
  interpret  $\text{finite\_measure } M \ s$  for  $s$  by  $\text{fact}$ 
  fix  $C$  assume  $\text{decseq } C \ \bigwedge i. \text{Measurable.pred } N \ (C \ i)$ 
  then show  $(\lambda s. \text{emeasure } (M \ s) \ \{x \in \text{space } N. (\text{INF } i. C \ i) \ x\}) = (\text{INF } i. (\lambda s. \text{emeasure } (M \ s) \ \{x \in \text{space } N. C \ i \ x\}))$ 
  unfolding  $\text{INF\_apply}[abs\_def]$ 
  by ( $\text{subst } \text{INF\_emeasure\_decseq}$ ) ( $\text{auto simp: antimono\_def fun\_eq\_iff intro!: arg\_cong2[where } f = \text{emeasure}]$ )
next
  show  $f \ x \leq (\lambda s. \text{emeasure } (M \ s) \ \{x \in \text{space } N. F \ \text{top } x\})$  for  $x$ 
  using  $\text{bound}$ [of  $x$ ]  $\text{sets\_eq\_imp\_space\_eq}$ [OF  $\text{sets}$ ] by ( $\text{simp add: iter}$ )
qed ( $\text{auto simp add: iter le\_fun\_def INF\_apply}[abs\_def] \ \text{intro!: meas cont}$ )

```

6.3.13 Counting space

```

lemma  $\text{strict\_monoI\_Suc}$ :
  assumes  $\text{ord } [simp]: (\bigwedge n. f \ n < f \ (\text{Suc } n))$  shows  $\text{strict\_mono } f$ 
  unfolding  $\text{strict\_mono\_def}$ 
proof  $\text{safe}$ 
  fix  $n \ m :: \text{nat}$  assume  $n < m$  then show  $f \ n < f \ m$ 
  by ( $\text{induct } m$ ) ( $\text{auto simp: less\_Suc\_eq intro: less\_trans ord}$ )
qed

lemma  $\text{emeasure\_count\_space}$ :
  assumes  $X \subseteq A$  shows  $\text{emeasure } (\text{count\_space } A) \ X = (\text{if finite } X \text{ then of\_nat } (\text{card } X) \text{ else } \infty)$ 

```

```

(is _ = ?M X)
unfolding count_space_def
proof (rule emeasure_measure_of_sigma)
  show  $X \in \text{Pow } A$  using  $\langle X \subseteq A \rangle$  by auto
  show sigma_algebra A (Pow A) by (rule sigma_algebra_Pow)
  show positive: positive (Pow A) ?M
    by (auto simp: positive_def)
  have additive: additive (Pow A) ?M
    by (auto simp: card_Un_disjoint additive_def)

interpret ring_of_sets A Pow A
  by (rule ring_of_setsI) auto
show countably_additive (Pow A) ?M
  unfolding countably_additive_iff_continuous_from_below[OF positive addi-
tive]
proof safe
  fix F :: nat  $\Rightarrow$  'a set assume incseq F
  show  $(\lambda i. ?M (F i)) \longrightarrow ?M (\bigcup i. F i)$ 
  proof cases
    assume  $\exists i. \forall j \geq i. F i = F j$ 
    then obtain i where  $i: \forall j \geq i. F i = F j$  ..
    with  $\langle \text{incseq } F \rangle$  have  $F j \subseteq F i$  for j
      by (cases  $i \leq j$ ) (auto simp: incseq_def)
    then have eq:  $(\bigcup i. F i) = F i$ 
      by auto
    with i show ?thesis
      by (auto intro!: Lim_transform_eventually[OF tendsto_const] eventu-
ally_sequentiallyI[where c=i])
  next
    assume  $\neg (\exists i. \forall j \geq i. F i = F j)$ 
    then obtain f where  $f: \bigwedge i. i \leq f i \wedge i. F i \neq F (f i)$  by metis
    then have  $\bigwedge i. F i \subseteq F (f i)$  using  $\langle \text{incseq } F \rangle$  by (auto simp: incseq_def)
    with f have *:  $\bigwedge i. F i \subset F (f i)$  by auto

    have incseq  $(\lambda i. ?M (F i))$ 
      using  $\langle \text{incseq } F \rangle$  unfolding incseq_def by (auto simp: card_mono dest:
finite_subset)
    then have  $(\lambda i. ?M (F i)) \longrightarrow (SUP n. ?M (F n))$ 
      by (rule LIMSEQ_SUP)

    moreover have  $(SUP n. ?M (F n)) = \text{top}$ 
  proof (rule ennreal_SUP_eq_top)
    fix n :: nat show  $\exists k::\text{nat} \in \text{UNIV}. \text{of\_nat } n \leq ?M (F k)$ 
    proof (induct n)
      case (Suc n)
      then obtain k where  $\text{of\_nat } n \leq ?M (F k)$  ..
      moreover have  $\text{finite } (F k) \implies \text{finite } (F (f k)) \implies \text{card } (F k) < \text{card } (F$ 
(f k))
        using  $\langle F k \subset F (f k) \rangle$  by (simp add: psubset_card_mono)

```

```

    moreover have finite (F (f k))  $\implies$  finite (F k)
    using  $\langle k \leq f k \rangle \langle \text{incseq } F \rangle$  by (auto simp: incseq_def dest: finite_subset)
    ultimately show ?case
    by (auto intro!: exI[of _ f k] simp del: of_nat_Suc)
  qed auto
qed

moreover
have inj  $(\lambda n. F ((f \smallfrown n) 0))$ 
  by (intro strict_mono_imp_inj_on strict_monoI_Suc) (simp add: *)
then have 1: infinite (range  $(\lambda i. F ((f \smallfrown i) 0))$ )
  by (rule range_inj_infinite)
have infinite (Pow  $(\bigcup i. F i)$ )
  by (rule infinite_super[OF _ 1]) auto
then have infinite  $(\bigcup i. F i)$ 
  by auto
ultimately show ?thesis by (simp only:) simp

qed
qed
qed

lemma distr_bij_count_space:
  assumes f: bij_betw f A B
  shows distr (count_space A) (count_space B) f = count_space B
proof (rule measure_eqI)
  have f':  $f \in \text{measurable } (\text{count\_space } A) (\text{count\_space } B)$ 
    using f unfolding Pi_def bij_betw_def by auto
  fix X assume X  $\in \text{sets } (\text{distr } (\text{count\_space } A) (\text{count\_space } B) f)$ 
  then have X:  $X \in \text{sets } (\text{count\_space } B)$  by auto
  moreover from X have  $f^{-1} X \cap A = \text{the\_inv\_into } A f^{-1} X$ 
    using f by (auto simp: bij_betw_def subset_image_iff image_iff the_inv_into_f_f
    intro: the_inv_into_f_f[symmetric])
  moreover have inj_on (the_inv_into A f) B
    using X f by (auto simp: bij_betw_def inj_on_the_inv_into)
  with X have inj_on (the_inv_into A f) X
    by (auto intro: subset_inj_on)
  ultimately show emeasure (distr (count_space A) (count_space B) f) X =
    emeasure (count_space B) X
    using f unfolding emeasure_distr[OF f' X]
    by (subst (1 2) emeasure_count_space) (auto simp: card_image dest: finite_imageD)
qed simp

lemma emeasure_count_space_finite[simp]:
   $X \subseteq A \implies \text{finite } X \implies \text{emeasure } (\text{count\_space } A) X = \text{of\_nat } (\text{card } X)$ 
  using emeasure_count_space[of X A] by simp

lemma emeasure_count_space_infinite[simp]:

```

$X \subseteq A \implies \text{infinite } X \implies \text{emeasure } (\text{count\_space } A) X = \infty$
using *emeasure_count_space[of X A]* **by** *simp*

lemma *measure_count_space*: $\text{measure } (\text{count\_space } A) X = (\text{if } X \subseteq A \text{ then of\_nat } (\text{card } X) \text{ else } 0)$
by *(cases finite X) (auto simp: measure_notin_sets ennreal_of_nat_eq_real_of_nat measure_zero_top measure_eq_emeasure_eq_ennreal)*

lemma *emeasure_count_space_eq_0*:
 $\text{emeasure } (\text{count\_space } A) X = 0 \iff (X \subseteq A \longrightarrow X = \{\})$
proof *cases*
assume *X*: $X \subseteq A$
then show *?thesis*
proof *(intro iffI impI)*
assume $\text{emeasure } (\text{count\_space } A) X = 0$
with *X* **show** $X = \{\}$
by *(subst (asm) emeasure_count_space) (auto split: if_split_asm)*
qed *simp*
qed *(simp add: emeasure_notin_sets)*

lemma *null_sets_count_space*: $\text{null\_sets } (\text{count\_space } A) = \{\{\}\}$
unfolding *null_sets_def* **by** *(auto simp add: emeasure_count_space_eq_0)*

lemma *AE_count_space*: $(AE x \text{ in } \text{count\_space } A. P x) \iff (\forall x \in A. P x)$
unfolding *eventually_ae_filter* **by** *(auto simp add: null_sets_count_space)*

lemma *sigma_finite_measure_count_space_countable*:
assumes *A*: *countable A*
shows $\text{sigma\_finite\_measure } (\text{count\_space } A)$
proof **qed** *(insert A, auto intro!: exI[of _ $(\lambda a. \{a\})$] 'A)*

lemma *sigma_finite_measure_count_space*:
fixes *A* :: '*a*::countable set' **shows** $\text{sigma\_finite\_measure } (\text{count\_space } A)$
by *(rule sigma_finite_measure_count_space_countable) auto*

lemma *finite_measure_count_space*:
assumes [*simp*]: *finite A*
shows $\text{finite\_measure } (\text{count\_space } A)$
by *rule simp*

lemma *sigma_finite_measure_count_space_finite*:
assumes *A*: *finite A* **shows** $\text{sigma\_finite\_measure } (\text{count\_space } A)$
proof –
interpret *finite_measure count_space A* **using** *A* **by** *(rule finite_measure_count_space)*
show $\text{sigma\_finite\_measure } (\text{count\_space } A)$ **..**
qed

6.3.14 Measure restricted to space

```

lemma emeasure_restrict_space:
  assumes  $\Omega \cap \text{space } M \in \text{sets } M$   $A \subseteq \Omega$ 
  shows  $\text{emeasure } (\text{restrict\_space } M \ \Omega) \ A = \text{emeasure } M \ A$ 
proof (cases  $A \in \text{sets } M$ )
  case True
  show ?thesis
  proof (rule emeasure_measure_of[OF restrict_space_def])
    show  $(\cap) \ \Omega \ \text{'sets } M \subseteq \text{Pow } (\Omega \cap \text{space } M) \ A \in \text{sets } (\text{restrict\_space } M \ \Omega)$ 
    using  $\langle A \subseteq \Omega \rangle \ \langle A \in \text{sets } M \rangle \ \text{sets.space\_closed}$  by (auto simp: sets_restrict_space)
    show positive  $(\text{sets } (\text{restrict\_space } M \ \Omega)) \ (\text{emeasure } M)$ 
    by (auto simp: positive_def)
    show countably_additive  $(\text{sets } (\text{restrict\_space } M \ \Omega)) \ (\text{emeasure } M)$ 
    proof (rule countably_additiveI)
      fix  $A :: \text{nat} \Rightarrow \_$  assume  $\text{range } A \subseteq \text{sets } (\text{restrict\_space } M \ \Omega)$  disjoint_family
    A
      with assms have  $\bigwedge i. A \ i \in \text{sets } M \ \bigwedge i. A \ i \subseteq \text{space } M$  disjoint_family  $A$ 
      by (fastforce simp: sets_restrict_space_iff[OF assms(1)] image_subset_iff
        dest: sets.sets_into_space)+
      then show  $(\sum i. \text{emeasure } M \ (A \ i)) = \text{emeasure } M \ (\bigcup i. A \ i)$ 
      by (subst suminf_emeasure) (auto simp: disjoint_family_subset)
    qed
  qed
next
  case False
  with assms have  $A \notin \text{sets } (\text{restrict\_space } M \ \Omega)$ 
  by (simp add: sets_restrict_space_iff)
  with False show ?thesis
  by (simp add: emeasure_notin_sets)
qed

lemma measure_restrict_space:
  assumes  $\Omega \cap \text{space } M \in \text{sets } M$   $A \subseteq \Omega$ 
  shows  $\text{measure } (\text{restrict\_space } M \ \Omega) \ A = \text{measure } M \ A$ 
  using emeasure_restrict_space[OF assms] by (simp add: measure_def)

lemma AE_restrict_space_iff:
  assumes  $\Omega \cap \text{space } M \in \text{sets } M$ 
  shows  $(AE \ x \ \text{in } \text{restrict\_space } M \ \Omega. \ P \ x) \longleftrightarrow (AE \ x \ \text{in } M. \ x \in \Omega \longrightarrow P \ x)$ 
proof –
  have ex_cong:  $\bigwedge P \ Q \ f. (\bigwedge x. P \ x \Longrightarrow Q \ x) \Longrightarrow (\bigwedge x. Q \ x \Longrightarrow P \ (f \ x)) \Longrightarrow (\exists x. P \ x) \longleftrightarrow (\exists x. Q \ x)$ 
  by auto
  { fix  $X$  assume  $X: X \in \text{sets } M$   $\text{emeasure } M \ X = 0$ 
    then have  $\text{emeasure } M \ (\Omega \cap \text{space } M \cap X) \leq \text{emeasure } M \ X$ 
    by (intro emeasure_mono) auto
    then have  $\text{emeasure } M \ (\Omega \cap \text{space } M \cap X) = 0$ 
    using  $X$  by (auto intro!: antisym) }
  with assms show ?thesis

```

```

    unfolding eventually_ae_filter
  by (auto simp add: space_restrict_space null_sets_def sets_restrict_space_iff
      emeasure_restrict_space cong: conj_cong
      intro!: ex_cong[where f= $\lambda X. (\Omega \cap \text{space } M) \cap X$ ])
qed

lemma restrict_restrict_space:
  assumes  $A \cap \text{space } M \in \text{sets } M$   $B \cap \text{space } M \in \text{sets } M$ 
  shows  $\text{restrict\_space } (\text{restrict\_space } M A) B = \text{restrict\_space } M (A \cap B)$  (is ?l
  = ?r)
proof (rule measure_eqI[symmetric])
  show  $\text{sets } ?r = \text{sets } ?l$ 
    unfolding sets_restrict_space image_comp by (intro image_cong) auto
next
  fix X assume  $X \in \text{sets } (\text{restrict\_space } M (A \cap B))$ 
  then obtain Y where  $Y \in \text{sets } M$   $X = Y \cap A \cap B$ 
    by (auto simp: sets_restrict_space)
  with assms sets.Int[OF assms] show  $\text{emeasure } ?r X = \text{emeasure } ?l X$ 
    by (subst (1 2) emeasure_restrict_space)
    (auto simp: space_restrict_space sets_restrict_space_iff emeasure_restrict_space
    ac_simps)
qed

lemma restrict_count_space:  $\text{restrict\_space } (\text{count\_space } B) A = \text{count\_space } (A \cap B)$ 
proof (rule measure_eqI)
  show  $\text{sets } (\text{restrict\_space } (\text{count\_space } B) A) = \text{sets } (\text{count\_space } (A \cap B))$ 
    by (subst sets_restrict_space) auto
  moreover fix X assume  $X \in \text{sets } (\text{restrict\_space } (\text{count\_space } B) A)$ 
  ultimately have  $X \subseteq A \cap B$  by auto
  then show  $\text{emeasure } (\text{restrict\_space } (\text{count\_space } B) A) X = \text{emeasure } (\text{count\_space } (A \cap B)) X$ 
    by (cases finite X) (auto simp add: emeasure_restrict_space)
qed

lemma sigma_finite_measure_restrict_space:
  assumes sigma_finite_measure M
  and A:  $A \in \text{sets } M$ 
  shows sigma_finite_measure ( $\text{restrict\_space } M A$ )
proof -
  interpret sigma_finite_measure M by fact
  from sigma_finite_countable obtain C
    where C: countable C  $C \subseteq \text{sets } M$   $(\bigcup C) = \text{space } M$   $\forall a \in C. \text{emeasure } M a \neq \infty$ 
  by blast
  let ?C =  $(\cap) A \restriction C$ 
  from C have countable ?C  $?C \subseteq \text{sets } (\text{restrict\_space } M A)$   $(\bigcup ?C) = \text{space } (\text{restrict\_space } M A)$ 
    by (auto simp add: sets_restrict_space space_restrict_space)

```

```

moreover {
  fix  $a$ 
  assume  $a \in ?C$ 
  then obtain  $a'$  where  $a = A \cap a'$   $a' \in C$  ..
  then have  $\text{emeasure } (\text{restrict\_space } M \ A) \ a \leq \text{emeasure } M \ a'$ 
  using  $A \ C$  by (auto simp add:  $\text{emeasure\_restrict\_space}$  intro:  $\text{emeasure\_mono}$ )
  also have  $\dots < \infty$  using  $C(4)[\text{rule\_format, of } a'] \langle a' \in C \rangle$  by (simp add:
less_top)
  finally have  $\text{emeasure } (\text{restrict\_space } M \ A) \ a \neq \infty$  by simp }
  ultimately show ?thesis
  by unfold_locales (rule exI conjI | assumption | blast) +
qed

```

```

lemma finite_measure_restrict_space:
  assumes finite_measure  $M$ 
  and  $A: A \in \text{sets } M$ 
  shows finite_measure ( $\text{restrict\_space } M \ A$ )
proof -
  interpret finite_measure  $M$  by fact
  show ?thesis
  by (rule finite_measureI) (simp add:  $\text{emeasure\_restrict\_space}$   $A$   $\text{space\_restrict\_space}$ )
qed

```

```

lemma restrict_distr:
  assumes [measurable]:  $f \in \text{measurable } M \ N$ 
  assumes [simp]:  $\Omega \cap \text{space } N \in \text{sets } N$  and  $\text{restrict}: f \in \text{space } M \rightarrow \Omega$ 
  shows  $\text{restrict\_space } (\text{distr } M \ N \ f) \ \Omega = \text{distr } M \ (\text{restrict\_space } N \ \Omega) \ f$ 
  (is ?l = ?r)
proof (rule measure_eqI)
  fix  $A$  assume  $A \in \text{sets } (\text{restrict\_space } (\text{distr } M \ N \ f) \ \Omega)$ 
  with  $\text{restrict}$  show  $\text{emeasure } ?l \ A = \text{emeasure } ?r \ A$ 
  by (subst  $\text{emeasure\_distr}$ )
  (auto simp:  $\text{sets\_restrict\_space\_iff}$   $\text{emeasure\_restrict\_space}$   $\text{emeasure\_distr}$ 
intro!:  $\text{measurable\_restrict\_space2}$ )
qed (simp add:  $\text{sets\_restrict\_space}$ )

```

```

lemma measure_eqI_restrict_generator:
  assumes  $E: \text{Int\_stable } E \ E \subseteq \text{Pow } \Omega \ \bigwedge X. X \in E \implies \text{emeasure } M \ X = \text{emeasure } N \ X$ 
  assumes  $\text{sets\_eq}: \text{sets } M = \text{sets } N$  and  $\Omega: \Omega \in \text{sets } M$ 
  assumes  $\text{sets } (\text{restrict\_space } M \ \Omega) = \text{sigma\_sets } \Omega \ E$ 
  assumes  $\text{sets } (\text{restrict\_space } N \ \Omega) = \text{sigma\_sets } \Omega \ E$ 
  assumes  $ae: AE \ x \text{ in } M. x \in \Omega \ AE \ x \text{ in } N. x \in \Omega$ 
  assumes  $A: \text{countable } A \ A \neq \{\} \ A \subseteq E \cup A = \Omega \ \bigwedge a. a \in A \implies \text{emeasure } M$ 
 $a \neq \infty$ 
  shows  $M = N$ 
proof (rule measure_eqI)
  fix  $X$  assume  $X: X \in \text{sets } M$ 
  then have  $\text{emeasure } M \ X = \text{emeasure } (\text{restrict\_space } M \ \Omega) \ (X \cap \Omega)$ 

```

```

    using ae  $\Omega$  by (auto simp add: emeasure_restrict_space intro!: emeasure_eq_AE)
  also have restrict_space  $M \ \Omega = \text{restrict\_space } N \ \Omega$ 
  proof (rule measure_eqI_generator_eq)
    fix  $X$  assume  $X \in E$ 
    then show emeasure (restrict_space  $M \ \Omega$ )  $X = \text{emeasure (restrict\_space } N \ \Omega) \ X$ 
  using  $E \ \Omega$  by (subst (1 2) emeasure_restrict_space) (auto simp: sets_eq
sets_eq[THEN sets_eq_imp_space_eq])
  next
    show range (from_nat_into  $A$ )  $\subseteq E$  ( $\bigcup i.$  from_nat_into  $A \ i$ )  $= \Omega$ 
    using  $A$  by (auto cong del: SUP_cong_simp)
  next
    fix  $i$ 
    have emeasure (restrict_space  $M \ \Omega$ ) (from_nat_into  $A \ i$ )  $= \text{emeasure } M$ 
    (from_nat_into  $A \ i$ )
    using  $A \ \Omega$  by (subst emeasure_restrict_space)
    (auto simp: sets_eq sets_eq[THEN sets_eq_imp_space_eq] intro:
from_nat_into)
    with  $A$  show emeasure (restrict_space  $M \ \Omega$ ) (from_nat_into  $A \ i$ )  $\neq \infty$ 
    by (auto intro: from_nat_into)
  qed fact+
  also have emeasure (restrict_space  $N \ \Omega$ ) ( $X \cap \Omega$ )  $= \text{emeasure } N \ X$ 
  using  $X \ ae \ \Omega$  by (auto simp add: emeasure_restrict_space sets_eq intro!:
emeasure_eq_AE)
  finally show emeasure  $M \ X = \text{emeasure } N \ X$  .
qed fact

```

6.3.15 Null measure

definition *null_measure* :: 'a measure $\Rightarrow$ 'a measure **where**
null_measure $M = \text{sigma (space } M) (\text{sets } M)$

lemma *space_null_measure*[simp]: *space (null_measure* $M) = \text{space } M$
by (simp add: null_measure_def)

lemma *sets_null_measure*[simp, measurable_cong]: *sets (null_measure* $M) = \text{sets } M$
by (simp add: null_measure_def)

lemma *emeasure_null_measure*[simp]: *emeasure (null_measure* $M) \ X = 0$
by (cases $X \in \text{sets } M$, rule emeasure_measure_of)
(auto simp: positive_def countably_additive_def emeasure_notin_sets null_measure_def
dest: sets.sets_into_space)

lemma *measure_null_measure*[simp]: *measure (null_measure* $M) \ X = 0$
by (intro measure_eq_emeasure_eq ennreal) auto

lemma *null_measure_idem* [simp]: *null_measure (null_measure* $M) = \text{null\_measure } M$

by(rule measure_eqI) simp_all

6.3.16 Scaling a measure

definition *scale_measure* :: *ennreal* $\Rightarrow$ 'a *measure* $\Rightarrow$ 'a *measure* **where**
scale_measure *r M* = *measure_of* (*space M*) (*sets M*) ($\lambda A. r * \text{emeasure } M A$)

lemma *space_scale_measure*: *space* (*scale_measure* *r M*) = *space M*
 by (simp add: *scale_measure_def*)

lemma *sets_scale_measure* [*simp*, *measurable_cong*]: *sets* (*scale_measure* *r M*)
 = *sets M*
 by (simp add: *scale_measure_def*)

lemma *emeasure_scale_measure* [*simp*]:
emeasure (*scale_measure* *r M*) *A* = *r* * *emeasure M A*
 (is _ = ? μ *A*)
proof(cases *A* $\in$ *sets M*)
 case *True*
 show ?thesis **unfolding** *scale_measure_def*
proof(rule *emeasure_measure_of_sigma*)
 show *sigma_algebra* (*space M*) (*sets M*) ..
 show *positive* (*sets M*) ? μ **by** (simp add: *positive_def*)
 show *countably_additive* (*sets M*) ? μ
proof (rule *countably_additiveI*)
 fix *A* :: *nat* $\Rightarrow$ _ **assume** *: *range A* $\subseteq$ *sets M* *disjoint_family A*
 have ($\sum i. ?\mu (A i)$) = *r* * ($\sum i. \text{emeasure } M (A i)$)
 by *simp*
 also have ... = ? μ ($\bigcup i. A i$) **using** * **by**(simp add: *suminf_emeasure*)
 finally show ($\sum i. ?\mu (A i)$) = ? μ ($\bigcup i. A i$) .
 qed
 qed(fact *True*)
 qed(simp add: *emeasure_notin_sets*)

lemma *scale_measure_1* [*simp*]: *scale_measure* 1 *M* = *M*
 by(rule measure_eqI) simp_all

lemma *scale_measure_0*[*simp*]: *scale_measure* 0 *M* = *null_measure M*
 by(rule measure_eqI) simp_all

lemma *measure_scale_measure* [*simp*]: $0 \leq r \implies \text{measure} (\text{scale\_measure } r M)$
 $A = r * \text{measure } M A$
using *emeasure_scale_measure*[of *r M A*]
emeasure_eq_ennreal_measure[of *M A*]
measure_eq_emeasure_eq_ennreal[of _ *scale_measure* *r M A*]
by (cases *emeasure* (*scale_measure* *r M*) *A* = *top*)
 (auto simp del: *emeasure_scale_measure*
 simp: *ennreal_top_eq_mult_iff* *ennreal_mult_eq_top_iff* *measure_zero_top*
ennreal_mult[symmetric])

lemma *scale_scale_measure* [simp]:
 $\text{scale\_measure } r (\text{scale\_measure } r' M) = \text{scale\_measure } (r * r') M$
by (rule *measure_eqI*) (simp_all add: *max_def mult.assoc*)

lemma *scale_null_measure* [simp]: $\text{scale\_measure } r (\text{null\_measure } M) = \text{null\_measure } M$
by (rule *measure_eqI*) simp_all

6.3.17 Complete lattice structure on measures

lemma (in *finite_measure*) *finite_measure_Diff'*:
 $A \in \text{sets } M \implies B \in \text{sets } M \implies \text{measure } M (A - B) = \text{measure } M A - \text{measure } M (A \cap B)$
using *finite_measure_Diff*[of $A \ A \cap B$] **by** (auto simp: *Diff_Int*)

lemma (in *finite_measure*) *finite_measure_Union'*:
 $A \in \text{sets } M \implies B \in \text{sets } M \implies \text{measure } M (A \cup B) = \text{measure } M A + \text{measure } M (B - A)$
using *finite_measure_Union*[of $A \ B - A$] **by** auto

lemma *finite_unsigned_Hahn_decomposition*:
assumes *finite_measure M finite_measure N* **and** [simp]: $\text{sets } N = \text{sets } M$
shows $\exists Y \in \text{sets } M. (\forall X \in \text{sets } M. X \subseteq Y \implies N X \leq M X) \wedge (\forall X \in \text{sets } M. X \cap Y = \{\} \implies M X \leq N X)$

proof –

interpret M : *finite_measure M* **by** fact

interpret N : *finite_measure N* **by** fact

define d **where** $d X = \text{measure } M X - \text{measure } N X$ **for** X

have [intro]: *bdd_above (d `sets M)*
using *sets.sets_into_space*[of $\_ M$]
by (intro *bdd_aboveI*[**where** $M = \text{measure } M (\text{space } M)$])
(auto simp: *d_def field_simps subset_eq intro!*: *add_increasing M.finite_measure_mono*)

define γ **where** $\gamma = (\text{SUP } X \in \text{sets } M. d X)$
have *le_γ*[intro]: $X \in \text{sets } M \implies d X \leq \gamma$ **for** X
by (auto simp: *γ_def intro!*: *cSUP_upper*)

have $\exists f. \forall n. f n \in \text{sets } M \wedge d (f n) > \gamma - 1 / 2^n$

proof (intro *choice_iff*[*THEN iffD1*] *allI*)

fix n

have $\exists X \in \text{sets } M. \gamma - 1 / 2^n < d X$

unfolding *γ_def* **by** (intro *less_cSUP_iff*[*THEN iffD1*]) auto

then show $\exists y. y \in \text{sets } M \wedge \gamma - 1 / 2^n < d y$

by auto

qed

then obtain E **where** [*measurable*]: $E n \in \text{sets } M$ **and** E : $d (E n) > \gamma - 1 /$

```

 $2^n$  for  $n$ 
  by auto

define F where  $F\ m\ n = (\text{if } m \leq n \text{ then } \bigcap_{i \in \{m..n\}} E\ i \text{ else space } M)$  for  $m$ 
n

have [measurable]:  $m \leq n \implies F\ m\ n \in \text{sets } M$  for  $m\ n$ 
  by (auto simp: F_def)

have 1:  $\gamma - 2 / 2^m + 1 / 2^n \leq d\ (F\ m\ n)$  if  $m \leq n$  for  $m\ n$ 
  using that
proof (induct rule: dec_induct)
  case base with  $E[\text{of } m]$  show ?case
    by (simp add: F_def field_simps)
  next
    case (step i)
    have F_Suc:  $F\ m\ (\text{Suc } i) = F\ m\ i \cap E\ (\text{Suc } i)$ 
      using  $\langle m \leq i \rangle$  by (auto simp: F_def le_Suc_eq)

    have  $\gamma + (\gamma - 2 / 2^m + 1 / 2^{\text{Suc } i}) \leq (\gamma - 1 / 2^{\text{Suc } i}) + (\gamma - 2 / 2^m + 1 / 2^i)$ 
      by (simp add: field_simps)
    also have  $\dots \leq d\ (E\ (\text{Suc } i)) + d\ (F\ m\ i)$ 
      using  $E[\text{of } \text{Suc } i]$  by (intro add_mono step) auto
    also have  $\dots = d\ (E\ (\text{Suc } i)) + d\ (F\ m\ i - E\ (\text{Suc } i)) + d\ (F\ m\ (\text{Suc } i))$ 
      using  $\langle m \leq i \rangle$  by (simp add: d_def field_simps F_Suc M.finite_measure_Diff'
N.finite_measure_Diff')
    also have  $\dots = d\ (E\ (\text{Suc } i) \cup F\ m\ i) + d\ (F\ m\ (\text{Suc } i))$ 
      using  $\langle m \leq i \rangle$  by (simp add: d_def field_simps M.finite_measure_Union'
N.finite_measure_Union')
    also have  $\dots \leq \gamma + d\ (F\ m\ (\text{Suc } i))$ 
      using  $\langle m \leq i \rangle$  by auto
    finally show ?case
      by auto
qed

define F' where  $F'\ m = (\bigcap_{i \in \{m.. \}} E\ i)$  for  $m$ 
have F'_eq:  $F'\ m = (\bigcap_{i. F\ m\ (i + m)})$  for  $m$ 
  by (fastforce simp: le_iff_add[of m] F'_def F_def)

have [measurable]:  $F'\ m \in \text{sets } M$  for  $m$ 
  by (auto simp: F'_def)

have  $\gamma_{le}$ :  $\gamma - 0 \leq d\ (\bigcup m. F'\ m)$ 
proof (rule LIMSEQ_le)
  show  $(\lambda n. \gamma - 2 / 2^n) \longrightarrow \gamma - 0$ 
    by (intro tendsto_intros LIMSEQ_divide_realpow_zero) auto
  have incseq F'
    by (auto simp: incseq_def F'_def)

```

```

then show ( $\lambda m. d (F' m)$ )  $\longrightarrow d (\bigcup m. F' m)$ 
  unfolding  $d\_def$ 
  by (intro tendsto_diff  $M.finite\_Lim\_measure\_incseq N.finite\_Lim\_measure\_incseq$ )
auto

have  $\gamma - 2 / 2^m + 0 \leq d (F' m)$  for  $m$ 
proof (rule LIMSEQ_le)
  have *:  $decseq (\lambda n. F m (n + m))$ 
  by (auto simp:  $decseq\_def F\_def$ )
  show ( $\lambda n. d (F m n)$ )  $\longrightarrow d (F' m)$ 
  unfolding  $d\_def F'\_eq$ 
  by (rule LIMSEQ_offset[where  $k=m$ ])
  (auto intro!:  $tendsto\_diff M.finite\_Lim\_measure\_decseq N.finite\_Lim\_measure\_decseq$ 
*)
  show ( $\lambda n. \gamma - 2 / 2^m + 1 / 2^n$ )  $\longrightarrow \gamma - 2 / 2^m + 0$ 
  by (intro tendsto_add LIMSEQ_divide_realpow_zero tendsto_const) auto
  show  $\exists N. \forall n \geq N. \gamma - 2 / 2^m + 1 / 2^n \leq d (F m n)$ 
  using 1[of  $m$ ] by (intro exI[of _  $m$ ]) auto
qed
then show  $\exists N. \forall n \geq N. \gamma - 2 / 2^n \leq d (F' n)$ 
  by auto
qed

show ?thesis
proof (safe intro!:  $bexI$ [of  $\bigcup m. F' m$ ])
  fix  $X$  assume [measurable]:  $X \in sets M$  and  $X: X \subseteq (\bigcup m. F' m)$ 
  have  $d (\bigcup m. F' m) - d X = d ((\bigcup m. F' m) - X)$ 
  using  $X$  by (auto simp:  $d\_def M.finite\_measure\_Diff N.finite\_measure\_Diff$ )
  also have  $\dots \leq \gamma$ 
  by auto
  finally have  $0 \leq d X$ 
  using  $\gamma\_le$  by auto
  then show  $emeasure N X \leq emeasure M X$ 
  by (auto simp:  $d\_def M.emeasure\_eq\_measure N.emeasure\_eq\_measure$ )
next
  fix  $X$  assume [measurable]:  $X \in sets M$  and  $X: X \cap (\bigcup m. F' m) = \{\}$ 
  then have  $d (\bigcup m. F' m) + d X = d (X \cup (\bigcup m. F' m))$ 
  by (auto simp:  $d\_def M.finite\_measure\_Union N.finite\_measure\_Union$ )
  also have  $\dots \leq \gamma$ 
  by auto
  finally have  $d X \leq 0$ 
  using  $\gamma\_le$  by auto
  then show  $emeasure M X \leq emeasure N X$ 
  by (auto simp:  $d\_def M.emeasure\_eq\_measure N.emeasure\_eq\_measure$ )
qed auto
qed

proposition unsigned_Hahn_decomposition:
  assumes [simp]:  $sets N = sets M$  and [measurable]:  $A \in sets M$ 

```

```

and [simp]: emeasure M A  $\neq$  top emeasure N A  $\neq$  top
shows  $\exists Y \in \text{sets } M. Y \subseteq A \wedge (\forall X \in \text{sets } M. X \subseteq Y \longrightarrow N X \leq M X) \wedge (\forall X \in \text{sets } M. X \subseteq A \longrightarrow X \cap Y = \{\} \longrightarrow M X \leq N X)$ 
proof -
  have  $\exists Y \in \text{sets } (\text{restrict\_space } M A).$ 
     $(\forall X \in \text{sets } (\text{restrict\_space } M A). X \subseteq Y \longrightarrow (\text{restrict\_space } N A) X \leq$ 
 $(\text{restrict\_space } M A) X) \wedge$ 
     $(\forall X \in \text{sets } (\text{restrict\_space } M A). X \cap Y = \{\} \longrightarrow (\text{restrict\_space } M A) X \leq$ 
 $(\text{restrict\_space } N A) X)$ 
  proof (rule finite_unsigned_Hahn_decomposition)
    show finite_measure (restrict_space M A) finite_measure (restrict_space N A)
    by (auto simp: space_restrict_space emeasure_restrict_space less_top intro!: finite_measureI)
  qed (simp add: sets_restrict_space)
  then show ?thesis
  apply -
  apply (erule beE)
  subgoal for Y
    apply (intro beI[of _ Y] conjI ballI conjI)
    apply (simp_all add: sets_restrict_space emeasure_restrict_space)
    apply safe
  subgoal for X Z
    by (erule ballE[of _ _ X]) (auto simp add: Int_absorb1)
  subgoal for X Z
    by (erule ballE[of _ _ X]) (auto simp add: Int_absorb1 ac_simps)
  apply auto
done
done
qed

```

Define a lexicographical order on *measure*, in the order space, sets and measure. The parts of the lexicographical order are point-wise ordered.

```

instantiation measure :: (type) order_bot
begin

```

```

inductive less_eq_measure :: 'a measure  $\Rightarrow$  'a measure  $\Rightarrow$  bool where
  space M  $\subset$  space N  $\Longrightarrow$  less_eq_measure M N
| space M = space N  $\Longrightarrow$  sets M  $\subset$  sets N  $\Longrightarrow$  less_eq_measure M N
| space M = space N  $\Longrightarrow$  sets M = sets N  $\Longrightarrow$  emeasure M  $\leq$  emeasure N  $\Longrightarrow$ 
less_eq_measure M N

```

```

lemma le_measure_iff:

```

```

  M  $\leq$  N  $\longleftrightarrow$  (if space M = space N then
    if sets M = sets N then emeasure M  $\leq$  emeasure N else sets M  $\subseteq$  sets N else
    space M  $\subseteq$  space N)
  by (auto elim: less_eq_measure.cases intro: less_eq_measure.intros)

```

```

definition less_measure :: 'a measure  $\Rightarrow$  'a measure  $\Rightarrow$  bool where
  less_measure M N  $\longleftrightarrow$  (M  $\leq$  N  $\wedge$   $\neg$  N  $\leq$  M)

```

definition *bot_measure* :: 'a measure **where**

bot_measure = *sigma* {} {}

lemma

shows *space_bot[simp]*: *space bot* = {}

and *sets_bot[simp]*: *sets bot* = { { }}

and *emeasure_bot[simp]*: *emeasure bot X* = 0

by (*auto simp: bot_measure_def sigma_sets_empty_eq emeasure_sigma*)

instance

proof *standard*

show *bot* ≤ *a* **for** *a* :: 'a measure

by (*simp add: le_measure_iff bot_measure_def sigma_sets_empty_eq emeasure_sigma le_fun_def*)

qed (*auto simp: le_measure_iff less_measure_def split: if_split_asm intro: measure_eqI*)

end

proposition *le_measure*: *sets M* = *sets N* ⇒ *M* ≤ *N* ⇔ (∀ *A* ∈ *sets M*. *emeasure M A* ≤ *emeasure N A*)

apply −

apply (*auto simp: le_measure_iff le_fun_def dest: sets_eq_imp_space_eq*)

subgoal for *X*

by (*cases X* ∈ *sets M*) (*auto simp: emeasure_notin_sets*)

done

definition *sup_measure'* :: 'a measure ⇒ 'a measure ⇒ 'a measure **where**

sup_measure' A B =

measure_of (*space A*) (*sets A*)

(λ*X*. *SUP* *Y* ∈ *sets A*. *emeasure A* (*X* ∩ *Y*) + *emeasure B* (*X* ∩ − *Y*))

lemma *assumes* [*simp*]: *sets B* = *sets A*

shows *space_sup_measure'[simp]*: *space (sup_measure' A B)* = *space A*

and *sets_sup_measure'[simp]*: *sets (sup_measure' A B)* = *sets A*

using *sets_eq_imp_space_eq[OF assms]* **by** (*simp_all add: sup_measure'_def*)

lemma *emeasure_sup_measure'*:

assumes *sets_eq[simp]*: *sets B* = *sets A* **and** [*simp, intro*]: *X* ∈ *sets A*

shows *emeasure (sup_measure' A B) X* = (*SUP* *Y* ∈ *sets A*. *emeasure A* (*X* ∩ *Y*) + *emeasure B* (*X* ∩ − *Y*))

(*is* _ = ?*S X*)

proof −

note *sets_eq_imp_space_eq[OF sets_eq, simp]*

show ?*thesis*

using *sup_measure'_def*

proof (*rule emeasure_measure_of*)

let ?*d* = λ*X Y*. *emeasure A* (*X* ∩ *Y*) + *emeasure B* (*X* ∩ − *Y*)

```

show countably_additive (sets (sup_measure' A B)) ( $\lambda X. \text{SUP } Y \in \text{sets } A. \text{emeasure } A (X \cap Y) + \text{emeasure } B (X \cap - Y)$ )
proof (rule countably_additiveI, goal_cases)
  case (1 X)
  then have [measurable]:  $\bigwedge i. X \ i \in \text{sets } A$  and disjoint_family X
  by auto
  have disjoint: disjoint_family ( $\lambda i. X \ i \cap Y$ ) disjoint_family ( $\lambda i. X \ i - Y$ )
for Y
  by (auto intro: disjoint_family_on_bisimulation [OF <disjoint_family X>,
simplified])
  have ( $\sum i. ?S (X \ i)$ ) = ( $\text{SUP } Y \in \text{sets } A. \sum i. ?d (X \ i) \ Y$ )
  proof (rule ennreal_suminf_SUP_eq_directed)
    fix J :: nat set and a b assume finite J and [measurable]:  $a \in \text{sets } A \ b \in \text{sets } A$ 
    have  $\exists c \in \text{sets } A. c \subseteq X \ i \wedge (\forall a \in \text{sets } A. ?d (X \ i) \ a \leq ?d (X \ i) \ c)$  for i
    proof cases
      assume  $\text{emeasure } A (X \ i) = \text{top} \vee \text{emeasure } B (X \ i) = \text{top}$ 
      then show ?thesis
      proof
        assume  $\text{emeasure } A (X \ i) = \text{top}$  then show ?thesis
        by (intro bexI[of _ X i]) auto
      next
        assume  $\text{emeasure } B (X \ i) = \text{top}$  then show ?thesis
        by (intro bexI[of _ {}]) auto
      qed
    next
    assume finite:  $\neg (\text{emeasure } A (X \ i) = \text{top} \vee \text{emeasure } B (X \ i) = \text{top})$ 
    then have  $\exists Y \in \text{sets } A. Y \subseteq X \ i \wedge (\forall C \in \text{sets } A. C \subseteq Y \longrightarrow B \ C \leq A \ C) \wedge (\forall C \in \text{sets } A. C \subseteq X \ i \longrightarrow C \cap Y = \{\} \longrightarrow A \ C \leq B \ C)$ 
    using unsigned_Hahn_decomposition[of B A X i] by simp
    then obtain Y where [measurable]:  $Y \in \text{sets } A$  and [simp]:  $Y \subseteq X \ i$ 
    and B_le_A:  $\bigwedge C. C \in \text{sets } A \Longrightarrow C \subseteq Y \Longrightarrow B \ C \leq A \ C$ 
    and A_le_B:  $\bigwedge C. C \in \text{sets } A \Longrightarrow C \subseteq X \ i \Longrightarrow C \cap Y = \{\} \Longrightarrow A \ C \leq B \ C$ 
    by auto

    show ?thesis
    proof (intro bexI[of _ Y] ballI conjI)
      fix a assume [measurable]:  $a \in \text{sets } A$ 
      have *:  $(X \ i \cap a \cap Y \cup (X \ i \cap a - Y)) = X \ i \cap a (X \ i - a) \cap Y \cup (X \ i - a - Y) = X \ i \cap - a$ 
      for a Y by auto
      then have ?d (X i) a =
         $(A (X \ i \cap a \cap Y) + A (X \ i \cap a \cap - Y)) + (B (X \ i \cap - a \cap Y) + B (X \ i \cap - a \cap - Y))$ 
      by (subst (1 2) plus_emeasure) (auto simp: Diff_eq[symmetric])
      also have ...  $\leq (A (X \ i \cap a \cap Y) + B (X \ i \cap a \cap - Y)) + (A (X \ i \cap - a \cap Y) + B (X \ i \cap - a \cap - Y))$ 
      by (intro add_mono order_refl B_le_A A_le_B) (auto simp:

```

```

Diff_eq[symmetric])
  also have ... ≤ (A (X i ∩ Y ∩ a) + A (X i ∩ Y ∩ - a)) + (B (X i ∩
- Y ∩ a) + B (X i ∩ - Y ∩ - a))
    by (simp add: ac_simps)
  also have ... ≤ A (X i ∩ Y) + B (X i ∩ - Y)
    by (subst (1 2) plus_emeasure) (auto simp: Diff_eq[symmetric] *)
  finally show ?d (X i) a ≤ ?d (X i) Y .
qed auto
qed
then obtain C where [measurable]: C i ∈ sets A and C i ⊆ X i
  and C: ⋀a. a ∈ sets A ⇒ ?d (X i) a ≤ ?d (X i) (C i) for i
  by metis
have *: X i ∩ (⋃ i. C i) = X i ∩ C i for i
proof safe
  fix x j assume x ∈ X i x ∈ C j
  moreover have i = j ∨ X i ∩ X j = {}
    using ⟨disjoint_family X⟩ by (auto simp: disjoint_family_on_def)
  ultimately show x ∈ C i
    using ⟨C i ⊆ X i⟩ ⟨C j ⊆ X j⟩ by auto
qed auto
have **: X i ∩ - (⋃ i. C i) = X i ∩ - C i for i
proof safe
  fix x j assume x ∈ X i x ∉ C i x ∈ C j
  moreover have i = j ∨ X i ∩ X j = {}
    using ⟨disjoint_family X⟩ by (auto simp: disjoint_family_on_def)
  ultimately show False
    using ⟨C i ⊆ X i⟩ ⟨C j ⊆ X j⟩ by auto
qed auto
show ∃ c ∈ sets A. ∀ i ∈ J. ?d (X i) a ≤ ?d (X i) c ∧ ?d (X i) b ≤ ?d (X i) c
  apply (intro bexI[of _ ⋃ i. C i])
  unfolding * **
  apply (intro C ballI conjI)
  apply auto
  done
qed
also have ... = ?S (⋃ i. X i)
  apply (simp only: UN_extend_simps(4))
  apply (rule arg_cong [of _ _ Sup])
  apply (rule image_cong)
  apply (fact refl)
  using disjoint
  apply (auto simp add: suminf_add [symmetric] Diff_eq [symmetric] im-
age_subset_iff suminf_emeasure simp del: UN_simps)
  done
finally show (∑ i. ?S (X i)) = ?S (⋃ i. X i) .
qed
qed (auto dest: sets.sets_into_space simp: positive_def intro!: SUP_const)
qed

```


lemma *le_emeasure_sup_measure'1:*

assumes *sets B = sets A X ∈ sets A* **shows** $\text{emeasure } A \ X \leq \text{emeasure } (\text{sup\_measure}' A \ B) \ X$
by (*subst emeasure_sup_measure'[OF assms]*) (*auto intro!: SUP_upper2[of X] assms*)

lemma *le_emeasure_sup_measure'2:*

assumes *sets B = sets A X ∈ sets A* **shows** $\text{emeasure } B \ X \leq \text{emeasure } (\text{sup\_measure}' A \ B) \ X$
by (*subst emeasure_sup_measure'[OF assms]*) (*auto intro!: SUP_upper2[of {}] assms*)

lemma *emeasure_sup_measure'_le2:*

assumes [*simp*]: *sets B = sets C sets A = sets C* **and** [*measurable*]: *X ∈ sets C*
assumes *A*: $\bigwedge Y. Y \subseteq X \implies Y \in \text{sets } A \implies \text{emeasure } A \ Y \leq \text{emeasure } C \ Y$
assumes *B*: $\bigwedge Y. Y \subseteq X \implies Y \in \text{sets } A \implies \text{emeasure } B \ Y \leq \text{emeasure } C \ Y$
shows $\text{emeasure } (\text{sup\_measure}' A \ B) \ X \leq \text{emeasure } C \ X$
proof (*subst emeasure_sup_measure'*)
show ($\text{SUP } Y \in \text{sets } A. \text{emeasure } A \ (X \cap Y) + \text{emeasure } B \ (X \cap - Y) \leq \text{emeasure } C \ X$)
unfolding $\langle \text{sets } A = \text{sets } C \rangle$
proof (*intro SUP_least*)
fix *Y* **assume** [*measurable*]: *Y ∈ sets C*
have [*simp*]: $X \cap Y \cup (X - Y) = X$
by *auto*
have $\text{emeasure } A \ (X \cap Y) + \text{emeasure } B \ (X \cap - Y) \leq \text{emeasure } C \ (X \cap Y) + \text{emeasure } C \ (X \cap - Y)$
by (*intro add_mono A B*) (*auto simp: Diff_eq[symmetric]*)
also have $\dots = \text{emeasure } C \ X$
by (*subst plus_emeasure*) (*auto simp: Diff_eq[symmetric]*)
finally show $\text{emeasure } A \ (X \cap Y) + \text{emeasure } B \ (X \cap - Y) \leq \text{emeasure } C \ X$.
qed
qed *simp_all*

definition *sup_lexord* :: $'a \Rightarrow 'a \Rightarrow ('a \Rightarrow 'b :: \text{order}) \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**

sup_lexord A B k s c =
(if k A = k B then c else
if $\neg k A \leq k B \wedge \neg k B \leq k A$ then s else
if $k B \leq k A$ then A else B)

lemma *sup_lexord:*

$(k A < k B \implies P B) \implies (k B < k A \implies P A) \implies (k A = k B \implies P c) \implies$
 $(\neg k B \leq k A \implies \neg k A \leq k B \implies P s) \implies P (\text{sup\_lexord } A \ B \ k \ s \ c)$
by (*auto simp: sup_lexord_def*)

lemmas *le_sup_lexord* = *sup_lexord*[**where** $P = \lambda a. c \leq a$ **for** *c*]

lemma *sup_lexord1*: $k A = k B \implies \text{sup\_lexord } A \ B \ k \ s \ c = c$

by (*simp add: sup_lexord_def*)

lemma *sup_lexord_commute*: *sup_lexord A B k s c = sup_lexord B A k s c*
by (*auto simp: sup_lexord_def*)

lemma *sigma_sets_le_sets_iff*: (*sigma_sets (space x) $\mathcal{A} \subseteq \text{sets } x$*) = (*$\mathcal{A} \subseteq \text{sets } x$*)
using *sets.sigma_sets_subset[of $\mathcal{A} x$]* **by** *auto*

lemma *sigma_le_iff*: *$\mathcal{A} \subseteq \text{Pow } \Omega \implies \text{sigma } \Omega \mathcal{A} \leq x \longleftrightarrow (\Omega \subseteq \text{space } x \wedge (\text{space } x = \Omega \longrightarrow \mathcal{A} \subseteq \text{sets } x))$*
by (*cases $\Omega = \text{space } x$*)
(simp_all add: eq_commute[of _ sets x] le_measure_iff emeasure_sigma le_fun_def
sigma_sets_superset_generator sigma_sets_le_sets_iff)

instantiation *measure* :: (*type*) *semilattice_sup*
begin

definition *sup_measure* :: '*a* *measure* $\Rightarrow$ '*a* *measure* $\Rightarrow$ '*a* *measure* **where**
sup_measure A B =
sup_lexord A B space (sigma (space A $\cup$ space B) {})
(sup_lexord A B sets (sigma (space A) (sets A $\cup$ sets B))) (sup_measure' A
B))

instance

proof

fix *x y z* :: '*a* *measure*

show *x $\leq$ sup x y*

unfolding *sup_measure_def*

proof (*intro le_sup_lexord*)

assume *space x = space y*

then have **: sets x $\cup$ sets y $\subseteq$ Pow (space x)*

using *sets.space_closed* **by** *auto*

assume *$\neg \text{sets } y \subseteq \text{sets } x \neg \text{sets } x \subseteq \text{sets } y$*

then have *sets x $\subset$ sets x $\cup$ sets y*

by *auto*

also have *$\dots \leq \text{sigma (space x) (sets x $\cup$ sets y)$*

by (*subst sets_measure_of[OF *]*) (*rule sigma_sets_superset_generator*)

finally show *x $\leq$ sigma (space x) (sets x $\cup$ sets y)*

by (*simp add: space_measure_of[OF *] less_eq_measure.intros(2)*)

next

assume *$\neg \text{space } y \subseteq \text{space } x \neg \text{space } x \subseteq \text{space } y$*

then show *x $\leq$ sigma (space x $\cup$ space y) {}*

by (*intro less_eq_measure.intros*) *auto*

next

assume *sets x = sets y* **then show** *x $\leq$ sup_measure' x y*

by (*simp add: le_measure le_emeasure_sup_measure'1*)

qed (*auto intro: less_eq_measure.intros*)

```

show  $y \leq \sup x y$ 
  unfolding sup_measure_def
proof (intro le_sup_lexord)
  assume **:  $\text{space } x = \text{space } y$ 
  then have *:  $\text{sets } x \cup \text{sets } y \subseteq \text{Pow } (\text{space } y)$ 
    using sets.space_closed by auto
  assume  $\neg \text{sets } y \subseteq \text{sets } x \neg \text{sets } x \subseteq \text{sets } y$ 
  then have  $\text{sets } y \subset \text{sets } x \cup \text{sets } y$ 
    by auto
  also have  $\dots \leq \sigma (\text{space } y) (\text{sets } x \cup \text{sets } y)$ 
    by (subst sets_measure_of[OF *]) (rule sigma_sets_superset_generator)
  finally show  $y \leq \sigma (\text{space } x) (\text{sets } x \cup \text{sets } y)$ 
    by (simp add: ** space_measure_of[OF *] less_eq_measure.intros(2))
next
  assume  $\neg \text{space } y \subseteq \text{space } x \neg \text{space } x \subseteq \text{space } y$ 
  then show  $y \leq \sigma (\text{space } x \cup \text{space } y) \{\}$ 
    by (intro less_eq_measure.intros) auto
next
  assume  $\text{sets } x = \text{sets } y$  then show  $y \leq \sup\_measure' x y$ 
    by (simp add: le_measure le_emeasure_sup_measure'2)
qed (auto intro: less_eq_measure.intros)
show  $x \leq y \implies z \leq y \implies \sup x z \leq y$ 
  unfolding sup_measure_def
proof (intro sup_lexord[where  $P = \lambda x. x \leq y$ ])
  assume  $x \leq y z \leq y$  and [simp]:  $\text{space } x = \text{space } z \text{ sets } x = \text{sets } z$ 
  from  $\langle x \leq y \rangle$  show  $\sup\_measure' x z \leq y$ 
  proof cases
    case 1 then show ?thesis
      by (intro less_eq_measure.intros(1)) simp
  next
    case 2 then show ?thesis
      by (intro less_eq_measure.intros(2)) simp_all
  next
    case 3 with  $\langle z \leq y \rangle \langle x \leq y \rangle$  show ?thesis
      by (auto simp add: le_measure intro!: emeasure_sup_measure'_le2)
  qed
next
  assume **:  $x \leq y z \leq y \text{ space } x = \text{space } z \neg \text{sets } z \subseteq \text{sets } x \neg \text{sets } x \subseteq \text{sets } z$ 
  then have *:  $\text{sets } x \cup \text{sets } z \subseteq \text{Pow } (\text{space } x)$ 
    using sets.space_closed by auto
  show  $\sigma (\text{space } x) (\text{sets } x \cup \text{sets } z) \leq y$ 
    unfolding sigma_le_iff[OF *] using ** by (auto simp: le_measure_iff split:
if_split_asm)
  next
    assume  $x \leq y z \leq y \neg \text{space } z \subseteq \text{space } x \neg \text{space } x \subseteq \text{space } z$ 
  then have  $\text{space } x \subseteq \text{space } y \text{ space } z \subseteq \text{space } y$ 
    by (auto simp: le_measure_iff split: if_split_asm)
  then show  $\sigma (\text{space } x \cup \text{space } z) \{\} \leq y$ 
    by (simp add: sigma_le_iff)

```

1368

qed
qed

end

lemma *space_empty_eq_bot*: $\text{space } a = \{\}$ $\longleftrightarrow a = \text{bot}$
using *space_empty[of a]* **by** (*auto intro!*: *measure_eqI*)

lemma *sets_eq_iff_bounded*: $A \leq B \implies B \leq C \implies \text{sets } A = \text{sets } C \implies \text{sets } B = \text{sets } A$
by (*auto dest: sets_eq_imp_space_eq simp add: le_measure_iff split: if_split_asm*)

lemma *sets_sup*: $\text{sets } A = \text{sets } M \implies \text{sets } B = \text{sets } M \implies \text{sets } (\text{sup } A \ B) = \text{sets } M$
by (*auto simp add: sup_measure_def sup_lexord_def dest: sets_eq_imp_space_eq*)

lemma *le_measureD1*: $A \leq B \implies \text{space } A \leq \text{space } B$
by (*auto simp: le_measure_iff split: if_split_asm*)

lemma *le_measureD2*: $A \leq B \implies \text{space } A = \text{space } B \implies \text{sets } A \leq \text{sets } B$
by (*auto simp: le_measure_iff split: if_split_asm*)

lemma *le_measureD3*: $A \leq B \implies \text{sets } A = \text{sets } B \implies \text{emeasure } A \ X \leq \text{emeasure } B \ X$
by (*auto simp: le_measure_iff le_fun_def dest: sets_eq_imp_space_eq split: if_split_asm*)

lemma *UN_space_closed*: $\bigcup (\text{sets } 'S) \subseteq \text{Pow } (\bigcup (\text{space } 'S))$
using *sets.space_closed* **by** *auto*

definition

Sup_lexord :: $('a \Rightarrow 'b :: \text{complete\_lattice}) \Rightarrow ('a \text{ set} \Rightarrow 'a) \Rightarrow ('a \text{ set} \Rightarrow 'a) \Rightarrow 'a$
 $\text{set} \Rightarrow 'a$

where

$\text{Sup\_lexord } k \ c \ s \ A =$
 $(\text{let } U = (\text{SUP } a \in A. \ k \ a)$
 $\text{in if } \exists a \in A. \ k \ a = U \text{ then } c \ \{a \in A. \ k \ a = U\} \text{ else } s \ A)$

lemma *Sup_lexord*:
 $(\bigwedge a \ S. \ a \in A \implies k \ a = (\text{SUP } a \in A. \ k \ a) \implies S = \{a' \in A. \ k \ a' = k \ a\} \implies P \ (c \ S)) \implies ((\bigwedge a. \ a \in A \implies k \ a \neq (\text{SUP } a \in A. \ k \ a)) \implies P \ (s \ A)) \implies P \ (\text{Sup\_lexord } k \ c \ s \ A)$
by (*auto simp: Sup_lexord_def Let_def*)

lemma *Sup_lexord1*:
assumes $A: A \neq \{\}$ $(\bigwedge a. \ a \in A \implies k \ a = (\bigcup a \in A. \ k \ a))$ $P \ (c \ A)$
shows $P \ (\text{Sup\_lexord } k \ c \ s \ A)$
unfolding *Sup_lexord_def Let_def*
proof (*clarsimp, safe*)

```

show  $\forall a \in A. k\ a \neq (\bigcup x \in A. k\ x) \implies P\ (s\ A)$ 
  by (metis assms(1,2) ex_in_conv)
next
  fix a assume  $a \in A\ k\ a = (\bigcup x \in A. k\ x)$ 
  then have  $\{a \in A. k\ a = (\bigcup x \in A. k\ x)\} = \{a \in A. k\ a = k\ a\}$ 
    by (metis A(2)[symmetric])
  then show  $P\ (c\ \{a \in A. k\ a = (\bigcup x \in A. k\ x)\})$ 
    by (simp add: A(3))
qed

instantiation measure :: (type) complete_lattice
begin

interpretation sup_measure: comm_monoid_set sup bot :: 'a measure
  by standard (auto intro!: antisym)

lemma sup_measure_F_mono':
  finite J  $\implies$  finite I  $\implies$  sup_measure.F id I  $\leq$  sup_measure.F id (I  $\cup$  J)
proof (induction J rule: finite_induct)
  case empty then show ?case
    by simp
next
  case (insert i J)
  show ?case
  proof cases
    assume  $i \in I$  with insert show ?thesis
      by (auto simp: insert_absorb)
  next
    assume  $i \notin I$ 
    have sup_measure.F id I  $\leq$  sup_measure.F id (I  $\cup$  J)
      by (intro insert)
    also have  $\dots \leq$  sup_measure.F id (insert i (I  $\cup$  J))
      using insert  $\langle i \notin I \rangle$  by (subst sup_measure.insert) auto
    finally show ?thesis
      by auto
  qed
qed

lemma sup_measure_F_mono: finite I  $\implies$   $J \subseteq I \implies$  sup_measure.F id J  $\leq$ 
sup_measure.F id I
  using sup_measure_F_mono'[of I J] by (auto simp: finite_subset Un_absorb1)

lemma sets_sup_measure_F:
  finite I  $\implies$   $I \neq \{\}$   $\implies$   $(\bigwedge i. i \in I \implies \text{sets } i = \text{sets } M) \implies \text{sets } (\text{sup\_measure.F id } I) = \text{sets } M$ 
  by (induction I rule: finite_ne_induct) (simp_all add: sets_sup)

definition Sup_measure' :: 'a measure set  $\Rightarrow$  'a measure where
Sup_measure' M =

```

$measure\_of (\bigcup a \in M. space\ a) (\bigcup a \in M. sets\ a)$
 $(\lambda X. (SUP\ P \in \{P. finite\ P \wedge P \subseteq M\}. sup\_measure.F\ id\ P\ X))$

lemma $space\_Sup\_measure'2$: $space\ (Sup\_measure'\ M) = (\bigcup m \in M. space\ m)$
unfolding $Sup\_measure'\_def$ **by** $(intro\ space\_measure\_of[OF\ UN\_space\_closed])$

lemma $sets\_Sup\_measure'2$: $sets\ (Sup\_measure'\ M) = sigma\_sets\ (\bigcup m \in M. space\ m) (\bigcup m \in M. sets\ m)$
unfolding $Sup\_measure'\_def$ **by** $(intro\ sets\_measure\_of[OF\ UN\_space\_closed])$

lemma $sets\_Sup\_measure'$:
assumes $sets\_eq[simp]$: $\bigwedge m. m \in M \implies sets\ m = sets\ A$ **and** $M \neq \{\}$
shows $sets\ (Sup\_measure'\ M) = sets\ A$
using $sets\_eq[THEN\ sets\_eq\_imp\_space\_eq, simp]$ $\langle M \neq \{\} \rangle$ **by** $(simp\ add:$
 $Sup\_measure'\_def)$

lemma $space\_Sup\_measure'$:
assumes $sets\_eq[simp]$: $\bigwedge m. m \in M \implies sets\ m = sets\ A$ **and** $M \neq \{\}$
shows $space\ (Sup\_measure'\ M) = space\ A$
using $sets\_eq[THEN\ sets\_eq\_imp\_space\_eq, simp]$ $\langle M \neq \{\} \rangle$
by $(simp\ add: Sup\_measure'\_def)$

lemma $emeasure\_Sup\_measure'$:
assumes $sets\_eq[simp]$: $\bigwedge m. m \in M \implies sets\ m = sets\ A$ **and** $X \in sets\ A\ M \neq \{\}$
shows $emeasure\ (Sup\_measure'\ M)\ X = (SUP\ P \in \{P. finite\ P \wedge P \subseteq M\}. sup\_measure.F\ id\ P\ X)$
 $(is\ \_ = ?S\ X)$
using $Sup\_measure'\_def$
proof $(rule\ emeasure\_measure\_of)$
note $sets\_eq[THEN\ sets\_eq\_imp\_space\_eq, simp]$
have $*$: $sets\ (Sup\_measure'\ M) = sets\ A$ $space\ (Sup\_measure'\ M) = space\ A$
using $\langle M \neq \{\} \rangle$ **by** $(simp\_all\ add: Sup\_measure'\_def)$
let $? \mu = sup\_measure.F\ id$
show $countably\_additive\ (sets\ (Sup\_measure'\ M))\ ?S$
proof $(rule\ countably\_additiveI, goal\_cases)$
case $(1\ F)$
then have $**$: $range\ F \subseteq sets\ A$
by $(auto\ simp: *)$
show $(\sum i. ?S\ (F\ i)) = ?S\ (\bigcup i. F\ i)$
proof $(subst\ ennreal\_suminf\_SUP\_eq\_directed)$
fix $i\ j$ **and** $N :: nat\ set$ **assume** ij : $i \in \{P. finite\ P \wedge P \subseteq M\}\ j \in \{P. finite\ P \wedge P \subseteq M\}$
have $(i \neq \{\} \longrightarrow sets\ (? \mu\ i) = sets\ A) \wedge (j \neq \{\} \longrightarrow sets\ (? \mu\ j) = sets\ A) \wedge$
 $(i \neq \{\} \vee j \neq \{\} \longrightarrow sets\ (? \mu\ (i \cup j)) = sets\ A)$
using ij **by** $(intro\ impI\ sets\_sup\_measure\_F\ conjI)\ auto$
then have $? \mu\ j\ (F\ n) \leq ? \mu\ (i \cup j)\ (F\ n) \wedge ? \mu\ i\ (F\ n) \leq ? \mu\ (i \cup j)\ (F\ n)$
for n
using ij

```

    by (cases i = {}; cases j = {})
      (auto intro!: le_measureD3 sup_measure_F_mono simp: sets_sup_measure_F
        simp del: id_apply)
  with ij show  $\exists k \in \{P. \text{finite } P \wedge P \subseteq M\}. \forall n \in N. ?\mu \ i \ (F \ n) \leq ?\mu \ k \ (F \ n)$ 
 $\wedge ?\mu \ j \ (F \ n) \leq ?\mu \ k \ (F \ n)$ 
    by (safe intro!: bexI[of _ i  $\cup$  j]) auto
  next
    show  $(\text{SUP } P \in \{P. \text{finite } P \wedge P \subseteq M\}. \sum n. ?\mu \ P \ (F \ n)) = (\text{SUP } P \in \{P. \text{finite } P \wedge P \subseteq M\}. ?\mu \ P \ (\bigcup (F \ ' \ \text{UNIV})))$ 
    proof (intro arg_cong [of _ _ Sup] image_cong refl)
      fix i assume i:  $i \in \{P. \text{finite } P \wedge P \subseteq M\}$ 
      show  $(\sum n. ?\mu \ i \ (F \ n)) = ?\mu \ i \ (\bigcup (F \ ' \ \text{UNIV}))$ 
      proof cases
        assume  $i \neq \{\}$  with i ** show ?thesis
          apply (intro suminf_emeasure ‹disjoint_family F›)
          apply (subst sets_sup_measure_F[OF _ _ sets_eq])
          apply auto
        done
      qed simp
    qed
  qed
  qed
  show positive (sets (Sup_measure' M)) ?S
    by (auto simp: positive_def bot_enreal[symmetric])
  show  $X \in \text{sets } (\text{Sup\_measure}' M)$ 
    using assms * by auto
qed (rule UN_space_closed)

```

definition *Sup_measure* :: 'a measure set $\Rightarrow$ 'a measure **where**
Sup_measure =
 Sup_lexord space
 (Sup_lexord sets *Sup_measure'*
 ($\lambda U. \text{sigma } (\bigcup u \in U. \text{space } u) (\bigcup u \in U. \text{sets } u)$))
 ($\lambda U. \text{sigma } (\bigcup u \in U. \text{space } u) \{\}$)

definition *Inf_measure* :: 'a measure set $\Rightarrow$ 'a measure **where**
Inf_measure $A = \text{Sup } \{x. \forall a \in A. x \leq a\}$

definition *inf_measure* :: 'a measure $\Rightarrow$ 'a measure $\Rightarrow$ 'a measure **where**
inf_measure $a \ b = \text{Inf } \{a, b\}$

definition *top_measure* :: 'a measure **where**
top_measure = *Inf* $\{\}$

instance

proof

note UN_space_closed [simp]

show upper: $x \leq \text{Sup } A$ if $x \in A$ for $x :: 'a$ measure **and** A
 unfolding *Sup_measure_def*

```

proof (intro Sup_lexord[where  $P = \lambda y. x \leq y$ ])
  assume  $\bigwedge a. a \in A \implies \text{space } a \neq (\bigcup_{a \in A. \text{space } a})$ 
  from this[OF  $\langle x \in A \rangle$ ]  $\langle x \in A \rangle$  show  $x \leq \text{sigma } (\bigcup_{a \in A. \text{space } a}) \{\}$ 
    by (intro less_eq_measure.intros) auto
next
  fix  $a \ S$  assume  $a \in A$  and  $a: \text{space } a = (\bigcup_{a \in A. \text{space } a})$  and  $S: S = \{a' \in$ 
 $A. \text{space } a' = \text{space } a\}$ 
    and  $\text{neg}: \bigwedge aa. aa \in S \implies \text{sets } aa \neq (\bigcup_{a \in S. \text{sets } a})$ 
  have  $\text{sp\_}a: \text{space } a = (\bigcup (\text{space } 'S))$ 
    using  $\langle a \in A \rangle$  by (auto simp:  $S$ )
  show  $x \leq \text{sigma } (\bigcup (\text{space } 'S)) (\bigcup (\text{sets } 'S))$ 
proof cases
  assume [simp]:  $\text{space } x = \text{space } a$ 
  have  $\text{sets } x \subset (\bigcup_{a \in S. \text{sets } a})$ 
    using  $\langle x \in A \rangle$   $\text{neg}[\text{of } x]$  by (auto simp:  $S$ )
  also have  $\dots \subseteq \text{sigma\_sets } (\bigcup_{x \in S. \text{space } x}) (\bigcup_{x \in S. \text{sets } x})$ 
    by (rule sigma_sets_superset_generator)
  finally show ?thesis
    by (intro less_eq_measure.intros(2)) (simp_all add:  $\text{sp\_}a$ )
next
  assume  $\text{space } x \neq \text{space } a$ 
  moreover have  $\text{space } x \leq \text{space } a$ 
    unfolding  $a$  using  $\langle x \in A \rangle$  by auto
  ultimately show ?thesis
    by (intro less_eq_measure.intros) (simp add: less_le  $\text{sp\_}a$ )
qed
next
  fix  $a \ b \ S \ S'$  assume  $a \in A$  and  $a: \text{space } a = (\bigcup_{a \in A. \text{space } a})$  and  $S: S =$ 
 $\{a' \in A. \text{space } a' = \text{space } a\}$ 
    and  $b \in S$  and  $b: \text{sets } b = (\bigcup_{a \in S. \text{sets } a})$  and  $S': S' = \{a' \in S. \text{sets } a' =$ 
 $\text{sets } b\}$ 
  then have  $S' \neq \{\}$   $\text{space } b = \text{space } a$ 
    by auto
  have  $\text{sets\_eq}: \bigwedge x. x \in S' \implies \text{sets } x = \text{sets } b$ 
    by (auto simp:  $S'$ )
  note  $\text{sets\_eq}[\text{THEN } \text{sets\_eq\_imp\_space\_eq}, \text{simp}]$ 
  have *:  $\text{sets } (\text{Sup\_measure}' S') = \text{sets } b \text{ space } (\text{Sup\_measure}' S') = \text{space } b$ 
    using  $\langle S' \neq \{\} \rangle$  by (simp_all add:  $\text{Sup\_measure}'\_def \text{sets\_eq}$ )
  show  $x \leq \text{Sup\_measure}' S'$ 
proof cases
  assume  $x \in S$ 
  with  $\langle b \in S \rangle$  have  $\text{space } x = \text{space } b$ 
    by (simp add:  $S$ )
  show ?thesis
proof cases
  assume  $x \in S'$ 
  show  $x \leq \text{Sup\_measure}' S'$ 
proof (intro le_measure[THEN iffD2] ballI)
  show  $\text{sets } x = \text{sets } (\text{Sup\_measure}' S')$ 

```



```

      using  $\langle x \in S' \rangle$  * by (simp add: S')
    fix X assume X  $\in$  sets x
    show emeasure x X  $\leq$  emeasure (Sup_measure' S') X
    proof (subst emeasure_Sup_measure'[OF _  $\langle X \in \text{sets } x \rangle$ ])
      show emeasure x X  $\leq$  (SUP P  $\in$  {P. finite P  $\wedge$  P  $\subseteq$  S'}). emeasure
(sup_measure.F id P) X
      using  $\langle x \in S' \rangle$  by (intro SUP_upper2[where i={x}]) auto
    qed (insert  $\langle x \in S' \rangle$  S', auto)
  qed
next
  assume x  $\notin$  S'
  then have sets x  $\neq$  sets b
  using  $\langle x \in S \rangle$  by (auto simp: S')
  moreover have sets x  $\leq$  sets b
  using  $\langle x \in S \rangle$  unfolding b by auto
  ultimately show ?thesis
  using *  $\langle x \in S \rangle$ 
  by (intro less_eq_measure.intros(2))
    (simp_all add: *  $\langle \text{space } x = \text{space } b \rangle$  less_le)
  qed
next
  assume x  $\notin$  S
  with  $\langle x \in A \rangle$   $\langle x \notin S \rangle$   $\langle \text{space } b = \text{space } a \rangle$  show ?thesis
  by (intro less_eq_measure.intros)
    (simp_all add: * less_le a SUP_upper S)
  qed
qed
show least: Sup A  $\leq$  x if x:  $\bigwedge z. z \in A \implies z \leq x$  for x :: 'a measure and A
  unfolding Sup_measure_def
  proof (intro Sup_lexord[where P= $\lambda y. y \leq x$ ])
    assume  $\bigwedge a. a \in A \implies \text{space } a \neq (\bigcup a \in A. \text{space } a)$ 
    show sigma ( $\bigcup (\text{space } 'A)$ ) {}  $\leq$  x
    using x[THEN le_measureD1] by (subst sigma_le_iff) auto
  next
    fix a S assume a  $\in$  A space a = ( $\bigcup a \in A. \text{space } a$ ) and S: S = {a'  $\in$  A. space
a' = space a}
     $\bigwedge a. a \in S \implies \text{sets } a \neq (\bigcup a \in S. \text{sets } a)$ 
    have  $\bigcup (\text{space } 'S) \subseteq \text{space } x$ 
    using S le_measureD1[OF x] by auto
    moreover
    have  $\bigcup (\text{space } 'S) = \text{space } a$ 
    using  $\langle a \in A \rangle$  S by auto
    then have space x =  $\bigcup (\text{space } 'S) \implies \bigcup (\text{sets } 'S) \subseteq \text{sets } x$ 
    using  $\langle a \in A \rangle$  le_measureD2[OF x] by (auto simp: S)
    ultimately show sigma ( $\bigcup (\text{space } 'S)$ ) ( $\bigcup (\text{sets } 'S)$ )  $\leq$  x
    by (subst sigma_le_iff) simp_all
  next
    fix a b S S' assume a  $\in$  A and a: space a = ( $\bigcup a \in A. \text{space } a$ ) and S: S =
{a'  $\in$  A. space a' = space a}

```

```

    and  $b \in S$  and  $b$ : sets  $b = (\bigcup_{a \in S} \text{sets } a)$  and  $S'$ :  $S' = \{a' \in S. \text{sets } a' = \text{sets } b\}$ 
  then have  $S' \neq \{\}$  space  $b = \text{space } a$ 
  by auto
  have sets_eq:  $\bigwedge x. x \in S' \implies \text{sets } x = \text{sets } b$ 
  by (auto simp:  $S'$ )
  note sets_eq[THEN sets_eq_imp_space_eq, simp]
  have *: sets (Sup_measure'  $S'$ ) = sets  $b$  space (Sup_measure'  $S'$ ) = space  $b$ 
  using  $\langle S' \neq \{\} \rangle$  by (simp_all add: Sup_measure'_def sets_eq)
  show Sup_measure'  $S' \leq x$ 
  proof cases
    assume space  $x = \text{space } a$ 
    show ?thesis
  proof cases
    assume **: sets  $x = \text{sets } b$ 
    show ?thesis
  proof (intro le_measure[THEN iffD2] ballI)
    show ***: sets (Sup_measure'  $S'$ ) = sets  $x$ 
    by (simp add: *)
  fix  $X$  assume  $X \in \text{sets } (Sup\_measure' S')$ 
  show emeasure (Sup_measure'  $S'$ )  $X \leq \text{emeasure } x X$ 
  unfolding ***
  proof (subst emeasure_Sup_measure'[OF _  $\langle X \in \text{sets } (Sup\_measure' S') \rangle$ ])
    show (SUP  $P \in \{P. \text{finite } P \wedge P \subseteq S'\}. \text{emeasure } (\text{sup\_measure.F id } P) X) \leq \text{emeasure } x X$ 
  proof (safe intro!: SUP_least)
    fix  $P$  assume  $P$ : finite  $P$   $P \subseteq S'$ 
    show emeasure (sup_measure.F id  $P$ )  $X \leq \text{emeasure } x X$ 
  proof cases
    assume  $P = \{\}$  then show ?thesis
    by auto
  next
    assume  $P \neq \{\}$ 
    from  $P$  have finite  $P$   $P \subseteq A$ 
    unfolding  $S' S$  by (simp_all add: subset_eq)
    then have sup_measure.F id  $P \leq x$ 
    by (induction  $P$ ) (auto simp:  $x$ )
    moreover have sets (sup_measure.F id  $P$ ) = sets  $x$ 
    using  $\langle \text{finite } P \rangle \langle P \neq \{\} \rangle \langle P \subseteq S' \rangle \langle \text{sets } x = \text{sets } b \rangle$ 
    by (intro sets_sup_measure_F) (auto simp:  $S'$ )
    ultimately show emeasure (sup_measure.F id  $P$ )  $X \leq \text{emeasure } x X$ 
    by (rule le_measureD3)
  qed
  qed
  show  $m \in S' \implies \text{sets } m = \text{sets } (Sup\_measure' S')$  for  $m$ 
  unfolding * by (simp add:  $S'$ )
  qed fact
  qed

```

```

next
  assume sets x ≠ sets b
  moreover have sets b ≤ sets x
  unfolding b S using x[THEN le_measureD2] ⟨space x = space a⟩ by auto
  ultimately show Sup_measure' S' ≤ x
    using ⟨space x = space a⟩ ⟨b ∈ S⟩
    by (intro less_eq_measure.intros(2)) (simp_all add: * S)
qed
next
  assume space x ≠ space a
  then have space a < space x
  using le_measureD1[OF x[OF ⟨a ∈ A⟩]] by auto
  then show Sup_measure' S' ≤ x
    by (intro less_eq_measure.intros) (simp add: * ⟨space b = space a⟩)
qed
qed
show Sup {} = (bot::'a measure) Inf {} = (top::'a measure)
  by (auto intro!: antisym least simp: top_measure_def)
show lower: x ∈ A ⇒ Inf A ≤ x for x :: 'a measure and A
  unfolding Inf_measure_def by (intro least) auto
show greatest: (⋀z. z ∈ A ⇒ x ≤ z) ⇒ x ≤ Inf A for x :: 'a measure and A
  unfolding Inf_measure_def by (intro upper) auto
show inf x y ≤ x inf x y ≤ y x ≤ y ⇒ x ≤ z ⇒ x ≤ inf y z for x y z :: 'a
measure
  by (auto simp: inf_measure_def intro!: lower greatest)
qed

end

lemma sets_SUP:
  assumes ⋀x. x ∈ I ⇒ sets (M x) = sets N
  shows I ≠ {} ⇒ sets (SUP i ∈ I. M i) = sets N
  unfolding Sup_measure_def
  using assms assms[THEN sets_eq_imp_space_eq]
  sets_Sup_measure'[where A=N and M=M'I]
  by (intro Sup_lexord1[where P=λx. sets x = sets N]) auto

lemma emeasure_SUP:
  assumes sets: ⋀i. i ∈ I ⇒ sets (M i) = sets N X ∈ sets N I ≠ {}
  shows emeasure (SUP i ∈ I. M i) X = (SUP J ∈ {J. J ≠ {} ∧ finite J ∧ J ⊆ I}.
emeasure (SUP i ∈ J. M i) X)
proof -
  interpret sup_measure: comm_monoid_set sup bot :: 'b measure
    by standard (auto intro!: antisym)
  have eq: finite J ⇒ sup_measure.F id J = (SUP i ∈ J. i) for J :: 'b measure
set
  by (induction J rule: finite_induct) auto
  have 1: J ≠ {} ⇒ J ⊆ I ⇒ sets (SUP x ∈ J. M x) = sets N for J
    by (intro sets_SUP sets) (auto)

```

```

from  $\langle I \neq \{\} \rangle$  obtain  $i$  where  $i \in I$  by auto
have  $\text{Sup\_measure}'(M'I) X = (\text{SUP } P \in \{P. \text{finite } P \wedge P \subseteq M'I\}. \text{sup\_measure}.F$ 
 $\text{id } P X)$ 
  using sets by (intro emeasure_Sup_measure') auto
also have  $\text{Sup\_measure}'(M'I) = (\text{SUP } i \in I. M i)$ 
unfolding Sup_measure_def using  $\langle I \neq \{\} \rangle$  sets  $\text{sets}(1)[\text{THEN sets\_eq\_imp\_space\_eq}]$ 
by (intro Sup_lexord1 [where  $P = \lambda x. \_ = x$ ]) auto
also have  $(\text{SUP } P \in \{P. \text{finite } P \wedge P \subseteq M'I\}. \text{sup\_measure}.F \text{id } P X) =$ 
 $(\text{SUP } J \in \{J. J \neq \{\} \wedge \text{finite } J \wedge J \subseteq I\}. (\text{SUP } i \in J. M i) X)$ 
proof (intro SUP_eq)
  fix  $J$  assume  $J \in \{P. \text{finite } P \wedge P \subseteq M'I\}$ 
  then obtain  $J'$  where  $J': J' \subseteq I$  finite  $J'$  and  $J: J = M'J'$  and finite  $J$ 
    using finite_subset_image [of  $J M I$ ] by auto
  show  $\exists j \in \{J. J \neq \{\} \wedge \text{finite } J \wedge J \subseteq I\}. \text{sup\_measure}.F \text{id } J X \leq (\text{SUP } i \in j.$ 
 $M i) X$ 
    proof cases
      assume  $J' = \{\}$  with  $\langle i \in I \rangle$  show ?thesis
        by (auto simp add: J)
      next
        assume  $J' \neq \{\}$  with  $J J'$  show ?thesis
          by (intro bexI [of  $\_ J'$ ]) (auto simp add: eq simp del: id_apply)
        qed
      next
        fix  $J$  assume  $J: J \in \{P. P \neq \{\} \wedge \text{finite } P \wedge P \subseteq I\}$ 
        show  $\exists J' \in \{J. \text{finite } J \wedge J \subseteq M'I\}. (\text{SUP } i \in J. M i) X \leq \text{sup\_measure}.F \text{id}$ 
 $J' X$ 
          using  $J$  by (intro bexI [of  $\_ M'J$ ]) (auto simp add: eq simp del: id_apply)
        qed
      finally show ?thesis .
    qed

```

lemma *emeasure_SUP_chain*:

```

assumes sets:  $\bigwedge i. i \in A \implies \text{sets } (M i) = \text{sets } N X \in \text{sets } N$ 
assumes ch: Complete_Partial_Order.chain  $(\leq) (M' A)$  and  $A \neq \{\}$ 
shows  $\text{emeasure } (\text{SUP } i \in A. M i) X = (\text{SUP } i \in A. \text{emeasure } (M i) X)$ 
proof (subst emeasure_SUP [OF sets  $\langle A \neq \{\} \rangle$ ])
  show  $(\text{SUP } J \in \{J. J \neq \{\} \wedge \text{finite } J \wedge J \subseteq A\}. \text{emeasure } (\text{Sup } (M' J)) X) =$ 
 $(\text{SUP } i \in A. \text{emeasure } (M i) X)$ 
    proof (rule SUP_eq)
      fix  $J$  assume  $J \in \{J. J \neq \{\} \wedge \text{finite } J \wedge J \subseteq A\}$ 
      then have  $J: \text{Complete_Partial_Order.chain } (\leq) (M' J)$  finite  $J$   $J \neq \{\}$  and
 $J \subseteq A$ 
        using ch [THEN chain_subset, of  $M'J$ ] by auto
        with in_chain_finite [OF  $J(1)$ ] obtain  $j$  where  $j \in J$   $(\text{SUP } j \in J. M j) = M j$ 
by auto
        with  $\langle J \subseteq A \rangle$  show  $\exists j \in A. \text{emeasure } (\text{Sup } (M' J)) X \leq \text{emeasure } (M j) X$ 
by auto
      next
        fix  $j$  assume  $j \in A$  then show  $\exists i \in \{J. J \neq \{\} \wedge \text{finite } J \wedge J \subseteq A\}. \text{emeasure}$ 

```

```

(M j) X ≤ emeasure (Sup (M ' i)) X
  by (intro bexI[of _ {j}]) auto
qed
qed

```

Supremum of a set of σ -algebras

```

lemma space_Sup_eq_UN: space (Sup M) = ( $\bigcup_{x \in M} \text{space } x$ )
  unfolding Sup_measure_def
  apply (intro Sup_lexord[where P= $\lambda x. \text{space } x = \_$ ])
  apply (subst space_Sup_measure'2)
  apply auto []
  apply (subst space_measure_of[OF UN_space_closed])
  apply auto
  done

```

```

lemma sets_Sup_eq:
  assumes *:  $\bigwedge m. m \in M \implies \text{space } m = X$  and  $M \neq \{\}$ 
  shows sets (Sup M) = sigma_sets X ( $\bigcup_{x \in M} \text{sets } x$ )
  unfolding Sup_measure_def
  apply (rule Sup_lexord1)
  apply fact
  apply (simp add: assms)
  apply (rule Sup_lexord)
  subgoal premises that for a S
    unfolding that(3) that(2)[symmetric]
    using that(1)
    apply (subst sets_Sup_measure'2)
    apply (intro arg_cong2[where f=sigma_sets])
    apply (auto simp: *)
    done
  apply (subst sets_measure_of[OF UN_space_closed])
  apply (simp add: assms)
  done

```

```

lemma in_sets_Sup: ( $\bigwedge m. m \in M \implies \text{space } m = X$ )  $\implies m \in M \implies A \in \text{sets } m \implies A \in \text{sets (Sup M)}$ 
  by (subst sets_Sup_eq[where X=X]) auto

```

```

lemma Sup_lexord_rel:
  assumes  $\bigwedge i. i \in I \implies k (A i) = k (B i)$ 
  R (c (A ' {a ∈ I. k (B a) = (SUP x ∈ I. k (B x))})) (c (B ' {a ∈ I. k (B a) = (SUP x ∈ I. k (B x))}))
  R (s (A'I)) (s (B'I))
  shows R (Sup_lexord k c s (A'I)) (Sup_lexord k c s (B'I))
proof -
  have A ' {a ∈ I. k (B a) = (SUP x ∈ I. k (B x))} = {a ∈ A ' I. k a = (SUP x ∈ I. k (B x))}
    using assms(1) by auto

```

```

    moreover have  $B \cdot \{a \in I. k (B a) = (SUP\ x \in I. k (B x))\} = \{a \in B \cdot I. k a$ 
    =  $(SUP\ x \in I. k (B x))\}$ 
    by auto
    ultimately show ?thesis
    using assms by (auto simp: Sup_lexord_def Let_def image_comp)
qed

```

```

lemma sets_SUP_cong:
  assumes eq:  $\bigwedge i. i \in I \implies sets (M i) = sets (N i)$  shows  $sets (SUP\ i \in I. M i)$ 
  =  $sets (SUP\ i \in I. N i)$ 
  unfolding Sup_measure_def
  using eq eq[THEN sets_eq_imp_space_eq]
  apply (intro Sup_lexord_rel[where R= $\lambda x y. sets\ x = sets\ y$ ])
  apply simp
  apply simp
  apply (simp add: sets_Sup_measure'2)
  apply (intro arg_cong2[where f= $\lambda x y. sets\ (sigma\ x\ y)$ ])
  apply auto
  done

```

```

lemma sets_Sup_in_sets:
  assumes  $M \neq \{\}$ 
  assumes  $\bigwedge m. m \in M \implies space\ m = space\ N$ 
  assumes  $\bigwedge m. m \in M \implies sets\ m \subseteq sets\ N$ 
  shows  $sets (Sup\ M) \subseteq sets\ N$ 
proof -
  have *:  $\bigcup (space \cdot M) = space\ N$ 
  using assms by auto
  show ?thesis
  unfolding * using assms by (subst sets_Sup_eq[of M space N]) (auto intro!:
sets.sigma_sets_subset)
qed

```

```

lemma measurable_Sup1:
  assumes  $m: m \in M$  and  $f: f \in measurable\ m\ N$ 
  and const_space:  $\bigwedge n. m \in M \implies n \in M \implies space\ m = space\ n$ 
  shows  $f \in measurable (Sup\ M)\ N$ 
proof -
  have  $space (Sup\ M) = space\ m$ 
  using m by (auto simp add: space_Sup_eq_UN dest: const_space)
  then show ?thesis
  using m f unfolding measurable_def by (auto intro: in_sets_Sup[OF const_space])
qed

```

```

lemma measurable_Sup2:
  assumes  $M: M \neq \{\}$ 
  assumes  $f: \bigwedge m. m \in M \implies f \in measurable\ m\ N$ 
  and const_space:  $\bigwedge m n. m \in M \implies n \in M \implies space\ m = space\ n$ 
  shows  $f \in measurable\ N (Sup\ M)$ 

```

```

proof -
  from M obtain m where m ∈ M by auto
  have space_eq:  $\bigwedge n. n \in M \implies \text{space } n = \text{space } m$ 
    by (intro const_space ‹m ∈ M›)
  have f ∈ measurable N (sigma ( $\bigcup_{m \in M. \text{space } m}$ ) ( $\bigcup_{m \in M. \text{sets } m}$ ))
  proof (rule measurable_measure_of)
    show f ∈ space N  $\rightarrow \bigcup (\text{space } ' M)$ 
      using measurable_space[OF f] M by auto
    qed (auto intro: measurable_sets f dest: sets_sets_into_space)
  also have measurable N (sigma ( $\bigcup_{m \in M. \text{space } m}$ ) ( $\bigcup_{m \in M. \text{sets } m}$ )) = mea-
surable N (Sup M)
    apply (intro measurable_cong_sets refl)
    apply (subst sets_Sup_eq[OF space_eq M])
    apply simp
    apply (subst sets_measure_of[OF UN_space_closed])
    apply (simp add: space_eq M)
    done
  finally show ?thesis .
qed

```

lemma measurable_SUP2:

```

I ≠ {}  $\implies (\bigwedge i. i \in I \implies f \in \text{measurable } N (M i)) \implies$ 
 $(\bigwedge i j. i \in I \implies j \in I \implies \text{space } (M i) = \text{space } (M j)) \implies f \in \text{measurable } N$ 
(SUP i ∈ I. M i)
  by (auto intro!: measurable_Sup2)

```

lemma sets_Sup_sigma:

```

assumes [simp]: M ≠ {} and M:  $\bigwedge m. m \in M \implies m \subseteq \text{Pow } \Omega$ 
shows sets (SUP m ∈ M. sigma Ω m) = sets (sigma Ω ( $\bigcup M$ ))
proof -
  { fix a m assume a ∈ sigma_sets Ω m m ∈ M
    then have a ∈ sigma_sets Ω ( $\bigcup M$ )
      by induction (auto intro: sigma_sets.intros(2-)) }
  then show sets (SUP m ∈ M. sigma Ω m) = sets (sigma Ω ( $\bigcup M$ ))
    apply (subst sets_Sup_eq[where X=Ω])
    apply (auto simp add: M) []
    apply auto []
    apply (simp add: space_measure_of_conv M Union_least)
    apply (rule sigma_sets_eqI)
    apply auto
    done
qed

```

lemma Sup_sigma:

```

assumes [simp]: M ≠ {} and M:  $\bigwedge m. m \in M \implies m \subseteq \text{Pow } \Omega$ 
shows (SUP m ∈ M. sigma Ω m) = (sigma Ω ( $\bigcup M$ ))
proof (intro antisym SUP_least)
  have *:  $\bigcup M \subseteq \text{Pow } \Omega$ 
    using M by auto

```

```

show  $\sigma \Omega (\bigcup M) \leq (\text{SUP } m \in M. \sigma \Omega m)$ 
proof (intro less_eq_measure.intros(3))
  show  $\text{space } (\sigma \Omega (\bigcup M)) = \text{space } (\text{SUP } m \in M. \sigma \Omega m)$ 
  sets  $(\sigma \Omega (\bigcup M)) = \text{sets } (\text{SUP } m \in M. \sigma \Omega m)$ 
  using sets_Sup_sigma[OF assms] sets_Sup_sigma[OF assms, THEN sets_eq_imp_space_eq]
  by auto
qed (simp add: emeasure_sigma le_fun_def)
fix m assume  $m \in M$  then show  $\sigma \Omega m \leq \sigma \Omega (\bigcup M)$ 
  by (subst sigma_le_iff) (auto simp add: M *)
qed

```

```

lemma SUP_sigma_sigma:
   $M \neq \{\} \implies (\bigwedge m. m \in M \implies f m \subseteq \text{Pow } \Omega) \implies (\text{SUP } m \in M. \sigma \Omega (f m))$ 
 $= \sigma \Omega (\bigcup m \in M. f m)$ 
  using Sup_sigma[of f' M  $\Omega$ ] by (auto simp: image_comp)

```

```

lemma sets_vimage_Sup_eq:
  assumes *:  $M \neq \{\} \implies f \in X \rightarrow Y \bigwedge m. m \in M \implies \text{space } m = Y$ 
  shows  $\text{sets } (\text{vimage\_algebra } X f (\text{Sup } M)) = \text{sets } (\text{SUP } m \in M. \text{vimage\_algebra } X f m)$ 
  (is ?IS = ?SI)

```

```

proof
  show ?IS  $\subseteq$  ?SI
    apply (intro sets_image_in_sets measurable_Sup2)
    apply (simp add: space_Sup_eq_UN *)
    apply (simp add: *)
    apply (intro measurable_Sup1)
    apply (rule imageI)
    apply assumption
    apply (rule measurable_vimage_algebra1)
    apply (auto simp: *)
    done
  show ?SI  $\subseteq$  ?IS
    apply (intro sets_Sup_in_sets)
    apply (auto simp: *) []
    apply (auto simp: *) []
    apply (elim imageE)
    apply simp
    apply (rule sets_image_in_sets)
    apply simp
    apply (simp add: measurable_def)
    apply (simp add: * space_Sup_eq_UN sets_vimage_algebra2)
    apply (auto intro: in_sets_Sup[OF *(3)])
    done

```

qed

```

lemma restrict_space_eq_vimage_algebra':
   $\text{sets } (\text{restrict\_space } M \Omega) = \text{sets } (\text{vimage\_algebra } (\Omega \cap \text{space } M) (\lambda x. x) M)$ 
proof -

```



```

have *:  $\{A \cap (\Omega \cap \text{space } M) \mid A. A \in \text{sets } M\} = \{A \cap \Omega \mid A. A \in \text{sets } M\}$ 
  using sets.sets_into_space[of _ M] by blast

show ?thesis
  unfolding restrict_space_def
  by (subst sets_measure_of)
    (auto simp add: image_subset_iff sets_vimage_algebra * dest: sets.sets_into_space
intro!: arg_cong2[where f=sigma_sets])
qed

```

```

lemma sigma_le_sets:
  assumes [simp]:  $A \subseteq \text{Pow } X$  shows  $\text{sets } (\text{sigma } X A) \subseteq \text{sets } N \longleftrightarrow X \in \text{sets } N \wedge A \subseteq \text{sets } N$ 
proof
  have  $X \in \text{sigma\_sets } X A \implies A \subseteq \text{sigma\_sets } X A$ 
    by (auto intro: sigma_sets_top)
  moreover assume  $\text{sets } (\text{sigma } X A) \subseteq \text{sets } N$ 
  ultimately show  $X \in \text{sets } N \wedge A \subseteq \text{sets } N$ 
    by auto
next
  assume *:  $X \in \text{sets } N \wedge A \subseteq \text{sets } N$ 
  { fix Y assume  $Y \in \text{sigma\_sets } X A$  from this * have  $Y \in \text{sets } N$ 
    by induction auto }
  then show  $\text{sets } (\text{sigma } X A) \subseteq \text{sets } N$ 
    by auto
qed

```

```

lemma measurable_iff_sets:
   $f \in \text{measurable } M N \longleftrightarrow (f \in \text{space } M \rightarrow \text{space } N \wedge \text{sets } (\text{vimage\_algebra } (\text{space } M) f N) \subseteq \text{sets } M)$ 
proof -
  have *:  $\{f \mid A \cap \text{space } M \mid A. A \in \text{sets } N\} \subseteq \text{Pow } (\text{space } M)$ 
    by auto
  show ?thesis
    unfolding measurable_def
    by (auto simp add: vimage_algebra_def sigma_le_sets[OF *])
qed

```

```

lemma sets_vimage_algebra_space:  $X \in \text{sets } (\text{vimage\_algebra } X f M)$ 
  using sets.top[of vimage_algebra X f M] by simp

```

```

lemma measurable_mono:
  assumes  $N: \text{sets } N' \leq \text{sets } N \text{ space } N = \text{space } N'$ 
  assumes  $M: \text{sets } M \leq \text{sets } M' \text{ space } M = \text{space } M'$ 
  shows  $\text{measurable } M N \subseteq \text{measurable } M' N'$ 
  unfolding measurable_def
proof safe
  fix f A assume  $f \in \text{space } M \rightarrow \text{space } N \wedge A \in \text{sets } N'$ 
  moreover assume  $\forall y \in \text{sets } N. f^{-1} y \cap \text{space } M \in \text{sets } M$  note this[THEN

```

1382

bspec, of A]

ultimately show $f - 'A \cap \text{space } M' \in \text{sets } M'$
using *assms by auto*
qed (*insert N M, auto*)

lemma *measurable_Sup_measurable:*

assumes $f: f \in \text{space } N \rightarrow A$
shows $f \in \text{measurable } N \text{ (Sup } \{M. \text{space } M = A \wedge f \in \text{measurable } N M\})$
proof (*rule measurable_Sup2*)
show $\{M. \text{space } M = A \wedge f \in \text{measurable } N M\} \neq \{\}$
using f **unfolding** *ex_in_conv[symmetric]*
by (*intro exI[of _ sigma A {}]*) (*auto intro!: measurable_measure_of*)
qed *auto*

lemma (*in sigma_algebra*) *sigma_sets_subset':*

assumes $a: a \subseteq M \ \Omega' \in M$
shows $\text{sigma\_sets } \Omega' a \subseteq M$
proof
show $x \in M$ **if** $x: x \in \text{sigma\_sets } \Omega' a$ **for** x
using x **by** (*induct rule: sigma_sets.induct*) (*insert a, auto*)
qed

lemma *in_sets_SUP:* $i \in I \implies (\bigwedge i. i \in I \implies \text{space } (M i) = Y) \implies X \in \text{sets } (M i) \implies X \in \text{sets } (\text{SUP } i \in I. M i)$
by (*intro in_sets_Sup[where X=Y]*) *auto*

lemma *measurable_SUP1:*

$i \in I \implies f \in \text{measurable } (M i) N \implies (\bigwedge m n. m \in I \implies n \in I \implies \text{space } (M m) = \text{space } (M n)) \implies$
 $f \in \text{measurable } (\text{SUP } i \in I. M i) N$
by (*auto intro: measurable_Sup1*)

lemma *sets_image_in_sets':*

assumes $X: X \in \text{sets } N$
assumes $f: \bigwedge A. A \in \text{sets } M \implies f - 'A \cap X \in \text{sets } N$
shows $\text{sets } (\text{vimage\_algebra } X f M) \subseteq \text{sets } N$
unfolding *sets_vimage_algebra*
by (*rule sets.sigma_sets_subset'*) (*auto intro!: measurable_sets X f*)

lemma *mono_vimage_algebra:*

$\text{sets } M \leq \text{sets } N \implies \text{sets } (\text{vimage\_algebra } X f M) \subseteq \text{sets } (\text{vimage\_algebra } X f N)$
using *sets.top[of sigma X {f - 'A \cap X | A. A \in sets N}]*
unfolding *vimage_algebra_def*
apply (*subst (asm) space_measure_of*)
apply *auto []*
apply (*subst sigma_le_sets*)
apply *auto*
done

lemma *mono_restrict_space*: $\text{sets } M \leq \text{sets } N \implies \text{sets } (\text{restrict\_space } M \ X) \subseteq \text{sets } (\text{restrict\_space } N \ X)$

unfolding *sets_restrict_space* **by** (rule *image_mono*)

lemma *sets_eq_bot*: $\text{sets } M = \{\{\}\} \longleftrightarrow M = \text{bot}$

apply *safe*

apply (intro *measure_eqI*)

apply *auto*

done

lemma *sets_eq_bot2*: $\{\{\}\} = \text{sets } M \longleftrightarrow M = \text{bot}$

using *sets_eq_bot*[of *M*] **by** *blast*

lemma (in *finite_measure*) *countable_support*:

countable $\{x. \text{measure } M \ \{x\} \neq 0\}$

proof *cases*

assume *measure* *M* (space *M*) = 0

with *bounded_measure* *measure_le_0_iff* **have** $\{x. \text{measure } M \ \{x\} \neq 0\} = \{\}$

by *auto*

then show *?thesis*

by *simp*

next

let *?M* = *measure* *M* (space *M*) **and** *?m* = $\lambda x. \text{measure } M \ \{x\}$

assume *?M* $\neq 0$

then have $\ast: \{x. ?m \ x \neq 0\} = (\bigcup n. \{x. ?M / \text{Suc } n < ?m \ x\})$

using *reals_Archimedean*[of *?m* *x* / *?M* **for** *x*]

by (auto *simp*: *field_simps* *not_le[symmetric]* *divide_le_0_iff* *measure_le_0_iff*)

have $\ast\ast: \bigwedge n. \text{finite } \{x. ?M / \text{Suc } n < ?m \ x\}$

proof (rule *ccontr*)

fix *n* **assume** *infinite* $\{x. ?M / \text{Suc } n < ?m \ x\}$ (is *infinite* *?X*)

then obtain *X* **where** *finite* *X* *card* *X* = *Suc* (*Suc* *n*) $X \subseteq ?X$

by (*metis* *infinite_arbitrarily_large*)

from *this*(*?*) **have** $\ast: \bigwedge x. x \in X \implies ?M / \text{Suc } n \leq ?m \ x$

by *auto*

{ fix *x* **assume** $x \in X$

from $\langle ?M \neq 0 \rangle \ast$ [*OF this*] **have** *?m* *x* $\neq 0$ **by** (auto *simp*: *field_simps* *measure_le_0_iff*)

then have $\{x\} \in \text{sets } M$ **by** (auto *dest*: *measure_notin_sets*) }

note *singleton_sets* = *this*

have $?M < (\sum x \in X. ?M / \text{Suc } n)$

using $\langle ?M \neq 0 \rangle$

by (*simp* *add*: $\langle \text{card } X = \text{Suc } (\text{Suc } n) \rangle$ *field_simps* *less_le*)

also have $\dots \leq (\sum x \in X. ?m \ x)$

by (rule *sum_mono*) *fact*

also have $\dots = \text{measure } M \ (\bigcup x \in X. \{x\})$

using *singleton_sets* $\langle \text{finite } X \rangle$

by (intro *finite_measure_finite_Union[symmetric]*) (auto *simp*: *disjoint_family_on_def*)

```

    finally have ?M < measure M ( $\bigcup_{x \in X}. \{x\}$ ) .
    moreover have measure M ( $\bigcup_{x \in X}. \{x\}$ )  $\leq$  ?M
    using singleton_sets[THEN sets.sets_into_space] by (intro finite_measure_mono)
  auto
    ultimately show False by simp
  qed
  show ?thesis
    unfolding * by (intro countable_UN countableI_type countable_finite[OF **])
  qed
end

```

6.4 Ordered Euclidean Space

```

theory Ordered_Euclidean_Space
imports
  Convex_Euclidean_Space
  HOL-Library.Product_Order
begin

```

An ordering on euclidean spaces that will allow us to talk about intervals

```

class ordered_euclidean_space = ord + inf + sup + abs + Inf + Sup + euclidean_space +
  assumes eucl_le:  $x \leq y \iff (\forall i \in \text{Basis}. x \cdot i \leq y \cdot i)$ 
  assumes eucl_less_le_not_le:  $x < y \iff x \leq y \wedge \neg y \leq x$ 
  assumes eucl_inf:  $\inf x y = (\sum i \in \text{Basis}. \inf (x \cdot i) (y \cdot i) *_R i)$ 
  assumes eucl_sup:  $\sup x y = (\sum i \in \text{Basis}. \sup (x \cdot i) (y \cdot i) *_R i)$ 
  assumes eucl_Inf:  $\text{Inf } X = (\sum i \in \text{Basis}. (\text{INF } x \in X. x \cdot i) *_R i)$ 
  assumes eucl_Sup:  $\text{Sup } X = (\sum i \in \text{Basis}. (\text{SUP } x \in X. x \cdot i) *_R i)$ 
  assumes eucl_abs:  $|x| = (\sum i \in \text{Basis}. |x \cdot i| *_R i)$ 
begin

subclass order
  by standard
  (auto simp: eucl_le eucl_less_le_not_le intro!: euclidean_eqI antisym intro: order.trans)

subclass ordered_ab_group_add_abs
  by standard (auto simp: eucl_le inner_add_left eucl_abs abs_leI)

subclass ordered_real_vector
  by standard (auto simp: eucl_le intro!: mult_left_mono mult_right_mono)

subclass lattice
  by standard (auto simp: eucl_inf eucl_sup eucl_le)

subclass distrib_lattice
  by standard (auto simp: eucl_inf eucl_sup sup_inf_distrib1 intro!: euclidean_eqI)

```

subclass *conditionally_complete_lattice*

proof

fix $z::'a$ **and** $X::'a \text{ set}$

assume $X \neq \{\}$

hence $\bigwedge i. (\lambda x. x \cdot i) \text{ ' } X \neq \{\}$ **by** *simp*

thus $(\bigwedge x. x \in X \implies z \leq x) \implies z \leq \text{Inf } X (\bigwedge x. x \in X \implies x \leq z) \implies \text{Sup } X \leq z$

by (*auto simp: eucl_Inf eucl_Sup eucl_le intro!: cInf_greatest cSup_least*)

qed (*force intro!: cInf_lower cSup_upper*

simp: bdd_below_def bdd_above_def preorder_class.bdd_below_def preorder_class.bdd_above_def eucl_Inf eucl_Sup eucl_le)

lemma *inner_Basis_inf_left*: $i \in \text{Basis} \implies \inf x \ y \cdot i = \inf (x \cdot i) (y \cdot i)$

and *inner_Basis_sup_left*: $i \in \text{Basis} \implies \sup x \ y \cdot i = \sup (x \cdot i) (y \cdot i)$

by (*simp_all add: eucl_inf eucl_sup inner_sum_left inner_Basis if_distrib cong: if_cong*)

lemma *inner_Basis_INF_left*: $i \in \text{Basis} \implies (\text{INF } x \in X. f \ x) \cdot i = (\text{INF } x \in X. f \ x \cdot i)$

and *inner_Basis_SUP_left*: $i \in \text{Basis} \implies (\text{SUP } x \in X. f \ x) \cdot i = (\text{SUP } x \in X. f \ x \cdot i)$

using *eucl_Sup [of f ' X] eucl_Inf [of f ' X]* **by** (*simp_all add: image_comp*)

lemma *abs_inner*: $i \in \text{Basis} \implies |x| \cdot i = |x \cdot i|$

by (*auto simp: eucl_abs*)

lemma

abs_scaleR: $|a \ *_R \ b| = |a| \ *_R \ |b|$

by (*auto simp: eucl_abs abs_mult intro!: euclidean_eqI*)

lemma *interval_inner_leI*:

assumes $x \in \{a \ .. \ b\}$ $0 \leq i$

shows $a \cdot i \leq x \cdot i \leq b \cdot i$

using *assms*

unfolding *euclidean_inner [of a i] euclidean_inner [of x i] euclidean_inner [of b i]*

by (*auto intro!: ordered_comm_monoid_add_class.sum_mono mult_right_mono simp: eucl_le*)

lemma *inner_nonneg_nonneg*:

shows $0 \leq a \implies 0 \leq b \implies 0 \leq a \cdot b$

using *interval_inner_leI [of a 0 a b]*

by *auto*

lemma *inner_Basis_mono*:

shows $a \leq b \implies c \in \text{Basis} \implies a \cdot c \leq b \cdot c$

by (*simp add: eucl_le*)

lemma *Basis_nonneg*[*intro*, *simp*]: $i \in \text{Basis} \implies 0 \leq i$
by (*auto simp: eucl_le inner_Basis*)

lemma *Sup_eq_maximum_componentwise*:
fixes $s::'a \text{ set}$
assumes $i: \bigwedge b. b \in \text{Basis} \implies X \cdot b = i \cdot b$
assumes $\text{sup}: \bigwedge b \ x. b \in \text{Basis} \implies x \in s \implies x \cdot b \leq X \cdot b$
assumes $i\_s: \bigwedge b. b \in \text{Basis} \implies (i \cdot b) \in (\lambda x. x \cdot b) \text{ ' } s$
shows $\text{Sup } s = X$
using *assms*
unfolding *eucl_Sup euclidean_representation_sum*
by (*auto intro!: conditionally_complete_lattice_class.cSup_eq_maximum*)

lemma *Inf_eq_minimum_componentwise*:
assumes $i: \bigwedge b. b \in \text{Basis} \implies X \cdot b = i \cdot b$
assumes $\text{sup}: \bigwedge b \ x. b \in \text{Basis} \implies x \in s \implies X \cdot b \leq x \cdot b$
assumes $i\_s: \bigwedge b. b \in \text{Basis} \implies (i \cdot b) \in (\lambda x. x \cdot b) \text{ ' } s$
shows $\text{Inf } s = X$
using *assms*
unfolding *eucl_Inf euclidean_representation_sum*
by (*auto intro!: conditionally_complete_lattice_class.cInf_eq_minimum*)

end

proposition *compact_attains_Inf_componentwise*:
fixes $b::'a::\text{ordered\_euclidean\_space}$
assumes $b \in \text{Basis}$ **assumes** $X \neq \{\}$ *compact* X
obtains x **where** $x \in X \ x \cdot b = \text{Inf } X \cdot b \ \bigwedge y. y \in X \implies x \cdot b \leq y \cdot b$
proof *atomize_elim*
let $?proj = (\lambda x. x \cdot b) \text{ ' } X$
from *assms* **have** *compact ?proj ?proj $\neq \{\}$*
by (*auto intro!: compact_continuous_image continuous_intros*)
from *compact_attains_inf[OF this]*
obtain $s \ x$
where $s: s \in (\lambda x. x \cdot b) \text{ ' } X \ \bigwedge t. t \in (\lambda x. x \cdot b) \text{ ' } X \implies s \leq t$
and $x: x \in X \ s = x \cdot b \ \bigwedge y. y \in X \implies x \cdot b \leq y \cdot b$
by *auto*
hence $\text{Inf } ?proj = x \cdot b$
by (*auto intro!: conditionally_complete_lattice_class.cInf_eq_minimum*)
hence $x \cdot b = \text{Inf } X \cdot b$
by (*auto simp: eucl_Inf inner_sum_left inner_Basis if_distrib <b $\in$ Basis>*
cong: if_cong)
with x **show** $\exists x. x \in X \ \wedge \ x \cdot b = \text{Inf } X \cdot b \ \wedge \ (\forall y. y \in X \longrightarrow x \cdot b \leq y \cdot b)$
by *blast*
qed

proposition
compact_attains_Sup_componentwise:
fixes $b::'a::\text{ordered\_euclidean\_space}$

```

assumes  $b \in \text{Basis}$  assumes  $X \neq \{\}$  compact  $X$ 
obtains  $x$  where  $x \in X$   $x \cdot b = \text{Sup } X \cdot b \wedge y. y \in X \implies y \cdot b \leq x \cdot b$ 
proof atomize_elim
  let  $?proj = (\lambda x. x \cdot b) \cdot X$ 
  from assms have compact  $?proj$   $?proj \neq \{\}$ 
  by (auto intro!: compact_continuous_image continuous_intros)
  from compact_attains_sup[OF this]
  obtain  $s$   $x$ 
    where  $s: s \in (\lambda x. x \cdot b) \cdot X \wedge t. t \in (\lambda x. x \cdot b) \cdot X \implies t \leq s$ 
    and  $x: x \in X$   $s = x \cdot b \wedge y. y \in X \implies y \cdot b \leq x \cdot b$ 
  by auto
  hence  $\text{Sup } ?proj = x \cdot b$ 
  by (auto intro!: cSup_eq_maximum)
  hence  $x \cdot b = \text{Sup } X \cdot b$ 
  by (auto simp: eucl_Sup[where  $'a = 'a$ ] inner_sum_left inner_Basis if_distrib
     $\langle b \in \text{Basis} \rangle$ 
    cong: if_cong)
  with  $x$  show  $\exists x. x \in X \wedge x \cdot b = \text{Sup } X \cdot b \wedge (\forall y. y \in X \longrightarrow y \cdot b \leq x \cdot b)$ 
by blast
qed

```

```

lemma tendsto_sup[tendsto_intros]:
  fixes  $X :: 'a \Rightarrow 'b :: \text{ordered\_euclidean\_space}$ 
  assumes  $(X \longrightarrow x)$  net  $(Y \longrightarrow y)$  net
  shows  $((\lambda i. \text{sup } (X \ i) (Y \ i)) \longrightarrow \text{sup } x \ y)$  net
  unfolding sup_max eucl_sup by (intro assms tendsto_intros)

```

```

lemma tendsto_inf[tendsto_intros]:
  fixes  $X :: 'a \Rightarrow 'b :: \text{ordered\_euclidean\_space}$ 
  assumes  $(X \longrightarrow x)$  net  $(Y \longrightarrow y)$  net
  shows  $((\lambda i. \text{inf } (X \ i) (Y \ i)) \longrightarrow \text{inf } x \ y)$  net
  unfolding inf_min eucl_inf by (intro assms tendsto_intros)

```

```

lemma tendsto_componentwise_max:
  assumes  $f: (f \longrightarrow l)$   $F$  and  $g: (g \longrightarrow m)$   $F$ 
  shows  $((\lambda x. (\sum_{i \in \text{Basis}} \text{max } (f \ x \cdot i) (g \ x \cdot i) *_R i)) \longrightarrow (\sum_{i \in \text{Basis}} \text{max } (l \cdot i) (m \cdot i) *_R i))$   $F$ 
  by (intro tendsto_intros assms)

```

```

lemma tendsto_componentwise_min:
  assumes  $f: (f \longrightarrow l)$   $F$  and  $g: (g \longrightarrow m)$   $F$ 
  shows  $((\lambda x. (\sum_{i \in \text{Basis}} \text{min } (f \ x \cdot i) (g \ x \cdot i) *_R i)) \longrightarrow (\sum_{i \in \text{Basis}} \text{min } (l \cdot i) (m \cdot i) *_R i))$   $F$ 
  by (intro tendsto_intros assms)

```

```

lemma (in order) atLeastatMost_empty'[simp]:
   $(\neg a \leq b) \implies \{a..b\} = \{\}$ 
  by (auto)

```

1388

instance *real* :: *ordered_euclidean_space*
 by *standard auto*

lemma *in_Basis_prod_iff*:
 fixes *i*::'*a*::*euclidean_space**'*b*::*euclidean_space*
 shows $i \in \text{Basis} \longleftrightarrow \text{fst } i = 0 \wedge \text{snd } i \in \text{Basis} \vee \text{snd } i = 0 \wedge \text{fst } i \in \text{Basis}$
 by (cases *i*) (auto simp: *Basis_prod_def*)

instantiation *prod* :: (*abs*, *abs*) *abs*
 begin

definition $|x| = (|\text{fst } x|, |\text{snd } x|)$

instance ..

end

instance *prod* :: (*ordered_euclidean_space*, *ordered_euclidean_space*) *ordered_euclidean_space*
 by *standard*
 (auto intro!: *add_mono simp add: euclidean_representation_sum' Ball_def*
inner_prod_def
in_Basis_prod_iff inner_Basis_inf_left inner_Basis_sup_left inner_Basis_INF_left
Inf_prod_def
inner_Basis_SUP_left Sup_prod_def less_prod_def less_eq_prod_def eucl_le[**where**
'*a*='*a*]
eucl_le[**where** '*a*='*b*] *abs_prod_def abs_inner*)

Instantiation for intervals on *ordered_euclidean_space*

proposition

fixes *a* :: '*a*::*ordered_euclidean_space*
 shows *cbox_interval*: $\text{cbox } a \ b = \{a..b\}$
 and *interval_cbox*: $\{a..b\} = \text{cbox } a \ b$
 and *eucl_le_atMost*: $\{x. \forall i \in \text{Basis}. x \cdot i \leq a \cdot i\} = \{..a\}$
 and *eucl_le_atLeast*: $\{x. \forall i \in \text{Basis}. a \cdot i \leq x \cdot i\} = \{a..\}$
 by (auto simp: *eucl_le*[**where** '*a*='*a*] *eucl_less_def box_def cbox_def*)

lemma *sums_vec_nth* :

assumes *f sums a*
 shows $(\lambda x. f \ x \ \$ \ i) \ \text{sums } a \ \$ \ i$
 using *assms unfolding sums_def*
 by (auto dest: *tendsto_vec_nth* [**where** *i=i*])

lemma *summable_vec_nth* :

assumes *summable f*
 shows *summable* $(\lambda x. f \ x \ \$ \ i)$
 using *assms unfolding summable_def*
 by (blast intro: *sums_vec_nth*)

lemma *closed_eucl_atLeastAtMost*[*simp*, *intro*]:


```

fixes  $a :: 'a::ordered\_euclidean\_space$ 
shows  $closed \{a..b\}$ 
by (simp add: cbox_interval[symmetric] closed_cbox)

lemma  $closed\_eucl\_atMost[simp, intro]$ :
  fixes  $a :: 'a::ordered\_euclidean\_space$ 
shows  $closed \{..a\}$ 
by (simp add: closed_interval_left eucl_le_atMost[symmetric])

lemma  $closed\_eucl\_atLeast[simp, intro]$ :
  fixes  $a :: 'a::ordered\_euclidean\_space$ 
shows  $closed \{a.. \}$ 
by (simp add: closed_interval_right eucl_le_atLeast[symmetric])

lemma  $bounded\_closed\_interval \ [simp]$ :
  fixes  $a :: 'a::ordered\_euclidean\_space$ 
shows  $bounded \{a .. b\}$ 
using  $bounded\_cbox[of \ a \ b]$ 
by (metis interval_cbox)

lemma  $convex\_closed\_interval \ [simp]$ :
  fixes  $a :: 'a::ordered\_euclidean\_space$ 
shows  $convex \{a .. b\}$ 
using  $convex\_box[of \ a \ b]$ 
by (metis interval_cbox)

lemma  $image\_smult\_interval:(\lambda x. m *_R (x::ordered\_euclidean\_space)) \ ' \{a .. b\} =$ 
   $(if \{a .. b\} = \{\} \ then \{\} \ else \ if \ 0 \leq m \ then \{m *_R a .. m *_R b\} \ else \{m *_R b .. m *_R a\})$ 
using  $image\_smult\_cbox[of \ m \ a \ b]$ 
by (simp add: cbox_interval)

lemma [simp]:
  fixes  $a \ b::'a::ordered\_euclidean\_space$ 
shows  $is\_interval\_ic: is\_interval \{..a\}$ 
  and  $is\_interval\_ci: is\_interval \{a.. \}$ 
  and  $is\_interval\_cc: is\_interval \{b..a\}$ 
by (force simp: is_interval_def eucl_le[where 'a='a])+)

lemma  $connected\_interval \ [simp]$ :
  fixes  $a \ b::'a::ordered\_euclidean\_space$ 
shows  $connected \{a..b\}$ 
using  $is\_interval\_cc \ is\_interval\_connected$  by blast

lemma  $compact\_interval \ [simp]$ :
  fixes  $a \ b::'a::ordered\_euclidean\_space$ 
shows  $compact \{a .. b\}$ 
by (metis compact_cbox interval_cbox)

```

1390

no_notation

eucl_less (**infix** $<_e$ 50)

lemma *One_nonneg*: $0 \leq (\sum \text{Basis}::'a::\text{ordered\_euclidean\_space})$

by (*auto intro: sum_nonneg*)

lemma

fixes *a b*::'a::ordered_euclidean_space

shows *bdd_above_cbox*[*intro, simp*]: *bdd_above* (*cbox a b*)

and *bdd_below_cbox*[*intro, simp*]: *bdd_below* (*cbox a b*)

and *bdd_above_box*[*intro, simp*]: *bdd_above* (*box a b*)

and *bdd_below_box*[*intro, simp*]: *bdd_below* (*box a b*)

unfolding *atomize_conj*

by (*metis bdd_above_Icc bdd_above_mono bdd_below_Icc bdd_below_mono*

bounded_box

bounded_subset_cbox_symmetric interval_cbox)

instantiation *vec* :: (*ordered_euclidean_space, finite*) *ordered_euclidean_space*

begin

definition *inf x y* = (χ *i. inf* (*x* \$ *i*) (*y* \$ *i*))

definition *sup x y* = (χ *i. sup* (*x* \$ *i*) (*y* \$ *i*))

definition *Inf X* = (χ *i. (INF* *x* ∈ *X. x* \$ *i*))

definition *Sup X* = (χ *i. (SUP* *x* ∈ *X. x* \$ *i*))

definition $|x|$ = (χ *i. |x* \$ *i*)

instance

apply *standard*

unfolding *euclidean_representation_sum'*

apply (*auto simp: less_eq_vec_def inf_vec_def sup_vec_def Inf_vec_def Sup_vec_def*
inner_axis

Basis_vec_def inner_Basis_inf_left inner_Basis_sup_left inner_Basis_INF_left

inner_Basis_SUP_left eucl_le [**where** '*a* = '*a*'] *less_le_not_le abs_vec_def abs_inner*)

done

end

end

6.5 Borel Space

theory *Borel_Space*

imports

Measurable Derivative Ordered_Euclidean_Space Extended_Real_Limits

begin

lemma *is_interval_real_ereal_oo: is_interval* (*real_of_ereal* ' $\{N <..<M::ereal\}$)

by (*auto simp: real_atLeastGreaterThan_eq*)

lemma *sets_Collect_eventually_sequentially*[measurable]:

$(\bigwedge i. \{x \in \text{space } M. P\ x\ i\} \in \text{sets } M) \implies \{x \in \text{space } M. \text{eventually } (P\ x)\ \text{sequentially}\} \in \text{sets } M$

unfolding *eventually_sequentially* **by** *simp*

lemma *topological_basis_trivial*: *topological_basis* $\{A. \text{open } A\}$

by (*auto simp: topological_basis_def*)

proposition *open_prod_generated*: *open* = *generate_topology* $\{A \times B \mid A\ B. \text{open } A \wedge \text{open } B\}$

proof –

have $\{A \times B :: ('a \times 'b)\ \text{set} \mid A\ B. \text{open } A \wedge \text{open } B\} = ((\lambda(a, b). a \times b)\ ` (\{A. \text{open } A\} \times \{A. \text{open } A\}))$

by *auto*

then show *?thesis*

by (*auto intro: topological_basis_prod topological_basis_trivial topological_basis_imp_subbasis*)

qed

proposition *mono_on_imp_deriv_nonneg*:

assumes *mono*: *mono_on* *f* *A* **and** *deriv*: (*f* *has_real_derivative* *D*) (at *x*)

assumes *x* $\in$ *interior* *A*

shows $D \geq 0$

proof (*rule tendsto_lowerbound*)

let $?A' = (\lambda y. y - x)\ ` \text{interior } A$

from *deriv* **show** $((\lambda h. (f\ (x + h) - f\ x) / h) \longrightarrow D)\ (at\ 0)$

by (*simp add: field_has_derivative_at has_field_derivative_def*)

from *mono* **have** *mono'*: *mono_on* *f* (*interior* *A*) **by** (*rule mono_on_subset*)
(*rule interior_subset*)

show *eventually* $(\lambda h. (f\ (x + h) - f\ x) / h \geq 0)\ (at\ 0)$

proof (*subst eventually_at_topological, intro exI conjI ballI impI*)

have *open* (*interior* *A*) **by** *simp*

hence *open* $((+) (-x)\ ` \text{interior } A)$ **by** (*rule open_translation*)

also have $((+) (-x)\ ` \text{interior } A) = ?A'$ **by** *auto*

finally show *open* $?A'$.

next

from $\langle x \in \text{interior } A \rangle$ **show** $0 \in ?A'$ **by** *auto*

next

fix *h* **assume** *h* $\in ?A'$

hence $x + h \in \text{interior } A$ **by** *auto*

with *mono'* **and** $\langle x \in \text{interior } A \rangle$ **show** $(f\ (x + h) - f\ x) / h \geq 0$

by (*cases h rule: linorder_cases[of _ 0]*)

(*simp_all add: divide_nonpos_neg divide_nonneg_pos mono_onD field_simps*)

qed

qed *simp*

proposition *mono_on_ctble_discont*:

fixes *f* :: *real* $\Rightarrow$ *real*

```

fixes  $A :: \text{real set}$ 
assumes  $\text{mono\_on } f \ A$ 
shows  $\text{countable } \{a \in A. \neg \text{continuous (at } a \text{ within } A) f\}$ 
proof -
  have  $\text{mono}: \bigwedge x \ y. x \in A \implies y \in A \implies x \leq y \implies f \ x \leq f \ y$ 
    using  $\langle \text{mono\_on } f \ A \rangle$  by  $(\text{simp add: mono\_on\_def})$ 
  have  $\forall a \in \{a \in A. \neg \text{continuous (at } a \text{ within } A) f\}. \exists q :: \text{nat} \times \text{rat.}$ 
     $(\text{fst } q = 0 \wedge \text{of\_rat (snd } q) < f \ a \wedge (\forall x \in A. x < a \longrightarrow f \ x < \text{of\_rat (snd } q))) \vee$ 
     $(\text{fst } q = 1 \wedge \text{of\_rat (snd } q) > f \ a \wedge (\forall x \in A. x > a \longrightarrow f \ x > \text{of\_rat (snd } q)))$ 
  proof  $(\text{clarsimp simp del: One\_nat\_def})$ 
    fix  $a$  assume  $a \in A$  assume  $\neg \text{continuous (at } a \text{ within } A) f$ 
    thus  $\exists q1 \ q2.$ 
       $q1 = 0 \wedge \text{real\_of\_rat } q2 < f \ a \wedge (\forall x \in A. x < a \longrightarrow f \ x < \text{real\_of\_rat } q2) \vee$ 
       $q1 = 1 \wedge f \ a < \text{real\_of\_rat } q2 \wedge (\forall x \in A. a < x \longrightarrow \text{real\_of\_rat } q2 < f \ x)$ 
    proof  $(\text{auto simp add: continuous\_within order\_tendsto\_iff eventually\_at})$ 
      fix  $l$  assume  $l < f \ a$ 
      then obtain  $q2$  where  $q2: l < \text{of\_rat } q2 \ \text{of\_rat } q2 < f \ a$ 
        using  $\text{of\_rat\_dense}$  by  $\text{blast}$ 
      assume  $*[\text{rule\_format}]: \forall d > 0. \exists x \in A. x \neq a \wedge \text{dist } x \ a < d \wedge \neg l < f \ x$ 
      from  $q2$  have  $\text{real\_of\_rat } q2 < f \ a \wedge (\forall x \in A. x < a \longrightarrow f \ x < \text{real\_of\_rat } q2)$ 
    proof  $\text{auto}$ 
      fix  $x$  assume  $x \in A \ x < a$ 
      with  $q2$   $*[\text{of } a - x]$  show  $f \ x < \text{real\_of\_rat } q2$ 
        apply  $(\text{auto simp add: dist\_real\_def not\_less})$ 
        apply  $(\text{subgoal\_tac } f \ x \leq f \ xa)$ 
        by  $(\text{auto intro: mono})$ 
      qed
    thus  $?thesis$  by  $\text{auto}$ 
  next
    fix  $u$  assume  $u > f \ a$ 
    then obtain  $q2$  where  $q2: f \ a < \text{of\_rat } q2 \ \text{of\_rat } q2 < u$ 
      using  $\text{of\_rat\_dense}$  by  $\text{blast}$ 
    assume  $*[\text{rule\_format}]: \forall d > 0. \exists x \in A. x \neq a \wedge \text{dist } x \ a < d \wedge \neg u > f \ x$ 
    from  $q2$  have  $\text{real\_of\_rat } q2 > f \ a \wedge (\forall x \in A. x > a \longrightarrow f \ x > \text{real\_of\_rat } q2)$ 
    proof  $\text{auto}$ 
      fix  $x$  assume  $x \in A \ x > a$ 
      with  $q2$   $*[\text{of } x - a]$  show  $f \ x > \text{real\_of\_rat } q2$ 
        apply  $(\text{auto simp add: dist\_real\_def})$ 
        apply  $(\text{subgoal\_tac } f \ x \geq f \ xa)$ 
        by  $(\text{auto intro: mono})$ 
      qed
    thus  $?thesis$  by  $\text{auto}$ 
  qed

```

```

qed
then obtain g :: real  $\Rightarrow$  nat  $\times$  rat where  $\forall a \in \{a \in A. \neg \text{continuous (at a within A) f}\}.$ 
  (fst (g a) = 0  $\wedge$  of_rat (snd (g a)) < f a  $\wedge$  ( $\forall x \in A. x < a \longrightarrow f x < \text{of\_rat (snd (g a))}$ )) |
  (fst (g a) = 1  $\wedge$  of_rat (snd (g a)) > f a  $\wedge$  ( $\forall x \in A. x > a \longrightarrow f x > \text{of\_rat (snd (g a))}$ ))
  by (rule bchoice [THEN exE]) blast
hence g:  $\bigwedge a x. a \in A \Longrightarrow \neg \text{continuous (at a within A) f} \Longrightarrow x \in A \Longrightarrow$ 
  (fst (g a) = 0  $\wedge$  of_rat (snd (g a)) < f a  $\wedge$  ( $x < a \longrightarrow f x < \text{of\_rat (snd (g a))}$ )) |
  (fst (g a) = 1  $\wedge$  of_rat (snd (g a)) > f a  $\wedge$  ( $x > a \longrightarrow f x > \text{of\_rat (snd (g a))}$ ))
  by auto
have inj_on g {a  $\in$  A.  $\neg \text{continuous (at a within A) f}$ }
proof (auto simp add: inj_on_def)
  fix w z
  assume 1: w  $\in$  A and 2:  $\neg \text{continuous (at w within A) f}$  and
    3: z  $\in$  A and 4:  $\neg \text{continuous (at z within A) f}$  and
    5: g w = g z
  from g [OF 1 2 3] g [OF 3 4 1] 5
  show w = z by auto
qed
thus ?thesis
  by (rule countableI')
qed

```

lemma *mono_on_ctble_discont_open:*

```

  fixes f :: real  $\Rightarrow$  real
  fixes A :: real set
  assumes open A mono_on f A
  shows countable {a  $\in$  A.  $\neg \text{isCont f a}$ }
proof -
  have {a  $\in$  A.  $\neg \text{isCont f a}$ } = {a  $\in$  A.  $\neg (\text{continuous (at a within A) f}$ )}
  by (auto simp add: continuous_within_open [OF _ ‹open A›])
  thus ?thesis
    apply (elim ssubst)
    by (rule mono_on_ctble_discont, rule assms)
qed

```

lemma *mono_ctble_discont:*

```

  fixes f :: real  $\Rightarrow$  real
  assumes mono f
  shows countable {a.  $\neg \text{isCont f a}$ }
  using assms mono_on_ctble_discont [of f UNIV] unfolding mono_on_def
  mono_def by auto

```

lemma *has_real_derivative_imp_continuous_on:*

```

  assumes  $\bigwedge x. x \in A \Longrightarrow (f \text{ has\_real\_derivative } f' x) \text{ (at } x)$ 

```

shows *continuous_on* A f
 apply (intro *differentiable_imp_continuous_on*, *unfold differentiable_on_def*)
 using *assms differentiable_at_withinI real_differentiable_def* by blast

lemma *continuous_interval_vimage_Int*:

assumes *continuous_on* $\{a::\text{real}..b\}$ g and *mono*: $\bigwedge x y. a \leq x \implies x \leq y \implies y \leq b \implies g x \leq g y$

assumes $a \leq b$ ($c::\text{real}$) $\leq d$ $\{c..d\} \subseteq \{g a..g b\}$

obtains $c' d'$ where $\{a..b\} \cap g - ' \{c..d\} = \{c'..d'\}$ $c' \leq d'$ $g c' = c$ $g d' = d$

proof—

let $?A = \{a..b\} \cap g - ' \{c..d\}$

from *IVT'*[of $g a c b$, *OF* $— \langle a \leq b \rangle$ *assms*(1)] *assms*(4,5)

obtain c'' where $c'': c'' \in ?A$ $g c'' = c$ by *auto*

from *IVT'*[of $g a d b$, *OF* $— \langle a \leq b \rangle$ *assms*(1)] *assms*(4,5)

obtain d'' where $d'': d'' \in ?A$ $g d'' = d$ by *auto*

hence [*simp*]: $?A \neq \{\}$ by *blast*

define c' where $c' = \text{Inf } ?A$

define d' where $d' = \text{Sup } ?A$

have $?A \subseteq \{c'..d'\}$ unfolding c'_def d'_def

by (intro *subsetI*) (auto intro: *cInf_lower cSup_upper*)

moreover from *assms* have *closed* $?A$

using *continuous_on_closed_vimage*[of $\{a..b\}$ g] by (*subst Int_commute*) *simp*

hence $c'd'_\text{in\_set}$: $c' \in ?A$ $d' \in ?A$ unfolding c'_def d'_def

by ((intro *closed_contains_Inf closed_contains_Sup*, *simp_all*))+

hence $\{c'..d'\} \subseteq ?A$ using *assms*

by (intro *subsetI*)

(auto intro!: *order_trans*[of $c g c' g x$ for x] *order_trans*[of $g x g d' d$ for x]
intro!: *mono*)

moreover have $c' \leq d'$ using $c'd'_\text{in\_set}$ (2) unfolding c'_def by (intro *cInf_lower*) *auto*

moreover have $g c' \leq c$ $g d' \geq d$

apply (insert $c'' d'' c'd'_\text{in\_set}$)

apply (*subst* c'' (2)[*symmetric*])

apply (auto *simp*: c'_def *intro!*: *mono cInf_lower* c'') []

apply (*subst* d'' (2)[*symmetric*])

apply (auto *simp*: d'_def *intro!*: *mono cSup_upper* d'') []

done

with $c'd'_\text{in\_set}$ have $g c' = c$ $g d' = d$ by *auto*

ultimately show *?thesis* using *that* by *blast*

qed

6.5.1 Generic Borel spaces

definition (in *topological_space*) *borel* :: 'a *measure* where

borel = *sigma UNIV* $\{S. \text{open } S\}$

abbreviation *borel_measurable* $M \equiv \text{measurable } M \text{ borel}$

lemma *in_borel_measurable*:

$f \in \text{borel\_measurable } M \longleftrightarrow$
 $(\forall S \in \text{sigma\_sets UNIV } \{S. \text{open } S\}. f - ' S \cap \text{space } M \in \text{sets } M)$
by (*auto simp add: measurable_def borel_def*)

lemma *in_borel_measurable_borel*:

$f \in \text{borel\_measurable } M \longleftrightarrow$
 $(\forall S \in \text{sets borel}.$
 $f - ' S \cap \text{space } M \in \text{sets } M)$
by (*auto simp add: measurable_def borel_def*)

lemma *space_borel[simp]*: $\text{space borel} = \text{UNIV}$

unfolding *borel_def* **by** *auto*

lemma *space_in_borel[measurable]*: $\text{UNIV} \in \text{sets borel}$

unfolding *borel_def* **by** *auto*

lemma *sets_borel*: $\text{sets borel} = \text{sigma\_sets UNIV } \{S. \text{open } S\}$

unfolding *borel_def* **by** (*rule sets_measure_of*) *simp*

lemma *measurable_sets_borel*:

$\llbracket f \in \text{measurable borel } M; A \in \text{sets } M \rrbracket \implies f - ' A \in \text{sets borel}$
by (*drule (1) measurable_sets*) *simp*

lemma *pred_Collect_borel[measurable (raw)]*: $\text{Measurable.pred borel } P \implies \{x. P\} \in \text{sets borel}$

unfolding *borel_def pred_def* **by** *auto*

lemma *borel_open[measurable (raw generic)]*:

assumes *open A* **shows** $A \in \text{sets borel}$

proof –

have $A \in \{S. \text{open } S\}$ **unfolding** *mem_Collect_eq* **using** *assms* .

thus *?thesis* **unfolding** *borel_def* **by** *auto*

qed

lemma *borel_closed[measurable (raw generic)]*:

assumes *closed A* **shows** $A \in \text{sets borel}$

proof –

have $\text{space borel} - (- A) \in \text{sets borel}$

using *assms* **unfolding** *closed_def* **by** (*blast intro: borel_open*)

thus *?thesis* **by** *simp*

qed

lemma *borel_singleton[measurable]*:

$A \in \text{sets borel} \implies \text{insert } x A \in \text{sets } (\text{borel} :: 'a::t1\_space \text{measure})$

unfolding *insert_def* **by** (*rule sets.Un*) *auto*

lemma *sets_borel_eq_count_space*: $\text{sets } (\text{borel} :: 'a::\{\text{countable}, t2\_space\} \text{measure}) = \text{count\_space UNIV}$

proof –

have $(\bigcup_{a \in A}. \{a\}) \in \text{sets borel}$ **for** $A :: 'a \text{ set}$
by $(\text{intro sets.countable\_UN})$ *auto*
then show *?thesis*
by *auto*

qed

lemma *borel_comp[measurable]*: $A \in \text{sets borel} \implies - A \in \text{sets borel}$
unfolding *Compl_eq_Diff_UNIV* **by** *simp*

lemma *borel_measurable_vimage*:

fixes $f :: 'a \Rightarrow 'x::t2\_space$
assumes *borel[measurable]*: $f \in \text{borel\_measurable } M$
shows $f - ' \{x\} \cap \text{space } M \in \text{sets } M$
by *simp*

lemma *borel_measurableI*:

fixes $f :: 'a \Rightarrow 'x::topological\_space$
assumes $\bigwedge S. \text{open } S \implies f - ' S \cap \text{space } M \in \text{sets } M$
shows $f \in \text{borel\_measurable } M$
unfolding *borel_def*

proof $(\text{rule measurable\_measure\_of, simp\_all})$

fix $S :: 'x \text{ set}$ **assume** *open S* **thus** $f - ' S \cap \text{space } M \in \text{sets } M$
using *assms[of S]* **by** *simp*

qed

lemma *borel_measurable_const*:

$(\lambda x. c) \in \text{borel\_measurable } M$
by *auto*

lemma *borel_measurable_indicator*:

assumes $A: A \in \text{sets } M$
shows *indicator A* $\in \text{borel\_measurable } M$
unfolding *indicator_def [abs_def]* **using** A
by $(\text{auto intro!: measurable\_If\_set})$

lemma *borel_measurable_count_space[measurable (raw)]*:

$f \in \text{borel\_measurable } (\text{count\_space } S)$
unfolding *measurable_def* **by** *auto*

lemma *borel_measurable_indicator'[measurable (raw)]*:

assumes *[measurable]*: $\{x \in \text{space } M. f x \in A\} \in \text{sets } M$
shows $(\lambda x. \text{indicator } (A x) (f x)) \in \text{borel\_measurable } M$
unfolding *indicator_def [abs_def]*
by $(\text{auto intro!: measurable\_If})$

lemma *borel_measurable_indicator_iff*:

$(\text{indicator } A :: 'a \Rightarrow 'x::\{t1\_space, \text{zero\_neq\_one}\}) \in \text{borel\_measurable } M \iff$
 $A \cap \text{space } M \in \text{sets } M$


```

  (is ?I ∈ borel_measurable M ⟷)
proof
  assume ?I ∈ borel_measurable M
  then have ?I - ' {1} ∩ space M ∈ sets M
    unfolding measurable_def by auto
  also have ?I - ' {1} ∩ space M = A ∩ space M
    unfolding indicator_def [abs_def] by auto
  finally show A ∩ space M ∈ sets M .
next
  assume A ∩ space M ∈ sets M
  moreover have ?I ∈ borel_measurable M ⟷
    (indicator (A ∩ space M) :: 'a ⇒ 'x) ∈ borel_measurable M
    by (intro measurable_cong) (auto simp: indicator_def)
  ultimately show ?I ∈ borel_measurable M by auto
qed

```

```

lemma borel_measurable_subalgebra:
  assumes sets N ⊆ sets M space N = space M f ∈ borel_measurable N
  shows f ∈ borel_measurable M
  using assms unfolding measurable_def by auto

```

```

lemma borel_measurable_restrict_space_iff_ereal:
  fixes f :: 'a ⇒ ereal
  assumes Ω[measurable, simp]: Ω ∩ space M ∈ sets M
  shows f ∈ borel_measurable (restrict_space M Ω) ⟷
    (λx. f x * indicator Ω x) ∈ borel_measurable M
  by (subst measurable_restrict_space_iff)
    (auto simp: indicator_def of_bool_def if_distrib[where f=λx. a * x for a]
    cong del: if_weak_cong)

```

```

lemma borel_measurable_restrict_space_iff_ennreal:
  fixes f :: 'a ⇒ ennreal
  assumes Ω[measurable, simp]: Ω ∩ space M ∈ sets M
  shows f ∈ borel_measurable (restrict_space M Ω) ⟷
    (λx. f x * indicator Ω x) ∈ borel_measurable M
  by (subst measurable_restrict_space_iff)
    (auto simp: indicator_def of_bool_def if_distrib[where f=λx. a * x for a]
    cong del: if_weak_cong)

```

```

lemma borel_measurable_restrict_space_iff:
  fixes f :: 'a ⇒ 'b::real_normed_vector
  assumes Ω[measurable, simp]: Ω ∩ space M ∈ sets M
  shows f ∈ borel_measurable (restrict_space M Ω) ⟷
    (λx. indicator Ω x *R f x) ∈ borel_measurable M
  by (subst measurable_restrict_space_iff)
    (auto simp: indicator_def of_bool_def if_distrib[where f=λx. x *R a for a]
    ac_simps
    cong del: if_weak_cong)

```

lemma *cbox_borel[measurable]*: *cbox* *a b* $\in$ *sets borel*
by (*auto intro: borel_closed*)

lemma *box_borel[measurable]*: *box* *a b* $\in$ *sets borel*
by (*auto intro: borel_open*)

lemma *borel_compact*: *compact* (*A::'a::t2_space set*) $\implies$ *A* $\in$ *sets borel*
by (*auto intro: borel_closed dest!: compact_imp_closed*)

lemma *borel_sigma_sets_subset*:
A $\subseteq$ *sets borel* $\implies$ *sigma_sets UNIV A* $\subseteq$ *sets borel*
using *sets.sigma_sets_subset[of A borel]* **by** *simp*

lemma *borel_eq_sigmaI1*:
fixes *F* :: '*i* $\Rightarrow$ '*a::topological_space set* **and** *X* :: '*a::topological_space set set*
assumes *borel_eq*: *borel* = *sigma UNIV X*
assumes *X*: $\bigwedge x. x \in X \implies x \in \text{sets } (\text{sigma UNIV } (F \text{ ' } A))$
assumes *F*: $\bigwedge i. i \in A \implies F \text{ } i \in \text{sets borel}$
shows *borel* = *sigma UNIV (F ' A)*
unfolding *borel_def*
proof (*intro sigma_eqI antisym*)
have *borel_rev_eq*: *sigma_sets UNIV {S::'a set. open S}* = *sets borel*
unfolding *borel_def* **by** *simp*
also have $\dots = \text{sigma\_sets UNIV } X$
unfolding *borel_eq* **by** *simp*
also have $\dots \subseteq \text{sigma\_sets UNIV } (F \text{ ' } A)$
using *X* **by** (*intro sigma_algebra.sigma_sets_subset[OF sigma_algebra_sigma_sets]*)
auto
finally show *sigma_sets UNIV {S. open S}* $\subseteq$ *sigma_sets UNIV (F ' A)* .
show *sigma_sets UNIV (F ' A)* $\subseteq$ *sigma_sets UNIV {S. open S}*
unfolding *borel_rev_eq* **using** *F* **by** (*intro borel_sigma_sets_subset*) *auto*
qed *auto*

lemma *borel_eq_sigmaI2*:
fixes *F* :: '*i* $\Rightarrow$ '*j* $\Rightarrow$ '*a::topological_space set*
and *G* :: '*l* $\Rightarrow$ '*k* $\Rightarrow$ '*a::topological_space set*
assumes *borel_eq*: *borel* = *sigma UNIV (($\lambda(i, j). G \text{ } i \text{ } j$). *G* *i j*) ' B)*
assumes *X*: $\bigwedge i \text{ } j. (i, j) \in B \implies G \text{ } i \text{ } j \in \text{sets } (\text{sigma UNIV } ((\lambda(i, j). F \text{ } i \text{ } j) \text{ ' } A))$
assumes *F*: $\bigwedge i \text{ } j. (i, j) \in A \implies F \text{ } i \text{ } j \in \text{sets borel}$
shows *borel* = *sigma UNIV (($\lambda(i, j). F \text{ } i \text{ } j$) ' A)*
using *assms*
by (*intro borel_eq_sigmaI1[where X=($\lambda(i, j). G \text{ } i \text{ } j$) ' B and F=($\lambda(i, j). F \text{ } i \text{ } j$)]*) *auto*

lemma *borel_eq_sigmaI3*:
fixes *F* :: '*i* $\Rightarrow$ '*j* $\Rightarrow$ '*a::topological_space set* **and** *X* :: '*a::topological_space set set*
assumes *borel_eq*: *borel* = *sigma UNIV X*
assumes *X*: $\bigwedge x. x \in X \implies x \in \text{sets } (\text{sigma UNIV } ((\lambda(i, j). F \text{ } i \text{ } j) \text{ ' } A))$

```

assumes F:  $\bigwedge i j. (i, j) \in A \implies F i j \in \text{sets borel}$ 
shows borel = sigma UNIV (( $\lambda(i, j). F i j$ ) ' A)
using assms by (intro borel_eq_sigmaI1[where X=X and F=( $\lambda(i, j). F i j$ )])
auto

```

```

lemma borel_eq_sigmaI4:
  fixes F :: 'i  $\Rightarrow$  'a::topological_space set
  and G :: 'l  $\Rightarrow$  'k  $\Rightarrow$  'a::topological_space set
  assumes borel_eq: borel = sigma UNIV (( $\lambda(i, j). G i j$ ) ' A)
  assumes X:  $\bigwedge i j. (i, j) \in A \implies G i j \in \text{sets (sigma UNIV (range F))}$ 
  assumes F:  $\bigwedge i. F i \in \text{sets borel}$ 
  shows borel = sigma UNIV (range F)
  using assms by (intro borel_eq_sigmaI1[where X=( $\lambda(i, j). G i j$ ) ' A and F=F]) auto

```

```

lemma borel_eq_sigmaI5:
  fixes F :: 'i  $\Rightarrow$  'j  $\Rightarrow$  'a::topological_space set and G :: 'l  $\Rightarrow$  'a::topological_space set
  assumes borel_eq: borel = sigma UNIV (range G)
  assumes X:  $\bigwedge i. G i \in \text{sets (sigma UNIV (range (\lambda(i, j). F i j)))}$ 
  assumes F:  $\bigwedge i j. F i j \in \text{sets borel}$ 
  shows borel = sigma UNIV (range ( $\lambda(i, j). F i j$ ))
  using assms by (intro borel_eq_sigmaI1[where X=range G and F=( $\lambda(i, j). F i j$ )]) auto

```

```

theorem second_countable_borel_measurable:
  fixes X :: 'a::second_countable_topology set set
  assumes eq: open = generate_topology X
  shows borel = sigma UNIV X
  unfolding borel_def
  proof (intro sigma_eqI sigma_sets_eqI)
    interpret X: sigma_algebra UNIV sigma_sets UNIV X
    by (rule sigma_algebra_sigma_sets) simp
  end

```

```

fix S :: 'a set assume S  $\in$  Collect open
then have generate_topology X S
  by (auto simp: eq)
then show S  $\in$  sigma_sets UNIV X
proof induction
  case (UN K)
  then have K:  $\bigwedge k. k \in K \implies \text{open } k$ 
  unfolding eq by auto
  from ex_countable_basis obtain B :: 'a set set where
    B:  $\bigwedge b. b \in B \implies \text{open } b \wedge X. \text{open } X \implies \exists b \subseteq B. (\bigcup b) = X$  and countable B
  by (auto simp: topological_basis_def)
  from B(2)[OF K] obtain m where m:  $\bigwedge k. k \in K \implies m k \subseteq B \wedge k. k \in K \implies \bigcup (m k) = k$ 
  by metis

```

```

define U where U = (⋃ k∈K. m k)
with m have countable U
  by (intro countable_subset[OF _ ⟨countable B⟩]) auto
have ⋃ U = (⋃ A∈U. A) by simp
also have ... = ⋃ K
  unfolding U_def UN_simps by (simp add: m)
finally have ⋃ U = ⋃ K .

have ∀ b∈U. ∃ k∈K. b ⊆ k
  using m by (auto simp: U_def)
then obtain u where u: ∧ b. b ∈ U ⇒ u b ∈ K and ∧ b. b ∈ U ⇒ b ⊆ u b
  by metis
then have (⋃ b∈U. u b) ⊆ ⋃ K ⋃ U ⊆ (⋃ b∈U. u b)
  by auto
then have ⋃ K = (⋃ b∈U. u b)
  unfolding ⟨⋃ U = ⋃ K⟩ by auto
also have ... ∈ sigma_sets UNIV X
  using u UN by (intro X.countable_UN' ⟨countable U⟩) auto
finally show ⋃ K ∈ sigma_sets UNIV X .
qed auto
qed (auto simp: eq intro: generate_topology.Basis)

lemma borel_eq_closed: borel = sigma UNIV (Collect closed)
  unfolding borel_def
proof (intro sigma_eqI sigma_sets_eqI, safe)
  fix x :: 'a set assume open x
  hence x = UNIV - (UNIV - x) by auto
  also have ... ∈ sigma_sets UNIV (Collect closed)
    by (force intro: sigma_sets.Compl simp: ⟨open x⟩)
  finally show x ∈ sigma_sets UNIV (Collect closed) by simp
next
  fix x :: 'a set assume closed x
  hence x = UNIV - (UNIV - x) by auto
  also have ... ∈ sigma_sets UNIV (Collect open)
    by (force intro: sigma_sets.Compl simp: ⟨closed x⟩)
  finally show x ∈ sigma_sets UNIV (Collect open) by simp
qed simp_all

proposition borel_eq_countable_basis:
  fixes B::'a::topological_space set set
  assumes countable B
  assumes topological_basis B
  shows borel = sigma UNIV B
  unfolding borel_def
proof (intro sigma_eqI sigma_sets_eqI, safe)
  interpret countable_basis open B using assms by (rule countable_basis_openI)
  fix X::'a set assume open X
  from open_countable_basisE[OF this] obtain B' where B': B' ⊆ B X = ⋃ B'
  .

```

```

    then show  $X \in \text{sigma\_sets UNIV } B$ 
      by (blast intro: sigma_sets_UNION ‹countable  $B$ › countable_subset)
next
  fix  $b$  assume  $b \in B$ 
  hence open  $b$  by (rule topological_basis_open[OF assms(2)])
  thus  $b \in \text{sigma\_sets UNIV (Collect open)}$  by auto
qed simp_all

lemma borel_measurable_continuous_on_restrict:
  fixes  $f :: 'a::\text{topological\_space} \Rightarrow 'b::\text{topological\_space}$ 
  assumes  $f: \text{continuous\_on } A \ f$ 
  shows  $f \in \text{borel\_measurable (restrict\_space borel } A)$ 
proof (rule borel_measurableI)
  fix  $S :: 'b \text{ set}$  assume open  $S$ 
  with  $f$  obtain  $T$  where  $f - 'S \cap A = T \cap A$  open  $T$ 
  by (metis continuous_on_open_invariant)
  then show  $f - 'S \cap \text{space (restrict\_space borel } A) \in \text{sets (restrict\_space borel } A)$ 
    by (force simp add: sets_restrict_space space_restrict_space)
qed

lemma borel_measurable_continuous_onI:  $\text{continuous\_on UNIV } f \Longrightarrow f \in \text{borel\_measurable borel}$ 
  by (drule borel_measurable_continuous_on_restrict) simp

lemma borel_measurable_continuous_on_if:
   $A \in \text{sets borel} \Longrightarrow \text{continuous\_on } A \ f \Longrightarrow \text{continuous\_on } (- A) \ g \Longrightarrow$ 
   $(\lambda x. \text{if } x \in A \text{ then } f \ x \text{ else } g \ x) \in \text{borel\_measurable borel}$ 
  by (auto simp add: measurable_If_restrict_space_iff Collect_neg_eq
    intro!: borel_measurable_continuous_on_restrict)

lemma borel_measurable_continuous_countable_exceptions:
  fixes  $f :: 'a::\text{t1\_space} \Rightarrow 'b::\text{topological\_space}$ 
  assumes  $X: \text{countable } X$ 
  assumes  $\text{continuous\_on } (- X) \ f$ 
  shows  $f \in \text{borel\_measurable borel}$ 
proof (rule measurable_discrete_difference[OF _  $X$ ])
  have  $X \in \text{sets borel}$ 
  by (rule sets.countable[OF _  $X$ ]) auto
  then show  $(\lambda x. \text{if } x \in X \text{ then undefined else } f \ x) \in \text{borel\_measurable borel}$ 
    by (intro borel_measurable_continuous_on_if assms continuous_intros)
qed auto

lemma borel_measurable_continuous_on:
  assumes  $f: \text{continuous\_on UNIV } f$  and  $g: g \in \text{borel\_measurable } M$ 
  shows  $(\lambda x. f \ (g \ x)) \in \text{borel\_measurable } M$ 
  using measurable_comp[OF  $g \text{ borel\_measurable\_continuous\_onI [OF } f]$ ] by (simp
  add: comp_def)

```

```

lemma borel_measurable_continuous_on_indicator:
  fixes  $f\ g :: 'a :: \text{topological\_space} \Rightarrow 'b :: \text{real\_normed\_vector}$ 
  shows  $A \in \text{sets borel} \implies \text{continuous\_on } A\ f \implies (\lambda x. \text{indicator } A\ x *_{\mathbb{R}} f\ x) \in$ 
    borel_measurable borel
  by (subst borel_measurable_restrict_space_iff[symmetric])
    (auto intro: borel_measurable_continuous_on_restrict)

lemma borel_measurable_Pair[measurable (raw)]:
  fixes  $f :: 'a \Rightarrow 'b :: \text{second\_countable\_topology}$  and  $g :: 'a \Rightarrow 'c :: \text{second\_countable\_topology}$ 
  assumes  $f[\text{measurable}]: f \in \text{borel\_measurable } M$ 
  assumes  $g[\text{measurable}]: g \in \text{borel\_measurable } M$ 
  shows  $(\lambda x. (f\ x, g\ x)) \in \text{borel\_measurable } M$ 
proof (subst borel_eq_countable_basis)
  let  $?B = \text{SOME } B :: 'b \text{ set set. countable } B \wedge \text{topological\_basis } B$ 
  let  $?C = \text{SOME } B :: 'c \text{ set set. countable } B \wedge \text{topological\_basis } B$ 
  let  $?P = (\lambda(b, c). b \times c) \text{ ' } (?B \times ?C)$ 
  show countable  $?P$  topological_basis  $?P$ 
    by (auto intro!: countable_basis topological_basis_prod is_basis)

  show  $(\lambda x. (f\ x, g\ x)) \in \text{measurable } M\ (\text{sigma UNIV } ?P)$ 
proof (rule measurable_measure_of)
  fix  $S$  assume  $S \in ?P$ 
  then obtain  $b\ c$  where  $b \in ?B\ c \in ?C$  and  $S: S = b \times c$  by auto
  then have borel: open  $b$  open  $c$ 
    by (auto intro: is_basis_topological_basis_open)
  have  $(\lambda x. (f\ x, g\ x)) \text{ -' } S \cap \text{space } M = (f \text{ -' } b \cap \text{space } M) \cap (g \text{ -' } c \cap \text{space } M)$ 
    unfolding  $S$  by auto
  also have  $\dots \in \text{sets } M$ 
    using borel by simp
  finally show  $(\lambda x. (f\ x, g\ x)) \text{ -' } S \cap \text{space } M \in \text{sets } M$  .
qed auto
qed

lemma borel_measurable_continuous_Pair:
  fixes  $f :: 'a \Rightarrow 'b :: \text{second\_countable\_topology}$  and  $g :: 'a \Rightarrow 'c :: \text{second\_countable\_topology}$ 
  assumes  $[ \text{measurable} ]: f \in \text{borel\_measurable } M$ 
  assumes  $[ \text{measurable} ]: g \in \text{borel\_measurable } M$ 
  assumes  $H: \text{continuous\_on UNIV } (\lambda x. H\ (\text{fst } x)\ (\text{snd } x))$ 
  shows  $(\lambda x. H\ (f\ x)\ (g\ x)) \in \text{borel\_measurable } M$ 
proof -
  have eq:  $(\lambda x. H\ (f\ x)\ (g\ x)) = (\lambda x. (\lambda x. H\ (\text{fst } x)\ (\text{snd } x))\ (f\ x, g\ x))$  by auto
  show ?thesis
    unfolding eq by (rule borel_measurable_continuous_on[OF H]) auto
qed

```

6.5.2 Borel spaces on order topologies

lemma $[\text{measurable}]:$

```

fixes  $a\ b :: 'a::linorder\_topology$ 
shows  $lessThan\_borel: \{.. $a$ \} \in sets\ borel$ 
  and  $greaterThan\_borel: \{a <..\} \in sets\ borel$ 
  and  $greaterThanLessThan\_borel: \{a <.. $b$ \} \in sets\ borel$ 
  and  $atMost\_borel: \{.. $a$ \} \in sets\ borel$ 
  and  $atLeast\_borel: \{a.. \} \in sets\ borel$ 
  and  $atLeastAtMost\_borel: \{a.. $b$ \} \in sets\ borel$ 
  and  $greaterThanAtMost\_borel: \{a <.. $b$ \} \in sets\ borel$ 
  and  $atLeastLessThan\_borel: \{a..< $b$ \} \in sets\ borel$ 
unfolding  $greaterThanAtMost\_def\ atLeastLessThan\_def$ 
by ( $blast\ intro: borel\_open\ borel\_closed\ open\_lessThan\ open\_greaterThan\ open\_greaterThanLessThan$ 
       $closed\_atMost\ closed\_atLeast\ closed\_atLeastAtMost$ ) $+$ 

lemma  $borel\_Iio$ :
   $borel = sigma\ UNIV\ (range\ lessThan :: 'a::\{linorder\_topology,\ second\_countable\_topology\}$ 
 $set\ set)$ 
  unfolding  $second\_countable\_borel\_measurable[OF\ open\_generated\_order]$ 
proof ( $intro\ sigma\_eqI\ sigma\_sets\_eqI$ )
  obtain  $D :: 'a\ set$  where  $D: countable\ D \wedge X. open\ X \implies X \neq \{\} \implies \exists d \in D.$ 
 $d \in X$ 
  by ( $rule\ countable\_dense\_setE$ )  $blast$ 

interpret  $L: sigma\_algebra\ UNIV\ sigma\_sets\ UNIV\ (range\ lessThan)$ 
  by ( $rule\ sigma\_algebra\_sigma\_sets$ )  $simp$ 

fix  $A :: 'a\ set$  assume  $A \in range\ lessThan \cup range\ greaterThan$ 
then obtain  $y$  where  $A = \{y <..\} \vee A = \{.. $y$ \}$ 
  by  $blast$ 
then show  $A \in sigma\_sets\ UNIV\ (range\ lessThan)$ 
proof
  assume  $A: A = \{y <..\}$ 
  show  $?thesis$ 
  proof cases
    assume  $\forall x > y. \exists d. y < d \wedge d < x$ 
    with  $D(2)[of\ \{y <.. $x$ \}\ for\ x]$  have  $\forall x > y. \exists d \in D. y < d \wedge d < x$ 
      by ( $auto\ simp: set\_eq\_iff$ )
    then have  $A = UNIV - (\bigcap d \in \{d \in D. y < d\}. \{.. $d$ \})$ 
      by ( $auto\ simp: A$ ) ( $metis\ less\_asym$ )
    also have  $\dots \in sigma\_sets\ UNIV\ (range\ lessThan)$ 
      using  $D(1)$  by ( $intro\ L.Diff\ L.top\ L.countable\_INT''$ )  $auto$ 
    finally show  $?thesis$  .
  next
    assume  $\neg (\forall x > y. \exists d. y < d \wedge d < x)$ 
    then obtain  $x$  where  $y < x \wedge d. y < d \implies \neg d < x$ 
      by  $auto$ 
    then have  $A = UNIV - \{.. $x$ \}$ 
      unfolding  $A$  by ( $auto\ simp: not\_less[symmetric]$ )
    also have  $\dots \in sigma\_sets\ UNIV\ (range\ lessThan)$ 
      by  $auto$ 

```

```

    finally show ?thesis .
  qed
qed auto
qed auto

```

lemma *borel_Ioi*:

borel = *sigma UNIV* (*range greaterThan* :: 'a::*{linorder_topology, second_countable_topology}*
set set)

unfolding *second_countable_borel_measurable*[*OF open_generated_order*]

proof (*intro sigma_eqI sigma_sets_eqI*)

obtain *D* :: 'a *set* **where** *D*: *countable D* $\wedge$ *X*. *open X* $\implies$ *X* $\neq$ {} $\implies \exists d \in D$.
d $\in$ *X*

by (*rule countable_dense_setE*) *blast*

interpret *L*: *sigma_algebra UNIV sigma_sets UNIV* (*range greaterThan*)

by (*rule sigma_algebra_sigma_sets*) *simp*

fix *A* :: 'a *set* **assume** *A* $\in$ *range lessThan* $\cup$ *range greaterThan*

then obtain *y* **where** *A* = {*y* <..*y*} $\vee$ *A* = {..*y*}

by *blast*

then show *A* $\in$ *sigma_sets UNIV* (*range greaterThan*)

proof

assume *A*: *A* = {..*y*}

show ?thesis

proof cases

assume $\forall x < y. \exists d. x < d \wedge d < y$

with *D*(2)[*of* {*x* <..*y*} **for** *x*] **have** $\forall x < y. \exists d \in D. x < d \wedge d < y$

by (*auto simp: set_eq_iff*)

then have *A* = *UNIV* - ($\bigcap d \in \{d \in D. d < y\}. \{d <..\}$)

by (*auto simp: A*) (*metis less_asym*)

also have ... $\in$ *sigma_sets UNIV* (*range greaterThan*)

using *D*(1) **by** (*intro L.Diff L.top L.countable_INT'*) *auto*

finally show ?thesis .

next

assume $\neg (\forall x < y. \exists d. x < d \wedge d < y)$

then obtain *x* **where** *x* < *y* $\wedge$ *y* > *d* $\implies x \geq d$

by (*auto simp: not_less[symmetric]*)

then have *A* = *UNIV* - {*x* <..*y*}

unfolding *A* *Compl_eq_Diff_UNIV[symmetric]* **by** *auto*

also have ... $\in$ *sigma_sets UNIV* (*range greaterThan*)

by *auto*

finally show ?thesis .

qed

qed auto

qed auto

lemma *borel_measurableI_less*:

fixes *f* :: 'a $\Rightarrow$ 'b::*{linorder_topology, second_countable_topology}*

shows ($\bigwedge y. \{x \in \text{space } M. f\ x < y\} \in \text{sets } M$) $\implies f \in \text{borel\_measurable } M$


```

unfolding borel_Ioi
by (rule measurable_measure_of) (auto simp: Int_def conj_commute)

lemma borel_measurableI_greater:
  fixes f :: 'a  $\Rightarrow$  'b::{linorder_topology, second_countable_topology}
  shows ( $\bigwedge y. \{x \in \text{space } M. y < f x\} \in \text{sets } M$ )  $\implies$  f  $\in$  borel_measurable M
  unfolding borel_Ioi
  by (rule measurable_measure_of) (auto simp: Int_def conj_commute)

lemma borel_measurableI_le:
  fixes f :: 'a  $\Rightarrow$  'b::{linorder_topology, second_countable_topology}
  shows ( $\bigwedge y. \{x \in \text{space } M. f x \leq y\} \in \text{sets } M$ )  $\implies$  f  $\in$  borel_measurable M
  by (rule borel_measurableI_greater) (auto simp: not_le[symmetric])

lemma borel_measurableI_ge:
  fixes f :: 'a  $\Rightarrow$  'b::{linorder_topology, second_countable_topology}
  shows ( $\bigwedge y. \{x \in \text{space } M. y \leq f x\} \in \text{sets } M$ )  $\implies$  f  $\in$  borel_measurable M
  by (rule borel_measurableI_less) (auto simp: not_le[symmetric])

lemma borel_measurable_less[measurable]:
  fixes f :: 'a  $\Rightarrow$  'b::{second_countable_topology, linorder_topology}
  assumes f  $\in$  borel_measurable M
  assumes g  $\in$  borel_measurable M
  shows  $\{w \in \text{space } M. f w < g w\} \in \text{sets } M$ 
proof -
  have  $\{w \in \text{space } M. f w < g w\} = (\lambda x. (f x, g x)) - ' \{x. \text{fst } x < \text{snd } x\} \cap \text{space } M$ 
  by auto
  also have  $\dots \in \text{sets } M$ 
  by (intro measurable_sets[OF borel_measurable_Pair borel_open, OF assms
    open_Collect_less]
    continuous_intros)
  finally show ?thesis .
qed

lemma
  fixes f :: 'a  $\Rightarrow$  'b::{second_countable_topology, linorder_topology}
  assumes f[measurable]: f  $\in$  borel_measurable M
  assumes g[measurable]: g  $\in$  borel_measurable M
  shows borel_measurable_le[measurable]:  $\{w \in \text{space } M. f w \leq g w\} \in \text{sets } M$ 
    and borel_measurable_eq[measurable]:  $\{w \in \text{space } M. f w = g w\} \in \text{sets } M$ 
    and borel_measurable_neq:  $\{w \in \text{space } M. f w \neq g w\} \in \text{sets } M$ 
  unfolding eq_iff_not_less[symmetric]
  by measurable

lemma borel_measurable_SUP[measurable (raw)]:
  fixes F ::  $\_ \Rightarrow \_ \Rightarrow \_::\{\text{complete\_linorder, linorder\_topology, second\_countable\_topology}\}$ 
  assumes [simp]: countable I
  assumes [measurable]:  $\bigwedge i. i \in I \implies F i \in \text{borel\_measurable } M$ 

```

shows $(\lambda x. \text{SUP } i \in I. F \ i \ x) \in \text{borel\_measurable } M$
by (rule borel_measurableI_greater) (simp add: less_SUP_iff)

lemma borel_measurable_INF[measurable (raw)]:
fixes $F :: \_ \Rightarrow \_ \Rightarrow \_ :: \{\text{complete\_linorder, linorder\_topology, second\_countable\_topology}\}$
assumes [simp]: countable I
assumes [measurable]: $\bigwedge i. i \in I \implies F \ i \in \text{borel\_measurable } M$
shows $(\lambda x. \text{INF } i \in I. F \ i \ x) \in \text{borel\_measurable } M$
by (rule borel_measurableI_less) (simp add: INF_less_iff)

lemma borel_measurable_cSUP[measurable (raw)]:
fixes $F :: \_ \Rightarrow \_ \Rightarrow 'a :: \{\text{conditionally\_complete\_linorder, linorder\_topology, second\_countable\_topology}\}$
assumes [simp]: countable I
assumes [measurable]: $\bigwedge i. i \in I \implies F \ i \in \text{borel\_measurable } M$
assumes bdd: $\bigwedge x. x \in \text{space } M \implies \text{bdd\_above } ((\lambda i. F \ i \ x) \text{ ` } I)$
shows $(\lambda x. \text{SUP } i \in I. F \ i \ x) \in \text{borel\_measurable } M$
proof cases
assume $I = \{\}$ **then show** ?thesis
unfolding $\langle I = \{\} \rangle$ image_empty **by** simp
next
assume $I \neq \{\}$
show ?thesis
proof (rule borel_measurableI_le)
fix y
have $\{x \in \text{space } M. \forall i \in I. F \ i \ x \leq y\} \in \text{sets } M$
by measurable
also have $\{x \in \text{space } M. \forall i \in I. F \ i \ x \leq y\} = \{x \in \text{space } M. (\text{SUP } i \in I. F \ i \ x) \leq y\}$
by (simp add: cSUP_le_iff $\langle I \neq \{\} \rangle$ bdd cong: conj_cong)
finally show $\{x \in \text{space } M. (\text{SUP } i \in I. F \ i \ x) \leq y\} \in \text{sets } M$.
qed
qed

lemma borel_measurable_cINF[measurable (raw)]:
fixes $F :: \_ \Rightarrow \_ \Rightarrow 'a :: \{\text{conditionally\_complete\_linorder, linorder\_topology, second\_countable\_topology}\}$
assumes [simp]: countable I
assumes [measurable]: $\bigwedge i. i \in I \implies F \ i \in \text{borel\_measurable } M$
assumes bdd: $\bigwedge x. x \in \text{space } M \implies \text{bdd\_below } ((\lambda i. F \ i \ x) \text{ ` } I)$
shows $(\lambda x. \text{INF } i \in I. F \ i \ x) \in \text{borel\_measurable } M$
proof cases
assume $I = \{\}$ **then show** ?thesis
unfolding $\langle I = \{\} \rangle$ image_empty **by** simp
next
assume $I \neq \{\}$
show ?thesis
proof (rule borel_measurableI_ge)
fix y

```

  have  $\{x \in \text{space } M. \forall i \in I. y \leq F i x\} \in \text{sets } M$ 
  by measurable
  also have  $\{x \in \text{space } M. \forall i \in I. y \leq F i x\} = \{x \in \text{space } M. y \leq (\text{INF } i \in I. F i x)\}$ 
  by (simp add: le_cINF_iff ‹ $I \neq \{\}$ › bdd cong: conj_cong)
  finally show  $\{x \in \text{space } M. y \leq (\text{INF } i \in I. F i x)\} \in \text{sets } M$  .
qed
qed

```

```

lemma borel_measurable_lfp[consumes 1, case_names continuity step]:
  fixes  $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b :: \{\text{complete\_linorder, linorder\_topology, second\_countable\_topology}\})$ 
  assumes  $\text{sup\_continuous } F$ 
  assumes  $*: \bigwedge f. f \in \text{borel\_measurable } M \implies F f \in \text{borel\_measurable } M$ 
  shows  $\text{lfp } F \in \text{borel\_measurable } M$ 
proof -
  { fix  $i$  have  $((F \smallfrown i) \text{ bot}) \in \text{borel\_measurable } M$ 
    by (induct  $i$ ) (auto intro!: *) }
  then have  $(\lambda x. \text{SUP } i. (F \smallfrown i) \text{ bot } x) \in \text{borel\_measurable } M$ 
  by measurable
  also have  $(\lambda x. \text{SUP } i. (F \smallfrown i) \text{ bot } x) = (\text{SUP } i. (F \smallfrown i) \text{ bot})$ 
  by (auto simp add: image_comp)
  also have  $(\text{SUP } i. (F \smallfrown i) \text{ bot}) = \text{lfp } F$ 
  by (rule sup_continuous_lfp[symmetric]) fact
  finally show ?thesis .
qed

```

```

lemma borel_measurable_gfp[consumes 1, case_names continuity step]:
  fixes  $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b :: \{\text{complete\_linorder, linorder\_topology, second\_countable\_topology}\})$ 
  assumes  $\text{inf\_continuous } F$ 
  assumes  $*: \bigwedge f. f \in \text{borel\_measurable } M \implies F f \in \text{borel\_measurable } M$ 
  shows  $\text{gfp } F \in \text{borel\_measurable } M$ 
proof -
  { fix  $i$  have  $((F \smallfrown i) \text{ top}) \in \text{borel\_measurable } M$ 
    by (induct  $i$ ) (auto intro!: * simp: bot_fun_def) }
  then have  $(\lambda x. \text{INF } i. (F \smallfrown i) \text{ top } x) \in \text{borel\_measurable } M$ 
  by measurable
  also have  $(\lambda x. \text{INF } i. (F \smallfrown i) \text{ top } x) = (\text{INF } i. (F \smallfrown i) \text{ top})$ 
  by (auto simp add: image_comp)
  also have  $\dots = \text{gfp } F$ 
  by (rule inf_continuous_gfp[symmetric]) fact
  finally show ?thesis .
qed

```

```

lemma borel_measurable_max[measurable (raw)]:
   $f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies (\lambda x. \text{max } (g x) (f x))$ 
   $:: 'b :: \{\text{second\_countable\_topology, linorder\_topology}\} \in \text{borel\_measurable } M$ 
  by (rule borel_measurableI_less) simp

```

lemma *borel_measurable_min*[*measurable (raw)*]:
 $f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies (\lambda x. \min (g \ x) (f \ x))$
 $:: 'b::\{\text{second\_countable\_topology}, \text{linorder\_topology}\} \in \text{borel\_measurable } M$
by (rule *borel_measurableI_greater*) *simp*

lemma *borel_measurable_Min*[*measurable (raw)*]:
 $\text{finite } I \implies (\bigwedge i. i \in I \implies f \ i \in \text{borel\_measurable } M) \implies (\lambda x. \text{Min } ((\lambda i. f \ i \ x) 'I))$
 $:: 'b::\{\text{second\_countable\_topology}, \text{linorder\_topology}\} \in \text{borel\_measurable } M$
proof (induct *I* rule: *finite_induct*)
case (*insert i I*) **then show** ?*case*
by (cases *I* = {}) *auto*
qed *auto*

lemma *borel_measurable_Max*[*measurable (raw)*]:
 $\text{finite } I \implies (\bigwedge i. i \in I \implies f \ i \in \text{borel\_measurable } M) \implies (\lambda x. \text{Max } ((\lambda i. f \ i \ x) 'I))$
 $:: 'b::\{\text{second\_countable\_topology}, \text{linorder\_topology}\} \in \text{borel\_measurable } M$
proof (induct *I* rule: *finite_induct*)
case (*insert i I*) **then show** ?*case*
by (cases *I* = {}) *auto*
qed *auto*

lemma *borel_measurable_sup*[*measurable (raw)*]:
 $f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies (\lambda x. \sup (g \ x) (f \ x))$
 $:: 'b::\{\text{lattice}, \text{second\_countable\_topology}, \text{linorder\_topology}\} \in \text{borel\_measurable } M$
unfolding *sup_max* **by** *measurable*

lemma *borel_measurable_inf*[*measurable (raw)*]:
 $f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies (\lambda x. \inf (g \ x) (f \ x))$
 $:: 'b::\{\text{lattice}, \text{second\_countable\_topology}, \text{linorder\_topology}\} \in \text{borel\_measurable } M$
unfolding *inf_min* **by** *measurable*

lemma [*measurable (raw)*]:
fixes *f* :: *nat* $\Rightarrow$ '*a* $\Rightarrow$ '*b*:: $\{\text{complete\_linorder}, \text{second\_countable\_topology}, \text{linorder\_topology}\}$
assumes $\bigwedge i. f \ i \in \text{borel\_measurable } M$
shows *borel_measurable_liminf*: $(\lambda x. \liminf (\lambda i. f \ i \ x)) \in \text{borel\_measurable } M$
and *borel_measurable_limsup*: $(\lambda x. \limsup (\lambda i. f \ i \ x)) \in \text{borel\_measurable } M$
unfolding *liminf_SUP_INF limsup_INF_SUP* **using** *assms* **by** *auto*

lemma *measurable_convergent*[*measurable (raw)*]:
fixes *f* :: *nat* $\Rightarrow$ '*a* $\Rightarrow$ '*b*:: $\{\text{complete\_linorder}, \text{second\_countable\_topology}, \text{linorder\_topology}\}$
assumes [*measurable*]: $\bigwedge i. f \ i \in \text{borel\_measurable } M$
shows *Measurable.pred* *M* $(\lambda x. \text{convergent } (\lambda i. f \ i \ x))$
unfolding *convergent_ereal* **by** *measurable*

lemma *sets_Collect_convergent*[*measurable*]:
fixes *f* :: *nat* $\Rightarrow$ '*a* $\Rightarrow$ '*b*:: $\{\text{complete\_linorder}, \text{second\_countable\_topology}, \text{linorder\_topology}\}$

assumes $f[\text{measurable}]$: $\bigwedge i. f\ i \in \text{borel\_measurable } M$
shows $\{x \in \text{space } M. \text{convergent } (\lambda i. f\ i\ x)\} \in \text{sets } M$
by *measurable*

lemma *borel_measurable_lim*[*measurable (raw)*]:
fixes $f :: \text{nat} \Rightarrow 'a \Rightarrow 'b :: \{\text{complete\_linorder, second\_countable\_topology, linorder\_topology}\}$
assumes $f[\text{measurable}]$: $\bigwedge i. f\ i \in \text{borel\_measurable } M$
shows $(\lambda x. \text{lim } (\lambda i. f\ i\ x)) \in \text{borel\_measurable } M$
proof –
have $\bigwedge x. \text{lim } (\lambda i. f\ i\ x) = (\text{if convergent } (\lambda i. f\ i\ x) \text{ then limsup } (\lambda i. f\ i\ x) \text{ else } (THE\ i. False))$
by (*simp add: lim_def convergent_def convergent_limsup_cl*)
then show *?thesis*
by *simp*
qed

lemma *borel_measurable_LIMSEQ_order*:
fixes $u :: \text{nat} \Rightarrow 'a \Rightarrow 'b :: \{\text{complete\_linorder, second\_countable\_topology, linorder\_topology}\}$
assumes u' : $\bigwedge x. x \in \text{space } M \implies (\lambda i. u\ i\ x) \longrightarrow u'\ x$
and u : $\bigwedge i. u\ i \in \text{borel\_measurable } M$
shows $u' \in \text{borel\_measurable } M$
proof –
have $\bigwedge x. x \in \text{space } M \implies u'\ x = \text{liminf } (\lambda n. u\ n\ x)$
using u' **by** (*simp add: lim_imp_Liminf[symmetric]*)
with u **show** *?thesis* **by** (*simp cong: measurable_cong*)
qed

6.5.3 Borel spaces on topological monoids

lemma *borel_measurable_add*[*measurable (raw)*]:
fixes $f\ g :: 'a \Rightarrow 'b :: \{\text{second\_countable\_topology, topological\_monoid\_add}\}$
assumes f : $f \in \text{borel\_measurable } M$
assumes g : $g \in \text{borel\_measurable } M$
shows $(\lambda x. f\ x + g\ x) \in \text{borel\_measurable } M$
using $f\ g$ **by** (*rule borel_measurable_continuous_Pair*) (*intro continuous_intros*)

lemma *borel_measurable_sum*[*measurable (raw)*]:
fixes $f :: 'c \Rightarrow 'a \Rightarrow 'b :: \{\text{second\_countable\_topology, topological\_comm\_monoid\_add}\}$
assumes $\bigwedge i. i \in S \implies f\ i \in \text{borel\_measurable } M$
shows $(\lambda x. \sum_{i \in S} f\ i\ x) \in \text{borel\_measurable } M$
proof *cases*
assume *finite S*
thus *?thesis* **using** *assms* **by** *induct auto*
qed *simp*

lemma *borel_measurable_suminf_order*[*measurable (raw)*]:
fixes $f :: \text{nat} \Rightarrow 'a \Rightarrow 'b :: \{\text{complete\_linorder, second\_countable\_topology, linorder\_topology, topological\_comm\_monoid\_add}\}$
assumes $f[\text{measurable}]$: $\bigwedge i. f\ i \in \text{borel\_measurable } M$

shows $(\lambda x. \text{suminf } (\lambda i. f \ i \ x)) \in \text{borel\_measurable } M$
unfolding *suminf_def sums_def[abs_def] lim_def[symmetric]* **by** *simp*

6.5.4 Borel spaces on Euclidean spaces

lemma *borel_measurable_inner[measurable (raw)]*:
fixes $f \ g :: 'a \Rightarrow 'b :: \{\text{second\_countable\_topology, real\_inner}\}$
assumes $f \in \text{borel\_measurable } M$
assumes $g \in \text{borel\_measurable } M$
shows $(\lambda x. f \ x \cdot g \ x) \in \text{borel\_measurable } M$
using *assms*
by *(rule borel_measurable_continuous_Pair) (intro continuous_intros)*

notation
eucl_less (**infix** $<e$ 50)

lemma *box_oc*: $\{x. a <e x \wedge x \leq b\} = \{x. a <e x\} \cap \{..b\}$
and *box_co*: $\{x. a \leq x \wedge x <e b\} = \{a.. \} \cap \{x. x <e b\}$
by *auto*

lemma *eucl_ivals[measurable]*:
fixes $a \ b :: 'a :: \{\text{ordered\_euclidean\_space}\}$
shows $\{x. x <e a\} \in \text{sets borel}$
and $\{x. a <e x\} \in \text{sets borel}$
and $\{..a\} \in \text{sets borel}$
and $\{a.. \} \in \text{sets borel}$
and $\{a..b\} \in \text{sets borel}$
and $\{x. a <e x \wedge x \leq b\} \in \text{sets borel}$
and $\{x. a \leq x \wedge x <e b\} \in \text{sets borel}$
unfolding *box_oc box_co*
by *(auto intro: borel_open borel_closed)*

lemma
fixes $i :: 'a :: \{\text{second\_countable\_topology, real\_inner}\}$
shows *hafspace_less_borel*: $\{x. a < x \cdot i\} \in \text{sets borel}$
and *hafspace_greater_borel*: $\{x. x \cdot i < a\} \in \text{sets borel}$
and *hafspace_less_eq_borel*: $\{x. a \leq x \cdot i\} \in \text{sets borel}$
and *hafspace_greater_eq_borel*: $\{x. x \cdot i \leq a\} \in \text{sets borel}$
by *simp_all*

lemma *borel_eq_box*:
 $\text{borel} = \text{sigma } \text{UNIV } (\text{range } (\lambda (a, b). \text{box } a \ b :: 'a :: \text{euclidean\_space set}))$
(is $\_ = ?\text{SIGMA}$ **)**
proof *(rule borel_eq_sigmaI1[OF borel_def])*
fix $M :: 'a \text{ set}$ **assume** $M \in \{S. \text{open } S\}$
then have *open M* **by** *simp*
show $M \in ?\text{SIGMA}$
apply *(subst open_UNION_box[OF open_M])*
apply *(safe intro!: sets.countable_UN' countable_PiE countable_Collect)*

```

    apply (auto intro: countable_rat)
  done
qed (auto simp: box_def)

lemma halfspace_gt_in_halfspace:
  assumes  $i: i \in A$ 
  shows  $\{x::'a. a < x \cdot i\} \in$ 
    sigma_sets UNIV  $((\lambda (a, i). \{x::'a::euclidean\_space. x \cdot i < a\}) \text{ ' } (UNIV \times A))$ 
  (is ?set  $\in$  ?SIGMA)
proof -
  interpret sigma_algebra UNIV ?SIGMA
  by (intro sigma_algebra_sigma_sets) simp_all
  have *: ?set =  $(\bigcup n. UNIV - \{x::'a. x \cdot i < a + 1 / \text{real } (Suc\ n)\})$ 
  proof (safe, simp_all add: not_less del: of_nat_Suc)
    fix  $x :: 'a$  assume  $a < x \cdot i$ 
    with reals_Archimedean[of  $x \cdot i - a$ ]
    obtain  $n$  where  $a + 1 / \text{real } (Suc\ n) < x \cdot i$ 
    by (auto simp: field_simps)
    then show  $\exists n. a + 1 / \text{real } (Suc\ n) \leq x \cdot i$ 
    by (blast intro: less_imp_le)
  next
    fix  $x\ n$ 
    have  $a < a + 1 / \text{real } (Suc\ n)$  by auto
    also assume  $\dots \leq x$ 
    finally show  $a < x$  .
  qed
  show ?set  $\in$  ?SIGMA unfolding *
    by (auto intro!: Diff sigma_sets_Inter i)
qed

lemma borel_eq_halfspace_less:
  borel = sigma UNIV  $((\lambda (a, i). \{x::'a::euclidean\_space. x \cdot i < a\}) \text{ ' } (UNIV \times Basis))$ 
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI2[OF borel_eq_box])
  fix  $a\ b :: 'a$ 
  have box  $a\ b = \{x \in space\ ?SIGMA. \forall i \in Basis. a \cdot i < x \cdot i \wedge x \cdot i < b \cdot i\}$ 
    by (auto simp: box_def)
  also have  $\dots \in sets\ ?SIGMA$ 
    by (intro sets.sets_Collect_conj sets.sets_Collect_finite_All sets.sets_Collect_const)
      (auto intro!: halfspace_gt_in_halfspace countable_PiE countable_rat)
  finally show box  $a\ b \in sets\ ?SIGMA$  .
qed auto

lemma borel_eq_halfspace_le:
  borel = sigma UNIV  $((\lambda (a, i). \{x::'a::euclidean\_space. x \cdot i \leq a\}) \text{ ' } (UNIV \times Basis))$ 
  (is _ = ?SIGMA)

```

```

proof (rule borel_eq_sigmaI2[OF borel_eq_halfspace_less])
  fix a :: real and i :: 'a assume (a, i) ∈ UNIV × Basis
  then have i: i ∈ Basis by auto
  have *: {x::'a. x·i < a} = (⋃ n. {x. x·i ≤ a - 1/real (Suc n)})
  proof (safe, simp_all del: of_nat_Suc)
    fix x::'a assume *: x·i < a
    with reals_Archimedean[of a - x·i]
    obtain n where x · i < a - 1 / (real (Suc n))
    by (auto simp: field_simps)
    then show ∃ n. x · i ≤ a - 1 / (real (Suc n))
    by (blast intro: less_imp_le)
  next
    fix x::'a and n
    assume x·i ≤ a - 1 / real (Suc n)
    also have ... < a by auto
    finally show x·i < a .
  qed
  show {x. x·i < a} ∈ ?SIGMA unfolding *
    by (intro sets.countable_UN) (auto intro: i)
qed auto

```

```

lemma borel_eq_halfspace_ge:
  borel = sigma UNIV ((λ (a, i). {x::'a::euclidean_space. a ≤ x · i}) ' (UNIV ×
  Basis))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI2[OF borel_eq_halfspace_less])
  fix a :: real and i :: 'a assume i: (a, i) ∈ UNIV × Basis
  have *: {x::'a. x·i < a} = space ?SIGMA - {x::'a. a ≤ x·i} by auto
  show {x. x·i < a} ∈ ?SIGMA unfolding *
    using i by (intro sets.compl_sets) auto
qed auto

```

```

lemma borel_eq_halfspace_greater:
  borel = sigma UNIV ((λ (a, i). {x::'a::euclidean_space. a < x · i}) ' (UNIV ×
  Basis))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI2[OF borel_eq_halfspace_le])
  fix a :: real and i :: 'a assume (a, i) ∈ (UNIV × Basis)
  then have i: i ∈ Basis by auto
  have *: {x::'a. x·i ≤ a} = space ?SIGMA - {x::'a. a < x·i} by auto
  show {x. x·i ≤ a} ∈ ?SIGMA unfolding *
    by (intro sets.compl_sets) (auto intro: i)
qed auto

```

```

lemma borel_eq_atMost:
  borel = sigma UNIV (range (λ a. {..a::'a::ordered_euclidean_space}))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI4[OF borel_eq_halfspace_le])
  fix a :: real and i :: 'a assume (a, i) ∈ UNIV × Basis

```



```

then have  $i \in \text{Basis}$  by auto
then have  $\ast: \{x::'a. x \cdot i \leq a\} = (\bigcup k::\text{nat}. \{.. (\sum n \in \text{Basis}. (\text{if } n = i \text{ then } a \text{ else } \text{real } k) \cdot_R n)\})$ 
proof (safe, simp_all add: eucl_le [where  $'a = 'a$ ] split: if_split_asm)
  fix  $x :: 'a$ 
  obtain  $k$  where  $\text{Max } ((\cdot) \ x \ 'a \ \text{Basis}) \leq \text{real } k$ 
  using real_arch_simple by blast
  then have  $\bigwedge i. i \in \text{Basis} \implies x \cdot i \leq \text{real } k$ 
  by (subst (asm) Max_le_iff) auto
  then show  $\exists k::\text{nat}. \forall ia \in \text{Basis}. ia \neq i \longrightarrow x \cdot ia \leq \text{real } k$ 
  by (auto intro!: exI[of _ k])
qed
show  $\{x. x \cdot i \leq a\} \in ?\text{SIGMA}$  unfolding  $\ast$ 
  by (intro sets.countable_UN) auto
qed auto

```

lemma *borel_eq_greaterThan:*

```

  borel = sigma UNIV (range ( $\lambda a::'a::\text{ordered\_euclidean\_space}. \{x. a < e \ x\}$ ))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI4 [OF borel_eq_halfspace_le])
  fix  $a :: \text{real}$  and  $i :: 'a$  assume  $(a, i) \in \text{UNIV} \times \text{Basis}$ 
  then have  $i \in \text{Basis}$  by auto
  have  $\{x::'a. x \cdot i \leq a\} = \text{UNIV} - \{x::'a. a < x \cdot i\}$  by auto
  also have  $\ast: \{x::'a. a < x \cdot i\} =$ 
     $(\bigcup k::\text{nat}. \{x. (\sum n \in \text{Basis}. (\text{if } n = i \text{ then } a \text{ else } -\text{real } k) \cdot_R n) < e \ x\})$  using
     $i$ 
  proof (safe, simp_all add: eucl_less_def split: if_split_asm)
    fix  $x :: 'a$ 
    obtain  $k$  where  $\text{Max } ((\cdot) \ (- \ x) \ 'a \ \text{Basis}) < \text{real } k$ 
    using reals_Archimedean2 by blast
    { fix  $i :: 'a$  assume  $i \in \text{Basis}$ 
      then have  $-x \cdot i < \text{real } k$ 
      using  $k$  by (subst (asm) Max_less_iff) auto
      then have  $-\text{real } k < x \cdot i$  by simp }
    then show  $\exists k::\text{nat}. \forall ia \in \text{Basis}. ia \neq i \longrightarrow -\text{real } k < x \cdot ia$ 
    by (auto intro!: exI[of _ k])
  qed
  finally show  $\{x. x \cdot i \leq a\} \in ?\text{SIGMA}$ 
  apply (simp only:)
  apply (intro sets.countable_UN sets.Diff)
  apply (auto intro: sigma_sets_top)
  done
qed auto

```

lemma *borel_eq_lessThan:*

```

  borel = sigma UNIV (range ( $\lambda a::'a::\text{ordered\_euclidean\_space}. \{x. x < e \ a\}$ ))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI4 [OF borel_eq_halfspace_ge])
  fix  $a :: \text{real}$  and  $i :: 'a$  assume  $(a, i) \in \text{UNIV} \times \text{Basis}$ 

```

```

then have i: i ∈ Basis by auto
have {x::'a. a ≤ x·i} = UNIV - {x::'a. x·i < a} by auto
also have *: {x::'a. x·i < a} = (⋃ k::nat. {x. x < e (∑ n∈Basis. (if n = i then
a else real k) *R n)}) using ‹i ∈ Basis›
proof (safe, simp_all add: eucl_less_def split: if_split_asm)
  fix x :: 'a
  obtain k where k: Max ((·) x ‹Basis) < real k
  using reals_Archimedean2 by blast
  { fix i :: 'a assume i ∈ Basis
    then have x·i < real k
      using k by (subst (asm) Max_less_iff) auto
    then have x·i < real k by simp }
  then show ∃ k::nat. ∀ i∈Basis. i ≠ i → x · i < real k
    by (auto intro!: exI[of _ k])
qed
finally show {x. a ≤ x·i} ∈ ?SIGMA
  apply (simp only:)
  apply (intro sets.countable_UN sets.Diff)
  apply (auto intro: sigma_sets_top)
done
qed auto

```

```

lemma borel_eq_atLeastAtMost:
  borel = sigma UNIV (range (λ(a,b). {a..b} :: 'a::ordered_euclidean_space set))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI5[OF borel_eq_atMost])
  fix a::'a
  have *: {..a} = (⋃ n::nat. { - real n *R One .. a })
  proof (safe, simp_all add: eucl_le[where 'a='a])
    fix x :: 'a
    obtain k where k: Max ((·) (- x) ‹Basis) ≤ real k
    using real_arch_simple by blast
    { fix i :: 'a assume i ∈ Basis
      with k have - x·i ≤ real k
        by (subst (asm) Max_le_iff) (auto simp: field_simps)
      then have - real k ≤ x·i by simp }
    then show ∃ n::nat. ∀ i∈Basis. - real n ≤ x · i
      by (auto intro!: exI[of _ k])
  qed
  show {..a} ∈ ?SIGMA unfolding *
    by (intro sets.countable_UN)
      (auto intro!: sigma_sets_top)
qed auto

```

```

lemma borel_set_induct[consumes 1, case_names empty interval compl union]:
  assumes A ∈ sets borel
  assumes empty: P {} and int: ∧ a b. a ≤ b ⇒ P {a..b} and compl: ∧ A. A ∈
sets borel ⇒ P A ⇒ P (-A) and
  un: ∧ f. disjoint_family f ⇒ (∧ i. f i ∈ sets borel) ⇒ (∧ i. P (f i)) ⇒

```

```

P ( $\bigcup i::nat. f\ i$ )
  shows P (A::real set)
proof -
  let ?G = range ( $\lambda(a,b). \{a..b::real\}$ )
  have Int_stable ?G ?G  $\subseteq$  Pow UNIV A  $\in$  sigma_sets UNIV ?G
    using assms(1) by (auto simp add: borel_eq_atLeastAtMost Int_stable_def)
  thus ?thesis
  proof (induction rule: sigma_sets_induct_disjoint)
    case (union f)
    from union.hyps(2) have  $\bigwedge i. f\ i \in$  sets borel by (auto simp: borel_eq_atLeastAtMost)
    with union show ?case by (auto intro: un)
  next
    case (basic A)
    then obtain a b where A = {a .. b} by auto
    then show ?case
      by (cases a  $\leq$  b) (auto intro: int empty)
  qed (auto intro: empty compl simp: Compl_eq_Diff_UNIV[symmetric] borel_eq_atLeastAtMost)
qed

```

```

lemma borel_sigma_sets_Ioc: borel = sigma UNIV (range ( $\lambda(a, b). \{a <.. b::real\}$ ))
proof (rule borel_eq_sigmaI5[OF borel_eq_atMost])
  fix i :: real
  have  $\{..i\} = (\bigcup j::nat. \{-j <.. i\})$ 
    by (auto simp: minus_less_iff reals_Archimedean2)
  also have  $\dots \in$  sets (sigma UNIV (range ( $\lambda(i, j). \{i <.. j\}$ )))
    by (intro sets.countable_nat_UN) auto
  finally show  $\{..i\} \in$  sets (sigma UNIV (range ( $\lambda(i, j). \{i <.. j\}$ ))) .
qed simp

```

```

lemma eucl_lessThan:  $\{x::real. x <_e a\} =$  lessThan a
  by (simp add: eucl_less_def lessThan_def)

```

```

lemma borel_eq_atLeastLessThan:
  borel = sigma UNIV (range ( $\lambda(a, b). \{a ..< b :: real\}$ )) (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI5[OF borel_eq_lessThan])
  have move_uminus:  $\bigwedge x\ y::real. -x \leq y \longleftrightarrow -y \leq x$  by auto
  fix x :: real
  have  $\{..<x\} = (\bigcup i::nat. \{-real\ i ..< x\})$ 
    by (auto simp: move_uminus real_arch_simple)
  then show  $\{y. y <_e x\} \in$  ?SIGMA
    by (auto intro: sigma_sets.intros(2-) simp: eucl_lessThan)
qed auto

```

```

lemma borel_measurable_halfspacesI:
  fixes f :: 'a  $\Rightarrow$  'c::euclidean_space
  assumes F: borel = sigma UNIV (F ' (UNIV  $\times$  Basis))
  and S_eq:  $\bigwedge a\ i. S\ a\ i = f - ' F\ (a,i) \cap$  space M
  shows f  $\in$  borel_measurable M = ( $\forall i \in$  Basis.  $\forall a::real. S\ a\ i \in$  sets M)
proof safe

```

```

fix  $a :: \text{real}$  and  $i :: 'b$  assume  $i: i \in \text{Basis}$  and  $f: f \in \text{borel\_measurable } M$ 
then show  $S \ a \ i \in \text{sets } M$  unfolding  $\text{assms}$ 
  by ( $\text{auto intro! measurable\_sets simp: assms}(1)$ )
next
  assume  $a: \forall i \in \text{Basis}. \forall a. S \ a \ i \in \text{sets } M$ 
  then show  $f \in \text{borel\_measurable } M$ 
    by ( $\text{auto intro! measurable\_measure\_of simp: } S\_eq \ F$ )
qed

lemma borel\_measurable\_iff\_halfspace\_le:
  fixes  $f :: 'a \Rightarrow 'c :: \text{euclidean\_space}$ 
  shows  $f \in \text{borel\_measurable } M = (\forall i \in \text{Basis}. \forall a. \{w \in \text{space } M. f \ w \cdot i \leq a\} \in \text{sets } M)$ 
  by ( $\text{rule borel\_measurable\_halfspacesI}[OF \text{ borel\_eq\_halfspace\_le}]$ ) auto

lemma borel\_measurable\_iff\_halfspace\_less:
  fixes  $f :: 'a \Rightarrow 'c :: \text{euclidean\_space}$ 
  shows  $f \in \text{borel\_measurable } M \longleftrightarrow (\forall i \in \text{Basis}. \forall a. \{w \in \text{space } M. f \ w \cdot i < a\} \in \text{sets } M)$ 
  by ( $\text{rule borel\_measurable\_halfspacesI}[OF \text{ borel\_eq\_halfspace\_less}]$ ) auto

lemma borel\_measurable\_iff\_halfspace\_ge:
  fixes  $f :: 'a \Rightarrow 'c :: \text{euclidean\_space}$ 
  shows  $f \in \text{borel\_measurable } M = (\forall i \in \text{Basis}. \forall a. \{w \in \text{space } M. a \leq f \ w \cdot i\} \in \text{sets } M)$ 
  by ( $\text{rule borel\_measurable\_halfspacesI}[OF \text{ borel\_eq\_halfspace\_ge}]$ ) auto

lemma borel\_measurable\_iff\_halfspace\_greater:
  fixes  $f :: 'a \Rightarrow 'c :: \text{euclidean\_space}$ 
  shows  $f \in \text{borel\_measurable } M \longleftrightarrow (\forall i \in \text{Basis}. \forall a. \{w \in \text{space } M. a < f \ w \cdot i\} \in \text{sets } M)$ 
  by ( $\text{rule borel\_measurable\_halfspacesI}[OF \text{ borel\_eq\_halfspace\_greater}]$ ) auto

lemma borel\_measurable\_iff\_le:
   $(f :: 'a \Rightarrow \text{real}) \in \text{borel\_measurable } M = (\forall a. \{w \in \text{space } M. f \ w \leq a\} \in \text{sets } M)$ 
  using  $\text{borel\_measurable\_iff\_halfspace\_le}[\text{where } 'c = \text{real}]$  by  $\text{simp}$ 

lemma borel\_measurable\_iff\_less:
   $(f :: 'a \Rightarrow \text{real}) \in \text{borel\_measurable } M = (\forall a. \{w \in \text{space } M. f \ w < a\} \in \text{sets } M)$ 
  using  $\text{borel\_measurable\_iff\_halfspace\_less}[\text{where } 'c = \text{real}]$  by  $\text{simp}$ 

lemma borel\_measurable\_iff\_ge:
   $(f :: 'a \Rightarrow \text{real}) \in \text{borel\_measurable } M = (\forall a. \{w \in \text{space } M. a \leq f \ w\} \in \text{sets } M)$ 
  using  $\text{borel\_measurable\_iff\_halfspace\_ge}[\text{where } 'c = \text{real}]$ 
  by  $\text{simp}$ 

lemma borel\_measurable\_iff\_greater:
   $(f :: 'a \Rightarrow \text{real}) \in \text{borel\_measurable } M = (\forall a. \{w \in \text{space } M. a < f \ w\} \in \text{sets } M)$ 
  using  $\text{borel\_measurable\_iff\_halfspace\_greater}[\text{where } 'c = \text{real}]$  by  $\text{simp}$ 

```

```

lemma borel_measurable_euclidean_space:
  fixes  $f :: 'a \Rightarrow 'c::\text{euclidean\_space}$ 
  shows  $f \in \text{borel\_measurable } M \iff (\forall i \in \text{Basis}. (\lambda x. f\ x \cdot i) \in \text{borel\_measurable } M)$ 
proof safe
  assume  $f: \forall i \in \text{Basis}. (\lambda x. f\ x \cdot i) \in \text{borel\_measurable } M$ 
  then show  $f \in \text{borel\_measurable } M$ 
    by (subst borel_measurable_iff_halfspace_le) auto
qed auto

```

6.5.5 Borel measurable operators

```

lemma borel_measurable_norm[measurable]:  $\text{norm} \in \text{borel\_measurable borel}$ 
  by (intro borel_measurable_continuous_onI continuous_intros)

```

```

lemma borel_measurable_sgn [measurable]:  $(\text{sgn}::'a::\text{real\_normed\_vector} \Rightarrow 'a) \in \text{borel\_measurable borel}$ 
  by (rule borel_measurable_continuous_countable_exceptions[where  $X=\{0\}$ ])
    (auto intro!: continuous_on_sgn continuous_on_id)

```

```

lemma borel_measurable_uminus[measurable (raw)]:
  fixes  $g :: 'a \Rightarrow 'b::\{\text{second\_countable\_topology, real\_normed\_vector}\}$ 
  assumes  $g: g \in \text{borel\_measurable } M$ 
  shows  $(\lambda x. -\ g\ x) \in \text{borel\_measurable } M$ 
  by (rule borel_measurable_continuous_on[OF  $g$ ]) (intro continuous_intros)

```

```

lemma borel_measurable_diff[measurable (raw)]:
  fixes  $f :: 'a \Rightarrow 'b::\{\text{second\_countable\_topology, real\_normed\_vector}\}$ 
  assumes  $f: f \in \text{borel\_measurable } M$ 
  assumes  $g: g \in \text{borel\_measurable } M$ 
  shows  $(\lambda x. f\ x -\ g\ x) \in \text{borel\_measurable } M$ 
  using borel_measurable_add [of  $f\ M -\ g$ ] assms by (simp add: fun_Compl_def)

```

```

lemma borel_measurable_times[measurable (raw)]:
  fixes  $f :: 'a \Rightarrow 'b::\{\text{second\_countable\_topology, real\_normed\_algebra}\}$ 
  assumes  $f: f \in \text{borel\_measurable } M$ 
  assumes  $g: g \in \text{borel\_measurable } M$ 
  shows  $(\lambda x. f\ x * g\ x) \in \text{borel\_measurable } M$ 
  using  $f\ g$  by (rule borel_measurable_continuous_Pair) (intro continuous_intros)

```

```

lemma borel_measurable_prod[measurable (raw)]:
  fixes  $f :: 'c \Rightarrow 'a \Rightarrow 'b::\{\text{second\_countable\_topology, real\_normed\_field}\}$ 
  assumes  $\bigwedge i. i \in S \implies f\ i \in \text{borel\_measurable } M$ 
  shows  $(\lambda x. \prod_{i \in S}. f\ i\ x) \in \text{borel\_measurable } M$ 
proof cases
  assume finite  $S$ 
  thus ?thesis using assms by induct auto
qed simp

```

```

lemma borel_measurable_dist[measurable (raw)]:
  fixes g f :: 'a  $\Rightarrow$  'b::{second_countable_topology, metric_space}
  assumes f: f  $\in$  borel_measurable M
  assumes g: g  $\in$  borel_measurable M
  shows  $(\lambda x. \text{dist } (f x) (g x)) \in \text{borel\_measurable } M$ 
  using f g by (rule borel_measurable_continuous_Pair) (intro continuous_intros)

```

```

lemma borel_measurable_scaleR[measurable (raw)]:
  fixes g :: 'a  $\Rightarrow$  'b::{second_countable_topology, real_normed_vector}
  assumes f: f  $\in$  borel_measurable M
  assumes g: g  $\in$  borel_measurable M
  shows  $(\lambda x. f x *_{\mathbb{R}} g x) \in \text{borel\_measurable } M$ 
  using f g by (rule borel_measurable_continuous_Pair) (intro continuous_intros)

```

```

lemma borel_measurable_uminus_eq [simp]:
  fixes f :: 'a  $\Rightarrow$  'b::{second_countable_topology, real_normed_vector}
  shows  $(\lambda x. - f x) \in \text{borel\_measurable } M \iff f \in \text{borel\_measurable } M$  (is ?l = ?r)
proof
  assume ?l from borel_measurable_uminus[OF this] show ?r by simp
qed auto

```

```

lemma affine_borel_measurable_vector:
  fixes f :: 'a  $\Rightarrow$  'x::real_normed_vector
  assumes f  $\in$  borel_measurable M
  shows  $(\lambda x. a + b *_{\mathbb{R}} f x) \in \text{borel\_measurable } M$ 
proof (rule borel_measurableI)
  fix S :: 'x set assume open S
  show  $(\lambda x. a + b *_{\mathbb{R}} f x) - ' S \cap \text{space } M \in \text{sets } M$ 
proof cases
  assume b  $\neq 0$ 
  with open S have open  $((\lambda x. (- a + x) /_{\mathbb{R}} b) - ' S)$  (is open ?S)
  using open_affinity [of S inverse b - a /_{\mathbb{R}} b]
  by (auto simp: algebra_simps)
  hence ?S  $\in \text{sets borel}$  by auto
moreover
  from b  $\neq 0$  have  $(\lambda x. a + b *_{\mathbb{R}} f x) - ' S = f - ' ?S$ 
  apply auto by (rule_tac x=a + b *_{\mathbb{R}} f x in image_eqI, simp_all)
  ultimately show ?thesis using assms unfolding in_borel_measurable_borel
  by auto
qed simp
qed

```

```

lemma borel_measurable_const_scaleR[measurable (raw)]:
  f  $\in$  borel_measurable M  $\implies (\lambda x. b *_{\mathbb{R}} f x :: 'a::\text{real\_normed\_vector}) \in \text{borel\_measurable } M$ 
  using affine_borel_measurable_vector[of f M 0 b] by simp

```

lemma *borel_measurable_const_add*[*measurable (raw)*]:
 $f \in \text{borel\_measurable } M \implies (\lambda x. a + f x :: 'a :: \text{real\_normed\_vector}) \in \text{borel\_measurable } M$
using *affine_borel_measurable_vector*[*of f M a 1*] **by** *simp*

lemma *borel_measurable_inverse*[*measurable (raw)*]:
fixes $f :: 'a \Rightarrow 'b :: \text{real\_normed\_div\_algebra}$
assumes $f: f \in \text{borel\_measurable } M$
shows $(\lambda x. \text{inverse } (f x)) \in \text{borel\_measurable } M$
apply (*rule measurable_compose*[*OF f*])
apply (*rule borel_measurable_continuous_countable_exceptions*[*of {0}*])
apply (*auto intro!: continuous_on_inverse continuous_on_id*)
done

lemma *borel_measurable_divide*[*measurable (raw)*]:
 $f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies$
 $(\lambda x. f x / g x :: 'b :: \{\text{second\_countable\_topology, real\_normed\_div\_algebra}\}) \in \text{borel\_measurable } M$
by (*simp add: divide_inverse*)

lemma *borel_measurable_abs*[*measurable (raw)*]:
 $f \in \text{borel\_measurable } M \implies (\lambda x. |f x :: \text{real}|) \in \text{borel\_measurable } M$
unfolding *abs_real_def* **by** *simp*

lemma *borel_measurable_nth*[*measurable (raw)*]:
 $(\lambda x :: \text{real}^n. x \$ i) \in \text{borel\_measurable borel}$
by (*simp add: cart_eq_inner_axis*)

lemma *convex_measurable*:
fixes $A :: 'a :: \text{euclidean\_space set}$
shows $X \in \text{borel\_measurable } M \implies X \text{ 'space } M \subseteq A \implies \text{open } A \implies \text{convex\_on } A \implies$
 $(\lambda x. q (X x)) \in \text{borel\_measurable } M$
by (*rule measurable_compose*[**where** $f=X$ **and** $N=\text{restrict\_space borel } A$])
(auto intro!: borel_measurable_continuous_on_restrict convex_on_continuous measurable_restrict_space2)

lemma *borel_measurable_ln*[*measurable (raw)*]:
assumes $f: f \in \text{borel\_measurable } M$
shows $(\lambda x. \ln (f x :: \text{real})) \in \text{borel\_measurable } M$
apply (*rule measurable_compose*[*OF f*])
apply (*rule borel_measurable_continuous_countable_exceptions*[*of {0}*])
apply (*auto intro!: continuous_on_ln continuous_on_id*)
done

lemma *borel_measurable_log*[*measurable (raw)*]:
 $f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies (\lambda x. \log (g x) (f x))$
 $\in \text{borel\_measurable } M$
unfolding *log_def* **by** *auto*

lemma *borel_measurable_exp* [*measurable*]:
 (*exp* :: 'a :: {*real_normed_field*, *banach*} $\Rightarrow$ 'a) $\in$ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_at_imp_continuous_on ballI isCont_exp*)

lemma *measurable_real_floor* [*measurable*]:
 (*floor* :: *real* $\Rightarrow$ *int*) $\in$ *measurable borel (count_space UNIV)*
proof –
have $\bigwedge a x. \lfloor x \rfloor = a \longleftrightarrow (\text{real\_of\_int } a \leq x \wedge x < \text{real\_of\_int } (a + 1))$
by (*auto intro: floor_eq2*)
then show ?*thesis*
by (*auto simp: vimage_def measurable_count_space_eq2_countable*)
qed

lemma *measurable_real_ceiling* [*measurable*]:
 (*ceiling* :: *real* $\Rightarrow$ *int*) $\in$ *measurable borel (count_space UNIV)*
unfolding *ceiling_def* [*abs_def*] **by** *simp*

lemma *borel_measurable_real_floor*: ($\lambda x :: \text{real}. \text{real\_of\_int } \lfloor x \rfloor$) $\in$ *borel_measurable borel*
by *simp*

lemma *borel_measurable_root* [*measurable*]: *root n* $\in$ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_sqrt* [*measurable*]: *sqrt* $\in$ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_power* [*measurable (raw)*]:
fixes *f* :: $\_ \Rightarrow$ 'b :: {*power*, *real_normed_algebra*}
assumes *f*: *f* $\in$ *borel_measurable M*
shows $(\lambda x. (f x) ^ n) \in \text{borel\_measurable } M$
by (*intro borel_measurable_continuous_on [OF _ f] continuous_intros*)

lemma *borel_measurable_Re* [*measurable*]: *Re* $\in$ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_Im* [*measurable*]: *Im* $\in$ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_of_real* [*measurable*]: (*of_real* :: $\_ \Rightarrow$ ($\_ :: \text{real\_normed\_algebra}$))
 $\in$ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_sin* [*measurable*]: (*sin* :: $\_ \Rightarrow$ ($\_ :: \{\text{real\_normed\_field}, \text{banach}\}$))
 $\in$ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_cos* [*measurable*]: (*cos* :: $\_ \Rightarrow (\_ :: \{\text{real\_normed\_field}, \text{banach}\})$)
 $\in \text{borel\_measurable borel}$

by (intro *borel_measurable_continuous_onI* *continuous_intros*)

lemma *borel_measurable_arctan* [*measurable*]: *arctan* $\in \text{borel\_measurable borel}$

by (intro *borel_measurable_continuous_onI* *continuous_intros*)

lemma *borel_measurable_complex_iff*:

$f \in \text{borel\_measurable } M \iff$

$(\lambda x. \text{Re } (f x)) \in \text{borel\_measurable } M \wedge (\lambda x. \text{Im } (f x)) \in \text{borel\_measurable } M$

apply *auto*

apply (*subst fun_complex_eq*)

apply (intro *borel_measurable_add*)

apply *auto*

done

lemma *powr_real_measurable* [*measurable*]:

assumes $f \in \text{measurable } M$ $\text{borel } g \in \text{measurable } M$ *borel*

shows $(\lambda x. f x \text{ powr } g x :: \text{real}) \in \text{measurable } M$ *borel*

using *assms* **by** (*simp_all add: powr_def*)

lemma *measurable_of_bool* [*measurable*]: *of_bool* $\in \text{count\_space UNIV} \rightarrow_M \text{borel}$

by *simp*

6.5.6 Borel space on the extended reals

lemma *borel_measurable_ereal* [*measurable* (*raw*)]:

assumes $f: f \in \text{borel\_measurable } M$ **shows** $(\lambda x. \text{ereal } (f x)) \in \text{borel\_measurable } M$

using *continuous_on_ereal f* **by** (*rule borel_measurable_continuous_on*) (*rule continuous_on_id*)

lemma *borel_measurable_real_of_ereal* [*measurable* (*raw*)]:

fixes $f :: 'a \Rightarrow \text{ereal}$

assumes $f: f \in \text{borel\_measurable } M$

shows $(\lambda x. \text{real\_of\_ereal } (f x)) \in \text{borel\_measurable } M$

apply (*rule measurable_compose[OF f]*)

apply (*rule borel_measurable_continuous_countable_exceptions[of \{\infty, -\infty\}]*)

apply (*auto intro: continuous_on_real simp: Compl_eq_Diff_UNIV*)

done

lemma *borel_measurable_ereal_cases*:

fixes $f :: 'a \Rightarrow \text{ereal}$

assumes $f: f \in \text{borel\_measurable } M$

assumes $H: (\lambda x. H (\text{ereal } (\text{real\_of\_ereal } (f x)))) \in \text{borel\_measurable } M$

shows $(\lambda x. H (f x)) \in \text{borel\_measurable } M$

proof –

let $?F = \lambda x. \text{if } f x = \infty \text{ then } H \infty \text{ else if } f x = -\infty \text{ then } H (-\infty) \text{ else } H (\text{ereal } (\text{real\_of\_ereal } (f x)))$

```

{ fix x have H (f x) = ?F x by (cases f x) auto }
with f H show ?thesis by simp
qed

```

lemma

```

fixes f :: 'a ⇒ ereal assumes f[measurable]: f ∈ borel_measurable M
shows borel_measurable_ereal_abs[measurable(raw)]: (λx. |f x|) ∈ borel_measurable
M
and borel_measurable_ereal_inverse[measurable(raw)]: (λx. inverse (f x) ::
ereal) ∈ borel_measurable M
and borel_measurable_uminus_ereal[measurable(raw)]: (λx. - f x :: ereal) ∈
borel_measurable M
by (auto simp del: abs_real_of_ereal simp: borel_measurable_ereal_cases[OF f]
measurable>If)

```

lemma *borel_measurable_uminus_eq_ereal[simp]:*

```

(λx. - f x :: ereal) ∈ borel_measurable M ⟷ f ∈ borel_measurable M (is ?l =
?r)

```

proof

```

assume ?l from borel_measurable_uminus_ereal[OF this] show ?r by simp
qed auto

```

lemma *set_Collect_ereal2:*

```

fixes f g :: 'a ⇒ ereal
assumes f: f ∈ borel_measurable M
assumes g: g ∈ borel_measurable M
assumes H: {x ∈ space M. H (ereal (real_of_ereal (f x))) (ereal (real_of_ereal
(g x)))} ∈ sets M
{x ∈ space borel. H (-∞) (ereal x)} ∈ sets borel
{x ∈ space borel. H (∞) (ereal x)} ∈ sets borel
{x ∈ space borel. H (ereal x) (-∞)} ∈ sets borel
{x ∈ space borel. H (ereal x) (∞)} ∈ sets borel
shows {x ∈ space M. H (f x) (g x)} ∈ sets M

```

proof –

```

let ?G = λy x. if g x = ∞ then H y ∞ else if g x = -∞ then H y (-∞) else H
y (ereal (real_of_ereal (g x)))

```

```

let ?F = λx. if f x = ∞ then ?G ∞ x else if f x = -∞ then ?G (-∞) x else ?G
(ereal (real_of_ereal (f x))) x

```

```

{ fix x have H (f x) (g x) = ?F x by (cases f x g x rule: ereal2_cases) auto }

```

```

note * = this

```

```

from assms show ?thesis

```

```

by (subst *) (simp del: space_borel split del: if_split)

```

qed

lemma *borel_measurable_ereal_iff:*

```

shows (λx. ereal (f x)) ∈ borel_measurable M ⟷ f ∈ borel_measurable M

```

proof

```

assume (λx. ereal (f x)) ∈ borel_measurable M

```

```

from borel_measurable_real_of_ereal[OF this]

```

show $f \in \text{borel\_measurable } M$ **by** *auto*
qed *auto*

lemma *borel_measurable_erealD[measurable_dest]*:
 $(\lambda x. \text{ereal } (f x)) \in \text{borel\_measurable } M \implies g \in \text{measurable } N \ M \implies (\lambda x. f (g x)) \in \text{borel\_measurable } N$
unfolding *borel_measurable_ereal_iff* **by** *simp*

theorem *borel_measurable_ereal_iff_real*:
fixes $f :: 'a \Rightarrow \text{ereal}$
shows $f \in \text{borel\_measurable } M \longleftrightarrow$
 $((\lambda x. \text{real\_of\_ereal } (f x)) \in \text{borel\_measurable } M \wedge f - ' \{\infty\} \cap \text{space } M \in \text{sets } M \wedge f - ' \{-\infty\} \cap \text{space } M \in \text{sets } M)$
proof *safe*
assume $*$: $(\lambda x. \text{real\_of\_ereal } (f x)) \in \text{borel\_measurable } M$ $f - ' \{\infty\} \cap \text{space } M \in \text{sets } M$ $f - ' \{-\infty\} \cap \text{space } M \in \text{sets } M$
have $f - ' \{\infty\} \cap \text{space } M = \{x \in \text{space } M. f x = \infty\}$ $f - ' \{-\infty\} \cap \text{space } M = \{x \in \text{space } M. f x = -\infty\}$ **by** *auto*
with $*$ **have** $**$: $\{x \in \text{space } M. f x = \infty\} \in \text{sets } M$ $\{x \in \text{space } M. f x = -\infty\} \in \text{sets } M$ **by** *simp_all*
let $?f = \lambda x. \text{if } f x = \infty \text{ then } \infty \text{ else if } f x = -\infty \text{ then } -\infty \text{ else } \text{ereal } (\text{real\_of\_ereal } (f x))$
have $?f \in \text{borel\_measurable } M$ **using** $*$ $**$ **by** (*intro measurable>If*) *auto*
also have $?f = f$ **by** (*auto simp: fun_eq_iff ereal_real*)
finally show $f \in \text{borel\_measurable } M$.
qed *simp_all*

lemma *borel_measurable_ereal_iff_Iio*:
 $(f :: 'a \Rightarrow \text{ereal}) \in \text{borel\_measurable } M \longleftrightarrow (\forall a. f - ' \{.. < a\} \cap \text{space } M \in \text{sets } M)$
by (*auto simp: borel_Iio measurable_iff_measure_of*)

lemma *borel_measurable_ereal_iff_Ioi*:
 $(f :: 'a \Rightarrow \text{ereal}) \in \text{borel\_measurable } M \longleftrightarrow (\forall a. f - ' \{a <.. \} \cap \text{space } M \in \text{sets } M)$
by (*auto simp: borel_Ioi measurable_iff_measure_of*)

lemma *vimage_sets_compl_iff*:
 $f - ' A \cap \text{space } M \in \text{sets } M \longleftrightarrow f - ' (- A) \cap \text{space } M \in \text{sets } M$
proof -
{ fix A **assume** $f - ' A \cap \text{space } M \in \text{sets } M$
moreover have $f - ' (- A) \cap \text{space } M = \text{space } M - f - ' A \cap \text{space } M$ **by** *auto*
ultimately have $f - ' (- A) \cap \text{space } M \in \text{sets } M$ **by** *auto* }
from *this[of A]* *this[of -A]* **show** *?thesis*
by (*metis double_complement*)
qed

lemma *borel_measurable_iff_Iic_ereal*:

$(f::'a \Rightarrow \text{ereal}) \in \text{borel\_measurable } M \longleftrightarrow (\forall a. f - ' \{..a\} \cap \text{space } M \in \text{sets } M)$
unfolding $\text{borel\_measurable\_ereal\_iff\_Ioi vimage\_sets\_compl\_iff}$ [where $A = \{a < ..\}$ for a] **by** *simp*

lemma $\text{borel\_measurable\_iff\_Ici\_ereal}$:

$(f::'a \Rightarrow \text{ereal}) \in \text{borel\_measurable } M \longleftrightarrow (\forall a. f - ' \{a.. \} \cap \text{space } M \in \text{sets } M)$
unfolding $\text{borel\_measurable\_ereal\_iff\_Ioi vimage\_sets\_compl\_iff}$ [where $A = \{.. < a\}$ for a] **by** *simp*

lemma $\text{borel\_measurable\_ereal2}$:

fixes $f\ g :: 'a \Rightarrow \text{ereal}$
assumes $f: f \in \text{borel\_measurable } M$
assumes $g: g \in \text{borel\_measurable } M$
assumes $H: (\lambda x. H (\text{ereal } (\text{real\_of\_ereal } (f\ x))) (\text{ereal } (\text{real\_of\_ereal } (g\ x))))$
 $\in \text{borel\_measurable } M$
 $(\lambda x. H (-\infty) (\text{ereal } (\text{real\_of\_ereal } (g\ x)))) \in \text{borel\_measurable } M$
 $(\lambda x. H (\infty) (\text{ereal } (\text{real\_of\_ereal } (g\ x)))) \in \text{borel\_measurable } M$
 $(\lambda x. H (\text{ereal } (\text{real\_of\_ereal } (f\ x))) (-\infty)) \in \text{borel\_measurable } M$
 $(\lambda x. H (\text{ereal } (\text{real\_of\_ereal } (f\ x))) (\infty)) \in \text{borel\_measurable } M$
shows $(\lambda x. H (f\ x) (g\ x)) \in \text{borel\_measurable } M$

proof –

let $?G = \lambda y\ x. \text{if } g\ x = \infty \text{ then } H\ y\ \infty \text{ else if } g\ x = -\infty \text{ then } H\ y\ (-\infty) \text{ else } H\ y\ (\text{ereal } (\text{real\_of\_ereal } (g\ x)))$

let $?F = \lambda x. \text{if } f\ x = \infty \text{ then } ?G\ \infty\ x \text{ else if } f\ x = -\infty \text{ then } ?G\ (-\infty)\ x \text{ else } ?G\ (\text{ereal } (\text{real\_of\_ereal } (f\ x)))\ x$

{ fix } x **have $H\ (f\ x)\ (g\ x) = ?F\ x$ **by** (cases $f\ x\ g\ x$ rule: ereal2_cases) **auto** }**

note $* = \text{this}$

from *assms* **show** $?thesis$ **unfolding** $*$ **by** *simp*

qed

lemma $[\text{measurable}(\text{raw})]$:

fixes $f :: 'a \Rightarrow \text{ereal}$

assumes $[\text{measurable}]: f \in \text{borel\_measurable } M\ g \in \text{borel\_measurable } M$

shows $\text{borel\_measurable\_ereal\_add}: (\lambda x. f\ x + g\ x) \in \text{borel\_measurable } M$

and $\text{borel\_measurable\_ereal\_times}: (\lambda x. f\ x * g\ x) \in \text{borel\_measurable } M$

by (*simp_all add: borel_measurable_ereal2*)

lemma $[\text{measurable}(\text{raw})]$:

fixes $f\ g :: 'a \Rightarrow \text{ereal}$

assumes $f \in \text{borel\_measurable } M$

assumes $g \in \text{borel\_measurable } M$

shows $\text{borel\_measurable\_ereal\_diff}: (\lambda x. f\ x - g\ x) \in \text{borel\_measurable } M$

and $\text{borel\_measurable\_ereal\_divide}: (\lambda x. f\ x / g\ x) \in \text{borel\_measurable } M$

using *assms* **by** (*simp_all add: minus_ereal_def divide_ereal_def*)

lemma $\text{borel\_measurable\_ereal\_sum}[\text{measurable } (\text{raw})]$:

fixes $f :: 'c \Rightarrow 'a \Rightarrow \text{ereal}$

assumes $\bigwedge i. i \in S \implies f\ i \in \text{borel\_measurable } M$

shows $(\lambda x. \sum_{i \in S. f\ i\ x}) \in \text{borel\_measurable } M$

using *assms* by (induction *S* rule: *infinite_finite_induct*) auto

lemma *borel_measurable_ereal_prod*[*measurable (raw)*]:
 fixes *f* :: 'c $\Rightarrow$ 'a $\Rightarrow$ *ereal*
 assumes $\bigwedge i. i \in S \implies f\ i \in \text{borel\_measurable } M$
 shows $(\lambda x. \prod_{i \in S}. f\ i\ x) \in \text{borel\_measurable } M$
 using *assms* by (induction *S* rule: *infinite_finite_induct*) auto

lemma *borel_measurable_extreal_suminf*[*measurable (raw)*]:
 fixes *f* :: nat $\Rightarrow$ 'a $\Rightarrow$ *ereal*
 assumes [*measurable*]: $\bigwedge i. f\ i \in \text{borel\_measurable } M$
 shows $(\lambda x. (\sum i. f\ i\ x)) \in \text{borel\_measurable } M$
 unfolding *suminf_def* *sums_def* [*abs_def*] *lim_def* [*symmetric*] by *simp*

6.5.7 Borel space on the extended non-negative reals

ennreal is a topological monoid, so no rules for plus are required, also all order statements are usually done on type classes.

lemma *measurable_enn2ereal*[*measurable*]: *enn2ereal* $\in$ *borel* $\rightarrow_M$ *borel*
 by (intro *borel_measurable_continuous_onI* *continuous_on_enn2ereal*)

lemma *measurable_e2ennreal*[*measurable*]: *e2ennreal* $\in$ *borel* $\rightarrow_M$ *borel*
 by (intro *borel_measurable_continuous_onI* *continuous_on_e2ennreal*)

lemma *borel_measurable_enn2real*[*measurable (raw)*]:
 $f \in M \rightarrow_M \text{borel} \implies (\lambda x. \text{enn2real } (f\ x)) \in M \rightarrow_M \text{borel}$
 unfolding *enn2real_def* [*abs_def*] by *measurable*

definition [*simp*]: *is_borel* *f* *M* $\longleftrightarrow f \in \text{borel\_measurable } M$

lemma *is_borel_transfer*[*transfer_rule*]: *rel_fun* (*rel_fun* (=) *pcr_ennreal*) (=)
is_borel *is_borel*

unfolding *is_borel_def* [*abs_def*]

proof (*safe intro!*: *rel_funI* *ext dest!*: *rel_fun_eq_pcr_ennreal* [*THEN iffD1*])

fix *f* and *M* :: 'a *measure*

show $f \in \text{borel\_measurable } M$ if $f: \text{enn2ereal} \circ f \in \text{borel\_measurable } M$

using *measurable_compose* [*OF f measurable_e2ennreal*] by *simp*

qed *simp*

context

includes *ennreal.lifting*

begin

lemma *measurable_ennreal*[*measurable*]: *ennreal* $\in$ *borel* $\rightarrow_M$ *borel*
 unfolding *is_borel_def* [*symmetric*]
 by *transfer simp*

lemma *borel_measurable_ennreal_iff*[*simp*]:
 assumes [*simp*]: $\bigwedge x. x \in \text{space } M \implies 0 \leq f\ x$

```

  shows  $(\lambda x. \text{ennreal } (f x)) \in M \rightarrow_M \text{borel} \longleftrightarrow f \in M \rightarrow_M \text{borel}$ 
proof safe
  assume  $(\lambda x. \text{ennreal } (f x)) \in M \rightarrow_M \text{borel}$ 
  then have  $(\lambda x. \text{enn2real } (\text{ennreal } (f x))) \in M \rightarrow_M \text{borel}$ 
    by measurable
  then show  $f \in M \rightarrow_M \text{borel}$ 
    by (rule measurable_cong[THEN iffD1, rotated]) auto
qed measurable

```

```

lemma borel_measurable_times_ennreal[measurable (raw)]:
  fixes  $f g :: 'a \Rightarrow \text{ennreal}$ 
  shows  $f \in M \rightarrow_M \text{borel} \implies g \in M \rightarrow_M \text{borel} \implies (\lambda x. f x * g x) \in M \rightarrow_M \text{borel}$ 
  unfolding is_borel_def[symmetric] by transfer simp

```

```

lemma borel_measurable_inverse_ennreal[measurable (raw)]:
  fixes  $f :: 'a \Rightarrow \text{ennreal}$ 
  shows  $f \in M \rightarrow_M \text{borel} \implies (\lambda x. \text{inverse } (f x)) \in M \rightarrow_M \text{borel}$ 
  unfolding is_borel_def[symmetric] by transfer simp

```

```

lemma borel_measurable_divide_ennreal[measurable (raw)]:
  fixes  $f :: 'a \Rightarrow \text{ennreal}$ 
  shows  $f \in M \rightarrow_M \text{borel} \implies g \in M \rightarrow_M \text{borel} \implies (\lambda x. f x / g x) \in M \rightarrow_M \text{borel}$ 
  unfolding divide_ennreal_def by simp

```

```

lemma borel_measurable_minus_ennreal[measurable (raw)]:
  fixes  $f :: 'a \Rightarrow \text{ennreal}$ 
  shows  $f \in M \rightarrow_M \text{borel} \implies g \in M \rightarrow_M \text{borel} \implies (\lambda x. f x - g x) \in M \rightarrow_M \text{borel}$ 
  unfolding is_borel_def[symmetric] by transfer simp

```

```

lemma borel_measurable_power_ennreal [measurable (raw)]:
  fixes  $f :: \_ \Rightarrow \text{ennreal}$ 
  assumes  $f: f \in \text{borel\_measurable } M$ 
  shows  $(\lambda x. (f x) ^ n) \in \text{borel\_measurable } M$ 
  by (induction n) (use f in auto)

```

```

lemma borel_measurable_prod_ennreal[measurable (raw)]:
  fixes  $f :: 'c \Rightarrow 'a \Rightarrow \text{ennreal}$ 
  assumes  $\bigwedge i. i \in S \implies f i \in \text{borel\_measurable } M$ 
  shows  $(\lambda x. \prod_{i \in S}. f i x) \in \text{borel\_measurable } M$ 
  using assms by (induction S rule: infinite_finite_induct) auto

```

end

```
hide_const (open) is_borel
```

6.5.8 LIMSEQ is borel measurable

lemma *borel_measurable_LIMSEQ_real*:

fixes $u :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$

assumes $u': \bigwedge x. x \in \text{space } M \implies (\lambda i. u \ i \ x) \longrightarrow u' \ x$

and $u: \bigwedge i. u \ i \in \text{borel\_measurable } M$

shows $u' \in \text{borel\_measurable } M$

proof –

have $\bigwedge x. x \in \text{space } M \implies \liminf (\lambda n. \text{ereal } (u \ n \ x)) = \text{ereal } (u' \ x)$

using u' **by** (*simp add: lim_imp_Liminf*)

moreover from u **have** $(\lambda x. \liminf (\lambda n. \text{ereal } (u \ n \ x))) \in \text{borel\_measurable } M$

by *auto*

ultimately show *?thesis* **by** (*simp cong: measurable_cong add: borel_measurable_ereal_iff*)

qed

lemma *borel_measurable_LIMSEQ_metric*:

fixes $f :: \text{nat} \Rightarrow 'a \Rightarrow 'b :: \text{metric\_space}$

assumes [*measurable*]: $\bigwedge i. f \ i \in \text{borel\_measurable } M$

assumes $\text{lim}: \bigwedge x. x \in \text{space } M \implies (\lambda i. f \ i \ x) \longrightarrow g \ x$

shows $g \in \text{borel\_measurable } M$

unfolding *borel_eq_closed*

proof (*safe intro!: measurable_measure_of*)

fix $A :: 'b \text{ set}$ **assume** *closed A*

have [*measurable*]: $(\lambda x. \text{infdist } (g \ x) \ A) \in \text{borel\_measurable } M$

proof (*rule borel_measurable_LIMSEQ_real*)

show $\bigwedge x. x \in \text{space } M \implies (\lambda i. \text{infdist } (f \ i \ x) \ A) \longrightarrow \text{infdist } (g \ x) \ A$

by (*intro tendsto_infdist lim*)

show $\bigwedge i. (\lambda x. \text{infdist } (f \ i \ x) \ A) \in \text{borel\_measurable } M$

by (*intro borel_measurable_continuous_on[where f= $\lambda x. \text{infdist } x \ A$]*)

continuous_at_imp_continuous_on ballI continuous_infdist continuous_ident)

auto

qed

show $g - 'A \cap \text{space } M \in \text{sets } M$

proof *cases*

assume $A \neq \{\}$

then have $\bigwedge x. \text{infdist } x \ A = 0 \longleftrightarrow x \in A$

using $\langle \text{closed } A \rangle$ **by** (*simp add: in_closed_iff_infdist_zero*)

then have $g - 'A \cap \text{space } M = \{x \in \text{space } M. \text{infdist } (g \ x) \ A = 0\}$

by *auto*

also have $\dots \in \text{sets } M$

by *measurable*

finally show *?thesis* .

qed *simp*

qed *auto*

lemma *sets_Collect_Cauchy*[*measurable*]:

fixes $f :: \text{nat} \Rightarrow 'a \Rightarrow 'b :: \{\text{metric\_space}, \text{second\_countable\_topology}\}$

assumes f [*measurable*]: $\bigwedge i. f \ i \in \text{borel\_measurable } M$

shows $\{x \in \text{space } M. \text{Cauchy } (\lambda i. f \ i \ x)\} \in \text{sets } M$
unfolding *metric_Cauchy_iff2* **using** *f* **by** *auto*

lemma *borel_measurable_lim_metric*[*measurable (raw)*]:
fixes *f* :: *nat* $\Rightarrow$ '*a* $\Rightarrow$ '*b*::{*banach*, *second_countable_topology*}
assumes *f*[*measurable*]: $\bigwedge i. f \ i \in \text{borel\_measurable } M$
shows $(\lambda x. \lim (\lambda i. f \ i \ x)) \in \text{borel\_measurable } M$
proof –
define *u'* **where** *u' x* = $\lim (\lambda i. \text{if } \text{Cauchy } (\lambda i. f \ i \ x) \text{ then } f \ i \ x \text{ else } 0)$ **for** *x*
then have *: $\bigwedge x. \lim (\lambda i. f \ i \ x) = (\text{if } \text{Cauchy } (\lambda i. f \ i \ x) \text{ then } u' \ x \text{ else } (\text{THE } x. \text{False}))$
by (*auto simp: lim_def convergent_eq_Cauchy[symmetric]*)
have *u' ∈ borel_measurable M*
proof (*rule borel_measurable_LIMSEQ_metric*)
fix *x*
have *convergent* $(\lambda i. \text{if } \text{Cauchy } (\lambda i. f \ i \ x) \text{ then } f \ i \ x \text{ else } 0)$
by (*cases Cauchy* $(\lambda i. f \ i \ x)$)
(auto simp add: convergent_eq_Cauchy[symmetric] convergent_def)
then show $(\lambda i. \text{if } \text{Cauchy } (\lambda i. f \ i \ x) \text{ then } f \ i \ x \text{ else } 0) \longrightarrow u' \ x$
unfolding *u'_def*
by (*rule convergent_LIMSEQ_iff[THEN iffD1]*)
qed *measurable*
then show *?thesis*
unfolding * **by** *measurable*
qed

lemma *borel_measurable_suminf*[*measurable (raw)*]:
fixes *f* :: *nat* $\Rightarrow$ '*a* $\Rightarrow$ '*b*::{*banach*, *second_countable_topology*}
assumes *f*[*measurable*]: $\bigwedge i. f \ i \in \text{borel\_measurable } M$
shows $(\lambda x. \text{suminf } (\lambda i. f \ i \ x)) \in \text{borel\_measurable } M$
unfolding *suminf_def sums_def[abs_def] lim_def[symmetric]* **by** *simp*

lemma *Collect_closed_imp_pred_borel*: $\text{closed } \{x. P \ x\} \Longrightarrow \text{Measurable.pred borel } P$
by (*simp add: pred_def*)

lemma *isCont_borel_pred*[*measurable*]:
fixes *f* :: '*b*::*metric_space* $\Rightarrow$ '*a*::*metric_space*
shows *Measurable.pred borel (isCont f)*
proof (*subst measurable_cong*)
let *?I* = $\lambda j. \text{inverse}(\text{real } (\text{Suc } j))$
show *isCont f x* = $(\forall i. \exists j. \forall y \ z. \text{dist } x \ y < ?I \ j \wedge \text{dist } x \ z < ?I \ j \longrightarrow \text{dist } (f \ y) (f \ z) \leq ?I \ i)$ **for** *x*
unfolding *continuous_at_eps_delta*
proof *safe*
fix *i* **assume** $\forall e > 0. \exists d > 0. \forall y. \text{dist } y \ x < d \longrightarrow \text{dist } (f \ y) (f \ x) < e$
moreover have $0 < ?I \ i / 2$
by *simp*


```

ultimately obtain d where d: 0 < d  $\wedge$   $\forall y. \text{dist } x \ y < d \implies \text{dist } (f \ y) \ (f \ x) <$ 
?I i / 2
  by (metis dist_commute)
then obtain j where j: ?I j < d
  by (metis reals_Archimedean)

show  $\exists j. \forall y \ z. \text{dist } x \ y < ?I \ j \wedge \text{dist } x \ z < ?I \ j \longrightarrow \text{dist } (f \ y) \ (f \ z) \leq ?I \ i$ 
proof (safe intro!: exI[where x=j])
  fix y z assume *:  $\text{dist } x \ y < ?I \ j \wedge \text{dist } x \ z < ?I \ j$ 
  have  $\text{dist } (f \ y) \ (f \ z) \leq \text{dist } (f \ y) \ (f \ x) + \text{dist } (f \ z) \ (f \ x)$ 
    by (rule dist_triangle2)
  also have  $\dots < ?I \ i / 2 + ?I \ i / 2$ 
    by (intro add_strict_mono d less_trans[OF _ j] *)
  also have  $\dots \leq ?I \ i$ 
    by (simp add: field_simps)
  finally show  $\text{dist } (f \ y) \ (f \ z) \leq ?I \ i$ 
    by simp
qed
next
fix e::real assume 0 < e
then obtain n where n: ?I n < e
  by (metis reals_Archimedean)
assume  $\forall i. \exists j. \forall y \ z. \text{dist } x \ y < ?I \ j \wedge \text{dist } x \ z < ?I \ j \longrightarrow \text{dist } (f \ y) \ (f \ z) \leq$ 
?I i
  from this[THEN spec, of Suc n]
  obtain j where j:  $\bigwedge y \ z. \text{dist } x \ y < ?I \ j \implies \text{dist } x \ z < ?I \ j \implies \text{dist } (f \ y) \ (f$ 
z)  $\leq ?I \ (Suc \ n)$ 
    by auto

show  $\exists d > 0. \forall y. \text{dist } y \ x < d \longrightarrow \text{dist } (f \ y) \ (f \ x) < e$ 
proof (safe intro!: exI[of _ ?I j])
  fix y assume  $\text{dist } y \ x < ?I \ j$ 
  then have  $\text{dist } (f \ y) \ (f \ x) \leq ?I \ (Suc \ n)$ 
    by (intro j) (auto simp: dist_commute)
  also have  $?I \ (Suc \ n) < ?I \ n$ 
    by simp
  also note n
  finally show  $\text{dist } (f \ y) \ (f \ x) < e$  .
qed simp
qed
qed (intro pred_intros_countable closed_Collect_all closed_Collect_le open_Collect_less
      Collect_closed_imp_pred_borel closed_Collect_imp open_Collect_conj
      continuous_intros)

lemma isCont_borel:
  fixes f :: 'b::metric_space  $\Rightarrow$  'a::metric_space
  shows  $\{x. \text{isCont } f \ x\} \in \text{sets borel}$ 
  by simp

```

```

lemma is_real_interval:
  assumes S: is_interval S
  shows  $\exists a b :: \text{real}. S = \{\}$   $\vee S = \text{UNIV}$   $\vee S = \{..<b\}$   $\vee S = \{..b\}$   $\vee S = \{a<..\}$ 
 $\vee S = \{a.. \}$   $\vee$ 
 $S = \{a<..b\}$   $\vee S = \{a<..b\}$   $\vee S = \{a..b\}$   $\vee S = \{a..b\}$ 
  using S unfolding is_interval_1 by (blast intro: interval_cases)

```

```

lemma real_interval_borel_measurable:
  assumes is_interval (S::real set)
  shows S  $\in$  sets borel
proof -
  from assms is_real_interval have  $\exists a b :: \text{real}. S = \{\}$   $\vee S = \text{UNIV}$   $\vee S = \{..<b\}$ 
 $\vee S = \{..b\}$   $\vee$ 
 $S = \{a<..\}$   $\vee S = \{a.. \}$   $\vee S = \{a<..b\}$   $\vee S = \{a<..b\}$   $\vee S = \{a..b\}$   $\vee S$ 
 $= \{a..b\}$  by auto
  then show ?thesis
    by auto
qed

```

The next lemmas hold in any second countable linorder (including ennreal or ereal for instance), but in the current state they are restricted to reals.

```

lemma borel_measurable_mono_on_fnc:
  fixes f :: real  $\Rightarrow$  real and A :: real set
  assumes mono_on f A
  shows f  $\in$  borel_measurable (restrict_space borel A)
  apply (rule measurable_restrict_countable[OF mono_on_ctble_discont[OF assms]])
  apply (auto intro!: image_eqI[where x= $\{x\}$  for x] simp: sets_restrict_space)
  apply (auto simp add: sets_restrict_restrict_space continuous_on_eq_continuous_within
    cong: measurable_cong_sets
    intro!: borel_measurable_continuous_on_restrict intro: continuous_within_subset)
  done

```

```

lemma borel_measurable_piecewise_mono:
  fixes f::real  $\Rightarrow$  real and C::real set set
  assumes countable C  $\wedge c. c \in C \Longrightarrow c \in \text{sets borel}$   $\wedge c. c \in C \Longrightarrow \text{mono\_on } f$ 
 $c (\bigcup C) = \text{UNIV}$ 
  shows f  $\in$  borel_measurable borel
  by (rule measurable_piecewise_restrict[of C], auto intro: borel_measurable_mono_on_fnc
simp: assms)

```

```

lemma borel_measurable_mono:
  fixes f :: real  $\Rightarrow$  real
  shows mono f  $\Longrightarrow f \in$  borel_measurable borel
  using borel_measurable_mono_on_fnc[of f UNIV] by (simp add: mono_def
mono_on_def)

```

```

lemma measurable_bdd_below_real[measurable (raw)]:
  fixes F :: 'a  $\Rightarrow$  'i  $\Rightarrow$  real

```

```

assumes [simp]: countable I and [measurable]:  $\bigwedge i. i \in I \implies F\ i \in M \rightarrow_M \text{borel}$ 
shows Measurable.pred M ( $\lambda x. \text{bdd\_below } ((\lambda i. F\ i\ x)'I)$ )
proof (subst measurable_cong)
  show  $\text{bdd\_below } ((\lambda i. F\ i\ x)'I) \longleftrightarrow (\exists q \in \mathbb{Z}. \forall i \in I. q \leq F\ i\ x) \text{ for } x$ 
  by (auto simp: bdd_below_def intro!: bexI[of _ of_int (floor _)] intro: order_trans of_int_floor_le)
  show Measurable.pred M ( $\lambda w. \exists q \in \mathbb{Z}. \forall i \in I. q \leq F\ i\ w$ )
  using countable_int by measurable
qed

lemma borel_measurable_cINF_real[measurable (raw)]:
  fixes F ::  $\_ \Rightarrow \_ \Rightarrow \text{real}$ 
  assumes [simp]: countable I
  assumes F[measurable]:  $\bigwedge i. i \in I \implies F\ i \in \text{borel\_measurable } M$ 
  shows ( $\lambda x. \text{INF } i \in I. F\ i\ x$ )  $\in \text{borel\_measurable } M$ 
proof (rule measurable_piecewise_restrict)
  let ? $\Omega$  = { $x \in \text{space } M. \text{bdd\_below } ((\lambda i. F\ i\ x)'I)$ }
  show countable {? $\Omega$ , - ? $\Omega$ }  $\text{space } M \subseteq \bigcup \{?\Omega, - ?\Omega\} \wedge X. X \in \{?\Omega, - ?\Omega\}$ 
 $\implies X \cap \text{space } M \in \text{sets } M$ 
  by auto
  fix X assume X  $\in \{?\Omega, - ?\Omega\}$  then show ( $\lambda x. \text{INF } i \in I. F\ i\ x$ )  $\in \text{borel\_measurable}$ 
    (restrict_space M X)
  proof safe
    show ( $\lambda x. \text{INF } i \in I. F\ i\ x$ )  $\in \text{borel\_measurable } (\text{restrict\_space } M\ ?\Omega)$ 
    by (intro borel_measurable_cINF_measurable_restrict_space1 F)
      (auto simp: space_restrict_space)
    show ( $\lambda x. \text{INF } i \in I. F\ i\ x$ )  $\in \text{borel\_measurable } (\text{restrict\_space } M\ (-?\Omega))$ 
  proof (subst measurable_cong)
    fix x assume x  $\in \text{space } (\text{restrict\_space } M\ (-?\Omega))$ 
    then have  $\neg (\forall i \in I. - F\ i\ x \leq y)$  for y
    by (auto simp: space_restrict_space bdd_above_def bdd_above_uminus[symmetric])
    then show ( $\text{INF } i \in I. F\ i\ x$ ) = - (THE x. False)
    by (auto simp: space_restrict_space Inf_real_def Sup_real_def Least_def
      simp del: Set.ball_simps(10))
    qed simp
  qed
qed

```

```

lemma borel_Ici: borel = sigma UNIV (range ( $\lambda x::\text{real}. \{x..\}$ ))
proof (safe intro!: borel_eq_sigmaI1[OF borel_Iio])
  fix x :: real
  have eq:  $\{..\leq x\} = \text{space } (\text{sigma UNIV } (\text{range atLeast})) - \{x..\}$ 
  by auto
  show  $\{..\leq x\} \in \text{sets } (\text{sigma UNIV } (\text{range atLeast}))$ 
  unfolding eq by (intro sets.compl_sets) auto
qed auto

```

```

lemma borel_measurable_pred_less[measurable (raw)]:
  fixes f :: 'a  $\Rightarrow$  'b::{second_countable_topology, linorder_topology}

```

shows $f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies \text{Measurable.pred } M (\lambda w. f w < g w)$

unfolding $\text{Measurable.pred\_def}$ **by** (rule $\text{borel\_measurable\_less}$)

no_notation

eucl_less (**infix** $<e$ 50)

lemma $\text{borel\_measurable\_Max2}[\text{measurable } (raw)]:$

fixes $f::\_ \Rightarrow \_ \Rightarrow 'a::\{\text{second\_countable\_topology, dense\_linorder, linorder\_topology}\}$

assumes $\text{finite } I$

and $[\text{measurable}]: \bigwedge i. f i \in \text{borel\_measurable } M$

shows $(\lambda x. \text{Max}\{f i x \mid i. i \in I\}) \in \text{borel\_measurable } M$

by ($\text{simp add: borel\_measurable\_Max}[\text{OF assms}(1)]$, **where** $?f=f$ **and** $?M=M$)
 Setcompr_eq_image)

lemma $\text{measurable\_compose\_n}[\text{measurable } (raw)]:$

assumes $T \in \text{measurable } M M$

shows $(T^{\sim n}) \in \text{measurable } M M$

by ($\text{induction } n$, **auto simp add: measurable_compose}[\text{OF } _ \text{ assms}])**

lemma $\text{measurable\_real\_imp\_nat}:$

fixes $f::'a \Rightarrow \text{nat}$

assumes $[\text{measurable}]: (\lambda x. \text{real}(f x)) \in \text{borel\_measurable } M$

shows $f \in \text{measurable } M (\text{count\_space UNIV})$

proof –

let $?g = (\lambda x. \text{real}(f x))$

have $\bigwedge(n::\text{nat}). ?g - \{ \text{real } n \} \cap \text{space } M = f - \{ n \} \cap \text{space } M$ **by** auto

moreover have $\bigwedge(n::\text{nat}). ?g - \{ \text{real } n \} \cap \text{space } M \in \text{sets } M$ **using** assms **by**
 measurable

ultimately have $\bigwedge(n::\text{nat}). f - \{ n \} \cap \text{space } M \in \text{sets } M$ **by** simp

then show $?thesis$ **using** $\text{measurable\_count\_space\_eq2\_countable}$ **by** blast

qed

lemma $\text{measurable\_equality\_set}[\text{measurable}]:$

fixes $f g::\_ \Rightarrow 'a::\{\text{second\_countable\_topology, t2\_space}\}$

assumes $[\text{measurable}]: f \in \text{borel\_measurable } M g \in \text{borel\_measurable } M$

shows $\{x \in \text{space } M. f x = g x\} \in \text{sets } M$

proof –

define A **where** $A = \{x \in \text{space } M. f x = g x\}$

define B **where** $B = \{y. \exists x::'a. y = (x,x)\}$

have $A = (\lambda x. (f x, g x)) - B \cap \text{space } M$ **unfolding** $A\_def B\_def$ **by** auto

moreover have $(\lambda x. (f x, g x)) \in \text{borel\_measurable } M$ **by** simp

moreover have $B \in \text{sets borel}$ **unfolding** B_def **by** ($\text{simp add: closed\_diagonal}$)

ultimately have $A \in \text{sets } M$ **by** simp

then show $?thesis$ **unfolding** A_def **by** simp

qed

lemma $\text{measurable\_inequality\_set}[\text{measurable}]:$

```

fixes  $f g :: \_ \Rightarrow 'a :: \{second\_countable\_topology, linorder\_topology\}$ 
assumes  $[measurable]: f \in borel\_measurable\ M\ g \in borel\_measurable\ M$ 
shows  $\{x \in space\ M. f\ x \leq g\ x\} \in sets\ M$ 
 $\{x \in space\ M. f\ x < g\ x\} \in sets\ M$ 
 $\{x \in space\ M. f\ x \geq g\ x\} \in sets\ M$ 
 $\{x \in space\ M. f\ x > g\ x\} \in sets\ M$ 
proof –
  define  $F$  where  $F = (\lambda x. (f\ x, g\ x))$ 
  have  $* [measurable]: F \in borel\_measurable\ M$  unfolding  $F\_def$  by simp

  have  $\{x \in space\ M. f\ x \leq g\ x\} = F - \{(x, y) \mid x\ y. x \leq y\} \cap space\ M$  unfolding
 $F\_def$  by auto
  moreover have  $\{(x, y) \mid x\ y. x \leq (y :: 'a)\} \in sets\ borel$  using closed_subdiagonal
 $borel\_closed$  by blast
  ultimately show  $\{x \in space\ M. f\ x \leq g\ x\} \in sets\ M$  using  $*$  by (metis
 $(mono\_tags, lifting)\ measurable\_sets$ )

  have  $\{x \in space\ M. f\ x < g\ x\} = F - \{(x, y) \mid x\ y. x < y\} \cap space\ M$  unfolding
 $F\_def$  by auto
  moreover have  $\{(x, y) \mid x\ y. x < (y :: 'a)\} \in sets\ borel$  using open_subdiagonal
 $borel\_open$  by blast
  ultimately show  $\{x \in space\ M. f\ x < g\ x\} \in sets\ M$  using  $*$  by (metis
 $(mono\_tags, lifting)\ measurable\_sets$ )

  have  $\{x \in space\ M. f\ x \geq g\ x\} = F - \{(x, y) \mid x\ y. x \geq y\} \cap space\ M$  unfolding
 $F\_def$  by auto
  moreover have  $\{(x, y) \mid x\ y. x \geq (y :: 'a)\} \in sets\ borel$  using closed_superdiagonal
 $borel\_closed$  by blast
  ultimately show  $\{x \in space\ M. f\ x \geq g\ x\} \in sets\ M$  using  $*$  by (metis
 $(mono\_tags, lifting)\ measurable\_sets$ )

  have  $\{x \in space\ M. f\ x > g\ x\} = F - \{(x, y) \mid x\ y. x > y\} \cap space\ M$  unfolding
 $F\_def$  by auto
  moreover have  $\{(x, y) \mid x\ y. x > (y :: 'a)\} \in sets\ borel$  using open_superdiagonal
 $borel\_open$  by blast
  ultimately show  $\{x \in space\ M. f\ x > g\ x\} \in sets\ M$  using  $*$  by (metis
 $(mono\_tags, lifting)\ measurable\_sets$ )
qed

```

proposition *measurable_limit* $[measurable]:$

```

fixes  $f :: nat \Rightarrow 'a \Rightarrow 'b :: first\_countable\_topology$ 
assumes  $[measurable]: \bigwedge n :: nat. f\ n \in borel\_measurable\ M$ 
shows  $Measurable.pred\ M\ (\lambda x. (\lambda n. f\ n\ x) \longrightarrow c)$ 

```

proof –

obtain $A :: nat \Rightarrow 'b$ **set** **where** $A:$

$\bigwedge i. open\ (A\ i)$

$\bigwedge i. c \in A\ i$

$\bigwedge S. open\ S \implies c \in S \implies eventually\ (\lambda i. A\ i \subseteq S)\ sequentially$

by (*rule countable_basis_at_decseq*) *blast*

have [measurable]: $\bigwedge N i. (f N) - (A i) \cap \text{space } M \in \text{sets } M$ using $A(1)$ by auto
 then have mes: $(\bigcap i. \bigcup n. \bigcap N \in \{n..\}. (f N) - (A i) \cap \text{space } M) \in \text{sets } M$ by
 blast

have $(u \longrightarrow c) \longleftrightarrow (\forall i. \text{eventually } (\lambda n. u n \in A i) \text{ sequentially})$ for $u::\text{nat} \Rightarrow 'b$

proof

assume $u \longrightarrow c$

then have eventually $(\lambda n. u n \in A i)$ sequentially for i using $A(1)[\text{of } i]$
 $A(2)[\text{of } i]$

by (simp add: topological_tendstoD)

then show $(\forall i. \text{eventually } (\lambda n. u n \in A i) \text{ sequentially})$ by auto

next

assume $H: (\forall i. \text{eventually } (\lambda n. u n \in A i) \text{ sequentially})$

show $(u \longrightarrow c)$

proof (rule topological_tendstoI)

fix S assume open S $c \in S$

with $A(3)[\text{OF this}]$ obtain i where $A i \subseteq S$

using eventually_False_sequentially eventually_mono by blast

moreover have eventually $(\lambda n. u n \in A i)$ sequentially using H by simp

ultimately show $\forall_F n$ in sequentially. $u n \in S$

by (simp add: eventually_mono subset_eq)

qed

qed

then have $\{x. (\lambda n. f n x) \longrightarrow c\} = (\bigcap i. \bigcup n. \bigcap N \in \{n..\}. (f N) - (A i))$

by (auto simp add: atLeast_def eventually_at_top_linorder)

then have $\{x \in \text{space } M. (\lambda n. f n x) \longrightarrow c\} = (\bigcap i. \bigcup n. \bigcap N \in \{n..\}. (f N) - (A i) \cap \text{space } M)$

by auto

then have $\{x \in \text{space } M. (\lambda n. f n x) \longrightarrow c\} \in \text{sets } M$ using mes by simp

then show ?thesis by auto

qed

lemma measurable_limit2 [measurable]:

fixes $u::\text{nat} \Rightarrow 'a \Rightarrow \text{real}$

assumes [measurable]: $\bigwedge n. u n \in \text{borel\_measurable } M$ $v \in \text{borel\_measurable } M$

shows $\text{Measurable.pred } M (\lambda x. (\lambda n. u n x) \longrightarrow v x)$

proof -

define w where $w = (\lambda n x. u n x - v x)$

have [measurable]: $w n \in \text{borel\_measurable } M$ for n unfolding w_def by auto

have $((\lambda n. u n x) \longrightarrow v x) \longleftrightarrow ((\lambda n. w n x) \longrightarrow 0)$ for x

unfolding w_def using Lim_null by auto

then show ?thesis using measurable_limit by auto

qed

lemma measurable_P_restriction [measurable (raw)]:

assumes [measurable]: $\text{Measurable.pred } M P$ $A \in \text{sets } M$

shows $\{x \in A. P x\} \in \text{sets } M$

```

proof –
  have  $A \subseteq \text{space } M$  using sets.sets_into_space[OF assms(2)].
  then have  $\{x \in A. P\ x\} = A \cap \{x \in \text{space } M. P\ x\}$  by blast
  then show ?thesis by auto
qed

```

```

lemma measurable_sum_nat [measurable (raw)]:
  fixes  $f :: 'c \Rightarrow 'a \Rightarrow \text{nat}$ 
  assumes  $\bigwedge i. i \in S \implies f\ i \in \text{measurable } M\ (\text{count\_space } UNIV)$ 
  shows  $(\lambda x. \sum_{i \in S}. f\ i\ x) \in \text{measurable } M\ (\text{count\_space } UNIV)$ 
proof cases
  assume finite S
  then show ?thesis using assms by induct auto
qed simp

```

```

lemma measurable_abs_powr [measurable]:
  fixes  $p :: \text{real}$ 
  assumes [measurable]:  $f \in \text{borel\_measurable } M$ 
  shows  $(\lambda x. |f\ x|^{\text{powr } p}) \in \text{borel\_measurable } M$ 
  by simp

```

The next one is a variation around *measurable_restrict_space*.

```

lemma measurable_restrict_space3:
  assumes  $f \in \text{measurable } M\ N$  and
     $f \in A \rightarrow B$ 
  shows  $f \in \text{measurable } (\text{restrict\_space } M\ A)\ (\text{restrict\_space } N\ B)$ 
proof –
  have  $f \in \text{measurable } (\text{restrict\_space } M\ A)\ N$  using assms(1) measurable_restrict_space1
by auto
  then show ?thesis by (metis Int_iff funcsetI funcset_mem
    measurable_restrict_space2[of f, of restrict_space M A, of B, of N] assms(2)
    space_restrict_space)
qed

```

```

lemma measurable_restrict_mono:
  assumes  $f: f \in \text{restrict\_space } M\ A \rightarrow_M N$  and  $B \subseteq A$ 
  shows  $f \in \text{restrict\_space } M\ B \rightarrow_M N$ 
by (rule measurable_compose[OF measurable_restrict_space3 f])
  (insert  $\langle B \subseteq A \rangle$ , auto)

```

The next one is a variation around *measurable_piecewise_restrict*.

```

lemma measurable_piecewise_restrict2:
  assumes [measurable]:  $\bigwedge n. A\ n \in \text{sets } M$ 
  and  $\text{space } M = (\bigcup (n :: \text{nat}). A\ n)$ 
   $\bigwedge n. \exists h \in \text{measurable } M\ N. (\forall x \in A\ n. f\ x = h\ x)$ 
  shows  $f \in \text{measurable } M\ N$ 
proof (rule measurableI)
  fix  $B$  assume [measurable]:  $B \in \text{sets } N$ 

```

```

{
  fix n::nat
  obtain h where [measurable]: h ∈ measurable M N and ∀ x ∈ A n. f x = h x
using assms(3) by blast
  then have *: f-‘B ∩ A n = h-‘B ∩ A n by auto
  have h-‘B ∩ A n = h-‘B ∩ space M ∩ A n using assms(2) sets.sets_into_space
by auto
  then have h-‘B ∩ A n ∈ sets M by simp
  then have f-‘B ∩ A n ∈ sets M using * by simp
}
then have (⋃ n. f-‘B ∩ A n) ∈ sets M by measurable
moreover have f-‘B ∩ space M = (⋃ n. f-‘B ∩ A n) using assms(2) by blast
ultimately show f-‘B ∩ space M ∈ sets M by simp
next
fix x assume x ∈ space M
then obtain n where x ∈ A n using assms(2) by blast
obtain h where [measurable]: h ∈ measurable M N and ∀ x ∈ A n. f x = h x
using assms(3) by blast
then have f x = h x using ⟨x ∈ A n⟩ by blast
moreover have h x ∈ space N by (metis measurable_space ⟨x ∈ space M⟩ ⟨h ∈
measurable M N⟩)
ultimately show f x ∈ space N by simp
qed
end

```

6.6 Lebesgue Integration for Nonnegative Functions

```

theory Nonnegative_Lebesgue_Integration
imports Measure_Space Borel_Space
begin

```

6.6.1 Approximating functions

```

lemma AE_upper_bound_inf_enreal:
  fixes F G::'a ⇒ ennreal
  assumes ⋀e. (e::real) > 0 ⇒ AE x in M. F x ≤ G x + e
  shows AE x in M. F x ≤ G x
proof -
  have AE x in M. ∀ n::nat. F x ≤ G x + ennreal (1 / Suc n)
  using assms by (auto simp: AE_all_countable)
  then show ?thesis
proof (eventually_elim)
  fix x assume x: ∀ n::nat. F x ≤ G x + ennreal (1 / Suc n)
  show F x ≤ G x
proof (rule ennreal_le_epsilon)
  fix e :: real assume 0 < e

```



```

    then obtain n where n: 1 / Suc n < e
    by (blast elim: nat_approx_posE)
    have F x ≤ G x + 1 / Suc n
    using x by simp
    also have ... ≤ G x + e
    using n by (intro add_mono ennreal_leI) auto
    finally show F x ≤ G x + ennreal e .
  qed
qed
qed

lemma AE_upper_bound_inf:
  fixes F G::'a ⇒ real
  assumes ⋀e. e > 0 ⇒ AE x in M. F x ≤ G x + e
  shows AE x in M. F x ≤ G x
proof -
  have AE x in M. F x ≤ G x + 1/real (n+1) for n::nat
  by (rule assms, auto)
  then have AE x in M. ∀ n::nat ∈ UNIV. F x ≤ G x + 1/real (n+1)
  by (rule AE_ball_countable', auto)
  moreover
  {
    fix x assume i: ∀ n::nat ∈ UNIV. F x ≤ G x + 1/real (n+1)
    have (λn. G x + 1/real (n+1)) ⟶ G x + 0
    by (rule tendsto_add, simp, rule LIMSEQ_ignore_initial_segment[OF lim_1_over_n,
of 1])
    then have F x ≤ G x using i LIMSEQ_le_const by fastforce
  }
  ultimately show ?thesis by auto
qed

lemma not_AE_zero_ennreal_E:
  fixes f::'a ⇒ ennreal
  assumes ¬ (AE x in M. f x = 0) and [measurable]: f ∈ borel_measurable M
  shows ∃ A ∈ sets M. ∃ e::real > 0. emeasure M A > 0 ∧ (∀ x ∈ A. f x ≥ e)
proof -
  { assume ¬ (∃ e::real > 0. {x ∈ space M. f x ≥ e} ∉ null_sets M)
    then have 0 < e ⇒ AE x in M. f x ≤ e for e::real
    by (auto simp: not_le less_imp_le dest!: AE_not_in)
    then have AE x in M. f x ≤ 0
    by (intro AE_upper_bound_inf_ennreal[where G=λ_. 0]) simp
    then have False
    using assms by auto }
  then obtain e::real where e: e > 0 {x ∈ space M. f x ≥ e} ∉ null_sets M by
auto
  define A where A = {x ∈ space M. f x ≥ e}
  have 1 [measurable]: A ∈ sets M unfolding A_def by auto
  have 2: emeasure M A > 0
  using e(2) A_def ⟨A ∈ sets M⟩ by auto

```

```

    have 3:  $\bigwedge x. x \in A \implies f\ x \geq e$  unfolding  $A\_def$  by auto
    show ?thesis using  $e(1)$  1 2 3 by blast
qed

lemma not_AE_zero_E:
  fixes  $f::'a \Rightarrow real$ 
  assumes  $AE\ x\ in\ M. f\ x \geq 0$ 
     $\neg(AE\ x\ in\ M. f\ x = 0)$ 
    and [measurable]:  $f \in borel\_measurable\ M$ 
  shows  $\exists A\ e. A \in sets\ M \wedge e > 0 \wedge emeasure\ M\ A > 0 \wedge (\forall x \in A. f\ x \geq e)$ 
proof -
  have  $\exists e. e > 0 \wedge \{x \in space\ M. f\ x \geq e\} \notin null\_sets\ M$ 
  proof (rule ccontr)
    assume *:  $\neg(\exists e. e > 0 \wedge \{x \in space\ M. f\ x \geq e\} \notin null\_sets\ M)$ 
    {
      fix  $e::real$  assume  $e > 0$ 
      then have  $\{x \in space\ M. f\ x \geq e\} \in null\_sets\ M$  using * by blast
      then have  $AE\ x\ in\ M. x \notin \{x \in space\ M. f\ x \geq e\}$  using  $AE\_not\_in$  by
        blast
      then have  $AE\ x\ in\ M. f\ x \leq e$  by auto
    }
    then have  $AE\ x\ in\ M. f\ x \leq 0$  by (rule AE_upper_bound_inf, auto)
    then have  $AE\ x\ in\ M. f\ x = 0$  using  $assms(1)$  by auto
    then show False using  $assms(2)$  by auto
  qed
  then obtain  $e$  where  $e: e > 0 \wedge \{x \in space\ M. f\ x \geq e\} \notin null\_sets\ M$  by auto
  define  $A$  where  $A = \{x \in space\ M. f\ x \geq e\}$ 
  have 1 [measurable]:  $A \in sets\ M$  unfolding  $A\_def$  by auto
  have 2:  $emeasure\ M\ A > 0$ 
    using  $e(2)$   $A\_def$   $\langle A \in sets\ M \rangle$  by auto
  have 3:  $\bigwedge x. x \in A \implies f\ x \geq e$  unfolding  $A\_def$  by auto
  show ?thesis
    using  $e(1)$  1 2 3 by blast
qed

```

6.6.2 Simple function

Our simple functions are not restricted to nonnegative real numbers. Instead they are just functions with a finite range and are measurable when singleton sets are measurable.

```

definition simple_function  $M\ g \longleftrightarrow$ 
   $finite\ (g\ 'space\ M) \wedge$ 
   $(\forall x \in g\ 'space\ M. g - \{x\} \cap space\ M \in sets\ M)$ 

```

```

lemma simple_functionD:
  assumes simple_function  $M\ g$ 
  shows  $finite\ (g\ 'space\ M)$  and  $g - X \cap space\ M \in sets\ M$ 
proof -
  show  $finite\ (g\ 'space\ M)$ 

```

```

    using assms unfolding simple_function_def by auto
  have  $g - ' X \cap \text{space } M = g - ' (X \cap g' \text{space } M) \cap \text{space } M$  by auto
  also have  $\dots = (\bigcup x \in X \cap g' \text{space } M. g - ' \{x\} \cap \text{space } M)$  by auto
  finally show  $g - ' X \cap \text{space } M \in \text{sets } M$  using assms
    by (auto simp del: UN_simps simp: simple_function_def)
qed

lemma measurable_simple_function[measurable_dest]:
  simple_function M f  $\implies f \in \text{measurable } M$  (count_space UNIV)
  unfolding simple_function_def measurable_def
proof safe
  fix A assume finite (f - ' space M)  $\forall x \in f - ' \text{space } M. f - ' \{x\} \cap \text{space } M \in \text{sets } M$ 
  then have  $(\bigcup x \in f - ' \text{space } M. \text{if } x \in A \text{ then } f - ' \{x\} \cap \text{space } M \text{ else } \{\}) \in \text{sets } M$ 
  by (intro sets.finite_UN) auto
  also have  $(\bigcup x \in f - ' \text{space } M. \text{if } x \in A \text{ then } f - ' \{x\} \cap \text{space } M \text{ else } \{\}) = f - ' A \cap \text{space } M$ 
    by (auto split: if_split_asm)
  finally show  $f - ' A \cap \text{space } M \in \text{sets } M$  .
qed simp

lemma borel_measurable_simple_function:
  simple_function M f  $\implies f \in \text{borel\_measurable } M$ 
  by (auto dest!: measurable_simple_function simp: measurable_def)

lemma simple_function_measurable2[intro]:
  assumes simple_function M f simple_function M g
  shows  $f - ' A \cap g - ' B \cap \text{space } M \in \text{sets } M$ 
proof -
  have  $f - ' A \cap g - ' B \cap \text{space } M = (f - ' A \cap \text{space } M) \cap (g - ' B \cap \text{space } M)$ 
    by auto
  then show ?thesis using assms[THEN simple_functionD(2)] by auto
qed

lemma simple_function_indicator_representation:
  fixes f :: 'a  $\Rightarrow$  ennreal
  assumes f: simple_function M f and x:  $x \in \text{space } M$ 
  shows  $f x = (\sum y \in f - ' \text{space } M. y * \text{indicator } (f - ' \{y\} \cap \text{space } M) x)$ 
    (is ?l = ?r)
proof -
  have  $?r = (\sum y \in f - ' \text{space } M. ( \text{if } y = f x \text{ then } y * \text{indicator } (f - ' \{y\} \cap \text{space } M) x \text{ else } 0 ))$ 
    by (auto intro!: sum.cong)
  also have  $\dots = f x * \text{indicator } (f - ' \{f x\} \cap \text{space } M) x$ 
    using assms by (auto dest: simple_functionD)
  also have  $\dots = f x$  using x by (auto simp: indicator_def)
  finally show ?thesis by auto
qed

```

lemma *simple_function_notspace*:
simple_function M $(\lambda x. h\ x * \text{indicator } (-\ \text{space } M)\ x :: \text{ennreal})$ **(is** *simple_function* M $?h$)
proof –
 have $?h\ ' \text{space } M \subseteq \{0\}$ **unfolding** *indicator_def* **by** *auto*
 hence $[simp, intro]: \text{finite } (?h\ ' \text{space } M)$ **by** $(\text{auto intro: finite\_subset})$
 have $?h\ -' \{0\} \cap \text{space } M = \text{space } M$ **by** *auto*
 thus $?thesis$ **unfolding** *simple_function_def* **by** $(\text{auto simp add: image\_constant\_conv})$
qed

lemma *simple_function_cong*:
assumes $\bigwedge t. t \in \text{space } M \implies f\ t = g\ t$
shows *simple_function* $M\ f \longleftrightarrow \text{simple\_function } M\ g$
proof –
 have $\bigwedge x. f\ -' \{x\} \cap \text{space } M = g\ -' \{x\} \cap \text{space } M$
 using *assms* **by** *auto*
 with *assms* **show** $?thesis$
 by $(\text{simp add: simple\_function\_def cong: image\_cong})$
qed

lemma *simple_function_cong_algebra*:
assumes $\text{sets } N = \text{sets } M\ \text{space } N = \text{space } M$
shows *simple_function* $M\ f \longleftrightarrow \text{simple\_function } N\ f$
unfolding *simple_function_def* *assms* ..

lemma *simple_function_borel_measurable*:
fixes $f :: 'a \Rightarrow 'x :: \{t2\_space\}$
assumes $f \in \text{borel\_measurable } M$ **and** *finite* $(f\ ' \text{space } M)$
shows *simple_function* $M\ f$
using *assms* **unfolding** *simple_function_def*
by $(\text{auto intro: borel\_measurable\_vimage})$

lemma *simple_function_iff_borel_measurable*:
fixes $f :: 'a \Rightarrow 'x :: \{t2\_space\}$
shows *simple_function* $M\ f \longleftrightarrow \text{finite } (f\ ' \text{space } M) \wedge f \in \text{borel\_measurable } M$
by $(\text{metis borel\_measurable\_simple\_function simple\_functionD(1) simple\_function\_borel\_measurable})$

lemma *simple_function_eq_measurable*:
simple_function $M\ f \longleftrightarrow \text{finite } (f\ ' \text{space } M) \wedge f \in \text{measurable } M\ (\text{count\_space } UNIV)$
using *measurable_simple_function* $[of\ M\ f]$ **by** $(\text{fastforce simp: simple\_function\_def})$

lemma *simple_function_const* $[intro, simp]$:
simple_function $M\ (\lambda x. c)$
by $(\text{auto intro: finite\_subset simp: simple\_function\_def})$
lemma *simple_function_compose* $[intro, simp]$:
assumes *simple_function* $M\ f$
shows *simple_function* $M\ (g \circ f)$
unfolding *simple_function_def*

```

proof safe
  show finite (( $g \circ f$ ) - ' space  $M$ )
    using assms unfolding simple_function_def image_comp [symmetric] by auto
next
  fix  $x$  assume  $x \in \text{space } M$ 
  let  $?G = g - ' \{g(f\ x)\} \cap (f' \text{space } M)$ 
  have *: ( $g \circ f$ ) - '  $\{(g \circ f)\ x\} \cap \text{space } M =$ 
    ( $\bigcup_{x \in ?G} f - ' \{x\} \cap \text{space } M$ ) by auto
  show ( $g \circ f$ ) - '  $\{(g \circ f)\ x\} \cap \text{space } M \in \text{sets } M$ 
    using assms unfolding simple_function_def *
    by (rule_tac sets.finite_UN) auto
qed

```

```

lemma simple_function_indicator[intro, simp]:
  assumes  $A \in \text{sets } M$ 
  shows simple_function  $M$  (indicator  $A$ )
proof -
  have indicator  $A - ' \text{space } M \subseteq \{0, 1\}$  (is  $?S \subseteq \_$ )
    by (auto simp: indicator_def)
  hence finite  $?S$  by (rule finite_subset) simp
  moreover have  $- A \cap \text{space } M = \text{space } M - A$  by auto
  ultimately show ?thesis unfolding simple_function_def
    using assms by (auto simp: indicator_def [abs_def])
qed

```

```

lemma simple_function_Pair[intro, simp]:
  assumes simple_function  $M$   $f$ 
  assumes simple_function  $M$   $g$ 
  shows simple_function  $M$  ( $\lambda x. (f\ x, g\ x)$ ) (is simple_function  $M$   $?p$ )
  unfolding simple_function_def
proof safe
  show finite ( $?p - ' \text{space } M$ )
    using assms unfolding simple_function_def
    by (rule_tac finite_subset[of  $\_ f' \text{space } M \times g' \text{space } M$ ]) auto
next
  fix  $x$  assume  $x \in \text{space } M$ 
  have ( $\lambda x. (f\ x, g\ x)$ ) - '  $\{(f\ x, g\ x)\} \cap \text{space } M =$ 
    ( $f - ' \{f\ x\} \cap \text{space } M$ )  $\cap (g - ' \{g\ x\} \cap \text{space } M)$ 
    by auto
  with  $\langle x \in \text{space } M \rangle$  show ( $\lambda x. (f\ x, g\ x)$ ) - '  $\{(f\ x, g\ x)\} \cap \text{space } M \in \text{sets } M$ 
    using assms unfolding simple_function_def by auto
qed

```

```

lemma simple_function_compose1:
  assumes simple_function  $M$   $f$ 
  shows simple_function  $M$  ( $\lambda x. g(f\ x)$ )
  using simple_function_compose[OF assms, of  $g$ ]
  by (simp add: comp_def)

```

```

lemma simple_function_compose2:
  assumes simple_function M f and simple_function M g
  shows simple_function M ( $\lambda x. h (f x) (g x)$ )
proof -
  have simple_function M ( $(\lambda(x, y). h x y) \circ (\lambda x. (f x, g x))$ )
    using assms by auto
  thus ?thesis by (simp_all add: comp_def)
qed

lemmas simple_function_add[intro, simp] = simple_function_compose2[where
h=(+)]
and simple_function_diff[intro, simp] = simple_function_compose2[where h=(-)]
and simple_function_uminus[intro, simp] = simple_function_compose2[where
g=uminus]
and simple_function_mult[intro, simp] = simple_function_compose2[where
h=(*)]
and simple_function_div[intro, simp] = simple_function_compose2[where h=(/)]
and simple_function_inverse[intro, simp] = simple_function_compose2[where
g=inverse]
and simple_function_max[intro, simp] = simple_function_compose2[where h=max]

lemma simple_function_sum[intro, simp]:
  assumes  $\bigwedge i. i \in P \implies \text{simple\_function } M (f i)$ 
  shows simple_function M ( $\lambda x. \sum_{i \in P} f i x$ )
proof cases
  assume finite P from this assms show ?thesis by induct auto
qed auto

lemma simple_function_enreal[intro, simp]:
  fixes f g :: 'a  $\Rightarrow$  real assumes sf: simple_function M f
  shows simple_function M ( $\lambda x. \text{enreal } (f x)$ )
  by (rule simple_function_compose1[OF sf])

lemma simple_function_real_of_nat[intro, simp]:
  fixes f g :: 'a  $\Rightarrow$  nat assumes sf: simple_function M f
  shows simple_function M ( $\lambda x. \text{real } (f x)$ )
  by (rule simple_function_compose1[OF sf])

lemma borel_measurable_implies_simple_function_sequence:
  fixes u :: 'a  $\Rightarrow$  enreal
  assumes u[measurable]:  $u \in \text{borel\_measurable } M$ 
  shows  $\exists f. \text{incseq } f \wedge (\forall i. (\forall x. f i x < \text{top}) \wedge \text{simple\_function } M (f i)) \wedge u =$ 
 $(\text{SUP } i. f i)$ 
proof -
  define f where [abs_def]:
     $f i x = \text{real\_of\_int } (\text{floor } (\text{enn2real } (\min i (u x)) * 2^i)) / 2^i \text{ for } i x$ 

  have [simp]:  $0 \leq f i x$  for i x
    by (auto simp: f_def intro!: divide_nonneg_nonneg mult_nonneg_nonneg

```

enn2real_nonneg)

have *: $2^n * \text{real\_of\_int } x = \text{real\_of\_int } (2^n * x)$ **for** n x
by *simp*

have $\text{real\_of\_int } \lfloor \text{real } i * 2^i \rfloor = \text{real\_of\_int } \lfloor i * 2^i \rfloor$ **for** i
by (*intro arg_cong[where f=real_of_int]*) *simp*
then have [*simp*]: $\text{real\_of\_int } \lfloor \text{real } i * 2^i \rfloor = i * 2^i$ **for** i
unfolding *floor_of_nat* **by** *simp*

have *incseq* f

proof (*intro monoI le_funI*)

fix $m n :: \text{nat}$ **and** x **assume** $m \leq n$

moreover

{ **fix** $d :: \text{nat}$

have $\lfloor 2^d :: \text{real} \rfloor * \lfloor 2^m * \text{enn2real } (\min (\text{of\_nat } m) (u \ x)) \rfloor \leq$
 $\lfloor 2^d * (2^m * \text{enn2real } (\min (\text{of\_nat } m) (u \ x))) \rfloor$

by (*rule le_mult_floor*) (*auto*)

also have ... $\leq \lfloor 2^d * (2^m * \text{enn2real } (\min (\text{of\_nat } d + \text{of\_nat } m) (u \ x))) \rfloor$

by (*intro floor_mono mult_mono enn2real_mono min.mono*)
(*auto simp: min_less_iff_disj of_nat_less_top*)

finally have $f \ m \ x \leq f \ (m + d) \ x$

unfolding *f_def*

by (*auto simp: field_simps power_add * simp del: of_int_mult*) }

ultimately show $f \ m \ x \leq f \ n \ x$

by (*auto simp add: le_iff_add*)

qed

then have *inc_f*: *incseq* $(\lambda i. \text{ennreal } (f \ i \ x))$ **for** x

by (*auto simp: incseq_def le_fun_def*)

then have *incseq* $(\lambda i \ x. \text{ennreal } (f \ i \ x))$

by (*auto simp: incseq_def le_fun_def*)

moreover

have *simple_function* $M \ (f \ i)$ **for** i

proof (*rule simple_function_borel_measurable*)

have $\lfloor \text{enn2real } (\min (\text{of\_nat } i) (u \ x)) * 2^i \rfloor \leq \lfloor \text{int } i * 2^i \rfloor$ **for** x

by (*cases u x rule: ennreal_cases*)

(*auto split: split_min intro!: floor_mono*)

then have $f \ i \text{ 'space } M \subseteq (\lambda n. \text{real\_of\_int } n / 2^i) \text{ ' } \{0 .. \text{of\_nat } i * 2^i\}$

unfolding *floor_of_int* **by** (*auto simp: f_def intro!: imageI*)

then show *finite* $(f \ i \text{ 'space } M)$

by (*rule finite_subset*) *auto*

show $f \ i \in \text{borel\_measurable } M$

unfolding *f_def* *enn2real_def* **by** *measurable*

qed

moreover

{ **fix** x

have $(\text{SUP } i. \text{ennreal } (f \ i \ x)) = u \ x$

proof (*cases u x rule: ennreal_cases*)

```

case top then show ?thesis
by (simp add: f_def inf_min[symmetric] ennreal_of_nat_eq_real_of_nat[symmetric]
      ennreal_SUP_of_nat_eq_top)
next
case (real r)
obtain n where  $r \leq \text{of\_nat } n$  using real_arch_simple by auto
then have min_eq_r:  $\forall_F x \text{ in sequentially. } \min(\text{real } x) \ r = r$ 
by (auto simp: eventually_sequentially intro!: exI[of _ n] split: split_min)

have ( $\lambda i. \text{real\_of\_int } \lfloor \min(\text{real } i) \ r * 2^i \rfloor / 2^i$ )  $\longrightarrow r$ 
proof (rule tendsto_sandwich)
show ( $\lambda n. r - (1/2)^n$ )  $\longrightarrow r$ 
by (auto intro!: tendsto_eq_intros LIMSEQ_power_zero)
show  $\forall_F n \text{ in sequentially. } \text{real\_of\_int } \lfloor \min(\text{real } n) \ r * 2^n \rfloor / 2^n \leq$ 
r
using min_eq_r by eventually_elim (auto simp: field_simps)
have *:  $r * (2^n * 2^n) \leq 2^n + 2^n * \text{real\_of\_int } \lfloor r * 2^n \rfloor$  for n
using real_of_int_floor_ge_diff_one[of r * 2^n, THEN mult_left_mono,
of 2^n]
by (auto simp: field_simps)
show  $\forall_F n \text{ in sequentially. } r - (1/2)^n \leq \text{real\_of\_int } \lfloor \min(\text{real } n) \ r * 2^n \rfloor / 2^n$ 
using min_eq_r by eventually_elim (insert *, auto simp: field_simps)
qed auto
then have ( $\lambda i. \text{ennreal } (f \ i \ x)$ )  $\longrightarrow \text{ennreal } r$ 
by (simp add: real_f_def ennreal_of_nat_eq_real_of_nat min_ennreal)
from LIMSEQ_unique[OF LIMSEQ_SUP[OF inc_f] this]
show ?thesis
by (simp add: real)
qed }
ultimately show ?thesis
by (intro exI [of _  $\lambda i \ x. \text{ennreal } (f \ i \ x)$ ]) (auto simp add: image_comp)
qed

lemma borel_measurable_implies_simple_function_sequence':
fixes u :: 'a  $\Rightarrow$  ennreal
assumes u:  $u \in \text{borel\_measurable } M$ 
obtains f where
 $\bigwedge i. \text{simple\_function } M \ (f \ i) \text{ incseq } f \ \bigwedge i \ x. f \ i \ x < \text{top} \ \bigwedge x. (\text{SUP } i. f \ i \ x) = u \ x$ 
using borel_measurable_implies_simple_function_sequence [OF u]
by (metis SUP_apply)

lemma simple_function_induct
[consumes 1, case_names cong set mult add, induct set: simple_function]:
fixes u :: 'a  $\Rightarrow$  ennreal
assumes u: simple_function M u
assumes cong:  $\bigwedge f \ g. \text{simple\_function } M \ f \ \Longrightarrow \text{simple\_function } M \ g \ \Longrightarrow (AE \ x$ 
in M.  $f \ x = g \ x) \ \Longrightarrow P \ f \ \Longrightarrow P \ g$ 
assumes set:  $\bigwedge A. A \in \text{sets } M \ \Longrightarrow P \ (\text{indicator } A)$ 

```



```

    assumes mult:  $\bigwedge u c. P u \implies P (\lambda x. c * u x)$ 
    assumes add:  $\bigwedge u v. P u \implies P v \implies P (\lambda x. v x + u x)$ 
    shows  $P u$ 
  proof (rule cong)
    from AE_space show  $AE\ x\ in\ M. (\sum y \in u \text{ ' } space\ M. y * indicator\ (u - \{y\} \cap space\ M)\ x) = u\ x$ 
    proof eventually_elim
      fix x assume x:  $x \in space\ M$ 
      from simple_function_indicator_representation[OF u x]
      show  $(\sum y \in u \text{ ' } space\ M. y * indicator\ (u - \{y\} \cap space\ M)\ x) = u\ x \ ..$ 
    qed
  next
    from u have finite (u ' space M)
    unfolding simple_function_def by auto
    then show  $P (\lambda x. \sum y \in u \text{ ' } space\ M. y * indicator\ (u - \{y\} \cap space\ M)\ x)$ 
    proof induct
      case empty show ?case
        using set[of {}] by (simp add: indicator_def[abs_def])
    qed (auto intro!: add mult set simple_functionD u)
  next
    show simple_function M  $(\lambda x. (\sum y \in u \text{ ' } space\ M. y * indicator\ (u - \{y\} \cap space\ M)\ x))$ 
    apply (subst simple_function_cong)
    apply (rule simple_function_indicator_representation[symmetric])
    apply (auto intro: u)
    done
  qed fact

lemma simple_function_induct_nn[consumes 1, case_names cong set mult add]:
  fixes u :: 'a  $\Rightarrow$  ennreal
  assumes u: simple_function M u
  assumes cong:  $\bigwedge f g. simple\_function\ M\ f \implies simple\_function\ M\ g \implies (\bigwedge x. x \in space\ M \implies f\ x = g\ x) \implies P\ f \implies P\ g$ 
  assumes set:  $\bigwedge A. A \in sets\ M \implies P\ (indicator\ A)$ 
  assumes mult:  $\bigwedge u c. simple\_function\ M\ u \implies P\ u \implies P (\lambda x. c * u\ x)$ 
  assumes add:  $\bigwedge u v. simple\_function\ M\ u \implies P\ u \implies simple\_function\ M\ v \implies (\bigwedge x. x \in space\ M \implies u\ x = 0 \vee v\ x = 0) \implies P\ v \implies P (\lambda x. v\ x + u\ x)$ 
  shows  $P\ u$ 
  proof -
    show ?thesis
    proof (rule cong)
      fix x assume x:  $x \in space\ M$ 
      from simple_function_indicator_representation[OF u x]
      show  $(\sum y \in u \text{ ' } space\ M. y * indicator\ (u - \{y\} \cap space\ M)\ x) = u\ x \ ..$ 
    next
      show simple_function M  $(\lambda x. (\sum y \in u \text{ ' } space\ M. y * indicator\ (u - \{y\} \cap space\ M)\ x))$ 
      apply (subst simple_function_cong)
      apply (rule simple_function_indicator_representation[symmetric])

```

```

    apply (auto intro: u)
  done
next
from u have finite (u ' space M)
  unfolding simple_function_def by auto
then show P (λx. ∑ y∈u ' space M. y * indicator (u - ' {y} ∩ space M) x)
proof induct
  case empty show ?case
    using set[of {}] by (simp add: indicator_def[abs_def])
  next
  case (insert x S)
  { fix z have (∑ y∈S. y * indicator (u - ' {y} ∩ space M) z) = 0 ∨
    x * indicator (u - ' {x} ∩ space M) z = 0
    using insert by (subst sum_eq_0_iff) (auto simp: indicator_def) }
  note disj = this
  from insert show ?case
  by (auto intro!: add_mult_set_simple_functionD u simple_function_sum disj)
qed
qed fact
qed

lemma borel_measurable_induct
  [consumes 1, case_names cong set mult add seq, induct set: borel_measurable]:
  fixes u :: 'a ⇒ ennreal
  assumes u: u ∈ borel_measurable M
  assumes cong: ∧f g. f ∈ borel_measurable M ⇒ g ∈ borel_measurable M ⇒
    (∧x. x ∈ space M ⇒ f x = g x) ⇒ P g ⇒ P f
  assumes set: ∧A. A ∈ sets M ⇒ P (indicator A)
  assumes mult': ∧u c. c < top ⇒ u ∈ borel_measurable M ⇒ (∧x. x ∈ space
    M ⇒ u x < top) ⇒ P u ⇒ P (λx. c * u x)
  assumes add: ∧u v. u ∈ borel_measurable M ⇒ (∧x. x ∈ space M ⇒ u x <
    top) ⇒ P u ⇒ v ∈ borel_measurable M ⇒ (∧x. x ∈ space M ⇒ v x < top)
    ⇒ (∧x. x ∈ space M ⇒ u x = 0 ∨ v x = 0) ⇒ P v ⇒ P (λx. v x + u x)
  assumes seq: ∧U. (∧i. U i ∈ borel_measurable M) ⇒ (∧i x. x ∈ space M ⇒
    U i x < top) ⇒ (∧i. P (U i)) ⇒ incseq U ⇒ u = (SUP i. U i) ⇒ P (SUP
    i. U i)
  shows P u
  using u
proof (induct rule: borel_measurable_implies_simple_function_sequence')
  fix U assume U: ∧i. simple_function M (U i) incseq U ∧i x. U i x < top and
  sup: ∧x. (SUP i. U i x) = u x
  have u_eq: u = (SUP i. U i)
  using u by (auto simp add: image_comp sup)

  have not_inf: ∧i x. x ∈ space M ⇒ U i x < top
  using U by (auto simp: image_iff eq_commute)

  from U have ∧i. U i ∈ borel_measurable M
  by (simp add: borel_measurable_simple_function)

```

```

show P u
  unfolding u_eq
proof (rule seq)
  fix i show P (U i)
    using ⟨simple_function M (U i)⟩ not_inf[of _ i]
  proof (induct rule: simple_function_induct_nn)
    case (mult u c)
    show ?case
    proof cases
      assume c = 0 ∨ space M = {} ∨ (∀ x ∈ space M. u x = 0)
      with mult(1) show ?thesis
        by (intro cong[of λx. c * u x indicator {}] set)
           (auto dest!: borel_measurable_simple_function)
    next
      assume ¬ (c = 0 ∨ space M = {} ∨ (∀ x ∈ space M. u x = 0))
      then obtain x where space M ≠ {} and x: x ∈ space M u x ≠ 0 c ≠ 0
        by auto
      with mult(3)[of x] have c < top
        by (auto simp: ennreal_mult_less_top)
      then have u_fin: x' ∈ space M ⟹ u x' < top for x'
        using mult(3)[of x] ⟨c ≠ 0⟩ by (auto simp: ennreal_mult_less_top)
      then have P u
        by (rule mult)
      with u_fin ⟨c < top⟩ mult(1) show ?thesis
        by (intro mult') (auto dest!: borel_measurable_simple_function)
    qed
  qed (auto intro: cong intro!: set add dest!: borel_measurable_simple_function)
qed fact+
qed

lemma simple_function>If_set:
  assumes sf: simple_function M f simple_function M g and A: A ∩ space M ∈
  sets M
  shows simple_function M (λx. if x ∈ A then f x else g x) (is simple_function
  M ?IF)
proof -
  define F where F x = f - ' {x} ∩ space M for x
  define G where G x = g - ' {x} ∩ space M for x
  show ?thesis unfolding simple_function_def
  proof safe
    have ?IF ' space M ⊆ f ' space M ∪ g ' space M by auto
    from finite_subset[OF this] assms
    show finite (?IF ' space M) unfolding simple_function_def by auto
  next
    fix x assume x ∈ space M
    then have *: ?IF - ' {?IF x} ∩ space M = (if x ∈ A
      then ((F (f x) ∩ (A ∩ space M)) ∪ (G (f x) - (G (f x) ∩ (A ∩ space M))))
      else ((F (g x) ∩ (A ∩ space M)) ∪ (G (g x) - (G (g x) ∩ (A ∩ space M)))))

```

```

    using sets.sets_into_space[OF A] by (auto split: if_split_asm simp: G_def
F_def)
    have [intro]:  $\bigwedge x. F\ x \in \text{sets } M \bigwedge x. G\ x \in \text{sets } M$ 
    unfolding F_def G_def using sf[THEN simple_functionD(2)] by auto
    show  $?IF - \{ ?IF\ x\} \cap \text{space } M \in \text{sets } M$  unfolding * using A by auto
  qed
qed

```

```

lemma simple_function_If:
  assumes sf: simple_function M f simple_function M g and P:  $\{x \in \text{space } M. P\ x\} \in \text{sets } M$ 
  shows simple_function M  $(\lambda x. \text{if } P\ x \text{ then } f\ x \text{ else } g\ x)$ 
proof -
  have  $\{x \in \text{space } M. P\ x\} = \{x. P\ x\} \cap \text{space } M$  by auto
  with simple_function_If_set[OF sf, of  $\{x. P\ x\}$ ] P show ?thesis by simp
qed

```

```

lemma simple_function_subalgebra:
  assumes simple_function N f
  and N_subalgebra:  $\text{sets } N \subseteq \text{sets } M \text{ space } N = \text{space } M$ 
  shows simple_function M f
  using assms unfolding simple_function_def by auto

```

```

lemma simple_function_comp:
  assumes T:  $T \in \text{measurable } M\ M'$ 
  and f: simple_function M' f
  shows simple_function M  $(\lambda x. f\ (T\ x))$ 
proof (intro simple_function_def[THEN iffD2] conjI ballI)
  have  $(\lambda x. f\ (T\ x)) - \{ \} \cap \text{space } M \subseteq f - \{ \} \cap \text{space } M'$ 
  using T unfolding measurable_def by auto
  then show finite  $((\lambda x. f\ (T\ x)) - \{ \} \cap \text{space } M)$ 
  using f unfolding simple_function_def by (auto intro: finite_subset)
  fix i assume i:  $i \in (\lambda x. f\ (T\ x)) - \{ \} \cap \text{space } M$ 
  then have  $i \in f - \{ \} \cap \text{space } M'$ 
  using T unfolding measurable_def by auto
  then have  $f - \{ \} \cap \text{space } M' \in \text{sets } M'$ 
  using f unfolding simple_function_def by auto
  then have  $T - \{ \} \cap (\text{space } M' \cap \text{space } M) \in \text{sets } M$ 
  using T unfolding measurable_def by auto
  also have  $T - \{ \} \cap (\text{space } M' \cap \text{space } M) = (\lambda x. f\ (T\ x)) - \{ \} \cap \text{space } M$ 
  using T unfolding measurable_def by auto
  finally show  $(\lambda x. f\ (T\ x)) - \{ \} \cap \text{space } M \in \text{sets } M$  .
qed

```

6.6.3 Simple integral

definition *simple_integral* :: $'a \text{ measure} \Rightarrow ('a \Rightarrow \text{ennreal}) \Rightarrow \text{ennreal } (\text{integral}^S)$
 where

$$\text{integral}^S M f = (\sum x \in f \text{ ' space } M. x * \text{emeasure } M (f \text{ - ' } \{x\} \cap \text{space } M))$$

syntax

`_simple_integral :: pttrn  $\Rightarrow$  ennreal  $\Rightarrow$  'a measure  $\Rightarrow$  ennreal ( $\int^S \_ . \_ \partial \_$  [60,61] 110)`

translations

$$\int^S x. f \partial M == \text{CONST simple\_integral } M (\%x. f)$$

lemma simple_integral_cong:

assumes $\bigwedge t. t \in \text{space } M \implies f t = g t$

shows $\text{integral}^S M f = \text{integral}^S M g$

proof –

have $f \text{ ' space } M = g \text{ ' space } M$

$\bigwedge x. f \text{ - ' } \{x\} \cap \text{space } M = g \text{ - ' } \{x\} \cap \text{space } M$

using *assms* **by** (*auto intro!: image_eqI*)

thus *?thesis* **unfolding** *simple_integral_def* **by** *simp*

qed

lemma simple_integral_const[simp]:

$(\int^S x. c \partial M) = c * (\text{emeasure } M) (\text{space } M)$

proof (*cases* $\text{space } M = \{\}$)

case *True* **thus** *?thesis* **unfolding** *simple_integral_def* **by** *simp*

next

case *False* **hence** $(\lambda x. c) \text{ ' space } M = \{c\}$ **by** *auto*

thus *?thesis* **unfolding** *simple_integral_def* **by** *simp*

qed

lemma simple_function_partition:

assumes *f*: *simple_function* *M f* **and** *g*: *simple_function* *M g*

assumes *sub*: $\bigwedge x y. x \in \text{space } M \implies y \in \text{space } M \implies g x = g y \implies f x = f y$

assumes *v*: $\bigwedge x. x \in \text{space } M \implies f x = v (g x)$

shows $\text{integral}^S M f = (\sum y \in g \text{ ' space } M. v y * \text{emeasure } M \{x \in \text{space } M. g x = y\})$

(*is* $\_ = ?r$)

proof –

from *f g* **have** [*simp*]: *finite* (*f* ' *space* *M*) *finite* (*g* ' *space* *M*)

by (*auto simp: simple_function_def*)

from *f g* **have** [*measurable*]: *f* $\in$ *measurable* *M* (*count_space* *UNIV*) *g* $\in$ *measurable* *M* (*count_space* *UNIV*)

by (*auto intro: measurable_simple_function*)

{ fix *y* **assume** $y \in \text{space } M$

then **have** $f \text{ ' space } M \cap \{i. \exists x \in \text{space } M. i = f x \wedge g y = g x\} = \{v (g y)\}$

by (*auto cong: sub simp: v[symmetric]*) }

note *eq = this*

have $\text{integral}^S M f =$

$(\sum y \in f \text{ ' space } M. y * (\sum z \in g \text{ ' space } M.$

```

      if  $\exists x \in \text{space } M. y = f x \wedge z = g x$  then  $\text{emeasure } M \{x \in \text{space } M. g x = z\}$ 
    else 0))
  unfolding simple_integral_def
  proof (safe intro!: sum.cong ennreal_mult_left_cong)
    fix y assume y:  $y \in \text{space } M$   $f y \neq 0$ 
    have [simp]:  $g \text{ 'space } M \cap \{z. \exists x \in \text{space } M. f y = f x \wedge z = g x\} =$ 
       $\{z. \exists x \in \text{space } M. f y = f x \wedge z = g x\}$ 
    by auto
    have eq:  $(\bigcup i \in \{z. \exists x \in \text{space } M. f y = f x \wedge z = g x\}. \{x \in \text{space } M. g x = i\})$ 
    =
       $f \text{ - '}\{f y\} \cap \text{space } M$ 
    by (auto simp: eq_commute cong: sub_rev_conj_cong)
    have finite (g'space M) by simp
    then have finite  $\{z. \exists x \in \text{space } M. f y = f x \wedge z = g x\}$ 
    by (rule rev_finite_subset) auto
    then show  $\text{emeasure } M (f \text{ - '}\{f y\} \cap \text{space } M) =$ 
       $(\sum z \in g \text{ 'space } M. \text{if } \exists x \in \text{space } M. f y = f x \wedge z = g x \text{ then } \text{emeasure } M \{x \in \text{space } M. g x = z\} \text{ else } 0)$ 
    apply (simp add: sum.If_cases)
    apply (subst sum_emeasure)
    apply (auto simp: disjoint_family_on_def eq)
    done
  qed
  also have  $\dots = (\sum y \in f \text{ 'space } M. (\sum z \in g \text{ 'space } M. \text{if } \exists x \in \text{space } M. y = f x \wedge z = g x \text{ then } y * \text{emeasure } M \{x \in \text{space } M. g x = z\} \text{ else } 0))$ 
  by (auto intro!: sum.cong simp: sum_distrib_left)
  also have  $\dots = ?r$ 
  by (subst sum.swap)
  (auto intro!: sum.cong simp: sum.If_cases scaleR_sum_right[symmetric] eq)
  finally show  $\text{integral}^S M f = ?r$  .
qed

lemma simple_integral_add[simp]:
  assumes f: simple_function M f and  $\bigwedge x. 0 \leq f x$  and g: simple_function M g
  and  $\bigwedge x. 0 \leq g x$ 
  shows  $(\int^S x. f x + g x \partial M) = \text{integral}^S M f + \text{integral}^S M g$ 
proof -
  have  $(\int^S x. f x + g x \partial M) =$ 
     $(\sum y \in (\lambda x. (f x, g x)) \text{ 'space } M. (\text{fst } y + \text{snd } y) * \text{emeasure } M \{x \in \text{space } M. (f x, g x) = y\})$ 
  by (intro simple_function_partition) (auto intro: f g)
  also have  $\dots = (\sum y \in (\lambda x. (f x, g x)) \text{ 'space } M. \text{fst } y * \text{emeasure } M \{x \in \text{space } M. (f x, g x) = y\}) +$ 
     $(\sum y \in (\lambda x. (f x, g x)) \text{ 'space } M. \text{snd } y * \text{emeasure } M \{x \in \text{space } M. (f x, g x) = y\})$ 
  using assms(2,4) by (auto intro!: sum.cong distrib_right simp: sum.distrib[symmetric])
  also have  $(\sum y \in (\lambda x. (f x, g x)) \text{ 'space } M. \text{fst } y * \text{emeasure } M \{x \in \text{space } M. (f x, g x) = y\}) = (\int^S x. f x \partial M)$ 

```

```

    by (intro simple_function_partition[symmetric]) (auto intro: f g)
    also have  $(\sum y \in (\lambda x. (f x, g x))' \text{space } M. \text{snd } y * \text{emeasure } M \{x \in \text{space } M. (f x, g x) = y\}) = (\int^S x. g x \partial M)$ 
    by (intro simple_function_partition[symmetric]) (auto intro: f g)
    finally show ?thesis .
qed

```

```

lemma simple_integral_sum[simp]:
  assumes  $\bigwedge i x. i \in P \implies 0 \leq f i x$ 
  assumes  $\bigwedge i. i \in P \implies \text{simple\_function } M (f i)$ 
  shows  $(\int^S x. (\sum i \in P. f i x) \partial M) = (\sum i \in P. \text{integral}^S M (f i))$ 
proof cases
  assume finite P
  from this assms show ?thesis
    by induct (auto)
qed auto

```

```

lemma simple_integral_mult[simp]:
  assumes f: simple_function M f
  shows  $(\int^S x. c * f x \partial M) = c * \text{integral}^S M f$ 
proof -
  have  $(\int^S x. c * f x \partial M) = (\sum y \in f' \text{space } M. (c * y) * \text{emeasure } M \{x \in \text{space } M. f x = y\})$ 
  using f by (intro simple_function_partition) auto
  also have  $\dots = c * \text{integral}^S M f$ 
  using f unfolding simple_integral_def
  by (subst sum_distrib_left) (auto simp: mult.assoc Int_def conj_commute)
  finally show ?thesis .
qed

```

```

lemma simple_integral_mono_AE:
  assumes f[measurable]: simple_function M f and g[measurable]: simple_function M g
  and mono: AE x in M. f x ≤ g x
  shows  $\text{integral}^S M f \leq \text{integral}^S M g$ 
proof -
  let ?μ = λP. emeasure M {x ∈ space M. P x}
  have  $\text{integral}^S M f = (\sum y \in (\lambda x. (f x, g x))' \text{space } M. \text{fst } y * ?μ (\lambda x. (f x, g x) = y))$ 
  using f g by (intro simple_function_partition) auto
  also have  $\dots \leq (\sum y \in (\lambda x. (f x, g x))' \text{space } M. \text{snd } y * ?μ (\lambda x. (f x, g x) = y))$ 
  proof (clarsimp intro!: sum_mono)
    fix x assume x ∈ space M
    let ?M = ?μ (λy. f y = f x ∧ g y = g x)
    show f x * ?M ≤ g x * ?M
  proof cases
    assume ?M ≠ 0
    then have 0 < ?M
      by (simp add: less_le)

```

```

    also have ... ≤ ?μ (λy. f x ≤ g x)
      using mono by (intro emeasure_mono_AE) auto
    finally have ¬ ¬ f x ≤ g x
      by (intro notI) auto
    then show ?thesis
      by (intro mult_right_mono) auto
  qed simp
qed
also have ... = integralS M g
  using f g by (intro simple_function_partition[symmetric]) auto
finally show ?thesis .
qed

```

```

lemma simple_integral_mono:
  assumes simple_function M f and simple_function M g
  and mono:  $\bigwedge x. x \in \text{space } M \implies f x \leq g x$ 
  shows integralS M f ≤ integralS M g
  using assms by (intro simple_integral_mono_AE) auto

```

```

lemma simple_integral_cong_AE:
  assumes simple_function M f and simple_function M g
  and AE x in M. f x = g x
  shows integralS M f = integralS M g
  using assms by (auto simp: eq_iff intro!: simple_integral_mono_AE)

```

```

lemma simple_integral_cong':
  assumes sf: simple_function M f simple_function M g
  and mea: (emeasure M) {x ∈ space M. f x ≠ g x} = 0
  shows integralS M f = integralS M g
proof (intro simple_integral_cong_AE sf AE_I)
  show (emeasure M) {x ∈ space M. f x ≠ g x} = 0 by fact
  show {x ∈ space M. f x ≠ g x} ∈ sets M
    using sf[THEN borel_measurable_simple_function] by auto
qed simp

```

```

lemma simple_integral_indicator:
  assumes A: A ∈ sets M
  assumes f: simple_function M f
  shows (∫S x. f x * indicator A x ∂M) =
    (∑ x ∈ f ' space M. x * emeasure M (f - ' {x} ∩ space M ∩ A))
proof -
  have eq: (λx. (f x, indicator A x)) ' space M ∩ {x. snd x = 1} = (λx. (f x,
    1::ennreal)) ' A
    using A[THEN sets_into_space] by (auto simp: indicator_def image_iff
    split: if_split_asm)
  have eq2:  $\bigwedge x. f x \notin f ' A \implies f - ' \{f x\} \cap \text{space } M \cap A = \{\}$ 
    by (auto simp: image_iff)

  have (∫S x. f x * indicator A x ∂M) =

```



```

  (∑ y ∈ (λ x. (f x, indicator A x)) 'space M. (fst y * snd y) * emeasure M {x ∈ space
M. (f x, indicator A x) = y})
  using assms by (intro simple_function_partition) auto
  also have ... = (∑ y ∈ (λ x. (f x, indicator A x :: ennreal)) 'space M.
    if snd y = 1 then fst y * emeasure M (f - ' {fst y} ∩ space M ∩ A) else 0)
  by (auto simp: indicator_def split: if_split_asm intro!: arg_cong2[where
f=(*)] arg_cong2[where f=emeasure] sum.cong)
  also have ... = (∑ y ∈ (λ x. (f x, 1 :: ennreal)) 'A. fst y * emeasure M (f - ' {fst
y} ∩ space M ∩ A))
  using assms by (subst sum.If_cases) (auto intro!: simple_functionD(1) simp:
eq)
  also have ... = (∑ y ∈ fst '(λ x. (f x, 1 :: ennreal)) 'A. y * emeasure M (f - ' {y}
∩ space M ∩ A))
  by (subst sum.reindex [of fst]) (auto simp: inj_on_def)
  also have ... = (∑ x ∈ f ' space M. x * emeasure M (f - ' {x} ∩ space M ∩
A))
  using A[THEN sets.sets_into_space]
  by (intro sum.mono_neutral_cong_left simple_functionD f) (auto simp: im-
age_comp comp_def eq2)
  finally show ?thesis .
qed

```

```

lemma simple_integral_indicator_only[simp]:
  assumes A ∈ sets M
  shows integralS M (indicator A) = emeasure M A
  using simple_integral_indicator[OF assms, of λ x. 1] sets.sets_into_space[OF
assms]
  by (simp_all add: image_constant_conv Int_absorb1 split: if_split_asm)

```

```

lemma simple_integral_null_set:
  assumes simple_function M u ∧ x. 0 ≤ u x and N ∈ null_sets M
  shows (∫S x. u x * indicator N x ∂M) = 0
proof -
  have AE x in M. indicator N x = (0 :: ennreal)
  using ⟨N ∈ null_sets M⟩ by (auto simp: indicator_def intro!: AE_I[of _ _
N])
  then have (∫S x. u x * indicator N x ∂M) = (∫S x. 0 ∂M)
  using assms apply (intro simple_integral_cong_AE) by auto
  then show ?thesis by simp
qed

```

```

lemma simple_integral_cong_AE_mult_indicator:
  assumes sf: simple_function M f and eq: AE x in M. x ∈ S and S ∈ sets M
  shows integralS M f = (∫S x. f x * indicator S x ∂M)
  using assms by (intro simple_integral_cong_AE) auto

```

```

lemma simple_integral_cmult_indicator:
  assumes A: A ∈ sets M
  shows (∫S x. c * indicator A x ∂M) = c * emeasure M A

```

```

using simple_integral_mult[OF simple_function_indicator[OF A]]
unfolding simple_integral_indicator_only[OF A] by simp

```

```

lemma simple_integral_nonneg:
  assumes f: simple_function M f and ae: AE x in M. 0 ≤ f x
  shows 0 ≤ integralS M f
proof -
  have integralS M (λx. 0) ≤ integralS M f
    using simple_integral_mono_AE[OF _ f ae] by auto
  then show ?thesis by simp
qed

```

6.6.4 Integral on nonnegative functions

definition $nn\_integral :: 'a \text{ measure} \Rightarrow ('a \Rightarrow \text{ennreal}) \Rightarrow \text{ennreal} \ (integral^N)$
where

$$integral^N M f = (SUP\ g \in \{g. \text{simple\_function } M\ g \wedge g \leq f\}. \text{integral}^S M g)$$

syntax

$$\_nn\_integral :: pttm \Rightarrow \text{ennreal} \Rightarrow 'a \text{ measure} \Rightarrow \text{ennreal} \ (\int^+ ((2\_./\_)/ \partial\_))$$

[60,61] 110)

translations

$$\int^+ x. f \partial M == \text{CONST } nn\_integral\ M\ (\lambda x. f)$$

lemma $nn\_integral\_def\_finite$:

$$integral^N M f = (SUP\ g \in \{g. \text{simple\_function } M\ g \wedge g \leq f \wedge (\forall x. g\ x < \text{top})\}. \text{integral}^S M g)$$

$$(\text{is } \_ = \text{Sup } (?A \text{ ' } ?f))$$

unfolding $nn\_integral\_def$

proof (safe intro!: antisym SUP_least)

fix g **assume** $g[\text{measurable}]$: $\text{simple\_function } M\ g\ g \leq f$

show $integral^S M\ g \leq \text{Sup } (?A \text{ ' } ?f)$

proof cases

assume ae : $AE\ x\ \text{in } M. g\ x \neq \text{top}$

let $?G = \{x \in \text{space } M. g\ x \neq \text{top}\}$

have $integral^S M\ g = integral^S M\ (\lambda x. g\ x * \text{indicator } ?G\ x)$

proof (rule simple_integral_cong_AE)

show $AE\ x\ \text{in } M. g\ x = g\ x * \text{indicator } ?G\ x$

using $ae\ AE\_space$ **by** eventually_elim auto

qed (insert g , auto)

also have $\dots \leq \text{Sup } (?A \text{ ' } ?f)$

using g **by** (intro SUP_upper) (auto simp: le_fun_def less_top split: split_indicator)

finally show ?thesis .

next

assume nAE : $\neg (AE\ x\ \text{in } M. g\ x \neq \text{top})$

then have $\text{emeasure } M\ \{x \in \text{space } M. g\ x = \text{top}\} \neq 0$ (is $\text{emeasure } M\ ?G \neq 0$)

```

    by (subst (asm) AE_iff_measurable[OF _ refl]) auto
  then have top = (SUP n. (∫S x. of_nat n * indicator ?G x ∂M))
  by (simp add: ennreal_SUP_of_nat_eq_top ennreal_top_eq_mult_iff SUP_mult_right_ennreal[symmetric])
  also have ... ≤ Sup (?A ‘ ?f)
    using g
    by (safe intro!: SUP_least SUP_upper)
      (auto simp: le_fun_def of_nat_less_top top_unique[symmetric] split:
split_indicator
intro: order_trans[of _ g x f x for x, OF order_trans[of _ top]])
  finally show ?thesis
    by (simp add: top_unique del: SUP_eq_top_iff Sup_eq_top_iff)
qed
qed (auto intro: SUP_upper)

```

```

lemma nn_integral_mono_AE:
  assumes ae: AE x in M. u x ≤ v x shows integralN M u ≤ integralN M v
  unfolding nn_integral_def
  proof (safe intro!: SUP_mono)
    fix n assume n: simple_function M n n ≤ u
    from ae[THEN AE_E] obtain N
      where N: {x ∈ space M. ¬ u x ≤ v x} ⊆ N emeasure M N = 0 N ∈ sets M
      by auto
    then have ae_N: AE x in M. x ∉ N by (auto intro: AE_not_in)
    let ?n = λx. n x * indicator (space M - N) x
    have AE x in M. n x ≤ ?n x simple_function M ?n
      using n N ae_N by auto
    moreover
    { fix x have ?n x ≤ v x
      proof cases
        assume x: x ∈ space M - N
        with N have u x ≤ v x by auto
        with n(2)[THEN le_funD, of x] x show ?thesis
          by (auto simp: max_def split: if_split_asm)
      qed simp }
    then have ?n ≤ v by (auto simp: le_funI)
    moreover have integralS M n ≤ integralS M ?n
      using ae_N N n by (auto intro!: simple_integral_mono_AE)
    ultimately show ∃ m ∈ {g. simple_function M g ∧ g ≤ v}. integralS M n ≤
integralS M m
      by force
  qed

```

```

lemma nn_integral_mono:
  (∧ x. x ∈ space M ⇒ u x ≤ v x) ⇒ integralN M u ≤ integralN M v
  by (auto intro: nn_integral_mono_AE)

```

```

lemma mono_nn_integral: mono F ⇒ mono (λx. integralN M (F x))
  by (auto simp add: mono_def le_fun_def intro!: nn_integral_mono)

```

lemma *nn_integral_cong_AE*:

AE x in M. u x = v x $\implies$ integral^N M u = integral^N M v
by (*auto simp: eq_iff intro!: nn_integral_mono_AE*)

lemma *nn_integral_cong*:

($\bigwedge x. x \in \text{space } M \implies u x = v x$) $\implies$ integral^N M u = integral^N M v
by (*auto intro: nn_integral_cong_AE*)

lemma *nn_integral_cong_simp*:

($\bigwedge x. x \in \text{space } M =_{\text{simp}} \implies u x = v x$) $\implies$ integral^N M u = integral^N M v
by (*auto intro: nn_integral_cong_simp: simp_implies_def*)

lemma *incseq_nn_integral*:

assumes *incseq f* **shows** *incseq ($\lambda i. \text{integral}^N M (f i)$)*

proof –

have $\bigwedge i x. f i x \leq f (\text{Suc } i) x$

using *assms* **by** (*auto dest!: incseq_SucD simp: le_fun_def*)

then show *?thesis*

by (*auto intro!: incseq_SucI nn_integral_mono*)

qed

lemma *nn_integral_eq_simple_integral*:

assumes *f: simple_function M f* **shows** *integral^N M f = integral^S M f*

proof –

let *?f = $\lambda x. f x * \text{indicator } (\text{space } M) x$*

have *f': simple_function M ?f* **using** *f* **by** *auto*

have *integral^N M ?f $\leq$ integral^S M ?f* **using** *f'*

by (*force intro!: SUP_least simple_integral_mono simp: le_fun_def nn_integral_def*)

moreover have *integral^S M ?f $\leq$ integral^N M ?f*

unfolding *nn_integral_def*

using *f'* **by** (*auto intro!: SUP_upper*)

ultimately show *?thesis*

by (*simp cong: nn_integral_cong simple_integral_cong*)

qed

Beppo-Levi monotone convergence theorem

lemma *nn_integral_monotone_convergence_SUP*:

assumes *f: incseq f* **and** [*measurable*]: $\bigwedge i. f i \in \text{borel\_measurable } M$

shows $(\int^+ x. (\text{SUP } i. f i x) \partial M) = (\text{SUP } i. \text{integral}^N M (f i))$

proof (*rule antisym*)

show $(\int^+ x. (\text{SUP } i. f i x) \partial M) \leq (\text{SUP } i. (\int^+ x. f i x \partial M))$

unfolding *nn_integral_def_finite* [*of $\lambda x. \text{SUP } i. f i x$*]

proof (*safe intro!: SUP_least*)

fix *u* **assume** *sf_u[simp]: simple_function M u* **and**

u: u $\leq$ ($\lambda x. \text{SUP } i. f i x$) **and** *u_range: $\forall x. u x < \text{top}$*

note *sf_u[THEN borel_measurable_simple_function, measurable]*

show *integral^S M u $\leq$ ($\text{SUP } j. \int^+ x. f j x \partial M$)*

proof (*rule ennreal_approx_unit*)

fix *a :: ennreal* **assume** *a < 1*

```

let ?au =  $\lambda x. a * u x$ 

let ?B =  $\lambda c i. \{x \in \text{space } M. ?au x = c \wedge c \leq f i x\}$ 
have integralS M ?au = ( $\sum c \in ?au' \text{space } M. c * (\text{SUP } i. \text{emeasure } M (?B c i))$ )
i)))
  unfolding simple_integral_def
proof (intro sum.cong ennreal_mult_left_cong refl)
  fix c assume c  $\in ?au' \text{space } M$  c  $\neq 0$ 
  { fix x' assume x':  $x' \in \text{space } M$  ?au x' = c
    with  $\langle c \neq 0 \rangle$  u_range have ?au x'  $< 1 * u x'$ 
      by (intro ennreal_mult_strict_right_mono  $\langle a < 1 \rangle$ ) (auto simp: less_le)
    also have ...  $\leq (\text{SUP } i. f i x')$ 
      using u by (auto simp: le_fun_def)
    finally have  $\exists i. ?au x' \leq f i x'$ 
      by (auto simp: less_SUP_iff intro: less_imp_le) }
  then have *: ?au -  $\{c\} \cap \text{space } M = (\bigcup i. ?B c i)$ 
    by auto
  show emeasure M (?au -  $\{c\} \cap \text{space } M$ ) = ( $\text{SUP } i. \text{emeasure } M (?B c i)$ )
  i))
  unfolding * using f
  by (intro SUP_emeasure_incseq[symmetric])
    (auto simp: incseq_def le_fun_def intro: order_trans)
qed
also have ... = ( $\text{SUP } i. \sum c \in ?au' \text{space } M. c * \text{emeasure } M (?B c i)$ )
  unfolding SUP_mult_left_ennreal using f
  by (intro ennreal_SUP_sum[symmetric])
    (auto intro!: mult_mono emeasure_mono simp: incseq_def le_fun_def
intro: order_trans)
  also have ...  $\leq (\text{SUP } i. \text{integral}^N M (f i))$ 
  proof (intro SUP_subset_mono order_refl)
    fix i
    have ( $\sum c \in ?au' \text{space } M. c * \text{emeasure } M (?B c i)$ ) =
      ( $\int^S x. (a * u x) * \text{indicator } \{x \in \text{space } M. a * u x \leq f i x\} x \partial M$ )
      by (subst simple_integral_indicator)
        (auto intro!: sum.cong ennreal_mult_left_cong arg_cong2[where
f=emeasure])
    also have ... = ( $\int^+ x. (a * u x) * \text{indicator } \{x \in \text{space } M. a * u x \leq f i x\} x \partial M$ )
      by (rule nn_integral_eq_simple_integral[symmetric]) simp
    also have ...  $\leq (\int^+ x. f i x \partial M)$ 
      by (intro nn_integral_mono) (auto split: split_indicator)
    finally show ( $\sum c \in ?au' \text{space } M. c * \text{emeasure } M (?B c i)$ )  $\leq (\int^+ x. f i x \partial M)$  .
  qed
  finally show a * integralS M u  $\leq (\text{SUP } i. \text{integral}^N M (f i))$ 
    by simp
  qed
qed
qed (auto intro!: SUP_least SUP_upper nn_integral_mono)

```

```

lemma sup_continuous_nn_integral[order_continuous_intros]:
  assumes f:  $\bigwedge y. \text{sup\_continuous } (f\ y)$ 
  assumes [measurable]:  $\bigwedge x. (\lambda y. f\ y\ x) \in \text{borel\_measurable } M$ 
  shows sup_continuous  $(\lambda x. (\int^+ y. f\ y\ x\ \partial M))$ 
  unfolding sup_continuous_def
proof safe
  fix C :: nat  $\Rightarrow$  'b assume C: incseq C
  with sup_continuous_mono[OF f] show  $(\int^+ y. f\ y\ (\text{Sup } (C\ ' \text{UNIV}))\ \partial M) =$ 
 $(\text{SUP } i. \int^+ y. f\ y\ (C\ i)\ \partial M)$ 
    unfolding sup_continuousD[OF f C]
    by (subst nn_integral_monotone_convergence_SUP) (auto simp: mono_def
le_fun_def)
qed

theorem nn_integral_monotone_convergence_SUP_AE:
  assumes f:  $\bigwedge i. \text{AE } x \text{ in } M. f\ i\ x \leq f\ (\text{Suc } i)\ x \wedge i. f\ i \in \text{borel\_measurable } M$ 
  shows  $(\int^+ x. (\text{SUP } i. f\ i\ x)\ \partial M) = (\text{SUP } i. \text{integral}^N M\ (f\ i))$ 
proof -
  from f have AE x in M.  $\forall i. f\ i\ x \leq f\ (\text{Suc } i)\ x$ 
    by (simp add: AE_all_countable)
  from this[THEN AE_E] obtain N
    where N:  $\{x \in \text{space } M. \neg (\forall i. f\ i\ x \leq f\ (\text{Suc } i)\ x)\} \subseteq N$  emeasure M N = 0
  N  $\in$  sets M
    by auto
  let ?f =  $\lambda i\ x. \text{if } x \in \text{space } M - N \text{ then } f\ i\ x \text{ else } 0$ 
  have f_eq: AE x in M.  $\forall i. ?f\ i\ x = f\ i\ x$  using N by (auto intro!: AE_I[of _
_ N])
  then have  $(\int^+ x. (\text{SUP } i. f\ i\ x)\ \partial M) = (\int^+ x. (\text{SUP } i. ?f\ i\ x)\ \partial M)$ 
    by (auto intro!: nn_integral_cong_AE)
  also have  $\dots = (\text{SUP } i. (\int^+ x. ?f\ i\ x\ \partial M))$ 
  proof (rule nn_integral_monotone_convergence_SUP)
    show incseq ?f using N(1) by (force intro!: incseq_SucI le_funI)
    { fix i show  $(\lambda x. \text{if } x \in \text{space } M - N \text{ then } f\ i\ x \text{ else } 0) \in \text{borel\_measurable } M$ 
      using f N(3) by (intro measurable_Iff_set) auto }
  qed
  also have  $\dots = (\text{SUP } i. (\int^+ x. f\ i\ x\ \partial M))$ 
    using f_eq by (force intro!: arg_cong[where f =  $\lambda f. \text{Sup } (\text{range } f)$ ] nn_integral_cong_AE
ext)
  finally show ?thesis .
qed

lemma nn_integral_monotone_convergence_simple:
  incseq f  $\implies (\bigwedge i. \text{simple\_function } M\ (f\ i)) \implies (\text{SUP } i. \int^S x. f\ i\ x\ \partial M) = (\int^+ x. (\text{SUP } i. f\ i\ x)\ \partial M)$ 
  using nn_integral_monotone_convergence_SUP[of f M]
  by (simp add: nn_integral_eq_simple_integral[symmetric] borel_measurable_simple_function)

lemma SUP_simple_integral_sequences:

```

```

    assumes f: incseq f  $\wedge$  i. simple_function M (f i)
    and g: incseq g  $\wedge$  i. simple_function M (g i)
    and eq: AE x in M. (SUP i. f i x) = (SUP i. g i x)
    shows (SUP i. integralS M (f i)) = (SUP i. integralS M (g i))
      (is Sup (?F ' _) = Sup (?G ' _))
  proof -
    have (SUP i. integralS M (f i)) = ( $\int$  +x. (SUP i. f i x)  $\partial$ M)
      using f by (rule nn_integral_monotone_convergence_simple)
    also have ... = ( $\int$  +x. (SUP i. g i x)  $\partial$ M)
      unfolding eq[THEN nn_integral_cong_AE] ..
    also have ... = (SUP i. ?G i)
      using g by (rule nn_integral_monotone_convergence_simple[symmetric])
    finally show ?thesis by simp
  qed

lemma nn_integral_const[simp]: ( $\int$  +x. c  $\partial$ M) = c * emeasure M (space M)
  by (subst nn_integral_eq_simple_integral) auto

lemma nn_integral_linear:
  assumes f: f  $\in$  borel_measurable M and g: g  $\in$  borel_measurable M
  shows ( $\int$  +x. a * f x + g x  $\partial$ M) = a * integralN M f + integralN M g
    (is integralN M ?L = _)
  proof -
    obtain u
      where  $\wedge$  i. simple_function M (u i) incseq u  $\wedge$  i x. u i x < top  $\wedge$  x. (SUP i. u
i x) = f x
      using borel_measurable_implies_simple_function_sequence' f(1)
      by auto
    note u = nn_integral_monotone_convergence_simple[OF this(2,1)] this

    obtain v where
       $\wedge$  i. simple_function M (v i) incseq v  $\wedge$  i x. v i x < top  $\wedge$  x. (SUP i. v i x) = g
x
      using borel_measurable_implies_simple_function_sequence' g(1)
      by auto
    note v = nn_integral_monotone_convergence_simple[OF this(2,1)] this

    let ?L' =  $\lambda$  i x. a * u i x + v i x

    have ?L  $\in$  borel_measurable M using assms by auto
    from borel_measurable_implies_simple_function_sequence'[OF this]
    obtain l where  $\wedge$  i. simple_function M (l i) incseq l  $\wedge$  i x. l i x < top  $\wedge$  x. (SUP
i. l i x) = a * f x + g x
      by auto
    note l = nn_integral_monotone_convergence_simple[OF this(2,1)] this

    have inc: incseq ( $\lambda$  i. a * integralS M (u i)) incseq ( $\lambda$  i. integralS M (v i))
      using u v by (auto simp: incseq_Suc_iff le_fun_def intro!: add_mono mult_left_mono
simple_integral_mono)

```

```

have l': (SUP i. integralS M (l i)) = (SUP i. integralS M (?L' i))
proof (rule SUP_simple_integral_sequences[OF l(3,2)])
  show incseq ?L' ∧ i. simple_function M (?L' i)
    using u v unfolding incseq_Suc_iff le_fun_def
    by (auto intro!: add_mono mult_left_mono)
  { fix x
    have (SUP i. a * u i x + v i x) = a * (SUP i. u i x) + (SUP i. v i x)
      using u(3) v(3) u(4)[of _ x] v(4)[of _ x] unfolding SUP_mult_left_enreal
      by (auto intro!: ennreal_SUP_add simp: incseq_Suc_iff le_fun_def add_mono
mult_left_mono) }
    then show AE x in M. (SUP i. l i x) = (SUP i. ?L' i x)
      unfolding l(5) using u(5) v(5) by (intro AE_I2) auto
  qed
also have ... = (SUP i. a * integralS M (u i) + integralS M (v i))
  using u(2) v(2) by auto
finally show ?thesis
  unfolding l(5)[symmetric] l(1)[symmetric]
  by (simp add: ennreal_SUP_add[OF inc] v u SUP_mult_left_enreal[symmetric])
qed

```

```

lemma nn_integral_cmult: f ∈ borel_measurable M ⇒ (∫+ x. c * f x ∂M) = c
* integralN M f
  using nn_integral_linear[of f M λx. 0 c] by simp

```

```

lemma nn_integral_multc: f ∈ borel_measurable M ⇒ (∫+ x. f x * c ∂M) =
integralN M f * c
  unfolding mult.commute[of _ c] nn_integral_cmult by simp

```

```

lemma nn_integral_divide: f ∈ borel_measurable M ⇒ (∫+ x. f x / c ∂M) =
(∫+ x. f x ∂M) / c
  unfolding divide_enreal_def by (rule nn_integral_multc)

```

```

lemma nn_integral_indicator[simp]: A ∈ sets M ⇒ (∫+ x. indicator A x ∂M)
= (emeasure M) A
  by (subst nn_integral_eq_simple_integral) (auto simp: simple_integral_indicator)

```

```

lemma nn_integral_cmult_indicator: A ∈ sets M ⇒ (∫+ x. c * indicator A x
∂M) = c * emeasure M A
  by (subst nn_integral_eq_simple_integral) (auto)

```

```

lemma nn_integral_indicator':
  assumes [measurable]: A ∩ space M ∈ sets M
  shows (∫+ x. indicator A x ∂M) = emeasure M (A ∩ space M)
proof -
  have (∫+ x. indicator A x ∂M) = (∫+ x. indicator (A ∩ space M) x ∂M)
    by (intro nn_integral_cong) (simp split: split_indicator)
  also have ... = emeasure M (A ∩ space M)
    by simp

```


finally show ?thesis .
qed

lemma *nn_integral_indicator_singleton[simp]*:
 assumes [measurable]: $\{y\} \in \text{sets } M$ shows $(\int^+ x. f x * \text{indicator } \{y\} x \partial M) = f y * \text{emeasure } M \{y\}$
proof –
 have $(\int^+ x. f x * \text{indicator } \{y\} x \partial M) = (\int^+ x. f y * \text{indicator } \{y\} x \partial M)$
 by (auto intro!: nn_integral_cong_split: split_indicator)
 then show ?thesis
 by (simp add: nn_integral_cmult)
 qed

lemma *nn_integral_set_ennreal*:
 $(\int^+ x. \text{ennreal } (f x) * \text{indicator } A x \partial M) = (\int^+ x. \text{ennreal } (f x * \text{indicator } A x) \partial M)$
 by (rule nn_integral_cong) (simp split: split_indicator)

lemma *nn_integral_indicator_singleton'[simp]*:
 assumes [measurable]: $\{y\} \in \text{sets } M$
 shows $(\int^+ x. \text{ennreal } (f x * \text{indicator } \{y\} x) \partial M) = f y * \text{emeasure } M \{y\}$
 by (subst nn_integral_set_ennreal[symmetric]) (simp)

lemma *nn_integral_add*:
 $f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies (\int^+ x. f x + g x \partial M) = \text{integral}^N M f + \text{integral}^N M g$
 using nn_integral_linear[of f M g 1] by simp

lemma *nn_integral_sum*:
 $(\bigwedge i. i \in P \implies f i \in \text{borel\_measurable } M) \implies (\int^+ x. (\sum_{i \in P. f i x} \partial M) = (\sum_{i \in P. \text{integral}^N M (f i))$
 by (induction P rule: infinite_finite_induct) (auto simp: nn_integral_add)

theorem *nn_integral_suminf*:
 assumes $f: \bigwedge i. f i \in \text{borel\_measurable } M$
 shows $(\int^+ x. (\sum i. f i x) \partial M) = (\sum i. \text{integral}^N M (f i))$
proof –
 have all_pos: $\text{AE } x \text{ in } M. \forall i. 0 \leq f i x$
 using assms by (auto simp: AE_all_countable)
 have $(\sum i. \text{integral}^N M (f i)) = (\text{SUP } n. \sum_{i < n. \text{integral}^N M (f i)})$
 by (rule suminf_eq_SUP)
 also have $\dots = (\text{SUP } n. \int^+ x. (\sum_{i < n. f i x} \partial M)$
 unfolding nn_integral_sum[OF f] ..
 also have $\dots = \int^+ x. (\text{SUP } n. \sum_{i < n. f i x} \partial M)$ using f all_pos
 by (intro nn_integral_monotone_convergence_SUP_AE[symmetric])
 (elim AE_mp, auto simp: sum_nonneg simp del: sum_lessThan_Suc intro!: AE_I2 sum_mono2)
 also have $\dots = \int^+ x. (\sum i. f i x) \partial M$ using all_pos
 by (intro nn_integral_cong_AE) (auto simp: suminf_eq_SUP)

finally show ?thesis by simp
qed

lemma nn_integral_bound_simple_function:
 assumes bnd: $\bigwedge x. x \in \text{space } M \implies f\ x < \infty$
 assumes f[measurable]: simple_function M f
 assumes supp: $\text{emeasure } M \ \{x \in \text{space } M. f\ x \neq 0\} < \infty$
 shows $\text{nn\_integral } M\ f < \infty$
 proof cases
 assume space M = {}
 then have $\text{nn\_integral } M\ f = (\int^+ x. 0\ \partial M)$
 by (intro nn_integral_cong) auto
 then show ?thesis by simp
 next
 assume space M $\neq \{\}$
 with simple_functionD(1)[OF f] bnd have bnd: $0 \leq \text{Max } (f'\text{space } M) \wedge \text{Max } (f'\text{space } M) < \infty$
 by (subst Max_less_iff) (auto simp: Max_ge_iff)

 have $\text{nn\_integral } M\ f \leq (\int^+ x. \text{Max } (f'\text{space } M) * \text{indicator } \{x \in \text{space } M. f\ x \neq 0\}\ x\ \partial M)$
 proof (rule nn_integral_mono)
 fix x assume $x \in \text{space } M$
 with f show $f\ x \leq \text{Max } (f'\text{space } M) * \text{indicator } \{x \in \text{space } M. f\ x \neq 0\}\ x$
 by (auto split: split_indicator intro!: Max_ge simple_functionD)
 qed
 also have $\dots < \infty$
 using bnd supp by (subst nn_integral_cmult) (auto simp: ennreal_mult_less_top)
 finally show ?thesis .
 qed

theorem nn_integral_Markov_inequality:
 assumes u: $(\lambda x. u\ x * \text{indicator } A\ x) \in \text{borel\_measurable } M$ and $A \in \text{sets } M$
 shows $(\text{emeasure } M) (\{x \in A. 1 \leq c * u\ x\}) \leq c * (\int^+ x. u\ x * \text{indicator } A\ x\ \partial M)$
 (is $(\text{emeasure } M) \ ?A \leq \_ * \ ?PI$)
 proof -
 define u' where $u' = (\lambda x. u\ x * \text{indicator } A\ x)$
 have [measurable]: $u' \in \text{borel\_measurable } M$
 using u unfolding u'_def .
 have $\{x \in \text{space } M. c * u'\ x \geq 1\} \in \text{sets } M$
 by measurable
 also have $\{x \in \text{space } M. c * u'\ x \geq 1\} = \ ?A$
 using sets.sets_into_space[OF $\langle A \in \text{sets } M \rangle$] by (auto simp: u'_def indicator_def)
 finally have $(\text{emeasure } M) \ ?A = (\int^+ x. \text{indicator } \ ?A\ x\ \partial M)$
 using nn_integral_indicator by simp
 also have $\dots \leq (\int^+ x. c * (u\ x * \text{indicator } A\ x)\ \partial M)$
 using u by (auto intro!: nn_integral_mono_AE simp: indicator_def)

```

also have ... = c * ( $\int^+ x. u\ x * \text{indicator } A\ x\ \partial M$ )
using assms by (auto intro!: nn_integral_cmult)
finally show ?thesis .
qed

lemma Chernoff_ineq_nn_integral_ge:
  assumes s:  $s > 0$  and [measurable]:  $A \in \text{sets } M$ 
  assumes [measurable]:  $(\lambda x. f\ x * \text{indicator } A\ x) \in \text{borel\_measurable } M$ 
  shows  $\text{emeasure } M\ \{x \in A. f\ x \geq a\} \leq$ 
     $\text{ennreal } (\exp (-s * a)) * \text{nn\_integral } M\ (\lambda x. \text{ennreal } (\exp (s * f\ x)) * \text{indicator } A\ x)$ 
proof -
  define f' where  $f' = (\lambda x. f\ x * \text{indicator } A\ x)$ 
  have [measurable]:  $f' \in \text{borel\_measurable } M$ 
  using assms(3) unfolding f'_def by assumption
  have  $(\lambda x. \text{ennreal } (\exp (s * f'\ x)) * \text{indicator } A\ x) \in \text{borel\_measurable } M$ 
  by simp
  also have  $(\lambda x. \text{ennreal } (\exp (s * f'\ x)) * \text{indicator } A\ x) =$ 
     $(\lambda x. \text{ennreal } (\exp (s * f\ x)) * \text{indicator } A\ x)$ 
  by (auto simp: f'_def indicator_def fun_eq_iff)
  finally have meas: ...  $\in \text{borel\_measurable } M$  .

  have  $\{x \in A. f\ x \geq a\} = \{x \in A. \text{ennreal } (\exp (-s * a)) * \text{ennreal } (\exp (s * f\ x)) \geq 1\}$ 
  using s by (auto simp: exp_minus field_simps simp flip: ennreal_mult)
  also have  $\text{emeasure } M\ \dots \leq \text{ennreal } (\exp (-s * a)) *$ 
     $(\int^+ x. \text{ennreal } (\exp (s * f\ x)) * \text{indicator } A\ x\ \partial M)$ 
  by (intro order.trans[OF nn_integral_Markov_inequality] meas) auto
  finally show ?thesis .
qed

lemma Chernoff_ineq_nn_integral_le:
  assumes s:  $s > 0$  and [measurable]:  $A \in \text{sets } M$ 
  assumes [measurable]:  $f \in \text{borel\_measurable } M$ 
  shows  $\text{emeasure } M\ \{x \in A. f\ x \leq a\} \leq$ 
     $\text{ennreal } (\exp (s * a)) * \text{nn\_integral } M\ (\lambda x. \text{ennreal } (\exp (-s * f\ x)) * \text{indicator } A\ x)$ 
  using Chernoff_ineq_nn_integral_ge[of s A M  $\lambda x. -f\ x - a$ ] assms by simp

lemma nn_integral_noteq_infinite:
  assumes g:  $g \in \text{borel\_measurable } M$  and  $\text{integral}^N\ M\ g \neq \infty$ 
  shows  $\text{AE } x \text{ in } M. g\ x \neq \infty$ 
proof (rule ccontr)
  assume c:  $\neg (\text{AE } x \text{ in } M. g\ x \neq \infty)$ 
  have  $(\text{emeasure } M)\ \{x \in \text{space } M. g\ x = \infty\} \neq 0$ 
  using c g by (auto simp add: AE_iff_null)
  then have  $0 < (\text{emeasure } M)\ \{x \in \text{space } M. g\ x = \infty\}$ 
  by (auto simp: zero_less_iff_neq_zero)
  then have  $\infty = \infty * (\text{emeasure } M)\ \{x \in \text{space } M. g\ x = \infty\}$ 

```

```

    by (auto simp: ennreal_top_eq_mult_iff)
  also have ...  $\leq (\int^+ x. \infty * \text{indicator } \{x \in \text{space } M. g \ x = \infty\} x \ \partial M)$ 
    using  $g$  by (subst nn_integral_cmult_indicator) auto
  also have ...  $\leq \text{integral}^N M \ g$ 
    using  $\text{assms}$  by (auto intro!: nn_integral_mono_AE simp: indicator_def)
  finally show  $\text{False}$ 
    using  $\langle \text{integral}^N M \ g \neq \infty \rangle$  by (auto simp: top_unique)
qed

```

lemma nn_integral_PInf :

assumes $f: f \in \text{borel\_measurable } M$ **and** $\text{not\_Inf}: \text{integral}^N M \ f \neq \infty$
shows $\text{emeasure } M \ (f - ' \{\infty\} \cap \text{space } M) = 0$

proof –

```

  have  $\infty * \text{emeasure } M \ (f - ' \{\infty\} \cap \text{space } M) = (\int^+ x. \infty * \text{indicator } (f - ' \{\infty\} \cap \text{space } M) \ x \ \partial M)$ 
    using  $f$  by (subst nn_integral_cmult_indicator) (auto simp: measurable_sets)
  also have ...  $\leq \text{integral}^N M \ f$ 
    by (auto intro!: nn_integral_mono simp: indicator_def)
  finally have  $\infty * (\text{emeasure } M) \ (f - ' \{\infty\} \cap \text{space } M) \leq \text{integral}^N M \ f$ 
    by  $\text{simp}$ 
  then show  $?thesis$ 
    using  $\text{assms}$  by (auto simp: ennreal_top_mult top_unique split: if_split_asm)
qed

```

lemma $\text{simple\_integral\_PInf}$:

$\text{simple\_function } M \ f \implies \text{integral}^S M \ f \neq \infty \implies \text{emeasure } M \ (f - ' \{\infty\} \cap \text{space } M) = 0$

by (rule nn_integral_PInf) (auto simp: $\text{nn\_integral\_eq\_simple\_integral}$ $\text{borel\_measurable\_simple\_fun}$)

lemma $\text{nn\_integral\_PInf\_AE}$:

assumes $f \in \text{borel\_measurable } M$ $\text{integral}^N M \ f \neq \infty$ **shows** $\text{AE } x \text{ in } M. f \ x \neq \infty$

proof (rule AE_I)

```

  show  $(\text{emeasure } M) \ (f - ' \{\infty\} \cap \text{space } M) = 0$ 
    by (rule  $\text{nn\_integral\_PInf}$  [OF  $\text{assms}$ ])
  show  $f - ' \{\infty\} \cap \text{space } M \in \text{sets } M$ 
    using  $\text{assms}$  by (auto intro:  $\text{borel\_measurable\_vimage}$ )

```

qed auto

lemma nn_integral_diff :

assumes $f: f \in \text{borel\_measurable } M$
and $g: g \in \text{borel\_measurable } M$
and $\text{fin}: \text{integral}^N M \ g \neq \infty$
and $\text{mono}: \text{AE } x \text{ in } M. g \ x \leq f \ x$
shows $(\int^+ x. f \ x - g \ x \ \partial M) = \text{integral}^N M \ f - \text{integral}^N M \ g$

proof –

```

  have  $\text{diff}: (\lambda x. f \ x - g \ x) \in \text{borel\_measurable } M$ 
    using  $\text{assms}$  by  $\text{auto}$ 
  have  $\text{AE } x \text{ in } M. f \ x = f \ x - g \ x + g \ x$ 

```

```

    using diff_add_cancel_ennreal mono nn_integral_noteq_infinite[OF g fin]
  assms by auto
  then have **:  $\text{integral}^N M f = (\int^+ x. f x - g x \partial M) + \text{integral}^N M g$ 
    unfolding nn_integral_add[OF diff g, symmetric]
    by (rule nn_integral_cong_AE)
  show ?thesis unfolding **
    using fin
    by (cases rule: ennreal2_cases[of  $\int^+ x. f x - g x \partial M \text{integral}^N M g$ ]) auto
qed

```

```

lemma nn_integral_mult_bounded_inf:
  assumes f:  $f \in \text{borel\_measurable } M$   $(\int^+ x. f x \partial M) < \infty$  and c:  $c \neq \infty$  and
  ae:  $AE x \text{ in } M. g x \leq c * f x$ 
  shows  $(\int^+ x. g x \partial M) < \infty$ 
proof -
  have  $(\int^+ x. g x \partial M) \leq (\int^+ x. c * f x \partial M)$ 
    by (intro nn_integral_mono_AE ae)
  also have  $(\int^+ x. c * f x \partial M) < \infty$ 
    using c f by (subst nn_integral_cmult) (auto simp: ennreal_mult_less_top
top_unique not_less)
  finally show ?thesis .
qed

```

Fatou's lemma: convergence theorem on limes inferior

```

lemma nn_integral_monotone_convergence_INF_AE':
  assumes f:  $\bigwedge i. AE x \text{ in } M. f (Suc i) x \leq f i x$  and [measurable]:  $\bigwedge i. f i \in \text{borel\_measurable } M$ 
  and *:  $(\int^+ x. f 0 x \partial M) < \infty$ 
  shows  $(\int^+ x. (\text{INF } i. f i x) \partial M) = (\text{INF } i. \text{integral}^N M (f i))$ 
proof (rule ennreal_minus_cancel)
  have  $\text{integral}^N M (f 0) - (\int^+ x. (\text{INF } i. f i x) \partial M) = (\int^+ x. f 0 x - (\text{INF } i. f i x) \partial M)$ 
  proof (rule nn_integral_diff[symmetric])
    have  $(\int^+ x. (\text{INF } i. f i x) \partial M) \leq (\int^+ x. f 0 x \partial M)$ 
      by (intro nn_integral_mono_INF_lower) simp
    with * show  $(\int^+ x. (\text{INF } i. f i x) \partial M) \neq \infty$ 
      by simp
  qed (auto intro: INF_lower)
  also have  $\dots = (\int^+ x. (\text{SUP } i. f 0 x - f i x) \partial M)$ 
    by (simp add: ennreal_INF_const_minus)
  also have  $\dots = (\text{SUP } i. (\int^+ x. f 0 x - f i x \partial M))$ 
  proof (intro nn_integral_monotone_convergence_SUP_AE)
    show  $AE x \text{ in } M. f 0 x - f i x \leq f 0 x - f (Suc i) x$  for i
      using f[of i] by eventually_elim (auto simp: ennreal_mono_minus)
  qed simp
  also have  $\dots = (\text{SUP } i. \text{nn\_integral } M (f 0) - (\int^+ x. f i x \partial M))$ 
  proof (subst nn_integral_diff[symmetric])
    fix i
    have dec:  $AE x \text{ in } M. \forall i. f (Suc i) x \leq f i x$ 

```

```

    unfolding AE_all_countable using f by auto
  then show AE x in M. f i x ≤ f 0 x
    using dec by eventually_elim (auto intro: lift_Suc_antimono_le[of λi. f i x
0 i for x])
  then have (∫+ x. f i x ∂M) ≤ (∫+ x. f 0 x ∂M)
    by (rule nn_integral_mono_AE)
  with * show (∫+ x. f i x ∂M) ≠ ∞
    by simp
qed (insert f, auto simp: decseq_def le_fun_def)
finally show integralN M (f 0) - (∫+ x. (INF i. f i x) ∂M) =
  integralN M (f 0) - (INF i. ∫+ x. f i x ∂M)
  by (simp add: ennreal_INF_const_minus)
qed (insert *, auto intro!: nn_integral_mono intro: INF_lower)

```

```

theorem nn_integral_monotone_convergence_INF_AE:
  fixes f :: nat ⇒ 'a ⇒ ennreal
  assumes f: ⋀i. AE x in M. f (Suc i) x ≤ f i x
    and [measurable]: ⋀i. f i ∈ borel_measurable M
    and fin: (∫+ x. f i x ∂M) < ∞
  shows (∫+ x. (INF i. f i x) ∂M) = (INF i. integralN M (f i))
proof -
  { fix f :: nat ⇒ ennreal and j assume decseq f
    then have (INF i. f i) = (INF i. f (i + j))
      apply (intro INF_eq)
      apply (rule_tac x=i in bexI)
      apply (auto simp: decseq_def le_fun_def)
      done }
  note INF_shift = this
  have mono: AE x in M. ∀i. f (Suc i) x ≤ f i x
    using f by (auto simp: AE_all_countable)
  then have AE x in M. (INF i. f i x) = (INF n. f (n + i) x)
    by eventually_elim (auto intro!: decseq_SucI INF_shift)
  then have (∫+ x. (INF i. f i x) ∂M) = (∫+ x. (INF n. f (n + i) x) ∂M)
    by (rule nn_integral_cong_AE)
  also have ... = (INF n. (∫+ x. f (n + i) x ∂M))
    by (rule nn_integral_monotone_convergence_INF_AE') (insert assms, auto)
  also have ... = (INF n. (∫+ x. f n x ∂M))
    by (intro INF_shift[symmetric] decseq_SucI nn_integral_mono_AE f)
  finally show ?thesis .
qed

```

```

lemma nn_integral_monotone_convergence_INF_decseq:
  assumes f: decseq f and *: ⋀i. f i ∈ borel_measurable M (∫+ x. f i x ∂M) <
∞
  shows (∫+ x. (INF i. f i x) ∂M) = (INF i. integralN M (f i))
    using nn_integral_monotone_convergence_INF_AE[of f M i, OF _ *] f by
(auto simp: decseq_Suc_iff le_fun_def)

```

```

theorem nn_integral_liminf:

```

```

fixes  $u :: \text{nat} \Rightarrow 'a \Rightarrow \text{ennreal}$ 
assumes  $u: \bigwedge i. u\ i \in \text{borel\_measurable } M$ 
shows  $(\int^+ x. \liminf (\lambda n. u\ n\ x) \partial M) \leq \liminf (\lambda n. \text{integral}^N M (u\ n))$ 
proof -
  have  $(\int^+ x. \liminf (\lambda n. u\ n\ x) \partial M) = (\text{SUP } n. \int^+ x. (\text{INF } i \in \{n..\}. u\ i\ x) \partial M)$ 
    unfolding  $\text{liminf\_SUP\_INF}$  using  $u$ 
    by ( $\text{intro } nn\_integral\_monotone\_convergence\_SUP\_AE$ )
    ( $\text{auto intro! : AE\_I2 intro : INF\_greatest INF\_superset\_mono}$ )
  also have  $\dots \leq \liminf (\lambda n. \text{integral}^N M (u\ n))$ 
    by ( $\text{auto simp : liminf\_SUP\_INF intro! : SUP\_mono INF\_greatest nn\_integral\_mono}$ 
 $\text{INF\_lower}$ )
  finally show ?thesis .
qed

```

```

theorem  $nn\_integral\_limsup$ :
  fixes  $u :: \text{nat} \Rightarrow 'a \Rightarrow \text{ennreal}$ 
  assumes  $[\text{measurable}] : \bigwedge i. u\ i \in \text{borel\_measurable } M$   $w \in \text{borel\_measurable } M$ 
  assumes  $\text{bounds} : \bigwedge i. AE\ x\ \text{in } M. u\ i\ x \leq w\ x$  and  $w : (\int^+ x. w\ x \partial M) < \infty$ 
  shows  $\limsup (\lambda n. \text{integral}^N M (u\ n)) \leq (\int^+ x. \limsup (\lambda n. u\ n\ x) \partial M)$ 
proof -
  have  $\text{bnd} : AE\ x\ \text{in } M. \forall i. u\ i\ x \leq w\ x$ 
    using  $\text{bounds}$  by ( $\text{auto simp : AE\_all\_countable}$ )
  then have  $(\int^+ x. (\text{SUP } n. u\ n\ x) \partial M) \leq (\int^+ x. w\ x \partial M)$ 
    by ( $\text{auto intro! : nn\_integral\_mono\_AE elim : eventually\_mono intro : SUP\_least}$ )
  then have  $(\int^+ x. \limsup (\lambda n. u\ n\ x) \partial M) = (\text{INF } n. \int^+ x. (\text{SUP } i \in \{n..\}. u\ i\ x) \partial M)$ 
    unfolding  $\text{limsup\_INF\_SUP}$  using  $\text{bnd } w$ 
    by ( $\text{intro } nn\_integral\_monotone\_convergence\_INF\_AE'$ )
    ( $\text{auto intro! : AE\_I2 intro : SUP\_least SUP\_subset\_mono}$ )
  also have  $\dots \geq \limsup (\lambda n. \text{integral}^N M (u\ n))$ 
    by ( $\text{auto simp : limsup\_INF\_SUP intro! : INF\_mono SUP\_least exI nn\_integral\_mono}$ 
 $\text{SUP\_upper}$ )
  finally (xtrans) show ?thesis .
qed

```

```

lemma  $nn\_integral\_LIMSEQ$ :
  assumes  $f : \text{incseq } f \bigwedge i. f\ i \in \text{borel\_measurable } M$ 
  and  $u : \bigwedge x. (\lambda i. f\ i\ x) \longrightarrow u\ x$ 
  shows  $(\lambda n. \text{integral}^N M (f\ n)) \longrightarrow \text{integral}^N M u$ 
proof -
  have  $(\lambda n. \text{integral}^N M (f\ n)) \longrightarrow (\text{SUP } n. \text{integral}^N M (f\ n))$ 
    using  $f$  by ( $\text{intro LIMSEQ\_SUP [of } \lambda n. \text{integral}^N M (f\ n)] \text{ incseq\_nn\_integral}$ )
  also have  $(\text{SUP } n. \text{integral}^N M (f\ n)) = \text{integral}^N M (\lambda x. \text{SUP } n. f\ n\ x)$ 
    using  $f$  by ( $\text{intro nn\_integral\_monotone\_convergence\_SUP [symmetric]}$ )
  also have  $\text{integral}^N M (\lambda x. \text{SUP } n. f\ n\ x) = \text{integral}^N M (\lambda x. u\ x)$ 
    using  $f$  by ( $\text{subst LIMSEQ\_SUP [THEN LIMSEQ\_unique, OF } \_ u] (\text{auto}$ 
 $\text{simp : incseq\_def le\_fun\_def})$ )
  finally show ?thesis .
qed

```

theorem *nn_integral_dominated_convergence*:

assumes [*measurable*]:

$\bigwedge i. u\ i \in \text{borel\_measurable } M\ u' \in \text{borel\_measurable } M\ w \in \text{borel\_measurable } M$

and *bound*: $\bigwedge j. \text{AE } x \text{ in } M. u\ j\ x \leq w\ x$

and *w*: $(\int^+ x. w\ x\ \partial M) < \infty$

and *u'*: $\text{AE } x \text{ in } M. (\lambda i. u\ i\ x) \longrightarrow u'\ x$

shows $(\lambda i. (\int^+ x. u\ i\ x\ \partial M)) \longrightarrow (\int^+ x. u'\ x\ \partial M)$

proof –

have $\text{limsup } (\lambda n. \text{integral}^N M (u\ n)) \leq (\int^+ x. \text{limsup } (\lambda n. u\ n\ x)\ \partial M)$

by (*intro nn_integral_limsup[OF _ bound w]*) *auto*

moreover have $(\int^+ x. \text{limsup } (\lambda n. u\ n\ x)\ \partial M) = (\int^+ x. u'\ x\ \partial M)$

using *u'* **by** (*intro nn_integral_cong_AE, eventually_elim*) (*metis tendsto_iff_Liminf_eq_Limsup sequentially_bot*)

moreover have $(\int^+ x. \text{liminf } (\lambda n. u\ n\ x)\ \partial M) = (\int^+ x. u'\ x\ \partial M)$

using *u'* **by** (*intro nn_integral_cong_AE, eventually_elim*) (*metis tendsto_iff_Liminf_eq_Limsup sequentially_bot*)

moreover have $(\int^+ x. \text{liminf } (\lambda n. u\ n\ x)\ \partial M) \leq \text{liminf } (\lambda n. \text{integral}^N M (u\ n))$

by (*intro nn_integral_liminf*) *auto*

moreover have $\text{liminf } (\lambda n. \text{integral}^N M (u\ n)) \leq \text{limsup } (\lambda n. \text{integral}^N M (u\ n))$

by (*intro Liminf_le_Limsup sequentially_bot*)

ultimately show *?thesis*

by (*intro Liminf_eq_Limsup*) *auto*

qed

lemma *inf_continuous_nn_integral[order_continuous_intros]*:

assumes *f*: $\bigwedge y. \text{inf\_continuous } (f\ y)$

assumes [*measurable*]: $\bigwedge x. (\lambda y. f\ y\ x) \in \text{borel\_measurable } M$

assumes *bnd*: $\bigwedge x. (\int^+ y. f\ y\ x\ \partial M) \neq \infty$

shows $\text{inf\_continuous } (\lambda x. (\int^+ y. f\ y\ x\ \partial M))$

unfolding *inf_continuous_def*

proof *safe*

fix *C* :: *nat* $\Rightarrow$ 'b **assume** *C*: *decseq C*

then show $(\int^+ y. f\ y\ (\text{Inf } (C\ ' \text{UNIV}))\ \partial M) = (\text{INF } i. \int^+ y. f\ y\ (C\ i)\ \partial M)$

using *inf_continuous_mono[OF f]* *bnd*

by (*auto simp add: inf_continuousD[OF f C] fun_eq_iff antimono_def mono_def le_fun_def less_top*)

intro!: *nn_integral_monotone_convergence_INF_decseq*)

qed

lemma *nn_integral_null_set*:

assumes *N* $\in \text{null\_sets } M$ **shows** $(\int^+ x. u\ x * \text{indicator } N\ x\ \partial M) = 0$

proof –

have $(\int^+ x. u\ x * \text{indicator } N\ x\ \partial M) = (\int^+ x. 0\ \partial M)$

proof (*intro nn_integral_cong_AE AE_I*)

show $\{x \in \text{space } M. u\ x * \text{indicator } N\ x \neq 0\} \subseteq N$


```

    by (auto simp: indicator_def)
  show (emeasure M) N = 0 N ∈ sets M
    using assms by auto
qed
then show ?thesis by simp
qed

lemma nn_integral_0_iff:
  assumes u [measurable]: u ∈ borel_measurable M
  shows  $\text{integral}^N M u = 0 \longleftrightarrow \text{emeasure } M \{x \in \text{space } M. u x \neq 0\} = 0$ 
    (is  $\_ \longleftrightarrow (\text{emeasure } M) ?A = 0$ )
proof -
  have u_eq:  $(\int^+ x. u x * \text{indicator } ?A x \partial M) = \text{integral}^N M u$ 
    by (auto intro!: nn_integral_cong simp: indicator_def)
  show ?thesis
  proof
    assume (emeasure M) ?A = 0
    with nn_integral_null_set[of ?A M u] u
    show  $\text{integral}^N M u = 0$  by (simp add: u_eq null_sets_def)
  next
    assume *:  $\text{integral}^N M u = 0$ 
    let ?M =  $\lambda n. \{x \in \text{space } M. 1 \leq \text{real } (n::\text{nat}) * u x\}$ 
    have 0 = (SUP n. (emeasure M) (?M n ∩ ?A))
    proof -
      { fix n :: nat
        have emeasure M {x ∈ ?A. 1 ≤ of_nat n * u x} ≤
          of_nat n *  $\int^+ x. u x * \text{indicator } ?A x \partial M$ 
          by (intro nn_integral_Markov_inequality) auto
        also have {x ∈ ?A. 1 ≤ of_nat n * u x} = (?M n ∩ ?A)
          by (auto simp: ennreal_of_nat_eq_real_of_nat u_eq *)
        finally have emeasure M (?M n ∩ ?A) ≤ 0
          by (simp add: ennreal_of_nat_eq_real_of_nat u_eq *)
        moreover have 0 ≤ (emeasure M) (?M n ∩ ?A) using u by auto
        ultimately have (emeasure M) (?M n ∩ ?A) = 0 by auto }
      thus ?thesis by simp
    }
  qed
  also have ... = (emeasure M) ( $\bigcup n. ?M n \cap ?A$ )
  proof (safe intro!: SUP_emeasure_incseq)
    fix n show ?M n ∩ ?A ∈ sets M
      using u by (auto intro!: sets.Int)
  next
    show incseq ( $\lambda n. \{x \in \text{space } M. 1 \leq \text{real } n * u x\} \cap \{x \in \text{space } M. u x \neq 0\}$ )
  proof (safe intro!: incseq_SucI)
    fix n :: nat and x
    assume *: 1 ≤ real n * u x
    also have real n * u x ≤ real (Suc n) * u x
      by (auto intro!: mult_right_mono)
    finally show 1 ≤ real (Suc n) * u x by auto
  qed

```

```

    qed
  qed
  also have ... = (emeasure M) {x ∈ space M. 0 < u x}
  proof (safe intro!: arg_cong[where f=(emeasure M)])
    fix x assume 0 < u x and [simp, intro]: x ∈ space M
    show x ∈ (⋃ n. ?M n ∩ ?A)
    proof (cases u x rule: ennreal_cases)
      case (real r) with ⟨0 < u x⟩ have 0 < r by auto
      obtain j :: nat where 1 / r ≤ real j using real_arch_simple ..
      hence 1 / r * r ≤ real j * r unfolding mult_le_cancel_right using ⟨0 <
r⟩ by auto
      hence 1 ≤ real j * r using real ⟨0 < r⟩ by auto
      thus ?thesis using ⟨0 < r⟩ real
        by (auto simp: ennreal_of_nat_eq_real_of_nat ennreal_1[symmetric]
ennreal_mult[symmetric]
simp del: ennreal_1)
    qed (insert ⟨0 < u x⟩, auto simp: ennreal_mult_top)
    qed (auto simp: zero_less_iff_neq_zero)
    finally show emeasure M ?A = 0
    by (simp add: zero_less_iff_neq_zero)
  qed
qed

```

```

lemma nn_integral_0_iff_AE:
  assumes u: u ∈ borel_measurable M
  shows integralN M u = 0 ⟷ (AE x in M. u x = 0)
proof -
  have sets: {x ∈ space M. u x ≠ 0} ∈ sets M
  using u by auto
  show integralN M u = 0 ⟷ (AE x in M. u x = 0)
  using nn_integral_0_iff[of u] AE_iff_null[OF sets] u by auto
qed

```

```

lemma AE_iff_nn_integral:
  {x ∈ space M. P x} ∈ sets M ⟹ (AE x in M. P x) ⟷ integralN M (indicator
{x. ¬ P x}) = 0
  by (subst nn_integral_0_iff_AE) (auto simp: indicator_def[abs_def])

```

```

lemma nn_integral_less:
  assumes [measurable]: f ∈ borel_measurable M g ∈ borel_measurable M
  assumes f: (∫+x. f x ∂M) ≠ ∞
  assumes ord: AE x in M. f x ≤ g x ∧ (AE x in M. g x ≤ f x)
  shows (∫+x. f x ∂M) < (∫+x. g x ∂M)
proof -
  have 0 < (∫+x. g x - f x ∂M)
  proof (intro order_le_neq_trans notI)
    assume 0 = (∫+x. g x - f x ∂M)
    then have AE x in M. g x - f x = 0
    using nn_integral_0_iff_AE[of λx. g x - f x M] by simp
  qed

```

```

    with ord(1) have AE x in M. g x ≤ f x
      by eventually_elim (auto simp: ennreal_minus_eq_0)
    with ord show False
      by simp
  qed simp
  also have ... = (∫+x. g x ∂M) - (∫+x. f x ∂M)
    using f by (subst nn_integral_diff) (auto simp: ord)
  finally show ?thesis
    using f by (auto dest!: ennreal_minus_pos_iff[rotated] simp: less_top)
  qed

lemma nn_integral_subalgebra:
  assumes f: f ∈ borel_measurable N
  and N: sets N ⊆ sets M space N = space M ∧ A. A ∈ sets N ⇒ emeasure N
  A = emeasure M A
  shows integralN N f = integralN M f
proof -
  have [simp]: ∧ f :: 'a ⇒ ennreal. f ∈ borel_measurable N ⇒ f ∈ borel_measurable
  M
    using N by (auto simp: measurable_def)
  have [simp]: ∧ P. (AE x in N. P x) ⇒ (AE x in M. P x)
    using N by (auto simp add: eventually_ae_filter null_sets_def subset_eq)
  have [simp]: ∧ A. A ∈ sets N ⇒ A ∈ sets M
    using N by auto
  from f show ?thesis
    apply induct
    apply (simp_all add: nn_integral_add nn_integral_cmult nn_integral_monotone_convergence_SUP
  N image_comp)
    apply (auto intro!: nn_integral_cong cong: nn_integral_cong simp: N(2)[symmetric])
    done
  qed

lemma nn_integral_nat_function:
  fixes f :: 'a ⇒ nat
  assumes f ∈ measurable M (count_space UNIV)
  shows (∫+x. of_nat (f x) ∂M) = (∑ t. emeasure M {x ∈ space M. t < f x})
proof -
  define F where F i = {x ∈ space M. i < f x} for i
  with assms have [measurable]: ∧ i. F i ∈ sets M
    by auto

  { fix x assume x ∈ space M
    have (λ i. if i < f x then 1 else 0) sums (of_nat (f x)::real)
      using sums_if_finite[of λ i. i < f x λ_. 1::real] by simp
    then have (λ i. ennreal (if i < f x then 1 else 0)) sums of_nat(f x)
      unfolding ennreal_of_nat_eq_real_of_nat
      by (subst sums_ennreal) auto
    moreover have ∧ i. ennreal (if i < f x then 1 else 0) = indicator (F i) x
      using ⟨x ∈ space M⟩ by (simp add: one_ennreal_def F_def)
  }

```

ultimately have $of\_nat (f x) = (\sum i. indicator (F i) x :: ennreal)$
 by (simp add: sums_iff) }
 then have $(\int^+ x. of\_nat (f x) \partial M) = (\int^+ x. (\sum i. indicator (F i) x) \partial M)$
 by (simp cong: nn_integral_cong)
 also have $\dots = (\sum i. emeasure M (F i))$
 by (simp add: nn_integral_suminf)
 finally show ?thesis
 by (simp add: F_def)
 qed

theorem nn_integral_lfp:
 assumes sets[simp]: $\bigwedge s. sets (M s) = sets N$
 assumes f: sup_continuous f
 assumes g: sup_continuous g
 assumes meas: $\bigwedge F. F \in borel\_measurable N \implies f F \in borel\_measurable N$
 assumes step: $\bigwedge F s. F \in borel\_measurable N \implies integral^N (M s) (f F) = g$
 ($\lambda s. integral^N (M s) F$) s
 shows $(\int^+ \omega. lfp f \omega \partial M s) = lfp g s$
proof (subst lfp_transfer_bounded[where $\alpha = \lambda F s. \int^+ x. F x \partial M s$ and $g = g$ and
 $f = f$ and $P = \lambda f. f \in borel\_measurable N$, symmetric])
 fix $C :: nat \Rightarrow 'b \Rightarrow ennreal$ assume incseq $C \bigwedge i. C i \in borel\_measurable N$
 then show $(\lambda s. \int^+ x. (SUP i. C i) x \partial M s) = (SUP i. (\lambda s. \int^+ x. C i x \partial M s))$
 unfolding SUP_apply[abs_def]
 by (subst nn_integral_monotone_convergence_SUP)
 (auto simp: mono_def fun_eq_iff intro!: arg_cong2[where $f = emeasure$]
 cong: measurable_cong_sets)
qed (auto simp add: step le_fun_def SUP_apply[abs_def] bot_fun_def bot_ennreal
 intro!: meas f g)

theorem nn_integral_gfp:
 assumes sets[simp]: $\bigwedge s. sets (M s) = sets N$
 assumes f: inf_continuous f and g: inf_continuous g
 assumes meas: $\bigwedge F. F \in borel\_measurable N \implies f F \in borel\_measurable N$
 assumes bound: $\bigwedge F s. F \in borel\_measurable N \implies (\int^+ x. f F x \partial M s) < \infty$
 assumes non_zero: $\bigwedge s. emeasure (M s) (space (M s)) \neq 0$
 assumes step: $\bigwedge F s. F \in borel\_measurable N \implies integral^N (M s) (f F) = g$
 ($\lambda s. integral^N (M s) F$) s
 shows $(\int^+ \omega. gfp f \omega \partial M s) = gfp g s$
proof (subst gfp_transfer_bounded[where $\alpha = \lambda F s. \int^+ x. F x \partial M s$ and $g = g$ and
 $f = f$
 and $P = \lambda F. F \in borel\_measurable N \wedge (\forall s. (\int^+ x. F x \partial M s) < \infty)$, symmetric])
 fix $C :: nat \Rightarrow 'b \Rightarrow ennreal$ assume decseq $C \bigwedge i. C i \in borel\_measurable N \wedge$
 ($\forall s. integral^N (M s) (C i) < \infty$)
 then show $(\lambda s. \int^+ x. (INF i. C i) x \partial M s) = (INF i. (\lambda s. \int^+ x. C i x \partial M s))$
 unfolding INF_apply[abs_def]
 by (subst nn_integral_monotone_convergence_INF_decseq)
 (auto simp: mono_def fun_eq_iff intro!: arg_cong2[where $f = emeasure$]
 cong: measurable_cong_sets)

```

next
  show  $\bigwedge x. g\ x \leq (\lambda s. \text{integral}^N\ (M\ s)\ (f\ \text{top}))$ 
  by (subst step)
    (auto simp add: top_fun_def less_le non_zero le_fun_def ennreal_top_mult
      cong del: if_weak_cong intro!: monoD[OF inf_continuous_mono[OF g],
        THEN le_funD])
next
  fix C assume  $\bigwedge i::\text{nat}. C\ i \in \text{borel\_measurable}\ N \wedge (\forall s. \text{integral}^N\ (M\ s)\ (C\ i) < \infty)$  decseq C
  with bound show  $\text{Inf}\ (C\ ` UNIV) \in \text{borel\_measurable}\ N \wedge (\forall s. \text{integral}^N\ (M\ s)\ (\text{Inf}\ (C\ ` UNIV)) < \infty)$ 
  unfolding INF_apply[abs_def]
  by (subst nn_integral_monotone_convergence_INF_decseq)
    (auto simp: INF_less_iff cong: measurable_cong_sets intro!: borel_measurable_INF)
next
  show  $\bigwedge x. x \in \text{borel\_measurable}\ N \wedge (\forall s. \text{integral}^N\ (M\ s)\ x < \infty) \implies$ 
     $(\lambda s. \text{integral}^N\ (M\ s)\ (f\ x)) = g\ (\lambda s. \text{integral}^N\ (M\ s)\ x)$ 
  by (subst step) auto
qed (insert bound, auto simp add: le_fun_def INF_apply[abs_def] top_fun_def
  intro!: meas f g)

```

Cauchy–Schwarz inequality for integral^N

```

lemma sum_of_squares_ge_ennreal:
  fixes a b :: ennreal
  shows  $2 * a * b \leq a^2 + b^2$ 
proof (cases a; cases b)
  fix x y
  assume xy:  $x \geq 0\ y \geq 0$  and [simp]:  $a = \text{ennreal}\ x\ b = \text{ennreal}\ y$ 
  have  $0 \leq (x - y)^2$ 
  by simp
  also have  $\dots = x^2 + y^2 - 2 * x * y$ 
  by (simp add: algebra_simps power2_eq_square)
  finally have  $2 * x * y \leq x^2 + y^2$ 
  by simp
  hence  $\text{ennreal}\ (2 * x * y) \leq \text{ennreal}\ (x^2 + y^2)$ 
  by (intro ennreal_leI)
  thus ?thesis using xy
  by (simp add: ennreal_mult ennreal_power)
qed auto

```

```

lemma Cauchy_Schwarz_nn_integral:
  assumes [measurable]:  $f \in \text{borel\_measurable}\ M\ g \in \text{borel\_measurable}\ M$ 
  shows  $(\int^{+x}. f\ x * g\ x\ \partial M)^2 \leq (\int^{+x}. f\ x^{\wedge} 2\ \partial M) * (\int^{+x}. g\ x^{\wedge} 2\ \partial M)$ 
proof (cases  $(\int^{+x}. f\ x * g\ x\ \partial M) = 0$ )
  case False
  define F where  $F = \text{nn\_integral}\ M\ (\lambda x. f\ x^{\wedge} 2)$ 
  define G where  $G = \text{nn\_integral}\ M\ (\lambda x. g\ x^{\wedge} 2)$ 
  from False have  $\neg(AE\ x\ \text{in}\ M. f\ x = 0 \vee g\ x = 0)$ 
  by (auto simp: nn_integral_0_iff_AE)

```

hence $\neg(AE\ x\ in\ M.\ f\ x = 0)$ and $\neg(AE\ x\ in\ M.\ g\ x = 0)$
 by (auto intro: AE_disjI1 AE_disjI2)
 hence $nz: F \neq 0\ G \neq 0$
 by (auto simp: nn_integral_0_iff_AE F_def G_def)

show ?thesis

proof (cases $F = \infty \vee G = \infty$)

case True

thus ?thesis using nz

by (auto simp: F_def G_def)

next

case False

define F' where $F' = \text{ennreal}(\text{sqrt}(\text{enn2real}\ F))$

define G' where $G' = \text{ennreal}(\text{sqrt}(\text{enn2real}\ G))$

from False have $\text{fin}: F < \text{top}\ G < \text{top}$

by (simp_all add: top.not_eq_extremum)

have $F'_{\text{sqr}}: F'^2 = F$

using False by (cases F) (auto simp: F'_def ennreal_power)

have $G'_{\text{sqr}}: G'^2 = G$

using False by (cases G) (auto simp: G'_def ennreal_power)

have $nz': F' \neq 0\ G' \neq 0$ and $\text{fin}': F' \neq \infty\ G' \neq \infty$

using F'_sqr G'_sqr nz fin by auto

from fin' have $\text{fin}'': F' < \text{top}\ G' < \text{top}$

by (auto simp: top.not_eq_extremum)

have $2 * (F' / F') * (G' / G') * (\int^+ x. f\ x * g\ x\ \partial M) =$

$F' * G' * (\int^+ x. 2 * (f\ x / F') * (g\ x / G')\ \partial M)$

using nz' fin''

by (simp add: divide_ennreal_def algebra_simps ennreal_inverse_mult flip:
nn_integral_cmult)

also have $F' / F' = 1$

using nz' fin'' by simp

also have $G' / G' = 1$

using nz' fin'' by simp

also have $2 * 1 * 1 = (2 :: \text{ennreal})$ by simp

also have $F' * G' * (\int^+ x. 2 * (f\ x / F') * (g\ x / G')\ \partial M) \leq$

$F' * G' * (\int^+ x. (f\ x / F')^2 + (g\ x / G')^2\ \partial M)$

by (intro mult_left_mono nn_integral_mono sum_of_squares_ge_ennreal)

auto

also have $\dots = F' * G' * (F / F'^2 + G / G'^2)$ using nz

by (auto simp: nn_integral_add algebra_simps nn_integral_divide F_def
G_def)

also have $F / F'^2 = 1$

using nz F'_sqr fin by simp

also have $G / G'^2 = 1$

using nz G'_sqr fin by simp

also have $F' * G' * (1 + 1) = 2 * (F' * G')$

by (simp add: mult_ac)

finally have $(\int^+ x. f\ x * g\ x\ \partial M) \leq F' * G'$

```

    by (subst (asm) ennreal_mult_le_mult_iff) auto
  hence  $(\int^+ x. f\ x * g\ x\ \partial M)^2 \leq (F' * G')^2$ 
    by (intro power_mono_ennreal)
  also have  $\dots = F * G$ 
    by (simp add: algebra_simps F'_sqr G'_sqr)
  finally show ?thesis
    by (simp add: F_def G_def)
qed
qed auto

```

6.6.5 Integral under concrete measures

```

lemma nn_integral_mono_measure:
  assumes sets M = sets N M ≤ N shows nn_integral M f ≤ nn_integral N f
  unfolding nn_integral_def
  proof (intro SUP_subset_mono)
    note ⟨sets M = sets N⟩[simp] ⟨sets M = sets N⟩[THEN sets_eq_imp_space_eq,
    simp]
    show {g. simple_function M g ∧ g ≤ f} ⊆ {g. simple_function N g ∧ g ≤ f}
      by (simp add: simple_function_def)
    show  $\text{integral}^S M\ x \leq \text{integral}^S N\ x$  for x
      using le_measureD3[OF ⟨M ≤ N⟩]
    by (auto simp add: simple_integral_def intro!: sum_mono mult_mono)
  qed

```

```

lemma nn_integral_empty:
  assumes space M = {}
  shows nn_integral M f = 0
  proof -
    have  $(\int^+ x. f\ x\ \partial M) = (\int^+ x. 0\ \partial M)$ 
      by (rule nn_integral_cong) (simp add: assms)
    thus ?thesis by simp
  qed

```

```

lemma nn_integral_bot[simp]: nn_integral bot f = 0
  by (simp add: nn_integral_empty)

```

Distributions

```

lemma nn_integral_distr:
  assumes T: T ∈ measurable M M' and f: f ∈ borel_measurable (distr M M' T)
  shows  $\text{integral}^N (\text{distr } M\ M'\ T)\ f = (\int^+ x. f\ (T\ x)\ \partial M)$ 
  using f
  proof induct
    case (cong f g)
    with T show ?case
      apply (subst nn_integral_cong[of _ f g])
      apply simp
      apply (subst nn_integral_cong[of _ λx. f (T x) λx. g (T x)])

```

```

    apply (simp add: measurable_def Pi_iff)
    apply simp
    done
next
  case (set A)
  then have eq:  $\bigwedge x. x \in \text{space } M \implies \text{indicator } A (T x) = \text{indicator } (T - ' A \cap \text{space } M) x$ 
    by (auto simp: indicator_def)
  from set T show ?case
    by (subst nn_integral_cong[OF eq])
      (auto simp add: emeasure_distr intro!: nn_integral_indicator[symmetric]
        measurable_sets)
qed (simp_all add: measurable_compose[OF T] T nn_integral_cmult nn_integral_add
  nn_integral_monotone_convergence_SUP le_fun_def incseq_def
  image_comp)

```

Counting space

```

lemma simple_function_count_space[simp]:
  simple_function (count_space A) f  $\longleftrightarrow$  finite (f ' A)
  unfolding simple_function_def by simp

lemma nn_integral_count_space:
  assumes A: finite {a ∈ A. 0 < f a}
  shows  $\text{integral}^N (\text{count\_space } A) f = (\sum a | a \in A \wedge 0 < f a. f a)$ 
proof -
  have *:  $(\int^+ x. \max 0 (f x) \partial \text{count\_space } A) =$ 
     $(\int^+ x. (\sum a | a \in A \wedge 0 < f a. f a * \text{indicator } \{a\} x) \partial \text{count\_space } A)$ 
    by (auto intro!: nn_integral_cong
      simp add: indicator_def of_bool_def if_distrib sum.If_cases[OF A]
      max_def le_less)
  also have ... =  $(\sum a | a \in A \wedge 0 < f a. \int^+ x. f a * \text{indicator } \{a\} x \partial \text{count\_space } A)$ 
    by (subst nn_integral_sum) (simp_all add: AE_count_space less_imp_le)
  also have ... =  $(\sum a | a \in A \wedge 0 < f a. f a)$ 
    by (auto intro!: sum.cong simp: one_enreal_def[symmetric] max_def)
  finally show ?thesis by (simp add: max.absorb2)
qed

```

```

lemma nn_integral_count_space_finite:
  finite A  $\implies (\int^+ x. f x \partial \text{count\_space } A) = (\sum a \in A. f a)$ 
  by (auto intro!: sum.mono_neutral_left simp: nn_integral_count_space less_le)

```

```

lemma nn_integral_count_space':
  assumes finite A  $\bigwedge x. x \in B \implies x \notin A \implies f x = 0$   $A \subseteq B$ 
  shows  $(\int^+ x. f x \partial \text{count\_space } B) = (\sum x \in A. f x)$ 
proof -
  have  $(\int^+ x. f x \partial \text{count\_space } B) = (\sum a | a \in B \wedge 0 < f a. f a)$ 
    using assms(2,3)

```



```

  by (intro nn_integral_count_space finite_subset[OF _ ‹finite A›]) (auto simp:
less_le)
  also have ... = (∑ a∈A. f a)
  using assms by (intro sum.mono_neutral_cong_left) (auto simp: less_le)
  finally show ?thesis .
qed

```

```

lemma nn_integral_bij_count_space:
  assumes g: bij_betw g A B
  shows (∫+x. f (g x) ∂count_space A) = (∫+x. f x ∂count_space B)
  using g[THEN bij_betw_imp_funcset]
  by (subst distr_bij_count_space[OF g, symmetric])
  (auto intro!: nn_integral_distr[symmetric])

```

```

lemma nn_integral_indicator_finite:
  fixes f :: 'a ⇒ ennreal
  assumes f: finite A and [measurable]: ∧a. a ∈ A ⇒ {a} ∈ sets M
  shows (∫+x. f x * indicator A x ∂M) = (∑ x∈A. f x * emeasure M {x})
proof -
  from f have (∫+x. f x * indicator A x ∂M) = (∫+x. (∑ a∈A. f a * indicator
{a} x) ∂M)
  by (intro nn_integral_cong) (auto simp: indicator_def if_distrib[where f=λa.
x * a for x] sum.If_cases)
  also have ... = (∑ a∈A. f a * emeasure M {a})
  by (subst nn_integral_sum) auto
  finally show ?thesis .
qed

```

```

lemma nn_integral_count_space_nat:
  fixes f :: nat ⇒ ennreal
  shows (∫+i. f i ∂count_space UNIV) = (∑ i. f i)
proof -
  have (∫+i. f i ∂count_space UNIV) =
    (∫+i. (∑ j. f j * indicator {j} i) ∂count_space UNIV)
  proof (intro nn_integral_cong)
    fix i
    have f i = (∑ j∈{i}. f j * indicator {j} i)
    by simp
    also have ... = (∑ j. f j * indicator {j} i)
    by (rule suminf_finite[symmetric]) auto
    finally show f i = (∑ j. f j * indicator {j} i) .
  qed
  also have ... = (∑ j. (∫+i. f j * indicator {j} i ∂count_space UNIV))
  by (rule nn_integral_suminf) auto
  finally show ?thesis
  by simp
qed

```

```

lemma nn_integral_enat_function:

```

```

assumes  $f: f \in \text{measurable } M \text{ (count\_space UNIV)}$ 
shows  $(\int^+ x. \text{ennreal\_of\_enat } (f x) \partial M) = (\sum t. \text{emeasure } M \{x \in \text{space } M. \\ t < f x\})$ 
proof -
define  $F$  where  $F i = \{x \in \text{space } M. i < f x\}$  for  $i :: \text{nat}$ 
with  $\text{assms}$  have  $[\text{measurable}]: \bigwedge i. F i \in \text{sets } M$ 
by auto

```

```

{ fix  $x$  assume  $x \in \text{space } M$ 
  have  $(\lambda i :: \text{nat}. \text{if } i < f x \text{ then } 1 \text{ else } 0) \text{ sums } \text{ennreal\_of\_enat } (f x)$ 
    using  $\text{sums\_If\_finite}[of \lambda r. r < f x \lambda \_. 1 :: \text{ennreal}]$ 
    by  $(\text{cases } f x) (\text{simp\_all add: sums\_def of\_nat\_tendsto\_top\_ennreal})$ 
  also have  $(\lambda i. (\text{if } i < f x \text{ then } 1 \text{ else } 0)) = (\lambda i. \text{indicator } (F i) x)$ 
    using  $\langle x \in \text{space } M \rangle$  by  $(\text{simp add: one\_ennreal\_def } F\_def \text{ fun\_eq\_iff})$ 
  finally have  $\text{ennreal\_of\_enat } (f x) = (\sum i. \text{indicator } (F i) x)$ 
    by  $(\text{simp add: sums\_iff})$  }
then have  $(\int^+ x. \text{ennreal\_of\_enat } (f x) \partial M) = (\int^+ x. (\sum i. \text{indicator } (F i) x) \partial M)$ 
  by  $(\text{simp cong: nn\_integral\_cong})$ 
also have  $\dots = (\sum i. \text{emeasure } M (F i))$ 
  by  $(\text{simp add: nn\_integral\_suminf})$ 
finally show ?thesis
  by  $(\text{simp add: } F\_def)$ 
qed

```

lemma *nn_integral_count_space_nn_integral*:

```

fixes  $f :: 'i \Rightarrow 'a \Rightarrow \text{ennreal}$ 
assumes countable I and  $[\text{measurable}]: \bigwedge i. i \in I \implies f i \in \text{borel\_measurable } M$ 
shows  $(\int^+ x. \int^+ i. f i x \partial \text{count\_space } I \partial M) = (\int^+ i. \int^+ x. f i x \partial M \partial \text{count\_space } I)$ 

```

proof *cases*

```

assume finite I then show ?thesis
  by  $(\text{simp add: nn\_integral\_count\_space\_finite nn\_integral\_sum})$ 
next
assume infinite I
then have  $[\text{simp}]: I \neq \{\}$ 
by auto
note  $*$   $= \text{bij\_betw\_from\_nat\_into}[OF \langle \text{countable } I \rangle \langle \text{infinite } I \rangle]$ 
have  $**$ :  $\bigwedge f. (\bigwedge i. 0 \leq f i) \implies (\int^+ i. f i \partial \text{count\_space } I) = (\sum n. f (\text{from\_nat\_into } I n))$ 
  by  $(\text{simp add: nn\_integral\_bij\_count\_space}[\text{symmetric, OF } *] \text{ nn\_integral\_count\_space\_nat})$ 
show ?thesis
  by  $(\text{simp add: ** nn\_integral\_suminf from\_nat\_into})$ 
qed

```

lemma *of_bool_Bex_eq_nn_integral*:

```

assumes unique:  $\bigwedge x y. x \in X \implies y \in X \implies P x \implies P y \implies x = y$ 
shows  $\text{of\_bool } (\exists y \in X. P y) = (\int^+ y. \text{of\_bool } (P y) \partial \text{count\_space } X)$ 
proof cases

```

```

  assume  $\exists y \in X. P y$ 
  then obtain  $y$  where  $P y$   $y \in X$  by auto
  then show ?thesis
    by (subst nn_integral_count_space'[where A={y}]) (auto dest: unique)
qed (auto cong: nn_integral_cong_simp)

```

lemma *emeasure_UN_countable:*

```

  assumes sets[measurable]:  $\bigwedge i. i \in I \implies X i \in \text{sets } M$  and  $I[simp]: \text{countable } I$ 
  assumes disj: disjoint_family_on  $X I$ 
  shows  $\text{emeasure } M (\bigcup (X \text{ ` } I)) = (\int^+ i. \text{emeasure } M (X i) \partial \text{count\_space } I)$ 
proof -
  have eq:  $\bigwedge x. \text{indicator} (\bigcup (X \text{ ` } I)) x = \int^+ i. \text{indicator} (X i) x \partial \text{count\_space } I$ 
  proof cases
    fix  $x$  assume  $x: x \in \bigcup (X \text{ ` } I)$ 
    then obtain  $j$  where  $j: x \in X j \wedge j \in I$ 
    by auto
    with disj have  $\bigwedge i. i \in I \implies \text{indicator} (X i) x = (\text{indicator } \{j\} i :: \text{ennreal})$ 
    by (auto simp: disjoint_family_on_def split: split_indicator)
    with  $x j$  show ?thesis  $x$ 
    by (simp cong: nn_integral_cong_simp)
  qed (auto simp: nn_integral_0_iff AE)

```

```

  note sets.countable_UN'[unfolded subset_eq, measurable]
  have  $\text{emeasure } M (\bigcup (X \text{ ` } I)) = (\int^+ x. \text{indicator} (\bigcup (X \text{ ` } I)) x \partial M)$ 
  by simp
  also have  $\dots = (\int^+ i. \int^+ x. \text{indicator} (X i) x \partial M \partial \text{count\_space } I)$ 
  by (simp add: eq nn_integral_count_space_nn_integral)
  finally show ?thesis
    by (simp cong: nn_integral_cong_simp)
qed

```

lemma *emeasure_countable_singleton:*

```

  assumes sets:  $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$  and  $X: \text{countable } X$ 
  shows  $\text{emeasure } M X = (\int^+ x. \text{emeasure } M \{x\} \partial \text{count\_space } X)$ 
proof -
  have  $\text{emeasure } M (\bigcup i \in X. \{i\}) = (\int^+ x. \text{emeasure } M \{x\} \partial \text{count\_space } X)$ 
  using assms by (intro emeasure_UN_countable) (auto simp: disjoint_family_on_def)
  also have  $(\bigcup i \in X. \{i\}) = X$  by auto
  finally show ?thesis .
qed

```

lemma *measure_eqI_countable:*

```

  assumes [simp]:  $\text{sets } M = \text{Pow } A$  sets  $N = \text{Pow } A$  and  $A: \text{countable } A$ 
  assumes eq:  $\bigwedge a. a \in A \implies \text{emeasure } M \{a\} = \text{emeasure } N \{a\}$ 
  shows  $M = N$ 
proof (rule measure_eqI)
  fix  $X$  assume  $X \in \text{sets } M$ 
  then have  $X: X \subseteq A$  by auto
  moreover from  $A X$  have  $\text{countable } X$  by (auto dest: countable_subset)

```

ultimately have
 $\text{emeasure } M \ X = (\int^+ a. \text{emeasure } M \ \{a\} \ \partial \text{count\_space } X)$
 $\text{emeasure } N \ X = (\int^+ a. \text{emeasure } N \ \{a\} \ \partial \text{count\_space } X)$
 by (auto intro!: *emeasure_countable_singleton*)
 moreover have $(\int^+ a. \text{emeasure } M \ \{a\} \ \partial \text{count\_space } X) = (\int^+ a. \text{emeasure } N \ \{a\} \ \partial \text{count\_space } X)$
 using *X* by (intro *nn_integral_cong* eq) auto
 ultimately show $\text{emeasure } M \ X = \text{emeasure } N \ X$
 by *simp*
 qed *simp*

lemma *measure_eqI_countable_AE*:
 assumes [*simp*]: *sets* $M = \text{UNIV}$ *sets* $N = \text{UNIV}$
 assumes *ae*: $AE \ x \text{ in } M. \ x \in \Omega \ AE \ x \text{ in } N. \ x \in \Omega$ and [*simp*]: *countable* Ω
 assumes *eq*: $\bigwedge x. \ x \in \Omega \implies \text{emeasure } M \ \{x\} = \text{emeasure } N \ \{x\}$
 shows $M = N$
 proof (rule *measure_eqI*)
 fix *A*
 have $\text{emeasure } N \ A = \text{emeasure } N \ \{x \in \Omega. \ x \in A\}$
 using *ae* by (intro *emeasure_eq_AE*) auto
 also have $\dots = (\int^+ x. \text{emeasure } N \ \{x\} \ \partial \text{count\_space } \{x \in \Omega. \ x \in A\})$
 by (intro *emeasure_countable_singleton*) auto
 also have $\dots = (\int^+ x. \text{emeasure } M \ \{x\} \ \partial \text{count\_space } \{x \in \Omega. \ x \in A\})$
 by (intro *nn_integral_cong* eq[*symmetric*]) auto
 also have $\dots = \text{emeasure } M \ \{x \in \Omega. \ x \in A\}$
 by (intro *emeasure_countable_singleton*[*symmetric*]) auto
 also have $\dots = \text{emeasure } M \ A$
 using *ae* by (intro *emeasure_eq_AE*) auto
 finally show $\text{emeasure } M \ A = \text{emeasure } N \ A$..
 qed *simp*

lemma *nn_integral_monotone_convergence_SUP_nat*:
 fixes $f :: 'a \Rightarrow \text{nat} \Rightarrow \text{ennreal}$
 assumes *chain*: *Complete_Partial_Order.chain* ($\leq$) ($f \ ' Y$)
 and *nonempty*: $Y \neq \{\}$
 shows $(\int^+ x. (\text{SUP } i \in Y. f \ i \ x) \ \partial \text{count\_space } \text{UNIV}) = (\text{SUP } i \in Y. (\int^+ x. f \ i \ x \ \partial \text{count\_space } \text{UNIV}))$
 (is $?lhs = ?rhs$ is *integral^N ?M _ = _*)
 proof (rule *order_class.order_antisym*)
 show $?rhs \leq ?lhs$
 by (auto intro!: *SUP_least SUP_upper nn_integral_mono*)
 next
 have $\exists g. \text{incseq } g \wedge \text{range } g \subseteq (\lambda i. f \ i \ x) \ ' Y \wedge (\text{SUP } i \in Y. f \ i \ x) = (\text{SUP } i. g \ i)$ for x
 by (rule *ennreal_Sup_countable_SUP*) (*simp add: nonempty*)
 then obtain g where *incseq*: $\bigwedge x. \text{incseq } (g \ x)$
 and *range*: $\bigwedge x. \text{range } (g \ x) \subseteq (\lambda i. f \ i \ x) \ ' Y$
 and *sup*: $\bigwedge x. (\text{SUP } i \in Y. f \ i \ x) = (\text{SUP } i. g \ x \ i)$ by *moura*
 from *incseq* have *incseq'*: $\text{incseq } (\lambda i \ x. g \ x \ i)$

```

by(blast intro: incseq_SucI le_funI dest: incseq_SucD)

have ?lhs =  $\int^+ x. (SUP i. g x i) \partial ?M$  by(simp add: sup)
also have ... =  $(SUP i. \int^+ x. g x i \partial ?M)$  using incseq'
  by(rule nn_integral_monotone_convergence_SUP) simp
also have ...  $\leq (SUP i \in Y. \int^+ x. f i x \partial ?M)$ 
proof(rule SUP_least)
  fix n
  have  $\bigwedge x. \exists i. g x n = f i x \wedge i \in Y$  using range by blast
  then obtain I where  $I: \bigwedge x. g x n = f (I x) x \wedge x. I x \in Y$  by moura

  have  $(\int^+ x. g x n \partial count\_space UNIV) = (\sum x. g x n)$ 
  by(rule nn_integral_count_space_nat)
  also have ... =  $(SUP m. \sum x < m. g x n)$ 
  by(rule suminf_eq_SUP)
  also have ...  $\leq (SUP i \in Y. \int^+ x. f i x \partial ?M)$ 
  proof(rule SUP_mono)
    fix m
    show  $\exists m' \in Y. (\sum x < m. g x n) \leq (\int^+ x. f m' x \partial ?M)$ 
    proof(cases  $m > 0$ )
      case False
      thus ?thesis using nonempty by auto
    next
      case True
      let ?Y =  $I^{-1} \{..<m\}$ 
      have  $f^{-1} ?Y \subseteq f^{-1} Y$  using I by auto
      with chain have chain':  $Complete\_Partial\_Order.chain (\leq) (f^{-1} ?Y)$  by(rule
chain_subset)
      hence  $Sup (f^{-1} ?Y) \in f^{-1} ?Y$ 
      by(rule ccpo_class.in_chain_finite)(auto simp add: True lessThan_empty_iff)
      then obtain m' where  $m' < m$  and m':  $(SUP i \in ?Y. f i) = f (I m')$  by
auto

      have  $I m' \in Y$  using I by blast
      have  $(\sum x < m. g x n) \leq (\sum x < m. f (I m') x)$ 
      proof(rule sum_mono)
        fix x
        assume  $x \in \{..<m\}$ 
        hence  $x < m$  by simp
        have  $g x n = f (I x) x$  by(simp add: I)
        also have ...  $\leq (SUP i \in ?Y. f i) x$  unfolding Sup_fun_def image_image
          using  $\langle x \in \{..<m\} \rangle$  by (rule Sup_upper [OF imageI])
        also have ... =  $f (I m') x$  unfolding m' by simp
        finally show  $g x n \leq f (I m') x$  .
      qed
      also have ...  $\leq (SUP m. (\sum x < m. f (I m') x))$ 
      by(rule SUP_upper) simp
      also have ... =  $(\sum x. f (I m') x)$ 
      by(rule suminf_eq_SUP[symmetric])
      also have ... =  $(\int^+ x. f (I m') x \partial ?M)$ 

```

```

      by(rule nn_integral_count_space_nat[symmetric])
      finally show ?thesis using ⟨I m' ∈ Y⟩ by blast
    qed
  qed
  finally show (∫+ x. g x n ∂count_space UNIV) ≤ ... .
  qed
  finally show ?lhs ≤ ?rhs .
  qed

lemma power_series_tendsto_at_left:
  assumes nonneg:  $\bigwedge i. 0 \leq f i$  and summable:  $\bigwedge z. 0 \leq z \implies z < 1 \implies \text{summable}$ 
     $(\lambda n. f n * z^n)$ 
  shows  $((\lambda z. \text{ennreal} (\sum n. f n * z^n)) \longrightarrow (\sum n. \text{ennreal} (f n)))$  (at_left
     $(1::\text{real}))$ 
  proof (intro tendsto_at_left_sequentially)
    show  $0 < (1::\text{real})$  by simp
    fix  $S :: \text{nat} \Rightarrow \text{real}$  assume  $S: \bigwedge n. S n < 1 \wedge \bigwedge n. 0 < S n \longrightarrow 1 \text{ incseq } S$ 
    then have  $S\_nonneg: \bigwedge i. 0 \leq S i$  by (auto intro: less_imp_le)

    have  $(\lambda i. (\int^{+n}. f n * S i^n \partial \text{count\_space UNIV})) \longrightarrow (\int^{+n}. \text{ennreal} (f n) \partial \text{count\_space UNIV})$ 
  proof (rule nn_integral_LIMSEQ)
    show incseq  $(\lambda i n. \text{ennreal} (f n * S i^n))$ 
      using  $S$  by (auto intro!: mult_mono power_mono nonneg ennreal_leI
        simp: incseq_def le_fun_def less_imp_le)
    fix  $n$  have  $(\lambda i. \text{ennreal} (f n * S i^n)) \longrightarrow \text{ennreal} (f n * 1^n)$ 
      by (intro tendsto_intros tendsto_ennrealI S)
    then show  $(\lambda i. \text{ennreal} (f n * S i^n)) \longrightarrow \text{ennreal} (f n)$ 
      by simp
  qed (auto simp:  $S\_nonneg$  intro!: mult_nonneg_nonneg nonneg)
  also have  $(\lambda i. (\int^{+n}. f n * S i^n \partial \text{count\_space UNIV})) = (\lambda i. \sum n. f n * S i^n)$ 
    by (subst nn_integral_count_space_nat)
      (intro ext suminf_ennreal2 mult_nonneg_nonneg nonneg  $S\_nonneg$ 
        zero_le_power summable S)+
  also have  $(\int^{+n}. \text{ennreal} (f n) \partial \text{count\_space UNIV}) = (\sum n. \text{ennreal} (f n))$ 
    by (simp add: nn_integral_count_space_nat nonneg)
  finally show  $(\lambda n. \text{ennreal} (\sum na. f na * S n^n)) \longrightarrow (\sum n. \text{ennreal} (f n))$ 
    .
  qed

```

Measures with Restricted Space

```

lemma simple_function_restrict_space_ennreal:
  fixes  $f :: 'a \Rightarrow \text{ennreal}$ 
  assumes  $\Omega \cap \text{space } M \in \text{sets } M$ 
  shows  $\text{simple\_function} (\text{restrict\_space } M \Omega) f \longleftrightarrow \text{simple\_function } M (\lambda x. f x$ 
     $* \text{indicator } \Omega x)$ 
  proof -

```

```

{ assume finite (f ' space (restrict_space M Ω))
  then have finite (f ' space (restrict_space M Ω) ∪ {0}) by simp
  then have finite ((λx. f x * indicator Ω x) ' space M)
  by (rule rev_finite_subset) (auto split: split_indicator simp: space_restrict_space)
}
moreover
{ assume finite ((λx. f x * indicator Ω x) ' space M)
  then have finite (f ' space (restrict_space M Ω))
  by (rule rev_finite_subset) (auto split: split_indicator simp: space_restrict_space)
}
ultimately show ?thesis
unfolding
  simple_function_iff_borel_measurable borel_measurable_restrict_space_iff_enreal[OF
assms]
  by auto
qed

```

```

lemma simple_function_restrict_space:
  fixes f :: 'a ⇒ 'b::real_normed_vector
  assumes Ω ∩ space M ∈ sets M
  shows simple_function (restrict_space M Ω) f ⟷ simple_function M (λx.
indicator Ω x *R f x)
proof -
  { assume finite (f ' space (restrict_space M Ω))
    then have finite (f ' space (restrict_space M Ω) ∪ {0}) by simp
    then have finite ((λx. indicator Ω x *R f x) ' space M)
    by (rule rev_finite_subset) (auto split: split_indicator simp: space_restrict_space)
  }
  moreover
  { assume finite ((λx. indicator Ω x *R f x) ' space M)
    then have finite (f ' space (restrict_space M Ω))
    by (rule rev_finite_subset) (auto split: split_indicator simp: space_restrict_space)
  }
  ultimately show ?thesis
  unfolding simple_function_iff_borel_measurable
    borel_measurable_restrict_space_iff[OF assms]
  by auto
qed

```

```

lemma simple_integral_restrict_space:
  assumes Ω: Ω ∩ space M ∈ sets M simple_function (restrict_space M Ω) f
  shows simple_integral (restrict_space M Ω) f = simple_integral M (λx. f x *
indicator Ω x)
  using simple_function_restrict_space_enreal[THEN iffD1, OF Ω, THEN sim-
ple_functionD(1)]
  by (auto simp add: space_restrict_space emeasure_restrict_space[OF Ω(1)] le_infI2
simple_integral_def
  split: split_indicator split_indicator_asm
  intro!: sum.mono_neutral_cong_left enreal_mult_left_cong arg_cong2)where

```

$f = \text{emeasure}]$)

```

lemma nn_integral_restrict_space:
  assumes  $\Omega[\text{simp}]$ :  $\Omega \cap \text{space } M \in \text{sets } M$ 
  shows  $\text{nn\_integral } (\text{restrict\_space } M \ \Omega) \ f = \text{nn\_integral } M \ (\lambda x. f \ x * \text{indicator } \Omega \ x)$ 
proof –
  let  $?R = \text{restrict\_space } M \ \Omega$  and  $?X = \lambda M \ f. \ \{s. \text{simple\_function } M \ s \wedge s \leq f \wedge (\forall x. s \ x < \text{top})\}$ 
  have  $\text{integral}^S \ ?R \ ' \ ?X \ ?R \ f = \text{integral}^S \ M \ ' \ ?X \ M \ (\lambda x. f \ x * \text{indicator } \Omega \ x)$ 
  proof (safe intro!: image_eqI)
    fix  $s$  assume  $s$ : simple_function  $?R \ s \ s \leq f \ \forall x. s \ x < \text{top}$ 
    from  $s$  show  $\text{integral}^S \ (\text{restrict\_space } M \ \Omega) \ s = \text{integral}^S \ M \ (\lambda x. s \ x * \text{indicator } \Omega \ x)$ 
    by (intro simple_integral_restrict_space) auto
    from  $s$  show simple_function  $M \ (\lambda x. s \ x * \text{indicator } \Omega \ x)$ 
    by (simp add: simple_function_restrict_space_ennreal)
    from  $s$  show  $(\lambda x. s \ x * \text{indicator } \Omega \ x) \leq (\lambda x. f \ x * \text{indicator } \Omega \ x)$ 
     $\wedge x. s \ x * \text{indicator } \Omega \ x < \text{top}$ 
    by (auto split: split_indicator simp: le_fun_def image_subset_iff)
  next
  fix  $s$  assume  $s$ : simple_function  $M \ s \ s \leq (\lambda x. f \ x * \text{indicator } \Omega \ x) \ \forall x. s \ x < \text{top}$ 
  then have simple_function  $M \ (\lambda x. s \ x * \text{indicator } (\Omega \cap \text{space } M) \ x)$  (is ?s')
  by (intro simple_function_mult simple_function_indicator) auto
  also have  $?s' \longleftrightarrow \text{simple\_function } M \ (\lambda x. s \ x * \text{indicator } \Omega \ x)$ 
  by (rule simple_function_cong) (auto split: split_indicator)
  finally show  $\text{sf: simple\_function } (\text{restrict\_space } M \ \Omega) \ s$ 
  by (simp add: simple_function_restrict_space_ennreal)

  from  $s$  have  $s\_eq: s = (\lambda x. s \ x * \text{indicator } \Omega \ x)$ 
  by (auto simp add: fun_eq_iff le_fun_def image_subset_iff
    split: split_indicator split_indicator_asm
    intro: antisym)

  show  $\text{integral}^S \ M \ s = \text{integral}^S \ (\text{restrict\_space } M \ \Omega) \ s$ 
  by (subst s_eq) (rule simple_integral_restrict_space[symmetric, OF  $\Omega$  sf])
  show  $\wedge x. s \ x < \text{top}$ 
  using  $s$  by (auto simp: image_subset_iff)
  from  $s$  show  $s \leq f$ 
  by (subst s_eq) (auto simp: image_subset_iff le_fun_def split: split_indicator split_indicator_asm)
qed
then show ?thesis
unfolding nn_integral_def_finite by (simp cong del: SUP_cong_simp)
qed

```

```

lemma nn_integral_count_space_indicator:
  assumes NO_MATCH (UNIV::'a set) (X::'a set)

```


shows $(\int^+ x. f\ x\ \partial\text{count\_space}\ X) = (\int^+ x. f\ x * \text{indicator}\ X\ x\ \partial\text{count\_space}\ \text{UNIV})$

by (*simp add: nn_integral_restrict_space[symmetric] restrict_count_space*)

lemma *nn_integral_count_space_eq*:

$(\bigwedge x. x \in A - B \implies f\ x = 0) \implies (\bigwedge x. x \in B - A \implies f\ x = 0) \implies$
 $(\int^+ x. f\ x\ \partial\text{count\_space}\ A) = (\int^+ x. f\ x\ \partial\text{count\_space}\ B)$

by (*auto simp: nn_integral_count_space_indicator intro!: nn_integral_cong split: split_indicator*)

lemma *nn_integral_ge_point*:

assumes $x \in A$

shows $p\ x \leq \int^+ x. p\ x\ \partial\text{count\_space}\ A$

proof –

from *assms* **have** $p\ x \leq \int^+ x. p\ x\ \partial\text{count\_space}\ \{x\}$

by(*auto simp add: nn_integral_count_space_finite max_def*)

also **have** $\dots = \int^+ x'. p\ x' * \text{indicator}\ \{x\}\ x'\ \partial\text{count\_space}\ A$

using *assms* **by**(*auto simp add: nn_integral_count_space_indicator indicator_def intro!: nn_integral_cong*)

also **have** $\dots \leq \int^+ x. p\ x\ \partial\text{count\_space}\ A$

by(*rule nn_integral_mono*)(*simp add: indicator_def*)

finally **show** *?thesis* .

qed

Measure spaces with an associated density

definition *density* :: *'a measure* $\Rightarrow$ (*'a* $\Rightarrow$ *ennreal*) $\Rightarrow$ *'a measure* **where**

density $M\ f = \text{measure\_of}\ (\text{space}\ M)\ (\text{sets}\ M)\ (\lambda A. \int^+ x. f\ x * \text{indicator}\ A\ x\ \partial M)$

lemma

shows *sets_density[simp, measurable_cong]*: $\text{sets}\ (\text{density}\ M\ f) = \text{sets}\ M$

and *space_density[simp]*: $\text{space}\ (\text{density}\ M\ f) = \text{space}\ M$

by (*auto simp: density_def*)

lemma *space_density_imp[measurable_dest]*:

$\bigwedge x\ M\ f. x \in \text{space}\ (\text{density}\ M\ f) \implies x \in \text{space}\ M$ **by** *auto*

lemma

shows *measurable_density_eq1[simp]*: $g \in \text{measurable}\ (\text{density}\ M\ f)\ Mg' \longleftrightarrow$
 $g \in \text{measurable}\ Mg\ Mg'$

and *measurable_density_eq2[simp]*: $h \in \text{measurable}\ Mh\ (\text{density}\ Mh'\ f) \longleftrightarrow$
 $h \in \text{measurable}\ Mh\ Mh'$

and *simple_function_density_eq[simp]*: $\text{simple\_function}\ (\text{density}\ Mu\ f)\ u \longleftrightarrow$
 $\text{simple\_function}\ Mu\ u$

unfolding *measurable_def simple_function_def* **by** *simp_all*

lemma *density_cong*: $f \in \text{borel\_measurable}\ M \implies f' \in \text{borel\_measurable}\ M \implies$

$(AE\ x\ in\ M.\ f\ x = f'\ x) \implies density\ M\ f = density\ M\ f'$
unfolding *density_def* **by** (auto intro!: *measure_of_eq_nn_integral_cong_AE* *sets.space_closed*)

lemma *emeasure_density*:

assumes $f[measurable]: f \in borel\_measurable\ M$ **and** $A[measurable]: A \in sets\ M$
shows $emeasure\ (density\ M\ f)\ A = (\int^+ x.\ f\ x * indicator\ A\ x\ \partial M)$
 (**is** $\_ = ?\mu\ A$)
unfolding *density_def*
proof (rule *emeasure_measure_of_sigma*)
show *sigma_algebra* (*space* M) (*sets* M) ..
show *positive* (*sets* M) $?\mu$
using f **by** (auto simp: *positive_def*)
show *countably_additive* (*sets* M) $?\mu$
proof (intro *countably_additiveI*)
fix $A :: nat \Rightarrow 'a\ set$ **assume** $range\ A \subseteq sets\ M$
then have $\bigwedge i.\ A\ i \in sets\ M$ **by** auto
then have $*: \bigwedge i.\ (\lambda x.\ f\ x * indicator\ (A\ i)\ x) \in borel\_measurable\ M$
by auto
assume *disj: disjoint_family* A
then have $(\sum n.\ ?\mu\ (A\ n)) = (\int^+ x.\ (\sum n.\ f\ x * indicator\ (A\ n)\ x)\ \partial M)$
using f **by** (subst *nn_integral_suminf*) auto
also have $(\int^+ x.\ (\sum n.\ f\ x * indicator\ (A\ n)\ x)\ \partial M) = (\int^+ x.\ f\ x * (\sum n.\ indicator\ (A\ n)\ x)\ \partial M)$
using f **by** (auto intro!: *ennreal_suminf_cmult_nn_integral_cong_AE*)
also have $\dots = (\int^+ x.\ f\ x * indicator\ (\bigcup n.\ A\ n)\ x\ \partial M)$
unfolding *suminf_indicator[OF disj]* ..
finally show $(\sum i.\ \int^+ x.\ f\ x * indicator\ (A\ i)\ x\ \partial M) = \int^+ x.\ f\ x * indicator\ (\bigcup i.\ A\ i)\ x\ \partial M$.
qed
qed fact

lemma *null_sets_density_iff*:

assumes $f: f \in borel\_measurable\ M$
shows $A \in null\_sets\ (density\ M\ f) \longleftrightarrow A \in sets\ M \wedge (AE\ x\ in\ M.\ x \in A \longrightarrow f\ x = 0)$
proof -
 { **assume** $A \in sets\ M$
have $(\int^+ x.\ f\ x * indicator\ A\ x\ \partial M) = 0 \longleftrightarrow emeasure\ M\ \{x \in space\ M.\ f\ x * indicator\ A\ x \neq 0\} = 0$
using $f\ \langle A \in sets\ M \rangle$ **by** (intro *nn_integral_0_iff*) auto
also have $\dots \longleftrightarrow (AE\ x\ in\ M.\ f\ x * indicator\ A\ x = 0)$
using $f\ \langle A \in sets\ M \rangle$ **by** (intro *AE_iff_measurable[OF _ refl, symmetric]*)
 auto
also have $(AE\ x\ in\ M.\ f\ x * indicator\ A\ x = 0) \longleftrightarrow (AE\ x\ in\ M.\ x \in A \longrightarrow f\ x \leq 0)$
by (auto simp add: *indicator_def* *max_def* *split: if_split_asm*)
finally have $(\int^+ x.\ f\ x * indicator\ A\ x\ \partial M) = 0 \longleftrightarrow (AE\ x\ in\ M.\ x \in A \longrightarrow f\ x \leq 0) .$ }

```

with f show ?thesis
  by (simp add: null_sets_def emeasure_density cong: conj_cong)
qed

lemma AE_density:
  assumes f: f ∈ borel_measurable M
  shows (AE x in density M f. P x) ⟷ (AE x in M. 0 < f x ⟶ P x)
proof
  assume AE x in density M f. P x
  with f obtain N where {x ∈ space M. ¬ P x} ⊆ N N ∈ sets M and ae: AE x
  in M. x ∈ N ⟶ f x = 0
  by (auto simp: eventually_ae_filter null_sets_density_iff)
  then have AE x in M. x ∉ N ⟶ P x by auto
  with ae show AE x in M. 0 < f x ⟶ P x
  by (rule eventually_elim2) auto
next
  fix N assume ae: AE x in M. 0 < f x ⟶ P x
  then obtain N where {x ∈ space M. ¬ (0 < f x ⟶ P x)} ⊆ N N ∈ null_sets
  M
  by (auto simp: eventually_ae_filter)
  then have *: {x ∈ space (density M f). ¬ P x} ⊆ N ∪ {x ∈ space M. f x = 0}
  N ∪ {x ∈ space M. f x = 0} ∈ sets M and ae2: AE x in M. x ∉ N
  using f by (auto simp: subset_eq zero_less_iff_neq_zero intro!: AE_not_in)
  show AE x in density M f. P x
  using ae2
  unfolding eventually_ae_filter[of _ density M f] Bex_def null_sets_density_iff[OF
  f]
  by (intro exI[of _ N ∪ {x ∈ space M. f x = 0}] conjI *) (auto elim: eventu-
  ally_elim2)
qed

lemma nn_integral_density:
  assumes f: f ∈ borel_measurable M
  assumes g: g ∈ borel_measurable M
  shows integralN (density M f) g = (∫+ x. f x * g x ∂M)
using g proof induct
  case (cong u v)
  then show ?case
    apply (subst nn_integral_cong[OF cong(3)])
    apply (simp_all cong: nn_integral_cong)
    done
next
  case (set A) then show ?case
    by (simp add: emeasure_density f)
next
  case (mult u c)
  moreover have ∧x. f x * (c * u x) = c * (f x * u x) by (simp add: field_simps)
  ultimately show ?case
    using f by (simp add: nn_integral_cmult)

```

```

next
  case (add u v)
  then have  $\bigwedge x. f\ x * (v\ x + u\ x) = f\ x * v\ x + f\ x * u\ x$ 
    by (simp add: distrib_left)
  with add f show ?case
    by (auto simp add: nn_integral_add intro!: nn_integral_add[symmetric])
next
  case (seq U)
  have eq:  $AE\ x\ in\ M. f\ x * (SUP\ i. U\ i\ x) = (SUP\ i. f\ x * U\ i\ x)$ 
    by eventually_elim (simp add: SUP_mult_left_enreal seq)
  from seq f show ?case
    apply (simp add: nn_integral_monotone_convergence_SUP image_comp)
    apply (subst nn_integral_cong_AE[OF eq])
    apply (subst nn_integral_monotone_convergence_SUP_AE)
    apply (auto simp: incseq_def le_fun_def intro!: mult_left_mono)
    done
qed

lemma density_distr:
  assumes [measurable]:  $f \in \text{borel\_measurable } N\ X \in \text{measurable } M\ N$ 
  shows  $\text{density } (distr\ M\ N\ X)\ f = distr\ (\text{density } M\ (\lambda x. f\ (X\ x)))\ N\ X$ 
  by (intro measure_eqI)
    (auto simp add: emeasure_density nn_integral_distr emeasure_distr
      split: split_indicator intro!: nn_integral_cong)

lemma emeasure_restricted:
  assumes  $S: S \in \text{sets } M$  and  $X: X \in \text{sets } M$ 
  shows  $\text{emeasure } (density\ M\ (\text{indicator } S))\ X = \text{emeasure } M\ (S \cap X)$ 
proof -
  have  $\text{emeasure } (density\ M\ (\text{indicator } S))\ X = (\int^{+x}. \text{indicator } S\ x * \text{indicator } X\ x\ \partial M)$ 
    using  $S\ X$  by (simp add: emeasure_density)
  also have  $\dots = (\int^{+x}. \text{indicator } (S \cap X)\ x\ \partial M)$ 
    by (auto intro!: nn_integral_cong simp: indicator_def)
  also have  $\dots = \text{emeasure } M\ (S \cap X)$ 
    using  $S\ X$  by (simp add: sets.Int)
  finally show ?thesis .
qed

lemma measure_restricted:
   $S \in \text{sets } M \implies X \in \text{sets } M \implies \text{measure } (density\ M\ (\text{indicator } S))\ X = \text{measure } M\ (S \cap X)$ 
  by (simp add: emeasure_restricted measure_def)

lemma (in finite_measure) finite_measure_restricted:
   $S \in \text{sets } M \implies \text{finite\_measure } (density\ M\ (\text{indicator } S))$ 
  by standard (simp add: emeasure_restricted)

lemma emeasure_density_const:

```

$A \in \text{sets } M \implies \text{emeasure } (\text{density } M (\lambda_. c)) A = c * \text{emeasure } M A$
by (auto simp: nn_integral_cmult_indicator emeasure_density)

lemma *measure_density_const*:

$A \in \text{sets } M \implies c \neq \infty \implies \text{measure } (\text{density } M (\lambda_. c)) A = \text{enn2real } c * \text{measure } M A$
by (auto simp: emeasure_density_const measure_def enn2real_mult)

lemma *density_density_eq*:

$f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies$
 $\text{density } (\text{density } M f) g = \text{density } M (\lambda x. f x * g x)$
by (auto intro!: measure_eqI simp: emeasure_density nn_integral_density ac_simps)

lemma *distr_density_distr*:

assumes $T: T \in \text{measurable } M M'$ **and** $T': T' \in \text{measurable } M' M$
and *inv*: $\forall x \in \text{space } M. T' (T x) = x$
assumes $f: f \in \text{borel\_measurable } M'$
shows $\text{distr } (\text{density } (\text{distr } M M' T) f) M T' = \text{density } M (f \circ T)$ (**is** ?R = ?L)
proof (rule measure_eqI)
fix A **assume** $A: A \in \text{sets } ?R$
{ **fix** x **assume** $x \in \text{space } M$
with *sets.sets_into_space*[OF A]
have $\text{indicator } (T' - 'A \cap \text{space } M') (T x) = (\text{indicator } A x :: \text{ennreal})$
using T *inv* **by** (auto simp: indicator_def measurable_space) **}**
with $A T T' f$ **show** $\text{emeasure } ?R A = \text{emeasure } ?L A$
by (simp add: measurable_comp emeasure_density emeasure_distr
nn_integral_distr measurable_sets cong: nn_integral_cong)

qed *simp*

lemma *density_density_divide*:

fixes $f g :: 'a \Rightarrow \text{real}$
assumes $f: f \in \text{borel\_measurable } M$ *AE* x *in* $M. 0 \leq f x$
assumes $g: g \in \text{borel\_measurable } M$ *AE* x *in* $M. 0 \leq g x$
assumes *ac*: *AE* x *in* $M. f x = 0 \longrightarrow g x = 0$
shows $\text{density } (\text{density } M f) (\lambda x. g x / f x) = \text{density } M g$
proof –
have $\text{density } M g = \text{density } M (\lambda x. \text{ennreal } (f x) * \text{ennreal } (g x / f x))$
using $f g$ *ac* **by** (auto intro!: density_cong measurable>If simp: ennreal_mult[symmetric])
then show ?thesis
using $f g$ **by** (subst density_density_eq) *auto*
qed

lemma *density_1*: $\text{density } M (\lambda_. 1) = M$

by (intro measure_eqI) (auto simp: emeasure_density)

lemma *emeasure_density_add*:

assumes $X: X \in \text{sets } M$
assumes $Mf[\text{measurable}]: f \in \text{borel\_measurable } M$

```

assumes  $Mg[\text{measurable}]$ :  $g \in \text{borel\_measurable } M$ 
shows  $\text{emeasure } (\text{density } M f) X + \text{emeasure } (\text{density } M g) X =$ 
 $\text{emeasure } (\text{density } M (\lambda x. f x + g x)) X$ 
using assms
apply (subst (1 2 3) emeasure_density, simp_all) []
apply (subst nn_integral_add[symmetric], simp_all) []
apply (intro nn_integral_cong, simp split: split_indicator)
done

```

Point measure

definition *point_measure* :: 'a set $\Rightarrow$ ('a $\Rightarrow$ ennreal) $\Rightarrow$ 'a measure **where**
point_measure A f = *density* (*count_space* A) f

lemma

```

shows space_point_measure: space (point_measure A f) = A
and sets_point_measure: sets (point_measure A f) = Pow A
by (auto simp: point_measure_def)

```

lemma *sets_point_measure_count_space[measurable_cong]*: *sets* (*point_measure* A f) = *sets* (*count_space* A)
by (*simp add: sets_point_measure*)

lemma *measurable_point_measure_eq1[simp]*:
 $g \in \text{measurable } (\text{point\_measure } A f) M \longleftrightarrow g \in A \rightarrow \text{space } M$
unfolding *point_measure_def* **by** *simp*

lemma *measurable_point_measure_eq2_finite[simp]*:
 $\text{finite } A \implies$
 $g \in \text{measurable } M (\text{point\_measure } A f) \longleftrightarrow$
 $(g \in \text{space } M \rightarrow A \wedge (\forall a \in A. g - \{a\} \cap \text{space } M \in \text{sets } M))$
unfolding *point_measure_def* **by** (*simp add: measurable_count_space_eq2*)

lemma *simple_function_point_measure[simp]*:
 $\text{simple\_function } (\text{point\_measure } A f) g \longleftrightarrow \text{finite } (g - A)$
by (*simp add: point_measure_def*)

lemma *emeasure_point_measure*:
assumes A: *finite* { $a \in X. 0 < f a$ } $X \subseteq A$
shows $\text{emeasure } (\text{point\_measure } A f) X = (\sum a | a \in X \wedge 0 < f a. f a)$
proof –
have { $a. (a \in X \longrightarrow a \in A \wedge 0 < f a) \wedge a \in X$ } = { $a \in X. 0 < f a$ }
using $\langle X \subseteq A \rangle$ **by** *auto*
with A **show** ?thesis
by (*simp add: emeasure_density nn_integral_count_space point_measure_def indicator_def of_bool_def*)
qed

lemma *emeasure_point_measure_finite*:

$finite\ A \implies X \subseteq A \implies \text{emeasure } (\text{point\_measure } A\ f)\ X = (\sum_{a \in X}. f\ a)$
by (*subst emeasure_point_measure*) (*auto dest: finite_subset intro!: sum.mono_neutral_left simp: less_le*)

lemma *emeasure_point_measure_finite2*:

$X \subseteq A \implies finite\ X \implies \text{emeasure } (\text{point\_measure } A\ f)\ X = (\sum_{a \in X}. f\ a)$
by (*subst emeasure_point_measure*)
(auto dest: finite_subset intro!: sum.mono_neutral_left simp: less_le)

lemma *null_sets_point_measure_iff*:

$X \in \text{null\_sets } (\text{point\_measure } A\ f) \longleftrightarrow X \subseteq A \wedge (\forall x \in X. f\ x = 0)$
by (*auto simp: AE_count_space null_sets_density_iff point_measure_def*)

lemma *AE_point_measure*:

$(AE\ x\ \text{in } \text{point\_measure } A\ f. P\ x) \longleftrightarrow (\forall x \in A. 0 < f\ x \longrightarrow P\ x)$
unfolding *point_measure_def*
by (*subst AE_density*) (*auto simp: AE_density AE_count_space point_measure_def*)

lemma *nn_integral_point_measure*:

$finite\ \{a \in A. 0 < f\ a \wedge 0 < g\ a\} \implies$
 $\text{integral}^N (\text{point\_measure } A\ f)\ g = (\sum_{a \in A \wedge 0 < f\ a \wedge 0 < g\ a}. f\ a * g\ a)$
unfolding *point_measure_def*
by (*subst nn_integral_density*)
(simp_all add: nn_integral_density nn_integral_count_space ennreal_zero_less_mult_iff)

lemma *nn_integral_point_measure_finite*:

$finite\ A \implies \text{integral}^N (\text{point\_measure } A\ f)\ g = (\sum_{a \in A}. f\ a * g\ a)$
by (*subst nn_integral_point_measure*) (*auto intro!: sum.mono_neutral_left simp: less_le*)

Uniform measure

definition *uniform_measure* $M\ A = \text{density } M\ (\lambda x. \text{indicator } A\ x / \text{emeasure } M\ A)$

lemma

shows *sets_uniform_measure[simp, measurable_cong]*: *sets (uniform_measure M A) = sets M*
and *space_uniform_measure[simp]*: *space (uniform_measure M A) = space M*
by (*auto simp: uniform_measure_def*)

lemma *emeasure_uniform_measure[simp]*:

assumes $A: A \in \text{sets } M$ **and** $B: B \in \text{sets } M$
shows $\text{emeasure } (\text{uniform\_measure } M\ A)\ B = \text{emeasure } M\ (A \cap B) / \text{emeasure } M\ A$

proof –

from $A\ B$ **have** $\text{emeasure } (\text{uniform\_measure } M\ A)\ B = (\int^+ x. (1 / \text{emeasure } M\ A) * \text{indicator } (A \cap B)\ x\ \partial M)$

by (*auto simp add: uniform_measure_def emeasure_density divide_ennreal_def*)

split: *split_indicator*
intro!: *nn_integral_cong*)
also have ... = *emeasure* *M* (*A* ∩ *B*) / *emeasure* *M* *A*
using *A B*
by (*subst nn_integral_cmult_indicator*) (*simp_all add: sets.Int divide_ennreal_def*
mult.commute)
finally show ?*thesis* .
qed

lemma *measure_uniform_measure[simp]*:
assumes *A*: *emeasure* *M* *A* ≠ 0 *emeasure* *M* *A* ≠ ∞ **and** *B*: *B* ∈ *sets* *M*
shows *measure* (*uniform_measure* *M* *A*) *B* = *measure* *M* (*A* ∩ *B*) / *measure* *M* *A*
using *emeasure_uniform_measure[OF emeasure_neq_0_sets[OF A(1)] B] A*
by (*cases emeasure M A emeasure M (A ∩ B) rule: ennreal2_cases*)
(simp_all add: measure_def divide_ennreal top_ennreal.rep_eq top_ereal_def
ennreal_top_divide)

lemma *AE_uniform_measureI*:
 $A \in \text{sets } M \implies (AE\ x \text{ in } M. x \in A \longrightarrow P\ x) \implies (AE\ x \text{ in } \text{uniform\_measure } M\ A. P\ x)$
unfolding *uniform_measure_def* **by** (*auto simp: AE_density divide_ennreal_def*)

lemma *emeasure_uniform_measure_1*:
 $\text{emeasure } M\ S \neq 0 \implies \text{emeasure } M\ S \neq \infty \implies \text{emeasure } (\text{uniform\_measure } M\ S)\ S = 1$
by (*subst emeasure_uniform_measure*)
(simp_all add: emeasure_neq_0_sets emeasure_eq_ennreal_measure divide_ennreal
zero_less_iff_neq_zero[symmetric])

lemma *nn_integral_uniform_measure*:
assumes *f[measurable]*: *f* ∈ *borel_measurable* *M* **and** *S[measurable]*: *S* ∈ *sets* *M*
shows $(\int^+ x. f\ x\ \partial \text{uniform\_measure } M\ S) = (\int^+ x. f\ x * \text{indicator } S\ x\ \partial M) / \text{emeasure } M\ S$

proof –
{ assume *emeasure* *M* *S* = ∞
then have ?*thesis*
by (*simp add: uniform_measure_def nn_integral_density f*) }
moreover
{ assume [*simp*]: *emeasure* *M* *S* = 0
then have *ae*: *AE* *x* in *M*. *x* ∉ *S*
using *sets.sets_into_space[OF S]*
by (*subst AE_iff_measurable[OF _ refl]*) (*simp_all add: subset_eq cong:*
rev_conj_cong)
from *ae* **have** $(\int^+ x. \text{indicator } S\ x / 0 * f\ x\ \partial M) = 0$
by (*subst nn_integral_0_iff_AE*) *auto*
moreover from *ae* **have** $(\int^+ x. f\ x * \text{indicator } S\ x\ \partial M) = 0$
by (*subst nn_integral_0_iff_AE*) *auto*
ultimately have ?*thesis*


```

    by (simp add: uniform_measure_def nn_integral_density f) }
  moreover have  $\text{emeasure } M \, S \neq 0 \implies \text{emeasure } M \, S \neq \infty \implies ?thesis$ 
    unfolding uniform_measure_def
    by (subst nn_integral_density)
    (auto simp: ennreal_times_divide f nn_integral_divide[symmetric] mult.commute)
  ultimately show ?thesis by blast
qed

```

lemma *AE_uniform_measure:*

```

  assumes  $\text{emeasure } M \, A \neq 0$   $\text{emeasure } M \, A < \infty$ 
  shows  $(AE \, x \text{ in } \text{uniform\_measure } M \, A. \, P \, x) \longleftrightarrow (AE \, x \text{ in } M. \, x \in A \longrightarrow P \, x)$ 
proof -
  have  $A \in \text{sets } M$ 
    using  $\langle \text{emeasure } M \, A \neq 0 \rangle$  by (metis emeasure_notin_sets)
  moreover have  $\bigwedge x. \, 0 < \text{indicator } A \, x / \text{emeasure } M \, A \longleftrightarrow x \in A$ 
    using assms
    by (cases emeasure M A) (auto split: split_indicator simp: ennreal_zero_less_divide)
  ultimately show ?thesis
    unfolding uniform_measure_def by (simp add: AE_density)
qed

```

Null measure

lemma *null_measure_eq_density:* $\text{null\_measure } M = \text{density } M \, (\lambda x. \, 0)$
 by (intro measure_eqI) (simp_all add: emeasure_density)

lemma *nn_integral_null_measure[simp]:* $(\int^+ x. \, f \, x \, \partial \text{null\_measure } M) = 0$
 by (auto simp add: nn_integral_def simple_integral_def SUP_constant bot_ennreal_def
 le_fun_def
 intro!: exI[of _ $\lambda x. \, 0$])

lemma *density_null_measure[simp]:* $\text{density } (\text{null\_measure } M) \, f = \text{null\_measure } M$

proof (intro measure_eqI)

fix A show $\text{emeasure } (\text{density } (\text{null\_measure } M) \, f) \, A = \text{emeasure } (\text{null\_measure } M) \, A$

by (simp add: density_def) (simp only: null_measure_def[symmetric] emeasure_null_measure)

qed simp

Uniform count measure

definition *uniform_count_measure* $A = \text{point\_measure } A \, (\lambda x. \, 1 / \text{card } A)$

lemma

shows *space_uniform_count_measure:* $\text{space } (\text{uniform\_count\_measure } A) = A$
 and *sets_uniform_count_measure:* $\text{sets } (\text{uniform\_count\_measure } A) = \text{Pow } A$

unfolding uniform_count_measure_def by (auto simp: space_point_measure sets_point_measure)

lemma *sets_uniform_count_measure_count_space*[*measurable_cong*]:

sets (*uniform_count_measure* *A*) = *sets* (*count_space* *A*)

by (*simp* *add*: *sets_uniform_count_measure*)

lemma *emeasure_uniform_count_measure*:

finite *A* $\implies X \subseteq A \implies \text{emeasure } (\text{uniform\_count\_measure } A) X = \text{card } X / \text{card } A$

by (*simp* *add*: *emeasure_point_measure_finite* *uniform_count_measure_def* *divide_inverse* *ennreal_mult*

ennreal_of_nat_eq_real_of_nat)

lemma *measure_uniform_count_measure*:

finite *A* $\implies X \subseteq A \implies \text{measure } (\text{uniform\_count\_measure } A) X = \text{card } X / \text{card } A$

by (*simp* *add*: *emeasure_point_measure_finite* *uniform_count_measure_def* *measure_def* *enn2real_mult*)

lemma *space_uniform_count_measure_empty_iff* [*simp*]:

space (*uniform_count_measure* *X*) = {} $\longleftrightarrow X = \{\}$

by(*simp* *add*: *space_uniform_count_measure*)

lemma *sets_uniform_count_measure_eq_UNIV* [*simp*]:

sets (*uniform_count_measure* *UNIV*) = *UNIV* $\longleftrightarrow \text{True}$

UNIV = *sets* (*uniform_count_measure* *UNIV*) $\longleftrightarrow \text{True}$

by(*simp_all* *add*: *sets_uniform_count_measure*)

Scaled measure

lemma *nn_integral_scale_measure*:

assumes *f*: *f* $\in$ *borel_measurable* *M*

shows *nn_integral* (*scale_measure* *r* *M*) *f* = *r* * *nn_integral* *M* *f*

using *f*

proof *induction*

case (*cong* *f* *g*)

thus ?*case*

by(*simp* *add*: *cong.hyps* *space_scale_measure* *cong*: *nn_integral_cong_simp*)

next

case (*mult* *f* *c*)

thus ?*case*

by(*simp* *add*: *nn_integral_cmult* *max_def* *mult.assoc* *mult.left_commute*)

next

case (*add* *f* *g*)

thus ?*case*

by(*simp* *add*: *nn_integral_add* *distrib_left*)

next

case (*seq* *U*)

thus ?*case*

by(*simp* *add*: *nn_integral_monotone_convergence_SUP* *SUP_mult_left_ennreal*)

```
image_comp)
qed simp
```

```
end
```

6.7 Binary Product Measure

```
theory Binary_Product_Measure
imports Nonnegative_Lebesgue_Integration
begin
```

```
lemma Pair_vimage_times[simp]: Pair x -' (A × B) = (if x ∈ A then B else {})
  by auto
```

```
lemma rev_Pair_vimage_times[simp]: (λx. (x, y)) -' (A × B) = (if y ∈ B then
A else {})
  by auto
```

6.7.1 Binary products

```
definition pair_measure (infixr  $\otimes_M$  80) where
  A  $\otimes_M$  B = measure_of (space A × space B)
    {a × b | a b. a ∈ sets A ∧ b ∈ sets B}
    (λX. ∫+x. (∫+y. indicator X (x,y) ∂B) ∂A)
```

```
lemma pair_measure_closed: {a × b | a b. a ∈ sets A ∧ b ∈ sets B} ⊆ Pow (space
A × space B)
  using sets.space_closed[of A] sets.space_closed[of B] by auto
```

```
lemma space_pair_measure:
  space (A  $\otimes_M$  B) = space A × space B
  unfolding pair_measure_def using pair_measure_closed[of A B]
  by (rule space_measure_of)
```

```
lemma SIGMA_Collect_eq: (SIGMA x:space M. {y∈space N. P x y}) = {x∈space
(M  $\otimes_M$  N). P (fst x) (snd x)}
  by (auto simp: space_pair_measure)
```

```
lemma sets_pair_measure:
  sets (A  $\otimes_M$  B) = sigma_sets (space A × space B) {a × b | a b. a ∈ sets A ∧
b ∈ sets B}
  unfolding pair_measure_def using pair_measure_closed[of A B]
  by (rule sets_measure_of)
```

```
lemma sets_pair_measure_cong[measurable_cong, cong]:
  sets M1 = sets M1'  $\implies$  sets M2 = sets M2'  $\implies$  sets (M1  $\otimes_M$  M2) = sets
(M1'  $\otimes_M$  M2')
  unfolding sets_pair_measure by (simp cong: sets_eq_imp_space_eq)
```

lemma *pair_measureI*[*intro*, *simp*, *measurable*]:
 $x \in \text{sets } A \implies y \in \text{sets } B \implies x \times y \in \text{sets } (A \otimes_M B)$
by (*auto simp: sets_pair_measure*)

lemma *sets_Pair*: $\{x\} \in \text{sets } M1 \implies \{y\} \in \text{sets } M2 \implies \{(x, y)\} \in \text{sets } (M1 \otimes_M M2)$
using *pair_measureI*[*of* $\{x\} M1 \{y\} M2$] **by** *simp*

lemma *measurable_pair_measureI*:
assumes 1: $f \in \text{space } M \rightarrow \text{space } M1 \times \text{space } M2$
assumes 2: $\bigwedge A B. A \in \text{sets } M1 \implies B \in \text{sets } M2 \implies f -' (A \times B) \cap \text{space } M \in \text{sets } M$
shows $f \in \text{measurable } M (M1 \otimes_M M2)$
unfolding *pair_measure_def* **using** 1 2
by (*intro measurable_measure_of*) (*auto dest: sets_sets_into_space*)

lemma *measurable_split_replace*[*measurable (raw)*]:
 $(\lambda x. f x (fst (g x)) (snd (g x))) \in \text{measurable } M N \implies (\lambda x. \text{case\_prod } (f x) (g x)) \in \text{measurable } M N$
unfolding *split_beta'* .

lemma *measurable_Pair*[*measurable (raw)*]:
assumes $f: f \in \text{measurable } M M1$ **and** $g: g \in \text{measurable } M M2$
shows $(\lambda x. (f x, g x)) \in \text{measurable } M (M1 \otimes_M M2)$
proof (*rule measurable_pair_measureI*)
show $(\lambda x. (f x, g x)) \in \text{space } M \rightarrow \text{space } M1 \times \text{space } M2$
using $f g$ **by** (*auto simp: measurable_def*)
fix $A B$ **assume** *: $A \in \text{sets } M1 \ B \in \text{sets } M2$
have $(\lambda x. (f x, g x)) -' (A \times B) \cap \text{space } M = (f -' A \cap \text{space } M) \cap (g -' B \cap \text{space } M)$
by *auto*
also have $\dots \in \text{sets } M$
by (*rule sets.Int*) (*auto intro!: measurable_sets * f g*)
finally show $(\lambda x. (f x, g x)) -' (A \times B) \cap \text{space } M \in \text{sets } M$.
qed

lemma *measurable_fst*[*intro!*, *simp*, *measurable*]: $\text{fst} \in \text{measurable } (M1 \otimes_M M2) M1$
by (*auto simp: fst_vimage_eq_Times space_pair_measure sets_sets_into_space Times_Int_Times measurable_def*)

lemma *measurable_snd*[*intro!*, *simp*, *measurable*]: $\text{snd} \in \text{measurable } (M1 \otimes_M M2) M2$
by (*auto simp: snd_vimage_eq_Times space_pair_measure sets_sets_into_space Times_Int_Times measurable_def*)

lemma *measurable_Pair_compose_split*[*measurable_dest*]:

assumes f : $\text{case\_prod } f \in \text{measurable } (M1 \otimes_M M2) N$
assumes g : $g \in \text{measurable } M M1$ **and** h : $h \in \text{measurable } M M2$
shows $(\lambda x. f (g x) (h x)) \in \text{measurable } M N$
using $\text{measurable\_compose}[OF \text{measurable\_Pair } f, OF g h]$ **by** *simp*

lemma $\text{measurable\_Pair1\_compose}[\text{measurable\_dest}]$:
assumes f : $(\lambda x. (f x, g x)) \in \text{measurable } M (M1 \otimes_M M2)$
assumes $[\text{measurable}]$: $h \in \text{measurable } N M$
shows $(\lambda x. f (h x)) \in \text{measurable } N M1$
using $\text{measurable\_compose}[OF f \text{measurable\_fst}]$ **by** *simp*

lemma $\text{measurable\_Pair2\_compose}[\text{measurable\_dest}]$:
assumes f : $(\lambda x. (f x, g x)) \in \text{measurable } M (M1 \otimes_M M2)$
assumes $[\text{measurable}]$: $h \in \text{measurable } N M$
shows $(\lambda x. g (h x)) \in \text{measurable } N M2$
using $\text{measurable\_compose}[OF f \text{measurable\_snd}]$ **by** *simp*

lemma measurable_pair :
assumes $(fst \circ f) \in \text{measurable } M M1$ $(snd \circ f) \in \text{measurable } M M2$
shows $f \in \text{measurable } M (M1 \otimes_M M2)$
using $\text{measurable\_Pair}[OF \text{assms}]$ **by** *simp*

lemma
assumes $f[\text{measurable}]$: $f \in \text{measurable } M (N \otimes_M P)$
shows $\text{measurable\_fst}'$: $(\lambda x. \text{fst } (f x)) \in \text{measurable } M N$
and $\text{measurable\_snd}'$: $(\lambda x. \text{snd } (f x)) \in \text{measurable } M P$
by *simp_all*

lemma
assumes $f[\text{measurable}]$: $f \in \text{measurable } M N$
shows $\text{measurable\_fst}''$: $(\lambda x. f (\text{fst } x)) \in \text{measurable } (M \otimes_M P) N$
and $\text{measurable\_snd}''$: $(\lambda x. f (\text{snd } x)) \in \text{measurable } (P \otimes_M M) N$
by *simp_all*

lemma sets_pair_in_sets :
assumes $\bigwedge a b. a \in \text{sets } A \implies b \in \text{sets } B \implies a \times b \in \text{sets } N$
shows $\text{sets } (A \otimes_M B) \subseteq \text{sets } N$
unfolding sets_pair_measure
by $(\text{intro } \text{sets.sigma\_sets\_subset'}) (\text{auto intro!} \text{ assms})$

lemma $\text{sets\_pair\_eq\_sets\_fst\_snd}$:
 $\text{sets } (A \otimes_M B) = \text{sets } (\text{Sup } \{\text{vimage\_algebra } (\text{space } A \times \text{space } B) \text{ fst } A, \text{vimage\_algebra } (\text{space } A \times \text{space } B) \text{ snd } B\})$
 $(\text{is } ?P = \text{sets } (\text{Sup } \{\text{?fst}, \text{?snd}\}))$
proof –
{ fix a b assume ab: a ∈ sets A b ∈ sets B
then have $a \times b = (\text{fst } -' a \cap (\text{space } A \times \text{space } B)) \cap (\text{snd } -' b \cap (\text{space } A \times \text{space } B))$
by $(\text{auto dest: sets.sets\_into\_space})$

```

also have ... ∈ sets (Sup {?fst, ?snd})
  apply (rule sets.Int)
  apply (rule in_sets_Sup)
  apply auto []
  apply (rule insertI1)
  apply (auto intro: ab in_vimage_algebra) []
  apply (rule in_sets_Sup)
  apply auto []
  apply (rule insertI2)
  apply (auto intro: ab in_vimage_algebra)
done
finally have  $a \times b \in \text{sets } (\text{Sup } \{?fst, ?snd\}) . \}$ 
moreover have  $\text{sets } ?fst \subseteq \text{sets } (A \otimes_M B)$ 
  by (rule sets_image_in_sets) (auto simp: space_pair_measure[symmetric])
moreover have  $\text{sets } ?snd \subseteq \text{sets } (A \otimes_M B)$ 
  by (rule sets_image_in_sets) (auto simp: space_pair_measure)
ultimately show ?thesis
  apply (intro antisym[of sets A for A] sets_Sup_in_sets sets_pair_in_sets)
  apply simp
  apply simp
  apply simp
  apply (elim disjE)
  apply (simp add: space_pair_measure)
  apply (simp add: space_pair_measure)
  apply (auto simp add: space_pair_measure)
done
qed

```

lemma *measurable_pair_iff*:

$f \in \text{measurable } M (M1 \otimes_M M2) \longleftrightarrow (fst \circ f) \in \text{measurable } M M1 \wedge (snd \circ f) \in \text{measurable } M M2$

by (auto intro: measurable_pair[of f M M1 M2])

lemma *measurable_split_conv*:

$(\lambda(x, y). f x y) \in \text{measurable } A B \longleftrightarrow (\lambda x. f (fst x) (snd x)) \in \text{measurable } A B$

by (intro arg_cong2[where f=($\in$)] auto)

lemma *measurable_pair_swap'*: $(\lambda(x, y). (y, x)) \in \text{measurable } (M1 \otimes_M M2) (M2 \otimes_M M1)$

by (auto intro!: measurable_Pair simp: measurable_split_conv)

lemma *measurable_pair_swap*:

assumes $f: f \in \text{measurable } (M1 \otimes_M M2) M$ **shows** $(\lambda(x, y). f (y, x)) \in \text{measurable } (M2 \otimes_M M1) M$

using *measurable_comp*[OF *measurable_Pair* f] **by** (auto simp: measurable_split_conv comp_def)

lemma *measurable_pair_swap_iff*:

$f \in \text{measurable } (M2 \otimes_M M1) M \longleftrightarrow (\lambda(x, y). f (y, x)) \in \text{measurable } (M1 \otimes_M M2) M$

$M2$) M
by (*auto dest: measurable_pair_swap*)

lemma *measurable_Pair1'*: $x \in \text{space } M1 \implies \text{Pair } x \in \text{measurable } M2 \ (M1 \otimes_M M2)$
by *simp*

lemma *sets_Pair1*[*measurable (raw)*]:
assumes $A: A \in \text{sets } (M1 \otimes_M M2)$ **shows** $\text{Pair } x \in A \implies A \in \text{sets } M2$
proof –
have $\text{Pair } x \in A = (\text{if } x \in \text{space } M1 \text{ then } \text{Pair } x \in A \cap \text{space } M2 \text{ else } \{\})$
using $A[\text{THEN sets.sets\_into\_space}]$ **by** (*auto simp: space_pair_measure*)
also have $\dots \in \text{sets } M2$
using A **by** (*auto simp add: measurable_Pair1' intro!: measurable_sets split: if_split_asm*)
finally show ?thesis .
qed

lemma *measurable_Pair2'*: $y \in \text{space } M2 \implies (\lambda x. (x, y)) \in \text{measurable } M1 \ (M1 \otimes_M M2)$
by (*auto intro!: measurable_Pair*)

lemma *sets_Pair2*: **assumes** $A: A \in \text{sets } (M1 \otimes_M M2)$ **shows** $(\lambda x. (x, y)) \in A \implies A \in \text{sets } M1$
proof –
have $(\lambda x. (x, y)) \in A = (\text{if } y \in \text{space } M2 \text{ then } (\lambda x. (x, y)) \in A \cap \text{space } M1 \text{ else } \{\})$
using $A[\text{THEN sets.sets\_into\_space}]$ **by** (*auto simp: space_pair_measure*)
also have $\dots \in \text{sets } M1$
using A **by** (*auto simp add: measurable_Pair2' intro!: measurable_sets split: if_split_asm*)
finally show ?thesis .
qed

lemma *measurable_Pair2*:
assumes $f: f \in \text{measurable } (M1 \otimes_M M2) \ M$ **and** $x: x \in \text{space } M1$
shows $(\lambda y. f \ (x, y)) \in \text{measurable } M2 \ M$
using *measurable_comp*[*OF measurable_Pair1' f, OF x*]
by (*simp add: comp_def*)

lemma *measurable_Pair1*:
assumes $f: f \in \text{measurable } (M1 \otimes_M M2) \ M$ **and** $y: y \in \text{space } M2$
shows $(\lambda x. f \ (x, y)) \in \text{measurable } M1 \ M$
using *measurable_comp*[*OF measurable_Pair2' f, OF y*]
by (*simp add: comp_def*)

lemma *Int_stable_pair_measure_generator*: $\text{Int\_stable } \{a \times b \mid a \in \text{sets } A \wedge b \in \text{sets } B\}$
unfolding *Int_stable_def*

by safe (auto simp add: Times_Int_Times)

lemma (in finite_measure) finite_measure_cut_measurable:
assumes [measurable]: $Q \in \text{sets } (N \otimes_M M)$
shows $(\lambda x. \text{emeasure } M (\text{Pair } x - ' Q)) \in \text{borel\_measurable } N$
 (is ?s $Q \in \_$)
using Int_stable_pair_measure_generator pair_measure_closed assms
unfolding sets_pair_measure
proof (induct rule: sigma_sets_induct_disjoint)
case (compl A)
with sets_sets_into_space **have** $\bigwedge x. \text{emeasure } M (\text{Pair } x - ' ((\text{space } N \times \text{space } M) - A)) =$
 (if $x \in \text{space } N$ then $\text{emeasure } M (\text{space } M) - ?s A \ x$ else 0)
unfolding sets_pair_measure[symmetric]
by (auto intro!: emeasure_compl simp: vimage_Diff sets_Pair1)
with compl_sets.top **show** ?case
by (auto intro!: measurable_If simp: space_pair_measure)
next
case (union F)
then have $\bigwedge x. \text{emeasure } M (\text{Pair } x - ' (\bigcup i. F \ i)) = (\sum i. ?s (F \ i) \ x)$
by (simp add: suminf_emeasure disjoint_family_on_vimageI subset_eq vimage_UN sets_pair_measure[symmetric])
with union **show** ?case
unfolding sets_pair_measure[symmetric] **by** simp
qed (auto simp add: if_distrib Int_def[symmetric] intro!: measurable_If)

lemma (in sigma_finite_measure) measurable_emeasure_Pair:
assumes $Q: Q \in \text{sets } (N \otimes_M M)$ **shows** $(\lambda x. \text{emeasure } M (\text{Pair } x - ' Q)) \in \text{borel\_measurable } N$ (is ?s $Q \in \_$)
proof -
obtain $F :: \text{nat} \Rightarrow 'a \text{ set}$ **where** $F:$
 range $F \subseteq \text{sets } M$
 $\bigcup (\text{range } F) = \text{space } M$
 $\bigwedge i. \text{emeasure } M (F \ i) \neq \infty$
 disjoint_family F **by** (blast intro: sigma_finite_disjoint)
then have $F\_sets: \bigwedge i. F \ i \in \text{sets } M$ **by** auto
let $?C = \lambda x \ i. F \ i \cap \text{Pair } x - ' Q$
{ fix i
have [simp]: $\text{space } N \times F \ i \cap \text{space } N \times \text{space } M = \text{space } N \times F \ i$
using F sets_sets_into_space **by** auto
let $?R = \text{density } M (\text{indicator } (F \ i))$
have finite_measure ?R
using F **by** (intro finite_measureI) (auto simp: emeasure_restricted_subset_eq)
then have $(\lambda x. \text{emeasure } ?R (\text{Pair } x - ' (\text{space } N \times \text{space } ?R \cap Q))) \in \text{borel\_measurable } N$
by (rule finite_measure.finite_measure_cut_measurable) (auto intro: Q)
moreover have $\bigwedge x. \text{emeasure } ?R (\text{Pair } x - ' (\text{space } N \times \text{space } ?R \cap Q))$
 = $\text{emeasure } M (F \ i \cap \text{Pair } x - ' (\text{space } N \times \text{space } ?R \cap Q))$


```

    using Q F_sets by (intro emeasure_restricted) (auto intro: sets_Pair1)
  moreover have  $\bigwedge x. F\ i \cap \text{Pair } x - ' (space\ N \times space\ ?R \cap Q) = ?C\ x\ i$ 
    using sets_sets_into_space[OF Q] by (auto simp: space_pair_measure)
  ultimately have  $(\lambda x. emeasure\ M\ (?C\ x\ i)) \in borel\_measurable\ N$ 
    by simp }
moreover
{ fix x
  have  $(\sum i. emeasure\ M\ (?C\ x\ i)) = emeasure\ M\ (\bigcup i. ?C\ x\ i)$ 
  proof (intro suminf_emeasure)
    show range (?C x)  $\subseteq$  sets M
      using F  $\langle Q \in sets\ (N \otimes_M M) \rangle$  by (auto intro!: sets_Pair1)
    have disjoint_family F using F by auto
    show disjoint_family (?C x)
      by (rule disjoint_family_on_bisimulation[OF  $\langle disjoint\_family\ F \rangle$ ]) auto
  qed
  also have  $(\bigcup i. ?C\ x\ i) = \text{Pair } x - ' Q$ 
    using F sets_sets_into_space[OF  $\langle Q \in sets\ (N \otimes_M M) \rangle$ ]
    by (auto simp: space_pair_measure)
  finally have  $emeasure\ M\ (\text{Pair } x - ' Q) = (\sum i. emeasure\ M\ (?C\ x\ i))$ 
    by simp }
ultimately show ?thesis using  $\langle Q \in sets\ (N \otimes_M M) \rangle$  F_sets
  by auto
qed

lemma (in sigma_finite_measure) measurable_emeasure[measurable (raw)]:
  assumes space:  $\bigwedge x. x \in space\ N \implies A\ x \subseteq space\ M$ 
  assumes A:  $\{x \in space\ (N \otimes_M M). snd\ x \in A\ (fst\ x)\} \in sets\ (N \otimes_M M)$ 
  shows  $(\lambda x. emeasure\ M\ (A\ x)) \in borel\_measurable\ N$ 
proof -
  from space have  $\bigwedge x. x \in space\ N \implies \text{Pair } x - ' \{x \in space\ (N \otimes_M M). snd\ x \in A\ (fst\ x)\} = A\ x$ 
  by (auto simp: space_pair_measure)
  with measurable_emeasure_Pair[OF A] show ?thesis
    by (auto cong: measurable_cong)
qed

lemma (in sigma_finite_measure) emeasure_pair_measure:
  assumes X  $\in sets\ (N \otimes_M M)$ 
  shows  $emeasure\ (N \otimes_M M)\ X = (\int^+ x. \int^+ y. indicator\ X\ (x, y)\ \partial M\ \partial N)$ 
  (is _ = ? $\mu\ X$ )
proof (rule emeasure_measure_of[OF pair_measure_def])
  show positive (sets (N  $\otimes_M$  M)) ? $\mu$ 
    by (auto simp: positive_def)
  have eq[simp]:  $\bigwedge A\ x\ y. indicator\ A\ (x, y) = indicator\ (\text{Pair } x - ' A)\ y$ 
    by (auto simp: indicator_def)
  show countably_additive (sets (N  $\otimes_M$  M)) ? $\mu$ 
  proof (rule countably_additiveI)
    fix F :: nat  $\Rightarrow$  ('b  $\times$  'a) set assume F: range F  $\subseteq sets\ (N \otimes_M M)$  disjoint_family F

```

```

from  $F$  have *:  $\bigwedge i. F\ i \in \text{sets } (N \otimes_M M)$  by auto
moreover have  $\bigwedge x. \text{disjoint\_family } (\lambda i. \text{Pair } x - ' F\ i)$ 
  by (intro disjoint_family_on_bisimulation[OF F(2)]) auto
moreover have  $\bigwedge x. \text{range } (\lambda i. \text{Pair } x - ' F\ i) \subseteq \text{sets } M$ 
  using  $F$  by (auto simp: sets_Pair1)
ultimately show  $(\sum n. ?\mu (F\ n)) = ?\mu (\bigcup i. F\ i)$ 
by (auto simp add: nn_integral_suminf[symmetric] vimage_UN suminf_emeasure
  intro!: nn_integral_cong nn_integral_indicator[symmetric])
qed
show  $\{a \times b \mid a\ b. a \in \text{sets } N \wedge b \in \text{sets } M\} \subseteq \text{Pow } (\text{space } N \times \text{space } M)$ 
  using sets.space_closed[of N] sets.space_closed[of M] by auto
qed fact

```

```

lemma (in sigma_finite_measure) emeasure_pair_measure_alt:
  assumes  $X: X \in \text{sets } (N \otimes_M M)$ 
  shows  $\text{emeasure } (N \otimes_M M) X = (\int^+ x. \text{emeasure } M (\text{Pair } x - ' X) \partial N)$ 
proof -
  have [simp]:  $\bigwedge x\ y. \text{indicator } X\ (x, y) = \text{indicator } (\text{Pair } x - ' X)\ y$ 
    by (auto simp: indicator_def)
  show ?thesis
    using  $X$  by (auto intro!: nn_integral_cong simp: emeasure_pair_measure
  sets_Pair1)
qed

```

```

proposition (in sigma_finite_measure) emeasure_pair_measure_Times:
  assumes  $A: A \in \text{sets } N$  and  $B: B \in \text{sets } M$ 
  shows  $\text{emeasure } (N \otimes_M M) (A \times B) = \text{emeasure } N\ A * \text{emeasure } M\ B$ 
proof -
  have  $\text{emeasure } (N \otimes_M M) (A \times B) = (\int^+ x. \text{emeasure } M\ B * \text{indicator } A\ x \partial N)$ 
    using  $A\ B$  by (auto intro!: nn_integral_cong simp: emeasure_pair_measure_alt)
  also have  $\dots = \text{emeasure } M\ B * \text{emeasure } N\ A$ 
    using  $A$  by (simp add: nn_integral_cmult_indicator)
  finally show ?thesis
    by (simp add: ac_simps)
qed

```

6.7.2 Binary products of σ -finite emeasure spaces

```

locale pair_sigma_finite =  $M1?$ : sigma_finite_measure  $M1$  +  $M2?$ : sigma_finite_measure  $M2$ 
  for  $M1 :: 'a\ \text{measure}$  and  $M2 :: 'b\ \text{measure}$ 

```

```

lemma (in pair_sigma_finite) measurable_emeasure_Pair1:
   $Q \in \text{sets } (M1 \otimes_M M2) \implies (\lambda x. \text{emeasure } M2 (\text{Pair } x - ' Q)) \in \text{borel\_measurable } M1$ 
  using  $M2.\text{measurable\_emeasure\_Pair}$  .

```

```

lemma (in pair_sigma_finite) measurable_emeasure_Pair2:

```

assumes $Q: Q \in \text{sets } (M1 \otimes_M M2)$ **shows** $(\lambda y. \text{emeasure } M1 ((\lambda x. (x, y)) - 'Q)) \in \text{borel\_measurable } M2$

proof –

have $(\lambda(x, y). (y, x)) - 'Q \cap \text{space } (M2 \otimes_M M1) \in \text{sets } (M2 \otimes_M M1)$
using $Q \text{ measurable\_pair\_swap'}$ **by** $(\text{auto intro: measurable\_sets})$
note $M1.\text{measurable\_emeasure\_Pair}[OF \text{ this}]$
moreover have $\bigwedge y. \text{Pair } y - '((\lambda(x, y). (y, x)) - 'Q \cap \text{space } (M2 \otimes_M M1))$
 $= (\lambda x. (x, y)) - 'Q$

using $Q[THEN \text{ sets.sets\_into\_space}]$ **by** $(\text{auto simp: space\_pair\_measure})$
ultimately show $?thesis$ **by** simp

qed

proposition $(\text{in pair\_sigma\_finite}) \text{ sigma\_finite\_up\_in\_pair\_measure\_generator:}$

defines $E \equiv \{A \times B \mid A \subseteq B. A \in \text{sets } M1 \wedge B \in \text{sets } M2\}$

shows $\exists F::\text{nat} \Rightarrow ('a \times 'b) \text{ set. range } F \subseteq E \wedge \text{incseq } F \wedge (\bigcup i. F i) = \text{space } M1 \times \text{space } M2 \wedge$

$(\forall i. \text{emeasure } (M1 \otimes_M M2) (F i) \neq \infty)$

proof –

obtain $F1$ **where** $F1: \text{range } F1 \subseteq \text{sets } M1$

$\bigcup (\text{range } F1) = \text{space } M1$

$\bigwedge i. \text{emeasure } M1 (F1 i) \neq \infty$

$\text{incseq } F1$

by $(\text{rule } M1.\text{sigma\_finite\_incseq}) \text{ blast}$

obtain $F2$ **where** $F2: \text{range } F2 \subseteq \text{sets } M2$

$\bigcup (\text{range } F2) = \text{space } M2$

$\bigwedge i. \text{emeasure } M2 (F2 i) \neq \infty$

$\text{incseq } F2$

by $(\text{rule } M2.\text{sigma\_finite\_incseq}) \text{ blast}$

from $F1 \ F2$ **have** $\text{space: space } M1 = (\bigcup i. F1 i) \text{ space } M2 = (\bigcup i. F2 i)$ **by** auto

let $?F = \lambda i. F1 i \times F2 i$

show $?thesis$

proof $(\text{intro exI}[of \_ ?F] \text{ conjI allI})$

show $\text{range } ?F \subseteq E$ **using** $F1 \ F2$ **by** $(\text{auto simp: } E\_def) (\text{metis range\_subsetD})$

next

have $\text{space } M1 \times \text{space } M2 \subseteq (\bigcup i. ?F i)$

proof (intro subsetI)

fix x **assume** $x \in \text{space } M1 \times \text{space } M2$

then obtain $i \ j$ **where** $\text{fst } x \in F1 i \ \text{snd } x \in F2 j$

by $(\text{auto simp: space})$

then have $\text{fst } x \in F1 (\max i j) \ \text{snd } x \in F2 (\max j i)$

using $\langle \text{incseq } F1 \rangle \langle \text{incseq } F2 \rangle$ **unfolding** incseq_def

by $(\text{force split: split\_max})+$

then have $(\text{fst } x, \text{snd } x) \in F1 (\max i j) \times F2 (\max i j)$

by $(\text{intro SigmaI}) (\text{auto simp add: max.commute})$

then show $x \in (\bigcup i. ?F i)$ **by** auto

qed

then show $(\bigcup i. ?F i) = \text{space } M1 \times \text{space } M2$

using space **by** $(\text{auto simp: space})$

```

next
  fix i show incseq ( $\lambda i. F1\ i \times F2\ i$ )
    using  $\langle incseq\ F1 \rangle \langle incseq\ F2 \rangle$  unfolding incseq_Suc_iff by auto
next
  fix i
  from F1 F2 have  $F1\ i \in sets\ M1\ F2\ i \in sets\ M2$  by auto
  with F1 F2 show  $emeasure\ (M1 \otimes_M M2)\ (F1\ i \times F2\ i) \neq \infty$ 
    by (auto simp add: emeasure_pair_measure_Times ennreal_mult_eq_top_iff)
qed
qed

sublocale pair_sigma_finite  $\subseteq P?$ : sigma_finite_measure  $M1 \otimes_M M2$ 
proof
  obtain  $F1 :: 'a\ set\ set$  and  $F2 :: 'b\ set\ set$  where
    countable  $F1 \wedge F1 \subseteq sets\ M1 \wedge \bigcup F1 = space\ M1 \wedge (\forall a \in F1. emeasure\ M1\ a \neq \infty)$ 
    countable  $F2 \wedge F2 \subseteq sets\ M2 \wedge \bigcup F2 = space\ M2 \wedge (\forall a \in F2. emeasure\ M2\ a \neq \infty)$ 
  using  $M1.sigma\_finite\_countable\ M2.sigma\_finite\_countable$  by auto
  then show
     $\exists A. countable\ A \wedge A \subseteq sets\ (M1 \otimes_M M2) \wedge \bigcup A = space\ (M1 \otimes_M M2) \wedge$ 
     $(\forall a \in A. emeasure\ (M1 \otimes_M M2)\ a \neq \infty)$ 
    by (intro exI[of  $\lambda(a, b). a \times b$ ]  $'(F1 \times F2)'$ ] conjI)
    (auto simp:  $M2.emeasure\_pair\_measure\_Times\ space\_pair\_measure\_set\_eq\_iff$ 
    subset_eq ennreal_mult_eq_top_iff)
qed

lemma sigma_finite_pair_measure:
  assumes  $A: sigma\_finite\_measure\ A$  and  $B: sigma\_finite\_measure\ B$ 
  shows  $sigma\_finite\_measure\ (A \otimes_M B)$ 
proof -
  interpret  $A: sigma\_finite\_measure\ A$  by fact
  interpret  $B: sigma\_finite\_measure\ B$  by fact
  interpret  $AB: pair\_sigma\_finite\ A\ B$  ..
  show ?thesis ..
qed

lemma sets_pair_swap:
  assumes  $A \in sets\ (M1 \otimes_M M2)$ 
  shows  $(\lambda(x, y). (y, x))^{-1} A \cap space\ (M2 \otimes_M M1) \in sets\ (M2 \otimes_M M1)$ 
  using measurable_pair_swap' assms by (rule measurable_sets)

lemma (in pair_sigma_finite) distr_pair_swap:
   $M1 \otimes_M M2 = distr\ (M2 \otimes_M M1)\ (M1 \otimes_M M2)\ (\lambda(x, y). (y, x))$  (is  $?P = ?D$ )
proof -
  let  $?E = \{a \times b \mid a \in sets\ M1 \wedge b \in sets\ M2\}$ 
  obtain  $F :: nat \Rightarrow ('a \times 'b)\ set$  where  $F: range\ F \subseteq ?E$ 
    incseq  $F \bigcup (range\ F) = space\ M1 \times space\ M2 \forall i. emeasure\ (M1 \otimes_M M2)$ 

```

```

(F i) ≠ ∞
  using sigma_finite_up_in_pair_measure_generator by auto
  show ?thesis
  proof (rule measure_eqI_generator_eq[OF Int_stable_pair_measure_generator[of
M1 M2]])
    show ?E ⊆ Pow (space ?P)
      using sets.space_closed[of M1] sets.space_closed[of M2] by (auto simp:
space_pair_measure)
    show sets ?P = sigma_sets (space ?P) ?E
      by (simp add: sets_pair_measure space_pair_measure)
    then show sets ?D = sigma_sets (space ?P) ?E
      by simp
    from F show range F ⊆ ?E (⋃ i. F i) = space ?P ∧ i. emeasure ?P (F i) ≠
∞
      by (auto simp: space_pair_measure)
  next
    fix X assume X ∈ ?E
    then obtain A B where X[simp]: X = A × B and A: A ∈ sets M1 and B:
B ∈ sets M2 by auto
    have (λ(y, x). (x, y)) - ' X ∩ space (M2 ⊗M M1) = B × A
      using sets.sets_into_space[OF A] sets.sets_into_space[OF B] by (auto simp:
space_pair_measure)
    with A B show emeasure (M1 ⊗M M2) X = emeasure ?D X
      by (simp add: M2.emeasure_pair_measure_Times M1.emeasure_pair_measure_Times
emeasure_distr
      measurable_pair_swap' ac_simps)
  qed
qed

lemma (in pair_sigma_finite) emeasure_pair_measure_alt2:
  assumes A: A ∈ sets (M1 ⊗M M2)
  shows emeasure (M1 ⊗M M2) A = (∫+ y. emeasure M1 ((λx. (x, y)) - ' A)
∂M2)
  (is _ = ?ν A)
  proof -
    have [simp]: ⋀ y. (Pair y - ' ((λ(x, y). (y, x)) - ' A ∩ space (M2 ⊗M M1))) =
(λx. (x, y)) - ' A
      using sets.sets_into_space[OF A] by (auto simp: space_pair_measure)
    show ?thesis using A
      by (subst distr_pair_swap)
      (simp_all del: vimage_Int add: measurable_sets[OF measurable_pair_swap']
      M1.emeasure_pair_measure_alt emeasure_distr[OF measur-
able_pair_swap' A])
  qed

lemma (in pair_sigma_finite) AE_pair:
  assumes AE x in (M1 ⊗M M2). Q x
  shows AE x in M1. (AE y in M2. Q (x, y))
  proof -

```

```

obtain  $N$  where  $N: N \in \text{sets } (M1 \otimes_M M2) \text{ emeasure } (M1 \otimes_M M2) N = 0$ 
 $\{x \in \text{space } (M1 \otimes_M M2). \neg Q x\} \subseteq N$ 
using assms unfolding eventually_ae_filter by auto
show ?thesis
proof (rule AE_I)
  from  $N \text{ measurable\_emeasure\_Pair1}[OF \langle N \in \text{sets } (M1 \otimes_M M2) \rangle]$ 
  show  $\text{emeasure } M1 \{x \in \text{space } M1. \text{emeasure } M2 (\text{Pair } x - 'N) \neq 0\} = 0$ 
    by (auto simp: M2.emeasure_pair_measure_alt nn_integral_0_iff)
  show  $\{x \in \text{space } M1. \text{emeasure } M2 (\text{Pair } x - 'N) \neq 0\} \in \text{sets } M1$ 
  by (intro borel_measurable_eq measurable_emeasure_Pair1 N sets.sets_Collect_neg
 $N$ ) simp
  { fix  $x$  assume  $x \in \text{space } M1 \text{ emeasure } M2 (\text{Pair } x - 'N) = 0$  }
    have  $AE \ y \text{ in } M2. Q(x, y)$ 
    proof (rule AE_I)
      show  $\text{emeasure } M2 (\text{Pair } x - 'N) = 0$  by fact
      show  $\text{Pair } x - 'N \in \text{sets } M2$  using  $N(1)$  by (rule sets_Pair1)
      show  $\{y \in \text{space } M2. \neg Q(x, y)\} \subseteq \text{Pair } x - 'N$ 
        using  $N \langle x \in \text{space } M1 \rangle$  unfolding space_pair_measure by auto
      qed
    then show  $\{x \in \text{space } M1. \neg (AE \ y \text{ in } M2. Q(x, y))\} \subseteq \{x \in \text{space } M1.$ 
 $\text{emeasure } M2 (\text{Pair } x - 'N) \neq 0\}$ 
      by auto
    qed
  qed

```

```

lemma (in pair_sigma_finite) AE_pair_measure:
  assumes  $\{x \in \text{space } (M1 \otimes_M M2). P x\} \in \text{sets } (M1 \otimes_M M2)$ 
  assumes ae:  $AE \ x \text{ in } M1. AE \ y \text{ in } M2. P(x, y)$ 
  shows  $AE \ x \text{ in } M1 \otimes_M M2. P x$ 
proof (subst AE_iff_measurable[OF _ refl])
  show  $\{x \in \text{space } (M1 \otimes_M M2). \neg P x\} \in \text{sets } (M1 \otimes_M M2)$ 
    by (rule sets.sets_Collect) fact
  then have  $\text{emeasure } (M1 \otimes_M M2) \{x \in \text{space } (M1 \otimes_M M2). \neg P x\} =$ 
 $(\int^+ x. \int^+ y. \text{indicator } \{x \in \text{space } (M1 \otimes_M M2). \neg P x\} (x, y) \partial M2 \partial M1)$ 
    by (simp add: M2.emeasure_pair_measure)
  also have  $\dots = (\int^+ x. \int^+ y. 0 \partial M2 \partial M1)$ 
    using ae
    apply (safe intro!: nn_integral_cong_AE)
    apply (intro AE_I2)
    apply (safe intro!: nn_integral_cong_AE)
    apply auto
    done
  finally show  $\text{emeasure } (M1 \otimes_M M2) \{x \in \text{space } (M1 \otimes_M M2). \neg P x\} = 0$ 
by simp
qed

```

```

lemma (in pair_sigma_finite) AE_pair_iff:
   $\{x \in \text{space } (M1 \otimes_M M2). P(\text{fst } x) (\text{snd } x)\} \in \text{sets } (M1 \otimes_M M2) \implies$ 
 $(AE \ x \text{ in } M1. AE \ y \text{ in } M2. P(x, y)) \longleftrightarrow (AE \ x \text{ in } (M1 \otimes_M M2). P(\text{fst } x))$ 

```

(snd x))
 using AE_pair[of $\lambda x. P (fst x) (snd x)$] AE_pair_measure[of $\lambda x. P (fst x) (snd x)$] by auto

lemma (in pair_sigma_finite) AE_commute:
 assumes $P: \{x \in \text{space } (M1 \otimes_M M2). P (fst x) (snd x)\} \in \text{sets } (M1 \otimes_M M2)$
 shows $(AE\ x\ \text{in } M1. AE\ y\ \text{in } M2. P\ x\ y) \longleftrightarrow (AE\ y\ \text{in } M2. AE\ x\ \text{in } M1. P\ x\ y)$
proof –
 interpret $Q: \text{pair\_sigma\_finite } M2\ M1 ..$
 have [simp]: $\bigwedge x. (fst (case\ x\ of\ (x, y) \Rightarrow (y, x))) = snd\ x \wedge x. (snd (case\ x\ of\ (x, y) \Rightarrow (y, x))) = fst\ x$
 by auto
 have $\{x \in \text{space } (M2 \otimes_M M1). P (snd x) (fst x)\} =$
 $(\lambda(x, y). (y, x)) - ' \{x \in \text{space } (M1 \otimes_M M2). P (fst x) (snd x)\} \cap \text{space } (M2 \otimes_M M1)$
 by (auto simp: space_pair_measure)
 also have $\dots \in \text{sets } (M2 \otimes_M M1)$
 by (intro sets_pair_swap P)
 finally show ?thesis
 apply (subst AE_pair_iff[OF P])
 apply (subst distr_pair_swap)
 apply (subst AE_distr_iff[OF measurable_pair_swap' P])
 apply (subst Q.AE_pair_iff)
 apply simp_all
 done
qed

6.7.3 Fubinis theorem

lemma measurable_compose_Pair1:
 $x \in \text{space } M1 \implies g \in \text{measurable } (M1 \otimes_M M2)\ L \implies (\lambda y. g\ (x, y)) \in \text{measurable } M2\ L$
 by simp

lemma (in sigma_finite_measure) borel_measurable_nn_integral_fst:
 assumes $f: f \in \text{borel\_measurable } (M1 \otimes_M M)$
 shows $(\lambda x. \int^+ y. f\ (x, y)\ \partial M) \in \text{borel\_measurable } M1$
using f **proof** induct
 case (cong u v)
 then have $\bigwedge w\ x. w \in \text{space } M1 \implies x \in \text{space } M \implies u\ (w, x) = v\ (w, x)$
 by (auto simp: space_pair_measure)
 show ?case
 apply (subst measurable_cong)
 apply (rule nn_integral_cong)
 apply fact+
 done
next
 case (set Q)

```

have [simp]:  $\bigwedge x y. \text{indicator } Q (x, y) = \text{indicator } (\text{Pair } x - ' Q) y$ 
by (auto simp: indicator_def)
have  $\bigwedge x. x \in \text{space } M1 \implies \text{emeasure } M (\text{Pair } x - ' Q) = \int^+ y. \text{indicator } Q$ 
 $(x, y) \partial M$ 
by (simp add: sets_Pair1[OF set])
from this measurable_emeasure_Pair[OF set] show ?case
by (rule measurable_cong[THEN iffD1])
qed (simp_all add: nn_integral_add nn_integral_cmult measurable_compose_Pair1
nn_integral_monotone_convergence_SUP incseq_def le_fun_def
image_comp
cong: measurable_cong)

```

```

lemma (in sigma_finite_measure) nn_integral_fst:
assumes  $f: f \in \text{borel\_measurable } (M1 \otimes_M M)$ 
shows  $(\int^+ x. \int^+ y. f (x, y) \partial M \partial M1) = \text{integral}^N (M1 \otimes_M M) f$  (is ?I f =
_)
using f proof induct
case (cong u v)
then have ?I u = ?I v
by (intro nn_integral_cong) (auto simp: space_pair_measure)
with cong show ?case
by (simp cong: nn_integral_cong)
qed (simp_all add: emeasure_pair_measure nn_integral_cmult nn_integral_add
nn_integral_monotone_convergence_SUP measurable_compose_Pair1
borel_measurable_nn_integral_fst nn_integral_mono incseq_def
le_fun_def image_comp
cong: nn_integral_cong)

```

```

lemma (in sigma_finite_measure) borel_measurable_nn_integral[measurable (raw)]:
case_prod  $f \in \text{borel\_measurable } (N \otimes_M M) \implies (\lambda x. \int^+ y. f x y \partial M) \in$ 
 $\text{borel\_measurable } N$ 
using borel_measurable_nn_integral_fst[of case_prod f N] by simp

```

```

proposition (in pair_sigma_finite) nn_integral_snd:
assumes  $f[\text{measurable}]: f \in \text{borel\_measurable } (M1 \otimes_M M2)$ 
shows  $(\int^+ y. (\int^+ x. f (x, y) \partial M1) \partial M2) = \text{integral}^N (M1 \otimes_M M2) f$ 
proof -
note measurable_pair_swap[OF f]
from M1.nn_integral_fst[OF this]
have  $(\int^+ y. (\int^+ x. f (x, y) \partial M1) \partial M2) = (\int^+ (x, y). f (y, x) \partial (M2 \otimes_M$ 
 $M1))$ 
by simp
also have  $(\int^+ (x, y). f (y, x) \partial (M2 \otimes_M M1)) = \text{integral}^N (M1 \otimes_M M2) f$ 
by (subst distr_pair_swap) (auto simp add: nn_integral_distr intro!: nn_integral_cong)
finally show ?thesis .
qed

```

```

theorem (in pair_sigma_finite) Fubini:
assumes  $f: f \in \text{borel\_measurable } (M1 \otimes_M M2)$ 

```


shows $(\int^+ y. (\int^+ x. f(x, y) \partial M1) \partial M2) = (\int^+ x. (\int^+ y. f(x, y) \partial M2) \partial M1)$

unfolding $nn\_integral\_snd[OF\ assms]\ M2.nn\_integral\_fst[OF\ assms] \dots$

theorem (in *pair_sigma_finite*) *Fubini'*:

assumes $f: case\_prod\ f \in borel\_measurable\ (M1 \otimes_M M2)$

shows $(\int^+ y. (\int^+ x. f\ x\ y\ \partial M1) \partial M2) = (\int^+ x. (\int^+ y. f\ x\ y\ \partial M2) \partial M1)$

using *Fubini*[*OF f*] **by** *simp*

6.7.4 Products on counting spaces, densities and distributions

proposition *sigma_prod*:

assumes $X\_cover: \exists E \subseteq A. countable\ E \wedge X = \bigcup E$ **and** $A: A \subseteq Pow\ X$

assumes $Y\_cover: \exists E \subseteq B. countable\ E \wedge Y = \bigcup E$ **and** $B: B \subseteq Pow\ Y$

shows $sigma\ X\ A \otimes_M sigma\ Y\ B = sigma\ (X \times Y)\ \{a \times b \mid a \in A \wedge b \in B\}$

(is $?P = ?S$)

proof (rule *measure_eqI*)

have [*simp*]: $snd \in X \times Y \rightarrow Y\ fst \in X \times Y \rightarrow X$

by *auto*

let $?XY = \{\{fst - 'a \cap X \times Y \mid a. a \in A\}, \{snd - 'b \cap X \times Y \mid b. b \in B\}\}$

have $sets\ ?P = sets\ (SUP\ xy \in ?XY. sigma\ (X \times Y)\ xy)$

by (*simp add: vimage_algebra_sigma_sets_pair_eq_sets_fst_snd A B*)

also have $\dots = sets\ (sigma\ (X \times Y)\ (\bigcup ?XY))$

by (*intro Sup_sigma_arg_cong[where f=sets] auto*)

also have $\dots = sets\ ?S$

proof (*intro arg_cong[where f=sets] sigma_eqI sigma_sets_eqI*)

show $\bigcup ?XY \subseteq Pow\ (X \times Y)\ \{a \times b \mid a \in A \wedge b \in B\} \subseteq Pow\ (X \times Y)$

using *A B* **by** *auto*

next

interpret $XY: sigma\_algebra\ X \times Y\ sigma\_sets\ (X \times Y)\ \{a \times b \mid a \in A \wedge b \in B\}$

using *A B* **by** (*intro sigma_algebra_sigma_sets auto*)

fix Z **assume** $Z \in \bigcup ?XY$

then show $Z \in sigma\_sets\ (X \times Y)\ \{a \times b \mid a \in A \wedge b \in B\}$

proof *safe*

fix a **assume** $a \in A$

from Y_cover **obtain** E **where** $E: E \subseteq B\ countable\ E$ **and** $Y = \bigcup E$

by *auto*

with $\langle a \in A \rangle\ A$ **have** $eq: fst - 'a \cap X \times Y = (\bigcup e \in E. a \times e)$

by *auto*

show $fst - 'a \cap X \times Y \in sigma\_sets\ (X \times Y)\ \{a \times b \mid a \in A \wedge b \in B\}$

using $\langle a \in A \rangle\ E$ **unfolding** *eq* **by** (*auto intro!: XY.countable_UN*)

next

fix b **assume** $b \in B$

from X_cover **obtain** E **where** $E: E \subseteq A\ countable\ E$ **and** $X = \bigcup E$

by *auto*

with $\langle b \in B \rangle\ B$ **have** $eq: snd - 'b \cap X \times Y = (\bigcup e \in E. e \times b)$

```

      by auto
    show snd - ' b  $\cap$  X  $\times$  Y  $\in$  sigma_sets (X  $\times$  Y) {a  $\times$  b | a b. a  $\in$  A  $\wedge$  b  $\in$ 
B}
    using <b  $\in$  B> E unfolding eq by (auto intro!: XY.countable_UN')
  qed
next
  fix Z assume Z  $\in$  {a  $\times$  b | a b. a  $\in$  A  $\wedge$  b  $\in$  B}
  then obtain a b where Z = a  $\times$  b and ab: a  $\in$  A b  $\in$  B
  by auto
  then have Z: Z = (fst - ' a  $\cap$  X  $\times$  Y)  $\cap$  (snd - ' b  $\cap$  X  $\times$  Y)
  using A B by auto
  interpret XY: sigma_algebra X  $\times$  Y sigma_sets (X  $\times$  Y) ( $\bigcup$  ?XY)
  by (intro sigma_algebra_sigma_sets) auto
  show Z  $\in$  sigma_sets (X  $\times$  Y) ( $\bigcup$  ?XY)
  unfolding Z by (rule XY.Int) (blast intro: ab)+
  qed
  finally show sets ?P = sets ?S .
next
  interpret finite_measure sigma X A for X A
  proof qed (simp add: emeasure_sigma)
  fix A assume A  $\in$  sets ?P then show emeasure ?P A = emeasure ?S A
  by (simp add: emeasure_pair_measure_alt emeasure_sigma)
qed

lemma sigma_sets_pair_measure_generator_finite:
  assumes finite A and finite B
  shows sigma_sets (A  $\times$  B) { a  $\times$  b | a b. a  $\subseteq$  A  $\wedge$  b  $\subseteq$  B} = Pow (A  $\times$  B)
  (is sigma_sets ?prod ?sets = _)
proof safe
  have fin: finite (A  $\times$  B) using assms by (rule finite_cartesian_product)
  fix x assume subset: x  $\subseteq$  A  $\times$  B
  hence finite x using fin by (rule finite_subset)
  from this subset show x  $\in$  sigma_sets ?prod ?sets
  proof (induct x)
    case empty show ?case by (rule sigma_sets.Empty)
  next
    case (insert a x)
    hence {a}  $\in$  sigma_sets ?prod ?sets by auto
    moreover have x  $\in$  sigma_sets ?prod ?sets using insert by auto
    ultimately show ?case unfolding insert_is_Un[of a x] by (rule sigma_sets_Un)
  qed
next
  fix x a b
  assume x  $\in$  sigma_sets ?prod ?sets and (a, b)  $\in$  x
  from sigma_sets_into_sp[OF _ this(1)] this(2)
  show a  $\in$  A and b  $\in$  B by auto
qed

proposition sets_pair_eq:

```

```

assumes Ea: Ea  $\subseteq$  Pow (space A) sets A = sigma_sets (space A) Ea
and Ca: countable Ca Ca  $\subseteq$  Ea  $\bigcup$  Ca = space A
and Eb: Eb  $\subseteq$  Pow (space B) sets B = sigma_sets (space B) Eb
and Cb: countable Cb Cb  $\subseteq$  Eb  $\bigcup$  Cb = space B
shows sets (A  $\otimes_M$  B) = sets (sigma (space A  $\times$  space B) { a  $\times$  b | a b. a  $\in$ 
Ea  $\wedge$  b  $\in$  Eb })
  (is _ = sets (sigma ? $\Omega$  ?E))
proof
  show sets (sigma ? $\Omega$  ?E)  $\subseteq$  sets (A  $\otimes_M$  B)
    using Ea(1) Eb(1) by (subst sigma_le_sets) (auto simp: Ea(2) Eb(2))
  have ?E  $\subseteq$  Pow ? $\Omega$ 
    using Ea(1) Eb(1) by auto
  then have E: a  $\in$  Ea  $\implies$  b  $\in$  Eb  $\implies$  a  $\times$  b  $\in$  sets (sigma ? $\Omega$  ?E) for a b
    by auto
  have sets (A  $\otimes_M$  B)  $\subseteq$  sets (Sup {vimage_algebra ? $\Omega$  fst A, vimage_algebra
? $\Omega$  snd B})
    unfolding sets_pair_eq_sets_fst_snd ..
  also have vimage_algebra ? $\Omega$  fst A = vimage_algebra ? $\Omega$  fst (sigma (space A)
Ea)
    by (intro vimage_algebra_cong[OF refl refl]) (simp add: Ea)
  also have ... = sigma ? $\Omega$  {fst - ' A  $\cap$  ? $\Omega$  | A. A  $\in$  Ea}
    by (intro Ea vimage_algebra_sigma) auto
  also have vimage_algebra ? $\Omega$  snd B = vimage_algebra ? $\Omega$  snd (sigma (space B)
Eb)
    by (intro vimage_algebra_cong[OF refl refl]) (simp add: Eb)
  also have ... = sigma ? $\Omega$  {snd - ' A  $\cap$  ? $\Omega$  | A. A  $\in$  Eb}
    by (intro Eb vimage_algebra_sigma) auto
  also have {sigma ? $\Omega$  {fst - ' Aa  $\cap$  ? $\Omega$  | Aa. Aa  $\in$  Ea}, sigma ? $\Omega$  {snd - ' Aa  $\cap$ 
? $\Omega$  | Aa. Aa  $\in$  Eb}} =
    sigma ? $\Omega$  { {fst - ' Aa  $\cap$  ? $\Omega$  | Aa. Aa  $\in$  Ea}, {snd - ' Aa  $\cap$  ? $\Omega$  | Aa. Aa  $\in$  Eb}}
    by auto
  also have sets (SUP S  $\in$  { {fst - ' Aa  $\cap$  ? $\Omega$  | Aa. Aa  $\in$  Ea}, {snd - ' Aa  $\cap$  ? $\Omega$  | Aa.
Aa  $\in$  Eb}}. sigma ? $\Omega$  S) =
    sets (sigma ? $\Omega$  ( $\bigcup$  { {fst - ' Aa  $\cap$  ? $\Omega$  | Aa. Aa  $\in$  Ea}, {snd - ' Aa  $\cap$  ? $\Omega$  | Aa. Aa
 $\in$  Eb}}))
    using Ea(1) Eb(1) by (intro sets_Sup_sigma) auto
  also have ...  $\subseteq$  sets (sigma ? $\Omega$  ?E)
proof (subst sigma_le_sets, safe intro!: space_in_measure_of)
  fix a assume a  $\in$  Ea
  then have fst - ' a  $\cap$  ? $\Omega$  = ( $\bigcup$  b  $\in$  Cb. a  $\times$  b)
    using Cb(3)[symmetric] Ea(1) by auto
  then show fst - ' a  $\cap$  ? $\Omega$   $\in$  sets (sigma ? $\Omega$  ?E)
    using Cb  $\langle$  a  $\in$  Ea  $\rangle$  by (auto intro!: sets.countable_UN' E)
next
  fix b assume b  $\in$  Eb
  then have snd - ' b  $\cap$  ? $\Omega$  = ( $\bigcup$  a  $\in$  Ca. a  $\times$  b)
    using Ca(3)[symmetric] Eb(1) by auto
  then show snd - ' b  $\cap$  ? $\Omega$   $\in$  sets (sigma ? $\Omega$  ?E)
    using Ca  $\langle$  b  $\in$  Eb  $\rangle$  by (auto intro!: sets.countable_UN' E)

```

1512

qed
 finally show $\text{sets } (A \otimes_M B) \subseteq \text{sets } (\text{sigma } ?\Omega \text{ } ?E)$.
 qed

proposition *borel_prod*:
 $(\text{borel } \otimes_M \text{ borel}) = (\text{borel} :: ('a::\text{second\_countable\_topology} \times 'b::\text{second\_countable\_topology})$
measure)
 (is $?P = ?B$)
proof –
 have $?B = \text{sigma } UNIV \{A \times B \mid A \text{ B. open } A \wedge \text{open } B\}$
 by (rule *second_countable_borel_measurable[OF open_prod_generated]*)
 also have $\dots = ?P$
 unfolding *borel_def*
 by (subst *sigma_prod*) (auto intro!: *exI*[of _ {UNIV}])
 finally show *?thesis* ..
 qed

proposition *pair_measure_count_space*:
 assumes *A*: *finite A* and *B*: *finite B*
 shows $\text{count\_space } A \otimes_M \text{count\_space } B = \text{count\_space } (A \times B)$ (is $?P = ?C$)
proof (rule *measure_eqI*)
 interpret *A*: *finite_measure count_space A* by (rule *finite_measure_count_space*)
 fact
 interpret *B*: *finite_measure count_space B* by (rule *finite_measure_count_space*)
 fact
 interpret *P*: *pair_sigma_finite count_space A count_space B* ..
 show *eq*: $\text{sets } ?P = \text{sets } ?C$
 by (simp add: *sets_pair_measure sigma_sets_pair_measure_generator_finite A B*)
 fix *X* assume *X*: $X \in \text{sets } ?P$
 with *eq* have *X_subset*: $X \subseteq A \times B$ by *simp*
 with *A B* have *fin_Pair*: $\bigwedge x. \text{finite } (\text{Pair } x - 'X)$
 by (intro *finite_subset[OF _ B]*) *auto*
 have *fin_X*: *finite X* using *X_subset* by (rule *finite_subset*) (auto simp: *A B*)
 have *card*: $0 < \text{card } (\text{Pair } a - 'X)$ if $(a, b) \in X$ for *a b*
 using *card_gt_0_iff fin_Pair* that by *auto*
 then have $\text{emeasure } ?P \ X = \int^+ x. \text{emeasure } (\text{count\_space } B) (\text{Pair } x - 'X)$
 $\partial \text{count\_space } A$
 by (simp add: *B.emeasure_pair_measure_alt X*)
 also have $\dots = \text{emeasure } ?C \ X$
 apply (subst *emeasure_count_space*)
 using *card X_subset A fin_Pair fin_X*
 apply (auto simp add: *nn_integral_count_space*
 $\text{of\_nat\_sum[symmetric]}$ *card_SigmaI[symmetric]*
simp del: card_SigmaI
intro!: arg_cong[where f=card])
 done
 finally show $\text{emeasure } ?P \ X = \text{emeasure } ?C \ X$.

qed

lemma *emeasure_prod_count_space*:

assumes $A: A \in \text{sets } (\text{count\_space } \text{UNIV} \otimes_M M)$ (**is** $A \in \text{sets } (?A \otimes_M ?B)$)
shows $\text{emeasure } (?A \otimes_M ?B) A = (\int^+ x. \int^+ y. \text{indicator } A (x, y) \partial ?B \partial ?A)$
by (rule *emeasure_measure_of*[*OF pair_measure_def*])
 (auto simp: *countably_additive_def positive_def suminf_indicator A*
nn_integral_suminf[symmetric] dest: *sets.sets_into_space*)

lemma *emeasure_prod_count_space_single*[*simp*]: $\text{emeasure } (\text{count\_space } \text{UNIV} \otimes_M \text{count\_space } \text{UNIV}) \{x\} = 1$

proof –

have [*simp*]: $\bigwedge a b x y. \text{indicator } \{(a, b)\} (x, y) = (\text{indicator } \{a\} x * \text{indicator } \{b\} y :: \text{ennreal})$
by (auto split: *split_indicator*)
show *?thesis*
by (cases x) (auto simp: *emeasure_prod_count_space nn_integral_cmult sets_Pair*)

qed

lemma *emeasure_count_space_prod_eq*:

fixes $A :: ('a \times 'b) \text{ set}$
assumes $A: A \in \text{sets } (\text{count\_space } \text{UNIV} \otimes_M \text{count\_space } \text{UNIV})$ (**is** $A \in \text{sets } (?A \otimes_M ?B)$)
shows $\text{emeasure } (?A \otimes_M ?B) A = \text{emeasure } (\text{count\_space } \text{UNIV}) A$
proof –
 { **fix** $A :: ('a \times 'b) \text{ set}$ **assume** *countable A*
then have $\text{emeasure } (?A \otimes_M ?B) (\bigcup a \in A. \{a\}) = (\int^+ a. \text{emeasure } (?A \otimes_M ?B) \{a\} \partial \text{count\_space } A)$
by (*intro emeasure_UN_countable*) (auto simp: *sets_Pair disjoint_family_on_def*)
also have $\dots = (\int^+ a. \text{indicator } A a \partial \text{count\_space } \text{UNIV})$
by (*subst nn_integral_count_space_indicator*) auto
finally have $\text{emeasure } (?A \otimes_M ?B) A = \text{emeasure } (\text{count\_space } \text{UNIV}) A$
by *simp* }
note $* = \text{this}$

show *?thesis*

proof *cases*

assume *finite A* **then show** *?thesis*

by (*intro * countable_finite*)

next

assume *infinite A*

then obtain C **where** *countable C* **and** *infinite C* **and** $C \subseteq A$

by (auto dest: *infinite_countable_subset'*)

with A **have** $\text{emeasure } (?A \otimes_M ?B) C \leq \text{emeasure } (?A \otimes_M ?B) A$

by (*intro emeasure_mono*) auto

also have $\text{emeasure } (?A \otimes_M ?B) C = \text{emeasure } (\text{count\_space } \text{UNIV}) C$

using $\langle \text{countable } C \rangle$ **by** (rule $*$)

finally show *?thesis*

```

    using ⟨infinite C⟩ ⟨infinite A⟩ by (simp add: top_unique)
  qed
qed

lemma nn_integral_count_space_prod_eq:
  nn_integral (count_space UNIV  $\otimes_M$  count_space UNIV) f = nn_integral
  (count_space UNIV) f
  (is nn_integral ?P f = _)
proof cases
  assume cntbl: countable {x. f x  $\neq$  0}
  have [simp]:  $\bigwedge x. \text{card } (\{x\} \cap \{x. f x \neq 0\}) = (\text{indicator } \{x. f x \neq 0\} x :: \text{ennreal})$ 
    by (auto split: split_indicator)
  have [measurable]:  $\bigwedge y. (\lambda x. \text{indicator } \{y\} x) \in \text{borel\_measurable } ?P$ 
    by (rule measurable_discrete_difference [of  $\lambda x. 0 \_ \text{borel } \{y\} \lambda x. \text{indicator } \{y\}$ 
  x for y])
    (auto intro: sets_Pair)

  have  $(\int^{+x}. f x \partial ?P) = (\int^{+x}. \int^{+x'}. f x * \text{indicator } \{x\} x' \partial \text{count\_space } \{x. f$ 
  x  $\neq 0\} \partial ?P)$ 
    by (auto simp add: nn_integral_cmult nn_integral_indicator' intro!: nn_integral_cong
  split: split_indicator)
  also have  $\dots = (\int^{+x}. \int^{+x'}. f x' * \text{indicator } \{x'\} x \partial \text{count\_space } \{x. f x \neq 0\}$ 
   $\partial ?P)$ 
    by (auto intro!: nn_integral_cong split: split_indicator)
  also have  $\dots = (\int^{+x'}. \int^{+x}. f x' * \text{indicator } \{x'\} x \partial ?P \partial \text{count\_space } \{x. f x$ 
   $\neq 0\})$ 
    by (intro nn_integral_count_space_nn_integral cntbl) auto
  also have  $\dots = (\int^{+x'}. f x' \partial \text{count\_space } \{x. f x \neq 0\})$ 
    by (intro nn_integral_cong) (auto simp: nn_integral_cmult sets_Pair)
  finally show ?thesis
    by (auto simp add: nn_integral_count_space_indicator intro!: nn_integral_cong
  split: split_indicator)
next
  { fix x assume f x  $\neq$  0
    then have  $(\exists r \geq 0. 0 < r \wedge f x = \text{ennreal } r) \vee f x = \infty$ 
      by (cases f x rule: ennreal_cases) (auto simp: less_le)
    then have  $\exists n. \text{ennreal } (1 / \text{real } (\text{Suc } n)) \leq f x$ 
      by (auto elim!: nat_approx_posE intro!: less_imp_le) }
  note * = this

  assume cntbl: uncountable {x. f x  $\neq$  0}
  also have  $\{x. f x \neq 0\} = (\bigcup n. \{x. 1 / \text{Suc } n \leq f x\})$ 
    using * by auto
  finally obtain n where infinite {x.  $1 / \text{Suc } n \leq f x$ }
    by (meson countableI_type countable_UN uncountable_infinite)
  then obtain C where C:  $C \subseteq \{x. 1 / \text{Suc } n \leq f x\}$  and countable C infinite C
    by (metis infinite_countable_subset')

  have [measurable]: C  $\in$  sets ?P

```

```

using sets.countable[OF _ ‹countable C›, of ?P] by (auto simp: sets_Pair)

have (∫+ x. ennreal (1/Suc n) * indicator C x ∂?P) ≤ nn_integral ?P f
  using C by (intro nn_integral_mono) (auto split: split_indicator simp: zero_ereal_def[symmetric])
moreover have (∫+ x. ennreal (1/Suc n) * indicator C x ∂?P) = ∞
  using ‹infinite C› by (simp add: nn_integral_cmult emeasure_count_space_prod_eq
ennreal_mult_top)
moreover have (∫+ x. ennreal (1/Suc n) * indicator C x ∂count_space UNIV)
≤ nn_integral (count_space UNIV) f
  using C by (intro nn_integral_mono) (auto split: split_indicator simp: zero_ereal_def[symmetric])
moreover have (∫+ x. ennreal (1/Suc n) * indicator C x ∂count_space UNIV)
= ∞
  using ‹infinite C› by (simp add: nn_integral_cmult ennreal_mult_top)
ultimately show ?thesis
  by (simp add: top_unique)
qed

```

```

theorem pair_measure_density:
  assumes f: f ∈ borel_measurable M1
  assumes g: g ∈ borel_measurable M2
  assumes sigma_finite_measure M2 sigma_finite_measure (density M2 g)
  shows density M1 f ⊗M density M2 g = density (M1 ⊗M M2) (λ(x,y). f x *
g y) (is ?L = ?R)
proof (rule measure_eqI)
  interpret M2: sigma_finite_measure M2 by fact
  interpret D2: sigma_finite_measure density M2 g by fact

  fix A assume A: A ∈ sets ?L
  with f g have (∫+ x. f x * ∫+ y. g y * indicator A (x, y) ∂M2 ∂M1) =
    (∫+ x. ∫+ y. f x * g y * indicator A (x, y) ∂M2 ∂M1)
    by (intro nn_integral_cong_AE)
    (auto simp add: nn_integral_cmult[symmetric] ac_simps)
  with A f g show emeasure ?L A = emeasure ?R A
    by (simp add: D2.emeasure_pair_measure_emeasure_density nn_integral_density
M2.nn_integral_fst[symmetric]
cong: nn_integral_cong)
qed simp

```

```

lemma sigma_finite_measure_distr:
  assumes sigma_finite_measure (distr M N f) and f: f ∈ measurable M N
  shows sigma_finite_measure M
proof -
  interpret sigma_finite_measure distr M N f by fact
  obtain A where A: countable A A ⊆ sets (distr M N f)
    ∪ A = space (distr M N f) ∀ a ∈ A. emeasure (distr M N f) a ≠ ∞
    using sigma_finite_countable by auto
  show ?thesis
proof
  show ∃ A. countable A ∧ A ⊆ sets M ∧ ∪ A = space M ∧ (∀ a ∈ A. emeasure

```

```

M a ≠ ∞)
  using A f
  by (intro exI[of_ (λa. f - ' a ∩ space M) ' A])
    (auto simp: emeasure_distr set_eq_iff subset_eq intro: measurable_space)
qed
qed

```

```

lemma pair_measure_distr:
  assumes f: f ∈ measurable M S and g: g ∈ measurable N T
  assumes sigma_finite_measure (distr N T g)
  shows distr M S f ⊗M distr N T g = distr (M ⊗M N) (S ⊗M T) (λ(x, y).
(f x, g y)) (is ?P = ?D)
proof (rule measure_eqI)
  interpret T: sigma_finite_measure distr N T g by fact
  interpret N: sigma_finite_measure N by (rule sigma_finite_measure_distr)
fact+

```

```

  fix A assume A: A ∈ sets ?P
  with f g show emeasure ?P A = emeasure ?D A
  by (auto simp add: N.emeasure_pair_measure_alt space_pair_measure_emeasure_distr
    T.emeasure_pair_measure_alt nn_integral_distr
    intro!: nn_integral_cong arg_cong[where f=emeasure N])
qed simp

```

```

lemma pair_measure_eqI:
  assumes sigma_finite_measure M1 sigma_finite_measure M2
  assumes sets: sets (M1 ⊗M M2) = sets M
  assumes emeasure: ∧A B. A ∈ sets M1 ⇒ B ∈ sets M2 ⇒ emeasure M1 A
* emeasure M2 B = emeasure M (A × B)
  shows M1 ⊗M M2 = M
proof -
  interpret M1: sigma_finite_measure M1 by fact
  interpret M2: sigma_finite_measure M2 by fact
  interpret pair_sigma_finite M1 M2 ..
  let ?E = {a × b | a b. a ∈ sets M1 ∧ b ∈ sets M2}
  let ?P = M1 ⊗M M2
  obtain F :: nat ⇒ ('a × 'b) set where F:
    range F ⊆ ?E incseq F ∪ (range F) = space M1 × space M2 ∀ i. emeasure ?P
(F i) ≠ ∞
  using sigma_finite_up_in_pair_measure_generator
  by blast
  show ?thesis
proof (rule measure_eqI_generator_eq[OF Int_stable_pair_measure_generator[of M1 M2]])
  show ?E ⊆ Pow (space ?P)
  using sets.space_closed[of M1] sets.space_closed[of M2] by (auto simp:
space_pair_measure)
  show sets ?P = sigma_sets (space ?P) ?E

```



```

    by (simp add: sets_pair_measure space_pair_measure)
  then show sets M = sigma_sets (space ?P) ?E
    using sets[symmetric] by simp
next
  show range F  $\subseteq$  ?E ( $\bigcup i. F i$ ) = space ?P  $\wedge$  i. emeasure ?P (F i)  $\neq \infty$ 
    using F by (auto simp: space_pair_measure)
next
  fix X assume X  $\in$  ?E
  then obtain A B where X[simp]: X = A  $\times$  B and A: A  $\in$  sets M1 and B:
B  $\in$  sets M2 by auto
  then have emeasure ?P X = emeasure M1 A * emeasure M2 B
    by (simp add: M2.emeasure_pair_measure_Times)
  also have ... = emeasure M (A  $\times$  B)
    using A B emeasure by auto
  finally show emeasure ?P X = emeasure M X
    by simp
qed
qed

```

lemma sets_pair_countable:

```

  assumes countable S1 countable S2
  assumes M: sets M = Pow S1 and N: sets N = Pow S2
  shows sets (M  $\otimes_M$  N) = Pow (S1  $\times$  S2)
proof auto
  fix x a b assume x: x  $\in$  sets (M  $\otimes_M$  N) (a, b)  $\in$  x
  from sets.sets_into_space[OF x(1)] x(2)
    sets_eq_imp_space_eq[of N count_space S2] sets_eq_imp_space_eq[of M
count_space S1] M N
  show a  $\in$  S1 b  $\in$  S2
    by (auto simp: space_pair_measure)
next
  fix X assume X: X  $\subseteq$  S1  $\times$  S2
  then have countable X
    by (metis countable_subset  $\langle$ countable S1 $\rangle$   $\langle$ countable S2 $\rangle$  countable_SIGMA)
  have X = ( $\bigcup (a, b) \in X. \{a\} \times \{b\}$ ) by auto
  also have ...  $\in$  sets (M  $\otimes_M$  N)
    using X
  by (safe intro!: sets.countable_UN'  $\langle$ countable X $\rangle$  subsetI pair_measureI) (auto
simp: M N)
  finally show X  $\in$  sets (M  $\otimes_M$  N) .
qed

```

lemma pair_measure_countable:

```

  assumes countable S1 countable S2
  shows count_space S1  $\otimes_M$  count_space S2 = count_space (S1  $\times$  S2)
proof (rule pair_measure_eqI)
  show sigma_finite_measure (count_space S1) sigma_finite_measure (count_space
S2)
    using assms by (auto intro!: sigma_finite_measure_count_space_countable)

```

```

show sets (count_space S1  $\otimes_M$  count_space S2) = sets (count_space (S1  $\times$ 
S2))
  by (subst sets_pair_countable[OF assms]) auto
next
  fix A B assume A  $\in$  sets (count_space S1) B  $\in$  sets (count_space S2)
  then show emeasure (count_space S1) A * emeasure (count_space S2) B =
    emeasure (count_space (S1  $\times$  S2)) (A  $\times$  B)
  by (subst (1 2 3) emeasure_count_space) (auto simp: finite_cartesian_product_iff
ennreal_mult_top ennreal_top_mult)
qed

```

proposition nn_integral_fst_count_space:

$(\int^+ x. \int^+ y. f(x, y) \partial \text{count\_space UNIV} \partial \text{count\_space UNIV}) = \text{integral}^N$
 $(\text{count\_space UNIV}) f$
(is ?lhs = ?rhs)

proof(cases)

```

assume *: countable {xy. f xy  $\neq$  0}
let ?A = fst ' {xy. f xy  $\neq$  0}
let ?B = snd ' {xy. f xy  $\neq$  0}
from * have [simp]: countable ?A countable ?B by (rule countable_image)+
have ?lhs =  $(\int^+ x. \int^+ y. f(x, y) \partial \text{count\_space UNIV} \partial \text{count\_space ?A})$ 
  by (rule nn_integral_count_space_eq)
  (auto simp add: nn_integral_0_iff AE AE_count_space not_le intro: rev_image_eqI)
also have ... =  $(\int^+ x. \int^+ y. f(x, y) \partial \text{count\_space ?B} \partial \text{count\_space ?A})$ 
  by (intro nn_integral_count_space_eq nn_integral_cong) (auto intro: rev_image_eqI)
also have ... =  $(\int^+ xy. f xy \partial \text{count\_space} (?A \times ?B))$ 
  by (subst sigma_finite_measure.nn_integral_fst)
  (simp_all add: sigma_finite_measure_count_space_countable pair_measure_countable)
also have ... = ?rhs
  by (rule nn_integral_count_space_eq) (auto intro: rev_image_eqI)
finally show ?thesis .

```

next

```

{ fix xy assume f xy  $\neq$  0
  then have  $(\exists r \geq 0. 0 < r \wedge f xy = \text{ennreal } r) \vee f xy = \infty$ 
    by (cases f xy rule: ennreal_cases) (auto simp: less_le)
  then have  $\exists n. \text{ennreal } (1 / \text{real } (\text{Suc } n)) \leq f xy$ 
    by (auto elim!: nat_approx_posE intro!: less_imp_le) }
note * = this

```

```

assume cntbl: uncountable {xy. f xy  $\neq$  0}
also have {xy. f xy  $\neq$  0} =  $(\bigcup n. \{xy. 1/\text{Suc } n \leq f xy\})$ 
  using * by auto
finally obtain n where infinite {xy.  $1/\text{Suc } n \leq f xy$ }
  by (meson countableI_type countable_UN uncountable_infinite)
then obtain C where C: C  $\subseteq$  {xy.  $1/\text{Suc } n \leq f xy$ } and countable C infinite
C
  by (metis infinite_countable_subset')

```

have $\infty = (\int^+ xy. \text{ennreal } (1 / \text{Suc } n) * \text{indicator } C xy \partial \text{count\_space UNIV})$

```

    using ‹infinite C› by (simp add: nn_integral_cmult ennreal_mult_top)
  also have ... ≤ ?rhs using C
    by (intro nn_integral_mono) (auto split: split_indicator)
  finally have ?rhs = ∞ by (simp add: top_unique)
  moreover have ?lhs = ∞
  proof (cases finite (fst ' C))
    case True
    then obtain x C' where x: x ∈ fst ' C
      and C': C' = fst - ' {x} ∩ C
      and infinite C'
      using ‹infinite C› by (auto elim!: inf_img_fin_domE')
    from x C C' have **: C' ⊆ {xy. 1 / Suc n ≤ f xy} by auto

    from C' ‹infinite C'› have infinite (snd ' C')
      by (auto dest!: finite_imageD simp add: inj_on_def)
    then have ∞ = (∫+ y. ennreal (1 / Suc n) * indicator (snd ' C') y ∂count_space UNIV)
    by (simp add: nn_integral_cmult ennreal_mult_top)
    also have ... = (∫+ y. ennreal (1 / Suc n) * indicator C' (x, y) ∂count_space UNIV)
    by (rule nn_integral_cong) (force split: split_indicator intro: rev_image_eqI simp add: C')
    also have ... = (∫+ x'. (∫+ y. ennreal (1 / Suc n) * indicator C' (x, y) ∂count_space UNIV) * indicator {x} x' ∂count_space UNIV)
    by (simp add: one_ereal_def[symmetric])
    also have ... ≤ (∫+ x. ∫+ y. ennreal (1 / Suc n) * indicator C' (x, y) ∂count_space UNIV ∂count_space UNIV)
    by (rule nn_integral_mono) (simp split: split_indicator)
    also have ... ≤ ?lhs using **
    by (intro nn_integral_mono) (auto split: split_indicator)
    finally show ?thesis by (simp add: top_unique)
  next
    case False
    define C' where C' = fst ' C
    have ∞ = ∫+ x. ennreal (1 / Suc n) * indicator C' x ∂count_space UNIV
      using C'_def False by (simp add: nn_integral_cmult ennreal_mult_top)
    also have ... = ∫+ x. ∫+ y. ennreal (1 / Suc n) * indicator C' x * indicator {SOME y. (x, y) ∈ C} y ∂count_space UNIV ∂count_space UNIV
    by (auto simp add: one_ereal_def[symmetric] max_def intro: nn_integral_cong)
    also have ... ≤ ∫+ x. ∫+ y. ennreal (1 / Suc n) * indicator C (x, y) ∂count_space UNIV ∂count_space UNIV
    by (intro nn_integral_mono) (auto simp add: C'_def split: split_indicator intro: someI)
    also have ... ≤ ?lhs using C
    by (intro nn_integral_mono) (auto split: split_indicator)
    finally show ?thesis by (simp add: top_unique)
  qed
  ultimately show ?thesis by simp
qed

```

proposition *nn_integral_snd_count_space:*

$(\int^+ y. \int^+ x. f(x, y) \partial \text{count\_space UNIV} \partial \text{count\_space UNIV}) = \text{integral}^N$
 $(\text{count\_space UNIV}) f$
 (is ?lhs = ?rhs)

proof –

have ?lhs = $(\int^+ y. \int^+ x. (\lambda(y, x). f(x, y)) (y, x) \partial \text{count\_space UNIV} \partial \text{count\_space UNIV})$

by(simp)

also have ... = $\int^+ yx. (\lambda(y, x). f(x, y)) yx \partial \text{count\_space UNIV}$

by(rule nn_integral_fst_count_space)

also have ... = $\int^+ xy. f xy \partial \text{count\_space} ((\lambda(x, y). (y, x)) ' \text{UNIV})$

by(subst nn_integral_bij_count_space[OF inj_on_imp_bij_betw, symmetric])
 (simp_all add: inj_on_def split_def)

also have ... = ?rhs **by**(rule nn_integral_count_space_eq) auto

finally show ?thesis .

qed

lemma *measurable_pair_measure_countable1:*

assumes countable A

and [measurable]: $\bigwedge x. x \in A \implies (\lambda y. f(x, y)) \in \text{measurable } N K$

shows $f \in \text{measurable} (\text{count\_space } A \otimes_M N) K$

using __ __ assms(1)

by(rule measurable_compose_countable'[**where** $f = \lambda a b. f(a, \text{snd } b)$ **and** $g = \text{fst}$
and $I = A$, simplified])simp_all

6.7.5 Product of Borel spaces

theorem *borel_Times:*

fixes A :: 'a::topological_space set **and** B :: 'b::topological_space set

assumes A: A ∈ sets borel **and** B: B ∈ sets borel

shows A × B ∈ sets borel

proof –

have A × B = (A × UNIV) ∩ (UNIV × B)

by auto

moreover

{ **have** A ∈ sigma_sets UNIV {S. open S} **using** A **by** (simp add: sets_borel)

then have A × UNIV ∈ sets borel

proof (induct A)

case (Basic S) **then show** ?case

by (auto intro!: borel_open open_Times)

next

case (Compl A)

moreover have *: (UNIV − A) × UNIV = UNIV − (A × UNIV)

by auto

ultimately show ?case

unfolding * **by** auto

next

case (Union A)

```

    moreover have *:  $(\bigcup (A \text{ ' } UNIV)) \times UNIV = \bigcup ((\lambda i. A \ i \times UNIV) \text{ ' } UNIV)$ 
      by auto
    ultimately show ?case
      unfolding * by auto
  qed simp }
moreover
{ have  $B \in \text{sigma\_sets } UNIV \{S. \text{open } S\}$  using B by (simp add: sets_borel)
  then have  $UNIV \times B \in \text{sets borel}$ 
  proof (induct B)
    case (Basic S) then show ?case
      by (auto intro!: borel_open open_Times)
  next
    case (Compl B)
    moreover have *:  $UNIV \times (UNIV - B) = UNIV - (UNIV \times B)$ 
      by auto
    ultimately show ?case
      unfolding * by auto
  next
    case (Union B)
    moreover have *:  $UNIV \times (\bigcup (B \text{ ' } UNIV)) = \bigcup ((\lambda i. UNIV \times B \ i) \text{ ' } UNIV)$ 
      by auto
    ultimately show ?case
      unfolding * by auto
  qed simp }
ultimately show ?thesis
  by auto
qed

lemma finite_measure_pair_measure:
  assumes finite_measure M finite_measure N
  shows finite_measure  $(N \otimes_M M)$ 
proof (rule finite_measureI)
  interpret M: finite_measure M by fact
  interpret N: finite_measure N by fact
  show  $\text{emeasure } (N \otimes_M M) (\text{space } (N \otimes_M M)) \neq \infty$ 
    by (auto simp: space_pair_measure M.emeasure_pair_measure_Times en-
nreal_mult_eq_top_iff)
qed

end

```

6.8 Finite Product Measure

```

theory Finite_Product_Measure
imports Binary_Product_Measure Function_Topology
begin

```

```

lemma PiE_choice:  $(\exists f \in \text{Pi}_E \ I \ F. \forall i \in I. P \ i \ (f \ i)) \longleftrightarrow (\forall i \in I. \exists x \in F \ i. P \ i \ x)$ 
  by (auto simp: Bex_def PiE_iff Ball_def dest!: choice_iff'[THEN iffD1])

```

(force intro: exI[of _ restrict f I for f])

lemma case_prod_const: $(\lambda(i, j). c) = (\lambda_. c)$
 by auto

6.8.1 More about Function restricted by *extensional*

definition

merge $I J = (\lambda(x, y) i. \text{if } i \in I \text{ then } x i \text{ else if } i \in J \text{ then } y i \text{ else undefined})$

lemma merge_apply[simp]:

$I \cap J = \{\} \implies i \in I \implies \text{merge } I J (x, y) i = x i$
 $I \cap J = \{\} \implies i \in J \implies \text{merge } I J (x, y) i = y i$
 $J \cap I = \{\} \implies i \in I \implies \text{merge } I J (x, y) i = x i$
 $J \cap I = \{\} \implies i \in J \implies \text{merge } I J (x, y) i = y i$
 $i \notin I \implies i \notin J \implies \text{merge } I J (x, y) i = \text{undefined}$
unfolding merge_def by auto

lemma merge_commute:

$I \cap J = \{\} \implies \text{merge } I J (x, y) = \text{merge } J I (y, x)$
 by (force simp: merge_def)

lemma Pi_cancel_merge_range[simp]:

$I \cap J = \{\} \implies x \in \text{Pi } I (\text{merge } I J (A, B)) \longleftrightarrow x \in \text{Pi } I A$
 $I \cap J = \{\} \implies x \in \text{Pi } I (\text{merge } J I (B, A)) \longleftrightarrow x \in \text{Pi } I A$
 $J \cap I = \{\} \implies x \in \text{Pi } I (\text{merge } I J (A, B)) \longleftrightarrow x \in \text{Pi } I A$
 $J \cap I = \{\} \implies x \in \text{Pi } I (\text{merge } J I (B, A)) \longleftrightarrow x \in \text{Pi } I A$
 by (auto simp: Pi_def)

lemma Pi_cancel_merge[simp]:

$I \cap J = \{\} \implies \text{merge } I J (x, y) \in \text{Pi } I B \longleftrightarrow x \in \text{Pi } I B$
 $J \cap I = \{\} \implies \text{merge } I J (x, y) \in \text{Pi } I B \longleftrightarrow x \in \text{Pi } I B$
 $I \cap J = \{\} \implies \text{merge } I J (x, y) \in \text{Pi } J B \longleftrightarrow y \in \text{Pi } J B$
 $J \cap I = \{\} \implies \text{merge } I J (x, y) \in \text{Pi } J B \longleftrightarrow y \in \text{Pi } J B$
 by (auto simp: Pi_def)

lemma extensional_merge[simp]: $\text{merge } I J (x, y) \in \text{extensional } (I \cup J)$
 by (auto simp: extensional_def)

lemma restrict_merge[simp]:

$I \cap J = \{\} \implies \text{restrict } (\text{merge } I J (x, y)) I = \text{restrict } x I$
 $I \cap J = \{\} \implies \text{restrict } (\text{merge } I J (x, y)) J = \text{restrict } y J$
 $J \cap I = \{\} \implies \text{restrict } (\text{merge } I J (x, y)) I = \text{restrict } x I$
 $J \cap I = \{\} \implies \text{restrict } (\text{merge } I J (x, y)) J = \text{restrict } y J$
 by (auto simp: restrict_def)

lemma split_merge: $P (\text{merge } I J (x, y) i) \longleftrightarrow (i \in I \longrightarrow P (x i)) \wedge (i \in J - I \longrightarrow P (y i)) \wedge (i \notin I \cup J \longrightarrow P \text{ undefined})$
unfolding merge_def by auto

lemma *PiE_cancel_merge*[simp]:

$I \cap J = \{\} \implies$
 $\text{merge } I \ J \ (x, y) \in \text{Pi}_E \ (I \cup J) \ B \longleftrightarrow x \in \text{Pi } I \ B \wedge y \in \text{Pi } J \ B$
by (auto simp: *PiE_def restrict_Pi_cancel*)

lemma *merge_singleton*[simp]: $i \notin I \implies \text{merge } I \ \{i\} \ (x, y) = \text{restrict } (x(i := y \ i)) \ (\text{insert } i \ I)$

unfolding *merge_def* **by** (auto simp: *fun_eq_iff*)

lemma *extensional_merge_sub*: $I \cup J \subseteq K \implies \text{merge } I \ J \ (x, y) \in \text{extensional } K$

unfolding *merge_def extensional_def* **by** auto

lemma *merge_restrict*[simp]:

$\text{merge } I \ J \ (\text{restrict } x \ I, y) = \text{merge } I \ J \ (x, y)$
 $\text{merge } I \ J \ (x, \text{restrict } y \ J) = \text{merge } I \ J \ (x, y)$
unfolding *merge_def* **by** auto

lemma *merge_x_x_eq_restrict*[simp]:

$\text{merge } I \ J \ (x, x) = \text{restrict } x \ (I \cup J)$
unfolding *merge_def* **by** auto

lemma *injective_vimage_restrict*:

assumes $J: J \subseteq I$

and *sets*: $A \subseteq (\Pi_E \ i \in J. S \ i) \ B \subseteq (\Pi_E \ i \in J. S \ i)$ **and** *ne*: $(\Pi_E \ i \in I. S \ i) \neq \{\}$

and *eq*: $(\lambda x. \text{restrict } x \ J) -' A \cap (\Pi_E \ i \in I. S \ i) = (\lambda x. \text{restrict } x \ J) -' B \cap (\Pi_E \ i \in I. S \ i)$

shows $A = B$

proof (intro *set_eqI*)

fix x

from *ne* **obtain** y **where** $y: \bigwedge i. i \in I \implies y \ i \in S \ i$ **by** auto

have $J \cap (I - J) = \{\}$ **by** auto

show $x \in A \longleftrightarrow x \in B$

proof *cases*

assume $x: x \in (\Pi_E \ i \in J. S \ i)$

have $x \in A \longleftrightarrow \text{merge } J \ (I - J) \ (x, y) \in (\lambda x. \text{restrict } x \ J) -' A \cap (\Pi_E \ i \in I. S \ i)$

using $y \ x \ \langle J \subseteq I \rangle$ *PiE_cancel_merge*[of $J \ I - J \ x \ y \ S$]

by (auto simp del: *PiE_cancel_merge* simp add: *Un_absorb1*)

then show $x \in A \longleftrightarrow x \in B$

using $y \ x \ \langle J \subseteq I \rangle$ *PiE_cancel_merge*[of $J \ I - J \ x \ y \ S$]

by (auto simp del: *PiE_cancel_merge* simp add: *Un_absorb1 eq*)

qed (insert *sets*, auto)

qed

lemma *restrict_vimage*:

$I \cap J = \{\} \implies$

$(\lambda x. (\text{restrict } x \ I, \text{restrict } x \ J)) -' (\text{Pi}_E \ I \ E \times \text{Pi}_E \ J \ F) = \text{Pi } (I \cup J) \ (\text{merge } I \ J \ (E, F))$

by (auto simp: restrict_Pi_cancel PiE_def)

lemma merge_vimage:

$I \cap J = \{\} \implies \text{merge } I \ J \text{ -- ' } \Pi_E (I \cup J) \ E = \Pi_E I \ E \times \Pi_E J \ E$

by (auto simp: restrict_Pi_cancel PiE_def)

6.8.2 Finite product spaces

definition prod_emb where

$\text{prod\_emb } I \ M \ K \ X = (\lambda x. \text{restrict } x \ K) \text{ -- ' } X \cap (\Pi_E i \in I. \text{space } (M \ i))$

lemma prod_emb_iff:

$f \in \text{prod\_emb } I \ M \ K \ X \longleftrightarrow f \in \text{extensional } I \wedge (\text{restrict } f \ K \in X) \wedge (\forall i \in I. f \ i \in \text{space } (M \ i))$

unfolding prod_emb_def PiE_def by auto

lemma

shows prod_emb_empty[simp]: $\text{prod\_emb } M \ L \ K \ \{\} = \{\}$

and prod_emb_Un[simp]: $\text{prod\_emb } M \ L \ K \ (A \cup B) = \text{prod\_emb } M \ L \ K \ A \cup \text{prod\_emb } M \ L \ K \ B$

and prod_emb_Int: $\text{prod\_emb } M \ L \ K \ (A \cap B) = \text{prod\_emb } M \ L \ K \ A \cap \text{prod\_emb } M \ L \ K \ B$

and prod_emb_UN[simp]: $\text{prod\_emb } M \ L \ K \ (\bigcup i \in I. F \ i) = (\bigcup i \in I. \text{prod\_emb } M \ L \ K \ (F \ i))$

and prod_emb_INT[simp]: $I \neq \{\} \implies \text{prod\_emb } M \ L \ K \ (\bigcap i \in I. F \ i) = (\bigcap i \in I. \text{prod\_emb } M \ L \ K \ (F \ i))$

and prod_emb_Diff[simp]: $\text{prod\_emb } M \ L \ K \ (A - B) = \text{prod\_emb } M \ L \ K \ A - \text{prod\_emb } M \ L \ K \ B$

by (auto simp: prod_emb_def)

lemma prod_emb_PiE: $J \subseteq I \implies (\bigwedge i. i \in J \implies E \ i \subseteq \text{space } (M \ i)) \implies$

$\text{prod\_emb } I \ M \ J \ (\Pi_E i \in J. E \ i) = (\Pi_E i \in I. \text{if } i \in J \text{ then } E \ i \text{ else } \text{space } (M \ i))$

by (force simp: prod_emb_def PiE_iff if_split_mem2)

lemma prod_emb_PiE_same_index[simp]:

$(\bigwedge i. i \in I \implies E \ i \subseteq \text{space } (M \ i)) \implies \text{prod\_emb } I \ M \ I \ (\Pi_E I \ E) = \Pi_E I \ E$

by (auto simp: prod_emb_def PiE_iff)

lemma prod_emb_trans[simp]:

$J \subseteq K \implies K \subseteq L \implies \text{prod\_emb } L \ M \ K \ (\text{prod\_emb } K \ M \ J \ X) = \text{prod\_emb } L \ M \ J \ X$

by (auto simp add: Int_absorb1 prod_emb_def PiE_def)

lemma prod_emb_Pi:

assumes $X \in (\Pi j \in J. \text{sets } (M \ j)) \ J \subseteq K$

shows $\text{prod\_emb } K \ M \ J \ (\Pi_E J \ X) = (\Pi_E i \in K. \text{if } i \in J \text{ then } X \ i \text{ else } \text{space } (M \ i))$

using assms sets.space_closed

by (auto simp: prod_emb_def PiE_iff split: if_split_asm) blast+

lemma *prod_emb_id*:

$B \subseteq (\Pi_E i \in L. \text{space } (M i)) \implies \text{prod\_emb } L \ M \ L \ B = B$
by (*auto simp: prod_emb_def subset_eq extensional_restrict*)

lemma *prod_emb_mono*:

$F \subseteq G \implies \text{prod\_emb } A \ M \ B \ F \subseteq \text{prod\_emb } A \ M \ B \ G$
by (*auto simp: prod_emb_def*)

definition *PiM* :: $'i \text{ set} \Rightarrow ('i \Rightarrow 'a \text{ measure}) \Rightarrow ('i \Rightarrow 'a) \text{ measure}$ **where**

$\text{PiM } I \ M = \text{extend\_measure } (\Pi_E i \in I. \text{space } (M i))$
 $\{(J, X). (J \neq \{\} \vee I = \{\}) \wedge \text{finite } J \wedge J \subseteq I \wedge X \in (\Pi j \in J. \text{sets } (M j))\}$
 $(\lambda(J, X). \text{prod\_emb } I \ M \ J \ (\Pi_E j \in J. X j))$
 $(\lambda(J, X). \prod j \in J \cup \{i \in I. \text{emeasure } (M i) (\text{space } (M i)) \neq 1\}. \text{ if } j \in J \text{ then } \text{emeasure } (M j) (X j) \text{ else } \text{emeasure } (M j) (\text{space } (M j)))$

definition *prod_algebra* :: $'i \text{ set} \Rightarrow ('i \Rightarrow 'a \text{ measure}) \Rightarrow ('i \Rightarrow 'a) \text{ set set}$ **where**

$\text{prod\_algebra } I \ M = (\lambda(J, X). \text{prod\_emb } I \ M \ J \ (\Pi_E j \in J. X j))$ ‘
 $\{(J, X). (J \neq \{\} \vee I = \{\}) \wedge \text{finite } J \wedge J \subseteq I \wedge X \in (\Pi j \in J. \text{sets } (M j))\}$

abbreviation

$Pi_M \ I \ M \equiv \text{PiM } I \ M$

syntax

$\text{PiM} :: \text{pttrn} \Rightarrow 'i \text{ set} \Rightarrow 'a \text{ measure} \Rightarrow ('i \Rightarrow 'a) \text{ measure} \ ((\exists \Pi_M \_ \in \_ / \_)$
 $10)$

translations

$\Pi_M \ x \in I. M == \text{CONST } \text{PiM } I \ (\%x. M)$

lemma *extend_measure_cong*:

assumes $\Omega = \Omega' \ I = I' \ G = G' \ \bigwedge i. i \in I' \implies \mu \ i = \mu' \ i$
shows $\text{extend\_measure } \Omega \ I \ G \ \mu = \text{extend\_measure } \Omega' \ I' \ G' \ \mu'$
unfolding *extend_measure_def* **by** (*auto simp add: assms*)

lemma *Pi_cong_sets*:

$\llbracket I = J; \bigwedge x. x \in I \implies M \ x = N \ x \rrbracket \implies \text{Pi } I \ M = \text{Pi } J \ N$
unfolding *Pi_def* **by** *auto*

lemma *PiM_cong*:

assumes $I = J \ \bigwedge x. x \in I \implies M \ x = N \ x$
shows $\text{PiM } I \ M = \text{PiM } J \ N$
unfolding *PiM_def*

proof (*rule extend_measure_cong, goal_cases*)

case 1

show ?case **using** *assms*

by (*subst assms(1), intro PiE_cong[of J $\lambda i. \text{space } (M i)$ $\lambda i. \text{space } (N i)$]*)

simp_all

next

case 2

```

have  $\bigwedge K. K \subseteq J \implies (\prod_{j \in K}. \text{sets } (M j)) = (\prod_{j \in K}. \text{sets } (N j))$ 
  using assms by (intro Pi_cong_sets) auto
thus ?case by (auto simp: assms)
next
case 3
show ?case using assms
  by (intro ext) (auto simp: prod_emb_def dest: PiE_mem)
next
case (4 x)
thus ?case using assms
  by (auto intro!: prod.cong_split: if_split_asm)
qed

```

```

lemma prod_algebra_sets_into_space:
   $\text{prod\_algebra } I M \subseteq \text{Pow } (\prod_{E \in I}. \text{space } (M i))$ 
  by (auto simp: prod_emb_def prod_algebra_def)

```

```

lemma prod_algebra_eq_finite:
  assumes I: finite I
  shows  $\text{prod\_algebra } I M = \{(\prod_{E \in I}. X i) \mid X. X \in (\prod_{j \in I}. \text{sets } (M j))\}$  (is ?L
  = ?R)
proof (intro iffI set_eqI)
  fix A assume A  $\in$  ?L
  then obtain J E where J:  $J \neq \{\}$   $\vee I = \{\}$  finite J  $J \subseteq I$   $\forall i \in J. E i \in \text{sets } (M i)$ 
    and A:  $A = \text{prod\_emb } I M J (\prod_{E \in J}. E j)$ 
    by (auto simp: prod_algebra_def)
  let ?A =  $\prod_{E \in I}. \text{if } i \in J \text{ then } E i \text{ else space } (M i)$ 
  have A:  $A = ?A$ 
    unfolding A using J by (intro prod_emb_PiE sets.sets_into_space) auto
  show A  $\in$  ?R unfolding A using J sets.top
    by (intro CollectI exI [of  $\lambda i. \text{if } i \in J \text{ then } E i \text{ else space } (M i)$ ]) simp
next
  fix A assume A  $\in$  ?R
  then obtain X where A:  $A = (\prod_{E \in I}. X i)$  and X:  $X \in (\prod_{j \in I}. \text{sets } (M j))$ 
  by auto
  then have A:  $A = \text{prod\_emb } I M I (\prod_{E \in I}. X i)$ 
    by (simp add: prod_emb_PiE_same_index [OF sets.sets_into_space] Pi_iff)
  from X I show A  $\in$  ?L unfolding A
    by (auto simp: prod_algebra_def)
qed

```

```

lemma prod_algebraI:
   $\text{finite } J \implies (J \neq \{\} \vee I = \{\}) \implies J \subseteq I \implies (\bigwedge i. i \in J \implies E i \in \text{sets } (M i))$ 
   $\implies \text{prod\_emb } I M J (\prod_{E \in J}. E j) \in \text{prod\_algebra } I M$ 
  by (auto simp: prod_algebra_def)

```

```

lemma prod_algebraI_finite:

```

$finite\ I \implies (\forall i \in I. E\ i \in sets\ (M\ i)) \implies (Pi_E\ I\ E) \in prod\_algebra\ I\ M$
using $prod\_algebraI[of\ I\ I\ E\ M]$ $prod\_emb\_PiE\_same\_index[of\ I\ E\ M, OF\ sets.sets\_into\_space]$ **by** $simp$

lemma Int_stable_PiE : $Int\_stable\ \{Pi_E\ J\ E \mid E. \forall i \in I. E\ i \in sets\ (M\ i)\}$

proof ($safe\ intro!$: $Int\_stableI$)

fix $E\ F$ **assume** $\forall i \in I. E\ i \in sets\ (M\ i) \ \forall i \in I. F\ i \in sets\ (M\ i)$

then show $\exists G. Pi_E\ J\ E \cap Pi_E\ J\ F = Pi_E\ J\ G \wedge (\forall i \in I. G\ i \in sets\ (M\ i))$

by ($auto\ intro!$: $exI[of\ \_ \lambda i. E\ i \cap F\ i]$ $simp$: PiE_Int)

qed

lemma $prod\_algebraE$:

assumes $A: A \in prod\_algebra\ I\ M$

obtains $J\ E$ **where** $A = prod\_emb\ I\ M\ J\ (\Pi_E\ j \in J. E\ j)$

$finite\ J \neq \{\} \vee I = \{\} \ J \subseteq I \wedge i. i \in J \implies E\ i \in sets\ (M\ i)$

using A **by** ($auto\ simp$: $prod\_algebra\_def$)

lemma $prod\_algebraE\_all$:

assumes $A: A \in prod\_algebra\ I\ M$

obtains E **where** $A = Pi_E\ I\ E \ E \in (\Pi\ i \in I. sets\ (M\ i))$

proof –

from A **obtain** $E\ J$ **where** $A: A = prod\_emb\ I\ M\ J\ (Pi_E\ J\ E)$

and $J: J \subseteq I$ **and** $E: E \in (\Pi\ i \in J. sets\ (M\ i))$

by ($auto\ simp$: $prod\_algebra\_def$)

from E **have** $\bigwedge i. i \in J \implies E\ i \subseteq space\ (M\ i)$

using $sets.sets\_into\_space$ **by** $auto$

then have $A = (\Pi_E\ i \in I. if\ i \in J\ then\ E\ i\ else\ space\ (M\ i))$

using $A\ J$ **by** ($auto\ simp$: $prod\_emb\_PiE$)

moreover have $(\lambda i. if\ i \in J\ then\ E\ i\ else\ space\ (M\ i)) \in (\Pi\ i \in I. sets\ (M\ i))$

using $sets.top\ E$ **by** $auto$

ultimately show $?thesis$ **using** $that$ **by** $auto$

qed

lemma $Int\_stable\_prod\_algebra$: $Int\_stable\ (prod\_algebra\ I\ M)$

unfolding Int_stable_def

proof $safe$

fix A **assume** $A \in prod\_algebra\ I\ M$

from $prod\_algebraE[OF\ this]$ **obtain** $J\ E$ **where** A :

$A = prod\_emb\ I\ M\ J\ (Pi_E\ J\ E)$

$finite\ J$

$J \neq \{\} \vee I = \{\}$

$J \subseteq I$

$\bigwedge i. i \in J \implies E\ i \in sets\ (M\ i)$

by $auto$

fix B **assume** $B \in prod\_algebra\ I\ M$

from $prod\_algebraE[OF\ this]$ **obtain** $K\ F$ **where** B :

$B = prod\_emb\ I\ M\ K\ (Pi_E\ K\ F)$

$finite\ K$

$K \neq \{\} \vee I = \{\}$

```

    K ⊆ I
    ∧ i. i ∈ K ⇒ F i ∈ sets (M i)
  by auto
  have A ∩ B = prod_emb I M (J ∪ K) (ΠE i ∈ J ∪ K. (if i ∈ J then E i else
space (M i)) ∩
    (if i ∈ K then F i else space (M i)))
  unfolding A B using A(2,3,4) A(5)[THEN sets.sets_into_space] B(2,3,4)
    B(5)[THEN sets.sets_into_space]
  apply (subst (1 2 3) prod_emb_PiE)
  apply (simp_all add: subset_eq PiE_Int)
  apply blast
  apply (intro PiE_cong)
  apply auto
  done
  also have ... ∈ prod_algebra I M
  using A B by (auto intro!: prod_algebraI)
  finally show A ∩ B ∈ prod_algebra I M .
qed

```

proposition *prod_algebra_mono*:

assumes *space*: $\bigwedge i. i \in I \Rightarrow \text{space } (E i) = \text{space } (F i)$

assumes *sets*: $\bigwedge i. i \in I \Rightarrow \text{sets } (E i) \subseteq \text{sets } (F i)$

shows $\text{prod\_algebra } I E \subseteq \text{prod\_algebra } I F$

proof

fix *A* **assume** $A \in \text{prod\_algebra } I E$

then obtain *J G* **where** $J: J \neq \{\} \vee I = \{\}$ *finite* $J \subseteq I$

and $A = \text{prod\_emb } I E J (\Pi_E i \in J. G i)$

and $G: \bigwedge i. i \in J \Rightarrow G i \in \text{sets } (E i)$

by (*auto simp: prod_algebra_def*)

moreover

from *space* **have** $(\Pi_E i \in I. \text{space } (E i)) = (\Pi_E i \in I. \text{space } (F i))$

by (*rule PiE_cong*)

with *A* **have** $A = \text{prod\_emb } I F J (\Pi_E i \in J. G i)$

by (*simp add: prod_emb_def*)

moreover

from *sets* *G J* **have** $\bigwedge i. i \in J \Rightarrow G i \in \text{sets } (F i)$

by *auto*

ultimately show $A \in \text{prod\_algebra } I F$

apply (*simp add: prod_algebra_def image_iff*)

apply (*intro exI[of _ J] exI[of _ G] conjI*)

apply *auto*

done

qed

proposition *prod_algebra_cong*:

assumes $I = J$ **and** *sets*: $\bigwedge i. i \in I \Rightarrow \text{sets } (M i) = \text{sets } (N i)$

shows $\text{prod\_algebra } I M = \text{prod\_algebra } J N$

proof —

have *space*: $\bigwedge i. i \in I \Rightarrow \text{space } (M i) = \text{space } (N i)$

using *sets_eq_imp_space_eq[OF sets]* **by** *auto*

```

with sets show ?thesis unfolding  $\langle I = J \rangle$ 
  by (intro antisym prod_algebra_mono) auto
qed

```

```

lemma space_in_prod_algebra:
   $(\Pi_E i \in I. \text{space } (M i)) \in \text{prod\_algebra } I M$ 
proof cases
  assume  $I = \{\}$  then show ?thesis
    by (auto simp add: prod_algebra_def image_iff prod_emb_def)
next
  assume  $I \neq \{\}$ 
  then obtain  $i$  where  $i \in I$  by auto
  then have  $(\Pi_E i \in I. \text{space } (M i)) = \text{prod\_emb } I M \{i\} (\Pi_E i \in \{i\}. \text{space } (M i))$ 
    by (auto simp: prod_emb_def)
  also have  $\dots \in \text{prod\_algebra } I M$ 
    using  $\langle i \in I \rangle$  by (intro prod_algebraI) auto
  finally show ?thesis .
qed

```

```

lemma space_PiM:  $\text{space } (\Pi_M i \in I. M i) = (\Pi_E i \in I. \text{space } (M i))$ 
  using prod_algebra_sets_into_space unfolding PiM_def prod_algebra_def by
  (intro space_extend_measure) simp

```

```

lemma prod_emb_subset_PiM[simp]:  $\text{prod\_emb } I M K X \subseteq \text{space } (PiM I M)$ 
  by (auto simp: prod_emb_def space_PiM)

```

```

lemma space_PiM_empty_iff[simp]:  $\text{space } (PiM I M) = \{\} \longleftrightarrow (\exists i \in I. \text{space } (M i) = \{\})$ 
  by (auto simp: space_PiM PiE_eq_empty_iff)

```

```

lemma undefined_in_PiM_empty[simp]:  $(\lambda x. \text{undefined}) \in \text{space } (PiM \{\} M)$ 
  by (auto simp: space_PiM)

```

```

lemma sets_PiM:  $\text{sets } (\Pi_M i \in I. M i) = \text{sigma\_sets } (\Pi_E i \in I. \text{space } (M i))$ 
   $(\text{prod\_algebra } I M)$ 
  using prod_algebra_sets_into_space unfolding PiM_def prod_algebra_def by
  (intro sets_extend_measure) simp

```

```

proposition sets_PiM_single:  $\text{sets } (PiM I M) =$ 
   $\text{sigma\_sets } (\Pi_E i \in I. \text{space } (M i)) \{ \{f \in \Pi_E i \in I. \text{space } (M i). f i \in A\} \mid i A. i$ 
 $\in I \wedge A \in \text{sets } (M i)\}$ 
  (is  $\_ = \text{sigma\_sets } ?\Omega ?R$ )
  unfolding sets_PiM
proof (rule sigma_sets_eqI)
  interpret R: sigma_algebra ? $\Omega$  ?sigma_sets ? $\Omega$  ?R by (rule sigma_algebra_sigma_sets)
  auto
  fix A assume  $A \in \text{prod\_algebra } I M$ 
  from prod_algebraE[OF this] obtain J X where X:
     $A = \text{prod\_emb } I M J (PiE J X)$ 

```

```

    finite J
    J ≠ {} ∨ I = {}
    J ⊆ I
    ∧ i. i ∈ J ⇒ X i ∈ sets (M i)
  by auto
show A ∈ sigma_sets ?Ω ?R
proof cases
  assume I = {}
  with X have A = {λx. undefined} by (auto simp: prod_emb_def)
  with ⟨I = {}⟩ show ?thesis by (auto intro!: sigma_sets_top)
next
  assume I ≠ {}
  with X have A = (∩ j ∈ J. {f ∈ (ΠE i ∈ I. space (M i)). f j ∈ X j})
    by (auto simp: prod_emb_def)
  also have ... ∈ sigma_sets ?Ω ?R
    using X ⟨I ≠ {}⟩ by (intro R.finite_INT sigma_sets.Basic) auto
  finally show A ∈ sigma_sets ?Ω ?R .
qed
next
fix A assume A ∈ ?R
then obtain i B where A: A = {f ∈ ΠE i ∈ I. space (M i). f i ∈ B} i ∈ I B ∈
sets (M i)
  by auto
then have A = prod_emb I M {i} (ΠE i ∈ {i}. B)
  by (auto simp: prod_emb_def)
also have ... ∈ sigma_sets ?Ω (prod_algebra I M)
  using A by (intro sigma_sets.Basic prod_algebraI) auto
finally show A ∈ sigma_sets ?Ω (prod_algebra I M) .
qed

lemma sets_PiM_eq_proj:
  I ≠ {} ⇒ sets (PiM I M) = sets (SUP i ∈ I. vimage_algebra (ΠE i ∈ I. space
(M i)) (λx. x i) (M i))
  apply (simp add: sets_PiM_single)
  apply (subst sets_Sup_eq[where X=ΠE i ∈ I. space (M i)])
  apply auto []
  apply auto []
  apply simp
  apply (subst arg_cong [of _ _ Sup, OF image_cong, OF refl])
  apply (rule sets_vimage_algebra2)
  apply (auto intro!: arg_cong2[where f=sigma_sets])
done

lemma
  shows space_PiM_empty: space (PiM {} M) = {λk. undefined}
    and sets_PiM_empty: sets (PiM {} M) = { {}, {λk. undefined} }
  by (simp_all add: space_PiM sets_PiM_single image_constant sigma_sets_empty_eq)

proposition sets_PiM_sigma:

```

```

assumes  $\Omega\_cover: \bigwedge i. i \in I \implies \exists S \subseteq E \ i. \text{countable } S \wedge \Omega \ i = \bigcup S$ 
assumes  $E: \bigwedge i. i \in I \implies E \ i \subseteq \text{Pow } (\Omega \ i)$ 
assumes  $J: \bigwedge j. j \in J \implies \text{finite } j \ \bigcup J = I$ 
defines  $P \equiv \{\{f \in (\prod_{E \ i \in I. \Omega \ i}). \forall i \in j. f \ i \in A \ i\} \mid A \ j. j \in J \wedge A \in \text{Pi } j \ E\}$ 
shows  $\text{sets } (\prod_M \ i \in I. \text{sigma } (\Omega \ i) (E \ i)) = \text{sets } (\text{sigma } (\prod_{E \ i \in I. \Omega \ i} P)$ 
proof cases
  assume  $I = \{\}$ 
  with  $\langle \bigcup J = I \rangle$  have  $P = \{\{\lambda \_. \text{undefined}\}\} \vee P = \{\}$ 
  by (auto simp: P_def)
  with  $\langle I = \{\} \rangle$  show ?thesis
  by (auto simp add: sets_PiM_empty sigma_sets_empty_eq)
next
  let  $?F = \lambda i. \{(\lambda x. x \ i) - 'A \cap \text{Pi}_E \ I \ \Omega \mid A. A \in E \ i\}$ 
  assume  $I \neq \{\}$ 
  then have  $\text{sets } (\text{Pi}_M \ I \ (\lambda i. \text{sigma } (\Omega \ i) (E \ i))) =$ 
     $\text{sets } (\text{SUP } i \in I. \text{vimage\_algebra } (\prod_{E \ i \in I. \Omega \ i} (\lambda x. x \ i) (\text{sigma } (\Omega \ i) (E \ i))))$ 
  by (subst sets_PiM_eq_proj) (auto simp: space_measure_of_conv)
  also have  $\dots = \text{sets } (\text{SUP } i \in I. \text{sigma } (\text{Pi}_E \ I \ \Omega) (?F \ i))$ 
  using  $E$  by (intro sets_SUP_cong_arg_cong[where f=sets] vimage_algebra_sigma)
  auto
  also have  $\dots = \text{sets } (\text{sigma } (\text{Pi}_E \ I \ \Omega) (\bigcup i \in I. ?F \ i))$ 
  using  $\langle I \neq \{\} \rangle$  by (intro arg_cong[where f=sets] SUP_sigma_sigma) auto
  also have  $\dots = \text{sets } (\text{sigma } (\text{Pi}_E \ I \ \Omega) P)$ 
  proof (intro arg_cong[where f=sets] sigma_eqI sigma_sets_eqI)
    show  $(\bigcup i \in I. ?F \ i) \subseteq \text{Pow } (\text{Pi}_E \ I \ \Omega) \ P \subseteq \text{Pow } (\text{Pi}_E \ I \ \Omega)$ 
    by (auto simp: P_def)
  next
    interpret  $P: \text{sigma\_algebra } \prod_{E \ i \in I. \Omega \ i} \text{sigma\_sets } (\prod_{E \ i \in I. \Omega \ i} P)$ 
    by (auto intro!: sigma_algebra_sigma_sets simp: P_def)

    fix  $Z$  assume  $Z \in (\bigcup i \in I. ?F \ i)$ 
    then obtain  $i \ A$  where  $i: i \in I \ A \in E \ i$  and  $Z\_def: Z = (\lambda x. x \ i) - 'A \cap$ 
 $\text{Pi}_E \ I \ \Omega$ 
    by auto
    from  $\langle i \in I \rangle \ J$  obtain  $j$  where  $j: i \in j \ j \in J \ j \subseteq I \ \text{finite } j$ 
    by auto
    obtain  $S$  where  $S: \bigwedge i. i \in j \implies S \ i \subseteq E \ i \ \bigwedge i. i \in j \implies \text{countable } (S \ i)$ 
     $\bigwedge i. i \in j \implies \Omega \ i = \bigcup (S \ i)$ 
    by (metis subset_eq  $\Omega\_cover \ \langle j \subseteq I \rangle$ )
    define  $A'$  where  $A' \ n = n(i := A)$  for  $n$ 
    then have  $A'_i: \bigwedge n. A' \ n \ i = A$ 
    by simp
    { fix  $n$  assume  $n \in \text{Pi}_E \ (j - \{i\}) \ S$ 
      then have  $A' \ n \in \text{Pi } j \ E$ 
      unfolding  $\text{Pi}_E\_def \ \text{Pi\_def}$  using  $S(1)$  by (auto simp:  $A'_def \ \langle A \in E \ i \rangle$ )
      with  $\langle j \in J \rangle$  have  $\{f \in \text{Pi}_E \ I \ \Omega. \forall i \in j. f \ i \in A' \ n \ i\} \in P$ 
      by (auto simp: P_def) }
    note  $A'\_in\_P = \text{this}$ 

```

```

{ fix x assume x i ∈ A x ∈ Pi_E I Ω
  with S(β) ⟨j ⊆ I⟩ have ∀ i ∈ j. ∃ s ∈ S i. x i ∈ s
    by (auto simp: Pi_E_def Pi_def)
  then obtain s where s: ∧ i. i ∈ j ⇒ s i ∈ S i ∧ i. i ∈ j ⇒ x i ∈ s i
    by metis
  with ⟨x i ∈ A⟩ have ∃ n ∈ Pi_E (j - {i}) S. ∀ i ∈ j. x i ∈ A' n i
    by (intro beI[of _ restrict (s(i := A)) (j - {i})]) (auto simp: A'_def split:
if_splits) }
  then have Z = (⋃ n ∈ Pi_E (j - {i}) S. {f ∈ (Π_E i ∈ I. Ω i). ∀ i ∈ j. f i ∈ A' n i})
    unfolding Z_def
  by (auto simp add: set_eq_iff ball_conj_distrib ⟨i ∈ j⟩ A'_i dest: bspec[OF _
⟨i ∈ j⟩]
      cong: conj_cong)
  also have ... ∈ sigma_sets (Π_E i ∈ I. Ω i) P
    using ⟨finite j⟩ S(2)
  by (intro P.countable_UN' countable_Pi_E) (simp_all add: image_subset_iff
A'_in_P)
  finally show Z ∈ sigma_sets (Π_E i ∈ I. Ω i) P .
next
  interpret F: sigma_algebra Π_E i ∈ I. Ω i sigma_sets (Π_E i ∈ I. Ω i) (⋃ i ∈ I.
?F i)
  by (auto intro!: sigma_algebra_sigma_sets)

fix b assume b ∈ P
then obtain A j where b: b = {f ∈ (Π_E i ∈ I. Ω i). ∀ i ∈ j. f i ∈ A i} j ∈ J A ∈
Pi j E
  by (auto simp: P_def)
show b ∈ sigma_sets (Pi_E I Ω) (⋃ i ∈ I. ?F i)
proof cases
  assume j = {}
  with b have b = (Π_E i ∈ I. Ω i)
    by auto
  then show ?thesis
    by blast
next
  assume j ≠ {}
  with J b(2,3) have eq: b = (⋂ i ∈ j. ((λ x. x i) - ' A i ∩ Pi_E I Ω))
    unfolding b(1)
  by (auto simp: Pi_E_def Pi_def)
  show ?thesis
    unfolding eq using ⟨A ∈ Pi j E⟩ ⟨j ∈ J⟩ J(2)
    by (intro F.finite_INT J ⟨j ∈ J⟩ ⟨j ≠ {}⟩ sigma_sets.Basic) blast
qed
qed
finally show ?thesis .
qed

```

```

lemma sets_PiM_in_sets:
  assumes space: space N = (Π_E i ∈ I. space (M i))

```


assumes $sets: \bigwedge i A. i \in I \implies A \in sets (M i) \implies \{x \in space N. x i \in A\} \in sets N$

shows $sets (\Pi_M i \in I. M i) \subseteq sets N$

unfolding $sets\_PiM\_single\ space[symmetric]$

by $(intro\ sets.sigma\_sets\_subset\ subsetI)\ (auto\ intro: sets)$

lemma $sets\_PiM\_cong[measurable\_cong]$:

assumes $I = J \wedge i. i \in J \implies sets (M i) = sets (N i)$ **shows** $sets (PiM I M) = sets (PiM J N)$

using $assms\ sets\_eq\_imp\_space\_eq[OF\ assms(2)]$ **by** $(simp\ add: sets\_PiM\_single\ cong: PiE\_cong\ conj\_cong)$

lemma $sets\_PiM\_I$:

assumes $finite\ J\ J \subseteq I \wedge i \in J. E i \in sets (M i)$

shows $prod\_emb\ I\ M\ J (\Pi_E j \in J. E j) \in sets (\Pi_M i \in I. M i)$

proof *cases*

assume $J = \{\}$

then have $prod\_emb\ I\ M\ J (\Pi_E j \in J. E j) = (\Pi_E j \in I. space (M j))$

by $(auto\ simp: prod\_emb\_def)$

then show *?thesis*

by $(auto\ simp\ add: sets\_PiM\ intro!: sigma\_sets\_top)$

next

assume $J \neq \{\}$ **with** $assms$ **show** *?thesis*

by $(force\ simp\ add: sets\_PiM\ prod\_algebra\_def)$

qed

proposition $measurable\_PiM$:

assumes $space: f \in space N \rightarrow (\Pi_E i \in I. space (M i))$

assumes $sets: \bigwedge X J. J \neq \{\} \vee I = \{\} \implies finite\ J \implies J \subseteq I \implies (\bigwedge i. i \in J \implies X i \in sets (M i)) \implies$

$f - 'prod\_emb\ I\ M\ J (PiE\ J\ X) \cap space\ N \in sets\ N$

shows $f \in measurable\ N (PiM\ I\ M)$

using $sets\_PiM\ prod\_algebra\_sets\_into\_space\ space$

proof $(rule\ measurable\_sigma\_sets)$

fix A **assume** $A \in prod\_algebra\ I\ M$

from $prod\_algebraE[OF\ this]$ **obtain** $J\ X$ **where**

$A = prod\_emb\ I\ M\ J (PiE\ J\ X)$

$finite\ J$

$J \neq \{\} \vee I = \{\}$

$J \subseteq I$

$\bigwedge i. i \in J \implies X i \in sets (M i)$

by *auto*

with $sets[of\ J\ X]$ **show** $f - 'A \cap space\ N \in sets\ N$ **by** *auto*

qed

lemma $measurable\_PiM\_Collect$:

assumes $space: f \in space N \rightarrow (\Pi_E i \in I. space (M i))$

assumes $sets: \bigwedge X J. J \neq \{\} \vee I = \{\} \implies finite\ J \implies J \subseteq I \implies (\bigwedge i. i \in J \implies X i \in sets (M i)) \implies$

```

    { $\omega \in \text{space } N. \forall i \in J. f \ \omega \ i \in X \ i$ }  $\in \text{sets } N$ 
  shows  $f \in \text{measurable } N \ (PiM \ I \ M)$ 
  using sets_PiM_prod_algebra_sets_into_space_space
proof (rule measurable_sigma_sets)
  fix  $A$  assume  $A \in \text{prod\_algebra } I \ M$ 
  from prod_algebraE[OF this] obtain  $J \ X$ 
  where  $X$ :
     $A = \text{prod\_emb } I \ M \ J \ (PiE \ J \ X)$ 
    finite  $J$ 
     $J \neq \{\}$   $\vee I = \{\}$ 
     $J \subseteq I$ 
     $\bigwedge i. i \in J \implies X \ i \in \text{sets } (M \ i)$ 
  by auto
  then have  $f - 'A \cap \text{space } N = \{\omega \in \text{space } N. \forall i \in J. f \ \omega \ i \in X \ i\}$ 
  using space by (auto simp: prod_emb_def del: PiE_I)
  also have  $\dots \in \text{sets } N$  using  $X(3,2,4,5)$  by (rule sets)
  finally show  $f - 'A \cap \text{space } N \in \text{sets } N$  .
qed

```

```

lemma measurable_PiM_single:
  assumes space:  $f \in \text{space } N \rightarrow (\Pi_E \ i \in I. \text{space } (M \ i))$ 
  assumes sets:  $\bigwedge A \ i. i \in I \implies A \in \text{sets } (M \ i) \implies \{\omega \in \text{space } N. f \ \omega \ i \in A\} \in \text{sets } N$ 
  shows  $f \in \text{measurable } N \ (PiM \ I \ M)$ 
  using sets_PiM_single
proof (rule measurable_sigma_sets)
  fix  $A$  assume  $A \in \{\{f \in \Pi_E \ i \in I. \text{space } (M \ i). f \ i \in A\} \mid i \ A. i \in I \wedge A \in \text{sets } (M \ i)\}$ 
  then obtain  $B \ i$  where  $A = \{f \in \Pi_E \ i \in I. \text{space } (M \ i). f \ i \in B\}$  and  $B: i \in I \ B \in \text{sets } (M \ i)$ 
  by auto
  with space have  $f - 'A \cap \text{space } N = \{\omega \in \text{space } N. f \ \omega \ i \in B\}$  by auto
  also have  $\dots \in \text{sets } N$  using  $B$  by (rule sets)
  finally show  $f - 'A \cap \text{space } N \in \text{sets } N$  .
qed (auto simp: space)

```

```

lemma measurable_PiM_single':
  assumes f:  $\bigwedge i. i \in I \implies f \ i \in \text{measurable } N \ (M \ i)$ 
  and  $(\lambda \omega \ i. f \ i \ \omega) \in \text{space } N \rightarrow (\Pi_E \ i \in I. \text{space } (M \ i))$ 
  shows  $(\lambda \omega \ i. f \ i \ \omega) \in \text{measurable } N \ (PiM \ I \ M)$ 
proof (rule measurable_PiM_single)
  fix  $A \ i$  assume  $A: i \in I \ A \in \text{sets } (M \ i)$ 
  then have  $\{\omega \in \text{space } N. f \ i \ \omega \in A\} = f \ i - 'A \cap \text{space } N$ 
  by auto
  then show  $\{\omega \in \text{space } N. f \ i \ \omega \in A\} \in \text{sets } N$ 
  using  $A \ f$  by (auto intro!: measurable_sets)
qed fact

```

```

lemma sets_PiM_I_finite[measurable]:

```

assumes *finite I* and *sets*: $(\bigwedge i. i \in I \implies E\ i \in \text{sets}\ (M\ i))$
 shows $(\Pi_{E\ j \in I}. E\ j) \in \text{sets}\ (\Pi_{M\ i \in I}. M\ i)$
 using *sets_PiM_I[of I I E M]* *sets.sets_into_space[OF sets]* $\langle \text{finite } I \rangle$ *sets by*
auto

lemma *measurable_component_singleton[measurable (raw)]*:
 assumes $i \in I$ **shows** $(\lambda x. x\ i) \in \text{measurable}\ (Pi_M\ I\ M)\ (M\ i)$
proof (*unfold measurable_def, intro CollectI conjI ballI*)
 fix *A* **assume** $A \in \text{sets}\ (M\ i)$
 then have $(\lambda x. x\ i) - 'A \cap \text{space}\ (Pi_M\ I\ M) = \text{prod\_emb}\ I\ M\ \{i\}\ (\Pi_{E\ j \in \{i\}}. A)$
 using *sets.sets_into_space* $\langle i \in I \rangle$
 by (*fastforce dest: Pi_mem simp: prod_emb_def space_PiM split: if_split_asm*)
 then show $(\lambda x. x\ i) - 'A \cap \text{space}\ (Pi_M\ I\ M) \in \text{sets}\ (Pi_M\ I\ M)$
 using $\langle A \in \text{sets}\ (M\ i) \rangle \langle i \in I \rangle$ **by** (*auto intro!: sets_PiM_I*)
qed (*insert* $\langle i \in I \rangle$, *auto simp: space_PiM*)

lemma *measurable_component_singleton'[measurable_dest]*:
 assumes $f: f \in \text{measurable}\ N\ (Pi_M\ I\ M)$
 assumes $g: g \in \text{measurable}\ L\ N$
 assumes $i: i \in I$
 shows $(\lambda x. (f\ (g\ x))\ i) \in \text{measurable}\ L\ (M\ i)$
 using *measurable_compose[OF measurable_compose[OF g f] measurable_component_singleton, OF i]*.

lemma *measurable_PiM_component_rev*:
 $i \in I \implies f \in \text{measurable}\ (M\ i)\ N \implies (\lambda x. f\ (x\ i)) \in \text{measurable}\ (Pi_M\ I\ M)\ N$
by *simp*

lemma *measurable_case_nat[measurable (raw)]*:
 assumes [*measurable (raw)*]: $i = 0 \implies f \in \text{measurable}\ M\ N$
 $\bigwedge j. i = \text{Suc}\ j \implies (\lambda x. g\ x\ j) \in \text{measurable}\ M\ N$
 shows $(\lambda x. \text{case\_nat}\ (f\ x)\ (g\ x)\ i) \in \text{measurable}\ M\ N$
by (*cases i*) *simp_all*

lemma *measurable_case_nat'[measurable (raw)]*:
 assumes *fg[measurable]*: $f \in \text{measurable}\ N\ M\ g \in \text{measurable}\ N\ (\Pi_{M\ i \in \text{UNIV}}. M)$
 shows $(\lambda x. \text{case\_nat}\ (f\ x)\ (g\ x)) \in \text{measurable}\ N\ (\Pi_{M\ i \in \text{UNIV}}. M)$
 using *fg[THEN measurable_space]*
by (*auto intro!: measurable_PiM_single' simp add: space_PiM PiE_iff split: nat.split*)

lemma *measurable_add_dim[measurable]*:
 $(\lambda(f, y). f(i := y)) \in \text{measurable}\ (Pi_M\ I\ M \otimes_M M\ i)\ (Pi_M\ (\text{insert}\ i\ I)\ M)$
 $(\text{is } ?f \in \text{measurable}\ ?P\ ?I)$
proof (*rule measurable_PiM_single*)
 fix *j A* **assume** $j: j \in \text{insert}\ i\ I$ and *A*: $A \in \text{sets}\ (M\ j)$
 have $\{\omega \in \text{space}\ ?P. (\lambda(f, x). \text{fun\_upd}\ f\ i\ x)\ \omega\ j \in A\} =$

```

    (if j = i then space (PiM I M) × A else ((λx. x j) ∘ fst) -‘ A ∩ space ?P)
    using sets.sets_into_space[OF A] by (auto simp add: space_pair_measure
space_PiM)
    also have ... ∈ sets ?P
    using A j
    by (auto intro!: measurable_sets[OF measurable_comp, OF__measurable_component_singleton])
    finally show {ω ∈ space ?P. case_prod (λf. fun_upd f i) ω j ∈ A} ∈ sets ?P .
qed (auto simp: space_pair_measure space_PiM PiE_def)

```

proposition *measurable_fun_upd*:

```

    assumes I: I = J ∪ {i}
    assumes f[measurable]: f ∈ measurable N (PiM J M)
    assumes h[measurable]: h ∈ measurable N (M i)
    shows (λx. (f x) (i := h x)) ∈ measurable N (PiM I M)
proof (intro measurable_PiM_single)
    fix j assume j ∈ I then show (λω. ((f ω)(i := h ω)) j) ∈ measurable N (M j)
    unfolding I by (cases j = i) auto
next
    show (λx. (f x)(i := h x)) ∈ space N → (ΠE i∈I. space (M i))
    using I f[THEN measurable_space] h[THEN measurable_space]
    by (auto simp: space_PiM PiE_iff extensional_def)
qed

```

lemma *measurable_component_update*:

```

    x ∈ space (PiM I M) ⇒ i ∉ I ⇒ (λv. x(i := v)) ∈ measurable (M i) (PiM
(insert i I) M)
    by simp

```

lemma *measurable_merge[measurable]*:

```

    merge I J ∈ measurable (PiM I M ⊗M PiM J M) (PiM (I ∪ J) M)
    (is ?f ∈ measurable ?P ?U)
proof (rule measurable_PiM_single)
    fix i assume A: A ∈ sets (M i) i ∈ I ∪ J
    then have {ω ∈ space ?P. merge I J ω i ∈ A} =
    (if i ∈ I then ((λx. x i) ∘ fst) -‘ A ∩ space ?P else ((λx. x i) ∘ snd) -‘ A ∩
space ?P)
    by (auto simp: merge_def)
    also have ... ∈ sets ?P
    using A
    by (auto intro!: measurable_sets[OF measurable_comp, OF__measurable_component_singleton])
    finally show {ω ∈ space ?P. merge I J ω i ∈ A} ∈ sets ?P .
qed (auto simp: space_pair_measure space_PiM PiE_iff merge_def extensional_def)

```

lemma *measurable_restrict[measurable (raw)]*:

```

    assumes X: ⋀i. i ∈ I ⇒ X i ∈ measurable N (M i)
    shows (λx. λi∈I. X i x) ∈ measurable N (PiM I M)
proof (rule measurable_PiM_single)
    fix A i assume A: i ∈ I A ∈ sets (M i)
    then have {ω ∈ space N. (λi∈I. X i ω) i ∈ A} = X i -‘ A ∩ space N

```

```

  by auto
  then show  $\{\omega \in \text{space } N. (\lambda i \in I. X \ i \ \omega) \ i \in A\} \in \text{sets } N$ 
    using  $A \ X$  by (auto intro!: measurable_sets)
qed (insert  $X$ , auto simp add:  $PiE\_def$  dest: measurable_space)

```

```

lemma measurable_abs_UNIV:
   $(\bigwedge n. (\lambda \omega. f \ n \ \omega) \in \text{measurable } M \ (N \ n)) \implies (\lambda \omega \ n. f \ n \ \omega) \in \text{measurable } M$ 
  ( $PiM \ UNIV \ N$ )
  by (intro measurable_PiM_single) (auto dest: measurable_space)

```

```

lemma measurable_restrict_subset:  $J \subseteq L \implies (\lambda f. \text{restrict } f \ J) \in \text{measurable}$ 
  ( $Pi_M \ L \ M$ ) ( $Pi_M \ J \ M$ )
  by (intro measurable_restrict measurable_component_singleton) auto

```

```

lemma measurable_restrict_subset':
  assumes  $J \subseteq L \ \bigwedge x. x \in J \implies \text{sets } (M \ x) = \text{sets } (N \ x)$ 
  shows  $(\lambda f. \text{restrict } f \ J) \in \text{measurable } (Pi_M \ L \ M) \ (Pi_M \ J \ N)$ 
proof-
  from assms(1) have  $(\lambda f. \text{restrict } f \ J) \in \text{measurable } (Pi_M \ L \ M) \ (Pi_M \ J \ M)$ 
    by (rule measurable_restrict_subset)
  also from assms(2) have  $\text{measurable } (Pi_M \ L \ M) \ (Pi_M \ J \ M) = \text{measurable}$ 
  ( $Pi_M \ L \ M$ ) ( $Pi_M \ J \ N$ )
    by (intro sets_PiM_cong measurable_cong_sets) simp_all
  finally show ?thesis .
qed

```

```

lemma measurable_prod_emb[intro, simp]:
   $J \subseteq L \implies X \in \text{sets } (Pi_M \ J \ M) \implies \text{prod\_emb } L \ M \ J \ X \in \text{sets } (Pi_M \ L \ M)$ 
  unfolding prod_emb_def space_PiM[symmetric]
  by (auto intro!: measurable_sets measurable_restrict measurable_component_singleton)

```

```

lemma merge_in_prod_emb:
  assumes  $y \in \text{space } (PiM \ I \ M) \ x \in X$  and  $X: X \in \text{sets } (Pi_M \ J \ M)$  and  $J \subseteq I$ 
  shows  $\text{merge } J \ I \ (x, y) \in \text{prod\_emb } I \ M \ J \ X$ 
  using assms sets_sets_into_space[OF  $X$ ]
  by (simp add: merge_def prod_emb_def subset_eq space_PiM PiE_def extensional_restrict Pi_iff
    cong: if_cong restrict_cong)
  (simp add: extensional_def)

```

```

lemma prod_emb_eq_emptyD:
  assumes  $J: J \subseteq I$  and  $ne: \text{space } (PiM \ I \ M) \neq \{\}$  and  $X: X \in \text{sets } (Pi_M \ J \ M)$ 
  and  $*: \text{prod\_emb } I \ M \ J \ X = \{\}$ 
  shows  $X = \{\}$ 
proof safe
  fix  $x$  assume  $x \in X$ 
  obtain  $\omega$  where  $\omega \in \text{space } (PiM \ I \ M)$ 
  using  $ne$  by blast
  from merge_in_prod_emb[OF this  $\langle x \in X \rangle \ X \ J$ ] * show  $x \in \{\}$  by auto

```

qed

lemma *sets_in_Pi_aux*:

finite I $\implies (\bigwedge j. j \in I \implies \{x \in \text{space } (M j). x \in F j\} \in \text{sets } (M j)) \implies$
 $\{x \in \text{space } (PiM I M). x \in Pi I F\} \in \text{sets } (PiM I M)$
by (*simp add: subset_eq Pi_iff*)

lemma *sets_in_Pi[measurable (raw)]*:

finite I $\implies f \in \text{measurable } N (PiM I M) \implies$
 $(\bigwedge j. j \in I \implies \{x \in \text{space } (M j). x \in F j\} \in \text{sets } (M j)) \implies$
 $\text{Measurable.pred } N (\lambda x. f x \in Pi I F)$
unfolding *pred_def*
by (*rule measurable_sets_Collect[of f N PiM I M, OF _ sets_in_Pi_aux]*) *auto*

lemma *sets_in_extensional_aux*:

$\{x \in \text{space } (PiM I M). x \in \text{extensional } I\} \in \text{sets } (PiM I M)$

proof –

have $\{x \in \text{space } (PiM I M). x \in \text{extensional } I\} = \text{space } (PiM I M)$

by (*auto simp add: extensional_def space_PiM*)

then show *?thesis* **by** *simp*

qed

lemma *sets_in_extensional[measurable (raw)]*:

$f \in \text{measurable } N (PiM I M) \implies \text{Measurable.pred } N (\lambda x. f x \in \text{extensional } I)$

unfolding *pred_def*

by (*rule measurable_sets_Collect[of f N PiM I M, OF _ sets_in_extensional_aux]*)
auto

lemma *sets_PiM_I_countable*:

assumes *I*: *countable I* **and** *E*: $\bigwedge i. i \in I \implies E i \in \text{sets } (M i)$ **shows** $Pi_E I E \in \text{sets } (PiM I M)$

proof *cases*

assume $I \neq \{\}$

then have $Pi_E I E = (\bigcap i \in I. \text{prod\_emb } I M \{i\} (Pi_E \{i\} E))$

using $E[THEN \text{sets.sets\_into\_space}]$ **by** (*auto simp: PiE_iff prod_emb_def fun_eq_iff*)

also have $\dots \in \text{sets } (PiM I M)$

using $I \neq \{\}$ **by** (*safe intro!: sets.countable_INT' measurable_prod_emb sets_PiM_I_finite E*)

finally show *?thesis* .

qed (*simp add: sets_PiM_empty*)

lemma *sets_PiM_D_countable*:

assumes *A*: $A \in PiM I M$

shows $\exists J \subseteq I. \exists X \in PiM J M. \text{countable } J \wedge A = \text{prod\_emb } I M J X$

using $A[\text{unfolded sets\_PiM\_single}]$

proof *induction*

case (*Basic A*)

then obtain *i X* **where** $i \in I X \in \text{sets } (M i)$ **and** $A = \{f \in \Pi_E i \in I. \text{space}$

```

(M i). f i ∈ X}
  by auto
  then have A: A = prod_emb I M {i} (ΠE _ ∈ {i}. X)
  by (auto simp: prod_emb_def)
  then show ?case
  by (intro exI[of _ {i}] conjI beXI[of _ ΠE _ ∈ {i}. X])
    (auto intro: countable_finite * sets_PiM_I_finite)
next
  case Empty then show ?case
  by (intro exI[of _ {}] conjI beXI[of _ {}]) auto
next
  case (Compl A)
  then obtain J X where J ⊆ I X ∈ sets (PiM J M) countable J A = prod_emb
I M J X
  by auto
  then show ?case
  by (intro exI[of _ J] beXI[of _ space (PiM J M) - X] conjI)
    (auto simp add: space_PiM prod_emb_PiE intro!: sets_PiM_I_countable)
next
  case (Union K)
  obtain J X where J: ∧ i. J i ⊆ I ∧ i. countable (J i) and X: ∧ i. X i ∈ sets
(PiM (J i) M)
  and K: ∧ i. K i = prod_emb I M (J i) (X i)
  by (metis Union.IH)
  show ?case
  proof (intro exI[of _ ∪ i. J i] beXI[of _ ∪ i. prod_emb (∪ i. J i) M (J i) (X i)]
conjI)
    show (∪ i. J i) ⊆ I countable (∪ i. J i) using J by auto
    with J show ∪ (K ' UNIV) = prod_emb I M (∪ i. J i) (∪ i. prod_emb (∪ i.
J i) M (J i) (X i))
    by (simp add: K[abs_def] SUP_upper)
  qed(auto intro: X)
qed

proposition measure_eqI_PiM_finite:
  assumes [simp]: finite I sets P = PiM I M sets Q = PiM I M
  assumes eq: ∧ A. (∧ i. i ∈ I ⇒ A i ∈ sets (M i)) ⇒ P (PiE I A) = Q (PiE
I A)
  assumes A: range A ⊆ prod_algebra I M (∪ i. A i) = space (PiM I M) ∧ i::nat.
P (A i) ≠ ∞
  shows P = Q
proof (rule measure_eqI_generator_eq[OF Int_stable_prod_algebra prod_algebra_sets_into_space])
  show range A ⊆ prod_algebra I M (∪ i. A i) = (ΠE i ∈ I. space (M i)) ∧ i. P
(A i) ≠ ∞
  unfolding space_PiM[symmetric] by fact+
  fix X assume X ∈ prod_algebra I M
  then obtain J E where X = prod_emb I M J (ΠE j ∈ J. E j)
  and J: finite J J ⊆ I ∧ j. j ∈ J ⇒ E j ∈ sets (M j)
  by (force elim!: prod_algebraE)

```

then show $\text{emeasure } P \ X = \text{emeasure } Q \ X$
unfolding X **by** $(\text{subst } (1 \ 2) \ \text{prod\_emb\_Pi}) \ (\text{auto simp: eq})$
qed $(\text{simp\_all add: sets\_PiM})$

proposition $\text{measure\_eqI\_PiM\_infinite}$:

assumes $[\text{simp}]$: $\text{sets } P = \text{PiM } I \ M \ \text{sets } Q = \text{PiM } I \ M$

assumes eq : $\bigwedge A \ J. \ \text{finite } J \implies J \subseteq I \implies (\bigwedge i. i \in J \implies A \ i \in \text{sets } (M \ i))$

$\implies$

$P \ (\text{prod\_emb } I \ M \ J \ (\text{PiE } J \ A)) = Q \ (\text{prod\_emb } I \ M \ J \ (\text{PiE } J \ A))$

assumes A : $\text{finite\_measure } P$

shows $P = Q$

proof $(\text{rule measure\_eqI\_generator\_eq}[\text{OF Int\_stable\_prod\_algebra prod\_algebra\_sets\_into\_space}])$

interpret $\text{finite\_measure } P$ **by fact**

define i **where** $i = (\text{SOME } i. i \in I)$

have $i: I \neq \{\} \implies i \in I$

unfolding i_def **by** (rule someI_ex) **auto**

define A **where** $A \ n =$

$(\text{if } I = \{\} \text{ then } \text{prod\_emb } I \ M \ \{\} \ (\Pi_E \ i \in \{\}. \ \{\}) \text{ else } \text{prod\_emb } I \ M \ \{i\} \ (\Pi_E$
 $i \in \{i\}. \ \text{space } (M \ i)))$

for $n :: \text{nat}$

then show $\text{range } A \subseteq \text{prod\_algebra } I \ M$

using $\text{prod\_algebraI}[\text{of } \{\} \ I \ \lambda i. \ \text{space } (M \ i) \ M]$ **by** $(\text{auto intro!: prod\_algebraI } i)$

have $\bigwedge i. A \ i = \text{space } (\text{PiM } I \ M)$

by $(\text{auto simp: prod\_emb\_def space\_PiM PiE\_iff A\_def i\_ex\_in\_conv}[\text{symmetric}])$

then show $(\bigcup i. A \ i) = (\Pi_E \ i \in I. \ \text{space } (M \ i)) \ \bigwedge i. \ \text{emeasure } P \ (A \ i) \neq \infty$

by $(\text{auto simp: space\_PiM})$

next

fix X **assume** X : $X \in \text{prod\_algebra } I \ M$

then obtain $J \ E$ **where** X : $X = \text{prod\_emb } I \ M \ J \ (\Pi_E \ j \in J. \ E \ j)$

and J : $\text{finite } J \ J \subseteq I \ \bigwedge j. j \in J \implies E \ j \in \text{sets } (M \ j)$

by $(\text{force elim!: prod\_algebraE})$

then show $\text{emeasure } P \ X = \text{emeasure } Q \ X$

by (auto intro!: eq)

qed $(\text{auto simp: sets\_PiM})$

locale $\text{product\_sigma\_finite} =$

fixes $M :: 'i \Rightarrow 'a \ \text{measure}$

assumes $\text{sigma\_finite\_measures}$: $\bigwedge i. \ \text{sigma\_finite\_measure } (M \ i)$

sublocale $\text{product\_sigma\_finite} \subseteq M? : \text{sigma\_finite\_measure } M \ i$ **for** i

by $(\text{rule sigma\_finite\_measures})$

locale $\text{finite\_product\_sigma\_finite} = \text{product\_sigma\_finite } M$ **for** $M :: 'i \Rightarrow 'a$
 $\text{measure} +$

fixes $I :: 'i \ \text{set}$

assumes finite_index : $\text{finite } I$

proposition (in *finite_product_sigma_finite*) *sigma_finite_pairs*:

$\exists F::'i \Rightarrow \text{nat} \Rightarrow 'a \text{ set.}$

$(\forall i \in I. \text{range } (F i) \subseteq \text{sets } (M i)) \wedge$

$(\forall k. \forall i \in I. \text{emeasure } (M i) (F i k) \neq \infty) \wedge \text{incseq } (\lambda k. \Pi_E i \in I. F i k) \wedge$

$(\bigcup k. \Pi_E i \in I. F i k) = \text{space } (PiM I M)$

proof –

have $\forall i::'i. \exists F::\text{nat} \Rightarrow 'a \text{ set. range } F \subseteq \text{sets } (M i) \wedge \text{incseq } F \wedge (\bigcup i. F i) = \text{space } (M i) \wedge (\forall k. \text{emeasure } (M i) (F k) \neq \infty)$

using *M.sigma_finite_incseq* **by** *metis*

then obtain $F :: 'i \Rightarrow \text{nat} \Rightarrow 'a \text{ set}$

where $\forall x. \text{range } (F x) \subseteq \text{sets } (M x) \wedge \text{incseq } (F x) \wedge \bigcup (\text{range } (F x)) = \text{space } (M x) \wedge (\forall k. \text{emeasure } (M x) (F x k) \neq \infty)$

by *metis*

then have $F: \bigwedge i. \text{range } (F i) \subseteq \text{sets } (M i) \wedge i. \text{incseq } (F i) \wedge i. (\bigcup j. F i j) = \text{space } (M i) \wedge i k. \text{emeasure } (M i) (F i k) \neq \infty$

by *auto*

let $?F = \lambda k. \Pi_E i \in I. F i k$

note *space_PiM[simp]*

show *?thesis*

proof (*intro exI[of _ F] conjI allI incseq_SucI set_eqI iffI ballI*)

fix *i* **show** $\text{range } (F i) \subseteq \text{sets } (M i)$ **by** *fact*

next

fix *i k* **show** $\text{emeasure } (M i) (F i k) \neq \infty$ **by** *fact*

next

fix *x* **assume** $x \in (\bigcup i. ?F i)$ **with** *F(1)* **show** $x \in \text{space } (PiM I M)$

by (*auto simp: PiE_def dest!: sets.sets_into_space*)

next

fix *f* **assume** $f \in \text{space } (PiM I M)$

with *Pi_UN[OF finite_index, of $\lambda k i. F i k$]* *F*

show $f \in (\bigcup i. ?F i)$ **by** (*auto simp: incseq_def PiE_def*)

next

fix *i* **show** $?F i \subseteq ?F (\text{Suc } i)$

using $\langle \bigwedge i. \text{incseq } (F i) \rangle [THEN \text{incseq\_SucD}]$ **by** *auto*

qed

qed

lemma *emeasure_PiM_empty[simp]*: $\text{emeasure } (PiM \{\} M) \{\lambda_. \text{undefined}\} = 1$

proof –

let $? \mu = \lambda A. \text{if } A = \{\} \text{ then } 0 \text{ else } (1::\text{ennreal})$

have $\text{emeasure } (PiM \{\} M) (\text{prod\_emb } \{\} M \{\} (\Pi_E i \in \{\}. \{\})) = 1$

proof (*subst emeasure_extend_measure_Pair[OF PiM_def]*)

show *positive* $(PiM \{\} M) ? \mu$

by (*auto simp: positive_def*)

show *countably_additive* $(PiM \{\} M) ? \mu$

by (*rule sets.countably_additiveI_finite*)

(*auto simp: additive_def positive_def sets_PiM_empty space_PiM_empty intro!*)

qed (*auto simp: prod_emb_def*)

also have $(\text{prod\_emb } \{\} M \{\} (\Pi_E i \in \{\}. \{\})) = \{\lambda_. \text{undefined}\}$

```

    by (auto simp: prod_emb_def)
  finally show ?thesis
    by simp
qed

```

```

lemma PiM_empty: PiM {} M = count_space {λ_. undefined}
  by (rule measure_eqI) (auto simp add: sets_PiM_empty)

```

```

lemma (in product_sigma_finite) emeasure_PiM:
  finite I  $\implies$  ( $\bigwedge i. i \in I \implies A\ i \in \text{sets } (M\ i)$ )  $\implies$  emeasure (PiM I M) (PiE I A)
= ( $\prod_{i \in I}. \text{emeasure } (M\ i) (A\ i)$ )
proof (induct I arbitrary: A rule: finite_induct)
  case (insert i I)
  interpret finite_product_sigma_finite M I by standard fact
  have finite (insert i I) using ⟨finite I⟩ by auto
  interpret I': finite_product_sigma_finite M insert i I by standard fact
  let ?h = (λ(f, y). f(i := y))

  let ?P = distr (PiM I M  $\otimes_M$  M i) (PiM (insert i I) M) ?h
  let ?μ = emeasure ?P
  let ?I = {j ∈ insert i I. emeasure (M j) (space (M j)) ≠ 1}
  let ?f = λJ E j. if j ∈ J then emeasure (M j) (E j) else emeasure (M j) (space
(M j))

```

```

  have emeasure (PiM (insert i I) M) (prod_emb (insert i I) M (insert i I) (PiE
(insert i I) A)) =
    ( $\prod_{i \in \text{insert } i\ I}. \text{emeasure } (M\ i) (A\ i)$ )
  proof (subst emeasure_extend_measure_Pair[OF PiM_def])
    fix J E assume (J ≠ {}  $\vee$  insert i I = {})  $\wedge$  finite J  $\wedge$  J  $\subseteq$  insert i I  $\wedge$  E ∈
( $\prod_{j \in J}. \text{sets } (M\ j)$ )
    then have J: J ≠ {} finite J J  $\subseteq$  insert i I and E:  $\forall j \in J. E\ j \in \text{sets } (M\ j)$ 
  by auto
  let ?p = prod_emb (insert i I) M J (PiE J E)
  let ?p' = prod_emb I M (J - {i}) (PiE j ∈ J - {i}. E j)
  have ?μ ?p =
    emeasure (PiM I M  $\otimes_M$  (M i)) (?h -' ?p ∩ space (PiM I M  $\otimes_M$  M i))
  by (intro emeasure_distr measurable_add_dim sets_PiM_I) fact+
  also have ?h -' ?p ∩ space (PiM I M  $\otimes_M$  M i) = ?p' × (if i ∈ J then E i
else space (M i))
  using J E [rule_format, THEN sets.sets_into_space]
  by (force simp: space_pair_measure space_PiM prod_emb_iff PiE_def Pi_iff
split: if_split_asm)
  also have emeasure (PiM I M  $\otimes_M$  (M i)) (?p' × (if i ∈ J then E i else space
(M i))) =
    emeasure (PiM I M) ?p' * emeasure (M i) (if i ∈ J then (E i) else space (M
i))
  using J E by (intro M.emeasure_pair_measure_Times sets_PiM_I) auto
  also have ?p' = (PiE j ∈ J - {i}. if j ∈ J - {i} then E j else space (M j))
  using J E [rule_format, THEN sets.sets_into_space]

```

```

    by (auto simp: prod_emb_iff PiE_def Pi_iff split: if_split_asm) blast+
  also have emeasure (PiM I M) (ΠE j∈I. if j ∈ J - {i} then E j else space (M
j)) =
    (Π j∈I. if j ∈ J - {i} then emeasure (M j) (E j) else emeasure (M j) (space
(M j)))
    using E by (subst insert) (auto intro!: prod.cong)
  also have (Π j∈I. if j ∈ J - {i} then emeasure (M j) (E j) else emeasure (M
j) (space (M j))) *
    emeasure (M i) (if i ∈ J then E i else space (M i)) = (Π j∈insert i I. ?f J E
j)
    using insert by (auto simp: mult.commute intro!: arg_cong2[where f=(*)]
prod.cong)
  also have ... = (Π j∈J ∪ ?I. ?f J E j)
    using insert(1,2) J E by (intro prod.mono_neutral_right) auto
  finally show ?μ ?p = ... .

  show prod_emb (insert i I) M J (PiE J E) ∈ Pow (ΠE i∈insert i I. space (M
i))
    using J E [rule_format, THEN sets.sets_into_space] by (auto simp: prod_emb_iff
PiE_def)
  next
    show positive (sets (PiM (insert i I) M)) ?μ countably_additive (sets (PiM
(insert i I) M)) ?μ
      using emeasure_positive[of ?P] emeasure_countably_additive[of ?P] by
simp_all
    next
      show (insert i I ≠ {} ∨ insert i I = {}) ∧ finite (insert i I) ∧
        insert i I ⊆ insert i I ∧ A ∈ (Π j∈insert i I. sets (M j))
        using insert by auto
      qed (auto intro!: prod.cong)
    with insert show ?case
      by (subst (asm) prod_emb_PiE_same_index) (auto intro!: sets.sets_into_space)
    qed simp

lemma (in product_sigma_finite) PiM_eqI:
  assumes I[simp]: finite I and P: sets P = PiM I M
  assumes eq: ∧ A. (∧ i. i ∈ I ⇒ A i ∈ sets (M i)) ⇒ P (PiE I A) = (Π i∈I.
emeasure (M i) (A i))
  shows P = PiM I M
proof -
  interpret finite_product_sigma_finite M I
    by standard fact
  from sigma_finite_pairs obtain C where C:
    ∀ i∈I. range (C i) ⊆ sets (M i) ∀ k. ∀ i∈I. emeasure (M i) (C i k) ≠ ∞
    incseq (λk. ΠE i∈I. C i k) (∪ k. ΠE i∈I. C i k) = space (PiM I M)
    by auto
  show ?thesis
proof (rule measure_eqI_PiM_finite[OF I refl P, symmetric])
  show (∧ i. i ∈ I ⇒ A i ∈ sets (M i)) ⇒ (PiM I M) (PiE I A) = P (PiE I

```

A) **for** A
 by (*simp add: eq_emeasure_PiM*)
 define A **where** $A\ n = (\Pi_E\ i \in I. C\ i\ n)$ **for** n
 with C **show** $\text{range } A \subseteq \text{prod\_algebra } I\ M \wedge i. \text{emeasure } (Pi_M\ I\ M) (A\ i) \neq \infty$
 $(\bigcup i. A\ i) = \text{space } (Pi_M\ I\ M)$
 by (*auto intro!: prod_algebraI_finite simp: emeasure_PiM subset_eq ennreal_prod_eq_top*)
 qed
qed

lemma (**in** *product_sigma_finite*) *sigma_finite*:
 assumes *finite I*
 shows *sigma_finite_measure (PiM I M)*
proof
 interpret *finite_product_sigma_finite M I* **by** *standard fact*

 obtain F **where** $F: \bigwedge j. \text{countable } (F\ j) \wedge j f. f \in F\ j \implies f \in \text{sets } (M\ j)$
 $\bigwedge j f. f \in F\ j \implies \text{emeasure } (M\ j) f \neq \infty$ **and**
 in_space: $\bigwedge j. \text{space } (M\ j) = \bigcup (F\ j)$
 using *sigma_finite_countable* **by** (*metis subset_eq*)
 moreover have $(\bigcup (Pi_E\ I\ ' Pi_E\ I\ F)) = \text{space } (Pi_M\ I\ M)$
 using *in_space* **by** (*auto simp: space_PiM PiE_iff intro!: PiE_choice [THEN iffD2]*)
 ultimately show $\exists A. \text{countable } A \wedge A \subseteq \text{sets } (Pi_M\ I\ M) \wedge \bigcup A = \text{space } (Pi_M\ I\ M)$
 $\wedge (\forall a \in A. \text{emeasure } (Pi_M\ I\ M) a \neq \infty)$
 by (*intro exI [of _ Pi_E\ I\ ' Pi_E\ I\ F]*)
 (*auto intro!: countable_PiE sets_PiM_I_finite*
 simp: PiE_iff emeasure_PiM finite_index ennreal_prod_eq_top)
 qed

sublocale *finite_product_sigma_finite* $\subseteq$ *sigma_finite_measure PiM I M*
 using *sigma_finite [OF finite_index]* .

lemma (**in** *finite_product_sigma_finite*) *measure_times*:
 $(\bigwedge i. i \in I \implies A\ i \in \text{sets } (M\ i)) \implies \text{emeasure } (Pi_M\ I\ M) (Pi_E\ I\ A) = (\prod i \in I. \text{emeasure } (M\ i) (A\ i))$
 using *emeasure_PiM [OF finite_index]* **by** *auto*

lemma (**in** *product_sigma_finite*) *nn_integral_empty*:
 $0 \leq f\ (\lambda k. \text{undefined}) \implies \text{integral}^N (Pi_M\ \{\}\ M) f = f\ (\lambda k. \text{undefined})$
 by (*simp add: PiM_empty nn_integral_count_space_finite max.absorb2*)

lemma (**in** *product_sigma_finite*) *distr_merge*:
 assumes $IJ[\text{simp}]: I \cap J = \{\}$ **and** *fin: finite I finite J*
 shows $\text{distr } (Pi_M\ I\ M \otimes_M Pi_M\ J\ M) (Pi_M\ (I \cup J)\ M) (\text{merge } I\ J) = Pi_M\ (I \cup J)\ M$
 (is ?D = ?P)
proof (*rule PiM_eqI*)
 interpret $I: \text{finite\_product\_sigma\_finite } M\ I$ **by** *standard fact*

```

interpret J: finite_product_sigma_finite M J by standard fact
fix A assume A:  $\bigwedge i. i \in I \cup J \implies A \ i \in \text{sets } (M \ i)$ 
have *: (merge I J -'  $Pi_E \ (I \cup J) \ A \cap \text{space } (Pi_M \ I \ M \ \otimes_M \ Pi_M \ J \ M)$ ) =
   $Pi_E \ I \ A \times Pi_E \ J \ A$ 
  using A[THEN sets.sets_into_space] by (auto simp: space_PiM space_pair_measure)
from A fin show emeasure (distr (Pi_M I M  $\otimes_M$  Pi_M J M) (Pi_M (I  $\cup$  J) M)
  (merge I J)) (Pi_E (I  $\cup$  J) A) =
  ( $\prod_{i \in I \cup J. \text{emeasure } (M \ i) \ (A \ i)$ )
  by (subst emeasure_distr)
  (auto simp: * J.emeasure_pair_measure_Times I.measure_times J.measure_times
    prod.union_disjoint)
qed (insert fin, simp_all)

```

proposition (in product_sigma_finite) product_nn_integral_fold:

```

assumes IJ:  $I \cap J = \{\}$  finite I finite J
and f[measurable]:  $f \in \text{borel\_measurable } (Pi_M \ (I \cup J) \ M)$ 
shows  $\text{integral}^N \ (Pi_M \ (I \cup J) \ M) \ f =$ 
  ( $\int^+ x. (\int^+ y. f \ (merge \ I \ J \ (x, y)) \ \partial(Pi_M \ J \ M)) \ \partial(Pi_M \ I \ M)$ )

```

proof –

```

interpret I: finite_product_sigma_finite M I by standard fact
interpret J: finite_product_sigma_finite M J by standard fact
interpret P: pair_sigma_finite Pi_M I M Pi_M J M by standard
have P_borel:  $(\lambda x. f \ (merge \ I \ J \ x)) \in \text{borel\_measurable } (Pi_M \ I \ M \ \otimes_M \ Pi_M \ J \ M)$ 
  using measurable_comp[OF measurable_merge f] by (simp add: comp_def)
show ?thesis
  apply (subst distr_merge[OF IJ, symmetric])
  apply (subst nn_integral_distr[OF measurable_merge])
  apply measurable []
  apply (subst J.nn_integral fst[symmetric, OF P_borel])
  apply simp
  done
qed

```

lemma (in product_sigma_finite) distr_singleton:

```

distr (Pi_M {i} M) (M i)  $(\lambda x. x \ i) = M \ i$  (is ?D = _)

```

proof (intro measure_eqI[symmetric])

```

interpret I: finite_product_sigma_finite M {i} by standard simp
fix A assume A:  $A \in \text{sets } (M \ i)$ 
then have  $(\lambda x. x \ i) -' A \cap \text{space } (Pi_M \ \{i\} \ M) = (\Pi_E \ i \in \{i\}. A)$ 
  using sets.sets_into_space by (auto simp: space_PiM)
then show emeasure (M i) A = emeasure ?D A
  using A I.measure_times[of  $\lambda_. A$ ]
  by (simp add: emeasure_distr measurable_component_singleton)
qed simp

```

lemma (in product_sigma_finite) product_nn_integral_singleton:

```

assumes f:  $f \in \text{borel\_measurable } (M \ i)$ 
shows  $\text{integral}^N \ (Pi_M \ \{i\} \ M) \ (\lambda x. f \ (x \ i)) = \text{integral}^N \ (M \ i) \ f$ 

```

proof –

```

interpret I: finite_product_sigma_finite M {i} by standard simp
from f show ?thesis
  apply (subst distr_singleton[symmetric])
  apply (subst nn_integral_distr[OF measurable_component_singleton])
  apply simp_all
done

```

qed

proposition (in product_sigma_finite) product_nn_integral_insert:

```

assumes I[simp]: finite I i ∉ I
  and f: f ∈ borel_measurable (Pi_M (insert i I) M)
shows integralN (Pi_M (insert i I) M) f = (∫+ x. (∫+ y. f (x(i := y)) ∂(M i))
∂(Pi_M I M))

```

proof –

```

interpret I: finite_product_sigma_finite M I by standard auto
interpret i: finite_product_sigma_finite M {i} by standard auto
have IJ: I ∩ {i} = {} and insert: I ∪ {i} = insert i I
  using f by auto
show ?thesis
  unfolding product_nn_integral_fold[OF IJ, unfolded insert, OF I(1) i.finite_index
f]

```

```

proof (rule nn_integral_cong, subst product_nn_integral_singleton[symmetric])
  fix x assume x: x ∈ space (Pi_M I M)
  let ?f = λy. f (x(i := y))
  show ?f ∈ borel_measurable (M i)
    using measurable_comp[OF measurable_component_update f, OF x ⟨i ∉ I⟩]
  unfolding comp_def .
  show (∫+ y. f (merge I {i} (x, y)) ∂Pi_M {i} M) = (∫+ y. f (x(i := y i))
∂Pi_M {i} M)
    using x
    by (auto intro!: nn_integral_cong arg_cong[where f=f]
simp add: space_PiM extensional_def PiE_def)

```

qed

qed

lemma (in product_sigma_finite) product_nn_integral_insert_rev:

```

assumes I[simp]: finite I i ∉ I
  and [measurable]: f ∈ borel_measurable (Pi_M (insert i I) M)
shows integralN (Pi_M (insert i I) M) f = (∫+ y. (∫+ x. f (x(i := y)) ∂(Pi_M
I M)) ∂(M i))
  apply (subst product_nn_integral_insert[OF assms])
  apply (rule pair_sigma_finite.Fubini')
  apply intro_locales []
  apply (rule sigma_finite[OF I(1)])
  apply measurable
done

```

lemma (in product_sigma_finite) product_nn_integral_prod:

```

    assumes finite I  $\wedge i. i \in I \implies f\ i \in \text{borel\_measurable } (M\ i)$ 
    shows  $(\int^+ x. (\prod_{i \in I}. f\ i\ (x\ i))\ \partial Pi_M\ I\ M) = (\prod_{i \in I}. \text{integral}^N\ (M\ i)\ (f\ i))$ 
using assms proof (induction I)
  case (insert i I)
  note insert.premis[measurable]
  note ⟨finite I⟩[intro, simp]
  interpret I: finite_product_sigma_finite M I by standard auto
  have *:  $\bigwedge x\ y. (\prod_{j \in I}. f\ j\ (\text{if } j = i \text{ then } y \text{ else } x\ j)) = (\prod_{j \in I}. f\ j\ (x\ j))$ 
    using insert by (auto intro!: prod.cong)
  have prod:  $\bigwedge J. J \subseteq \text{insert } i\ I \implies (\lambda x. (\prod_{i \in J}. f\ i\ (x\ i))) \in \text{borel\_measurable } (Pi_M\ J\ M)$ 
    using sets.sets_into_space insert
    by (intro borel_measurable_prod ennreal
        measurable_comp[OF measurable_component_singleton, unfolded
comp_def])
    auto
  then show ?case
    apply (simp add: product_nn_integral_insert[OF insert(1,2)])
    apply (simp add: insert(2-) * nn_integral_multc)
    apply (subst nn_integral_cmult)
    apply (auto simp add: insert(2-))
    done
qed (simp add: space_PiM)

```

```

proposition (in product_sigma_finite) product_nn_integral_pair:
  assumes [measurable]: case_prod f  $\in \text{borel\_measurable } (M\ x \otimes_M M\ y)$ 
  assumes xy:  $x \neq y$ 
  shows  $(\int^+ \sigma. f\ (\sigma\ x)\ (\sigma\ y)\ \partial Pi_M\ \{x, y\}\ M) = (\int^+ z. f\ (\text{fst } z)\ (\text{snd } z)\ \partial(M\ x \otimes_M M\ y))$ 
proof -
  interpret psm: pair_sigma_finite M x M y
  unfolding pair_sigma_finite_def using sigma_finite_measures by simp_all
  have  $\{x, y\} = \{y, x\}$  by auto
  also have  $(\int^+ \sigma. f\ (\sigma\ x)\ (\sigma\ y)\ \partial Pi_M\ \{y, x\}\ M) = (\int^+ y. \int^+ \sigma. f\ (\sigma\ x)\ y\ \partial Pi_M\ \{x\}\ M\ \partial M\ y)$ 
    using xy by (subst product_nn_integral_insert_rev) simp_all
  also have  $\dots = (\int^+ y. \int^+ x. f\ x\ y\ \partial M\ x\ \partial M\ y)$ 
    by (intro nn_integral_cong, subst product_nn_integral_singleton) simp_all
  also have  $\dots = (\int^+ z. f\ (\text{fst } z)\ (\text{snd } z)\ \partial(M\ x \otimes_M M\ y))$ 
    by (subst psm.nn_integral_snd[symmetric]) simp_all
  finally show ?thesis .
qed

```

```

lemma (in product_sigma_finite) distr_component:
   $\text{distr } (M\ i)\ (Pi_M\ \{i\}\ M)\ (\lambda x. \lambda i \in \{i\}. x) = Pi_M\ \{i\}\ M\ (\text{is } ?D = ?P)$ 
proof (intro PiM_eqI)
  fix A assume A:  $\bigwedge ia. ia \in \{i\} \implies A\ ia \in \text{sets } (M\ ia)$ 
  then have  $(\lambda x. \lambda i \in \{i\}. x) - ' Pi_E\ \{i\}\ A \cap \text{space } (M\ i) = A\ i$ 
    by (fastforce dest: sets.sets_into_space)

```

```

with A show emeasure (distr (M i) (Pi_M {i} M) (λx. λi∈{i}. x)) (Pi_E {i} A)
= (∏ i∈{i}. emeasure (M i) (A i))
by (subst emeasure_distr) (auto intro!: sets_PiM_I_finite measurable_restrict)
qed simp_all

```

```

lemma (in product_sigma_finite)
assumes IJ:  $I \cap J = \{\}$  finite I finite J and A:  $A \in \text{sets } (Pi_M (I \cup J) M)$ 
shows emeasure_fold_integral:
  emeasure (Pi_M (I ∪ J) M) A =  $(\int^{+x}. \text{emeasure } (Pi_M J M) ((\lambda y. \text{merge } I J$ 
   $(x, y)) - 'A \cap \text{space } (Pi_M J M)) \partial Pi_M I M)$  (is ?I)
and emeasure_fold_measurable:
   $(\lambda x. \text{emeasure } (Pi_M J M) ((\lambda y. \text{merge } I J (x, y)) - 'A \cap \text{space } (Pi_M J M)))$ 
 $\in \text{borel\_measurable } (Pi_M I M)$  (is ?B)
proof -
interpret I: finite_product_sigma_finite M I by standard fact
interpret J: finite_product_sigma_finite M J by standard fact
interpret IJ: pair_sigma_finite Pi_M I M Pi_M J M ..
have merge:  $\text{merge } I J - 'A \cap \text{space } (Pi_M I M \otimes_M Pi_M J M) \in \text{sets } (Pi_M I$ 
 $M \otimes_M Pi_M J M)$ 
by (intro measurable_sets[OF _ A] measurable_merge assms)

```

```

show ?I
apply (subst distr_merge[symmetric, OF IJ])
apply (subst emeasure_distr[OF measurable_merge A])
apply (subst J.emeasure_pair_measure_alt[OF merge])
apply (auto intro!: nn_integral_cong arg_cong2[where f=emeasure] simp:
space_pair_measure)
done

```

```

show ?B
using IJ.measurable_emeasure_Pair1[OF merge]
by (simp add: vimage_comp comp_def space_pair_measure cong: measurable_cong)
qed

```

```

lemma sets_Collect_single:
 $i \in I \implies A \in \text{sets } (M i) \implies \{ x \in \text{space } (Pi_M I M). x i \in A \} \in \text{sets } (Pi_M I$ 
 $M)$ 
by simp

```

```

lemma pair_measure_eq_distr_PiM:
fixes M1 :: 'a measure and M2 :: 'a measure
assumes sigma_finite_measure M1 sigma_finite_measure M2
shows  $(M1 \otimes_M M2) = \text{distr } (Pi_M \text{ UNIV } (\text{case\_bool } M1 M2)) (M1 \otimes_M M2)$ 
 $(\lambda x. (x \text{ True}, x \text{ False}))$ 
(is ?P = ?D)
proof (rule pair_measure_eqI[OF assms])
interpret B: product_sigma_finite case_bool M1 M2
unfolding product_sigma_finite_def using assms by (auto split: bool.split)

```



```

let ?B = PiM UNIV (case_bool M1 M2)

have [simp]: fst ∘ (λx. (x True, x False)) = (λx. x True) snd ∘ (λx. (x True, x
False)) = (λx. x False)
by auto
fix A B assume A: A ∈ sets M1 and B: B ∈ sets M2
have emeasure M1 A * emeasure M2 B = (∏i ∈ UNIV. emeasure (case_bool
M1 M2 i) (case_bool A B i))
by (simp add: UNIV_bool ac_simps)
also have ... = emeasure ?B (PiE UNIV (case_bool A B))
using A B by (subst B.emeasure_PiM) (auto split: bool.split)
also have PiE UNIV (case_bool A B) = (λx. (x True, x False)) - ‘(A × B) ∩
space ?B
using A [THEN sets.sets_into_space] B [THEN sets.sets_into_space]
by (auto simp: PiE_iff_all_bool_eq space_PiM split: bool.split)
finally show emeasure M1 A * emeasure M2 B = emeasure ?D (A × B)
using A B
    measurable_component_singleton[of True UNIV case_bool M1 M2]
    measurable_component_singleton[of False UNIV case_bool M1 M2]
by (subst emeasure_distr) (auto simp: measurable_pair_iff)
qed simp

lemma infprod_in_sets[intro]:
  fixes E :: nat ⇒ 'a set assumes E: ∧i. E i ∈ sets (M i)
  shows Pi UNIV E ∈ sets (ΠM i ∈ UNIV::nat set. M i)
proof -
  have Pi UNIV E = (∩i. prod_emb UNIV M {..i} (ΠE j ∈ {..i}. E j))
  using E E [THEN sets.sets_into_space]
  by (auto simp: prod_emb_def Pi_iff_extensional_def)
  with E show ?thesis by auto
qed

```

6.8.3 Measurability

There are two natural sigma-algebras on a product space: the borel sigma algebra, generated by open sets in the product, and the product sigma algebra, countably generated by products of measurable sets along finitely many coordinates. The second one is defined and studied in `Finite_Product_Measure.thy`.

These sigma-algebras share a lot of natural properties (measurability of coordinates, for instance), but there is a fundamental difference: open sets are generated by arbitrary unions, not only countable ones, so typically many open sets will not be measurable with respect to the product sigma algebra (while all sets in the product sigma algebra are borel). The two sigma algebras coincide only when everything is countable (i.e., the product is countable, and the borel sigma algebra in the factor is countably generated). In this paragraph, we develop basic measurability properties for the borel sigma algebra, and compare it with the product sigma algebra as explained

above.

lemma *measurable_product_coordinates* [measurable (raw)]:
 $(\lambda x. x \ i) \in \text{measurable borel borel}$
by (rule borel_measurable_continuous_onI[OF continuous_on_product_coordinates])

lemma *measurable_product_then_coordinatewise*:

fixes $f::'a \Rightarrow 'b \Rightarrow ('c::\text{topological\_space})$

assumes [measurable]: $f \in \text{borel\_measurable } M$

shows $(\lambda x. f \ x \ i) \in \text{borel\_measurable } M$

proof –

have $(\lambda x. f \ x \ i) = (\lambda y. y \ i) \circ f$

unfolding *comp_def* **by** *auto*

then show *?thesis* **by** *simp*

qed

To compare the Borel sigma algebra with the product sigma algebra, we give a presentation of the product sigma algebra that is more similar to the one we used above for the product topology.

lemma *sets_PiM_finite*:

$\text{sets } (Pi_M \ I \ M) = \text{sigma\_sets } (\Pi_E \ i \in I. \text{space } (M \ i))$

$\{(\Pi_E \ i \in I. X \ i) \mid X. (\forall i. X \ i \in \text{sets } (M \ i)) \wedge \text{finite } \{i. X \ i \neq \text{space } (M \ i)\}\}$

proof

have $\{(\Pi_E \ i \in I. X \ i) \mid X. (\forall i. X \ i \in \text{sets } (M \ i)) \wedge \text{finite } \{i. X \ i \neq \text{space } (M \ i)\}\}$

$\subseteq \text{sets } (Pi_M \ I \ M)$

proof (*auto*)

fix X **assume** $H: \forall i. X \ i \in \text{sets } (M \ i) \text{ finite } \{i. X \ i \neq \text{space } (M \ i)\}$

then have $*$: $X \ i \in \text{sets } (M \ i)$ **for** i **by** *simp*

define J **where** $J = \{i \in I. X \ i \neq \text{space } (M \ i)\}$

have $\text{finite } J \ J \subseteq I$ **unfolding** J_def **using** H **by** *auto*

define Y **where** $Y = (\Pi_E \ j \in J. X \ j)$

have $\text{prod\_emb } I \ M \ J \ Y \in \text{sets } (Pi_M \ I \ M)$

unfolding Y_def **apply** (rule *sets_PiM_I*) **using** $\langle \text{finite } J \rangle \langle J \subseteq I \rangle *$ **by**

auto

moreover have $\text{prod\_emb } I \ M \ J \ Y = (\Pi_E \ i \in I. X \ i)$

unfolding $\text{prod\_emb\_def } Y\_def \ J\_def$ **using** $H \text{ sets.sets\_into\_space}[OF \ *]$

by (*auto simp add: PiE_iff, blast*)

ultimately show $Pi_E \ I \ X \in \text{sets } (Pi_M \ I \ M)$ **by** *simp*

qed

then show $\text{sigma\_sets } (\Pi_E \ i \in I. \text{space } (M \ i)) \{(\Pi_E \ i \in I. X \ i) \mid X. (\forall i. X \ i \in \text{sets } (M \ i)) \wedge \text{finite } \{i. X \ i \neq \text{space } (M \ i)\}\}$

$\subseteq \text{sets } (Pi_M \ I \ M)$

by (*metis (mono_tags, lifting) sets.sigma_sets_subset' sets.top_space_PiM*)

have $*$: $\exists X. \{f. (\forall i \in I. f \ i \in \text{space } (M \ i)) \wedge f \in \text{extensional } I \wedge f \ i \in A\} = Pi_E \ I \ X \wedge$

$(\forall i. X \ i \in \text{sets } (M \ i)) \wedge \text{finite } \{i. X \ i \neq \text{space } (M \ i)\}$

if $i \in I \ A \in \text{sets } (M \ i)$ **for** $i \ A$

proof –

define X **where** $X = (\lambda j. \text{if } j = i \text{ then } A \text{ else } \text{space } (M \ j))$

```

have {f. (∀ i ∈ I. f i ∈ space (M i)) ∧ f ∈ extensional I ∧ f i ∈ A} = Pi_E I X
  unfolding X_def using sets.sets_into_space[OF ‹A ∈ sets (M i)›] ‹i ∈ I›
  by (auto simp add: Pi_E_iff extensional_def, metis subsetCE, metis)
moreover have X j ∈ sets (M j) for j
  unfolding X_def using ‹A ∈ sets (M i)› by auto
moreover have finite {j. X j ≠ space (M j)}
  unfolding X_def by simp
ultimately show ?thesis by auto
qed
show sets (Pi_M I M) ⊆ sigma_sets (Pi_E i ∈ I. space (M i)) {(Pi_E i ∈ I. X i) | X.
  (∀ i. X i ∈ sets (M i)) ∧ finite {i. X i ≠ space (M i)}}
  unfolding sets_PiM_single
  apply (rule sigma_sets_mono')
  apply (auto simp add: Pi_E_iff *)
  done
qed

```

lemma sets_PiM_subset_borel:

sets (Pi_M UNIV (λ_. borel)) ⊆ sets borel

proof –

have *: Pi_E UNIV X ∈ sets borel if [measurable]: ∧ i. X i ∈ sets borel finite {i.
X i ≠ UNIV} for X::'a ⇒ 'b set

proof –

define I where I = {i. X i ≠ UNIV}

have finite I unfolding I_def using that by simp

have Pi_E UNIV X = (∩ i ∈ I. (λ x. x i) - (X i) ∩ space borel) ∩ space borel

unfolding I_def by auto

also have ... ∈ sets borel

using that ‹finite I› by measurable

finally show ?thesis by simp

qed

then have {(Pi_E i ∈ UNIV. X i) | X::('a ⇒ 'b set). (∀ i. X i ∈ sets borel) ∧ finite
{i. X i ≠ space borel}} ⊆ sets borel

by auto

then show ?thesis unfolding sets_PiM_finite space_borel

by (simp add: * sets.sigma_sets_subset')

qed

proposition sets_PiM_equal_borel:

sets (Pi_M UNIV (λ i::('a::countable). borel::('b::second_countable_topology mea-
sure))) = sets borel

proof

obtain K::('a ⇒ 'b) set set where K: topological_basis K countable K

∧ k. k ∈ K ⇒ ∃ X. (k = Pi_E UNIV X) ∧ (∀ i. open (X i)) ∧ finite {i.
X i ≠ UNIV}

using product_topology_countable_basis by fast

have *: k ∈ sets (Pi_M UNIV (λ_. borel)) if k ∈ K for k

proof –

obtain X where H: k = Pi_E UNIV X ∧ i. open (X i) finite {i. X i ≠ UNIV}

```

    using K(3)[OF ⟨k ∈ K⟩] by blast
    show ?thesis unfolding H(1) sets_PiM_finite_space_borel using borel_open[OF
H(2)] H(3) by auto
qed
have **: U ∈ sets (Pi_M UNIV (λ_. borel)) if open U for U::('a ⇒ 'b) set
proof -
  obtain B where B ⊆ K U = (⋃ B)
  using ⟨open U⟩ ⟨topological_basis K⟩ by (metis topological_basis_def)
  have countable B using ⟨B ⊆ K⟩ ⟨countable K⟩ countable_subset by blast
  moreover have k ∈ sets (Pi_M UNIV (λ_. borel)) if k ∈ B for k
  using ⟨B ⊆ K⟩ * that by auto
  ultimately show ?thesis unfolding ⟨U = (⋃ B)⟩ by auto
qed
have sigma_sets UNIV (Collect open) ⊆ sets (Pi_M UNIV (λi::'a. (borel::('b
measure))))
  apply (rule sets.sigma_sets_subset') using ** by auto
  then show sets (borel::('a ⇒ 'b) measure) ⊆ sets (Pi_M UNIV (λ_. borel))
  unfolding borel_def by auto
qed (simp add: sets_PiM_subset_borel)

lemma measurable_coordinatewise_then_product:
  fixes f::'a ⇒ ('b::countable) ⇒ ('c::second_countable_topology)
  assumes [measurable]: ∧i. (λx. f x i) ∈ borel_measurable M
  shows f ∈ borel_measurable M
proof -
  have f ∈ measurable M (Pi_M UNIV (λ_. borel))
  by (rule measurable_PiM_single', auto simp add: assms)
  then show ?thesis using sets_PiM_equal_borel measurable_cong_sets by blast
qed

end

```

6.9 Caratheodory Extension Theorem

```

theory Caratheodory
imports Measure_Space
begin

```

Originally from the Hurd/Coble measure theory development, translated by Lawrence Paulson.

```

lemma suminf_enreal_2dimen:
  fixes f:: nat × nat ⇒ enreal
  assumes ∧m. g m = (∑ n. f (m,n))
  shows (∑ i. f (prod_decode i)) = suminf g
proof -
  have g_def: g = (λm. (∑ n. f (m,n)))
  using assms by (simp add: fun_eq_iff)
  have reindex: ∧B. (∑ x∈B. f (prod_decode x)) = sum f (prod_decode ` B)

```

```

  by (simp add: sum.reindex[OF inj_prod_decode] comp_def)
  have (SUP n.  $\sum_{i < n} f \text{ (prod\_decode } i)$ ) = (SUP  $p \in \text{UNIV} \times \text{UNIV}$ .  $\sum_{i < \text{fst } p} \sum_{n < \text{snd } p} f \text{ (} i, n)$ )
  proof (intro SUP_eq; clarsimp simp: sum.cartesian_product reindex)
    fix n
    let ?M =  $\lambda f$ . Suc (Max (f ' prod\_decode ' {.. $n$ }))
    { fix a b x assume  $x < n$  and [symmetric]:  $(a, b) = \text{prod\_decode } x$ 
      then have  $a < ?M \text{ fst } b < ?M \text{ snd}$ 
        by (auto intro!: Max_ge le_imp_less_Suc image_eqI) }
    then have  $\text{sum } f \text{ (prod\_decode ' \{.. $n$ \})} \leq \text{sum } f \text{ (\{.. $?M \text{ fst}$ \}  $\times$  \{.. $?M \text{ snd}$ \})}$ 
      by (auto intro!: sum_mono2)
    then show  $\exists a b. \text{sum } f \text{ (prod\_decode ' \{.. $n$ \})} \leq \text{sum } f \text{ (\{.. $a$ \}  $\times$  \{.. $b$ \})}$  by
  auto
  next
    fix a b
    let ?M = prod\_decode ' {.. $\text{Suc (Max (prod\_encode ' (\{.. $a$ \}  $\times$  \{.. $b$ \})))}$ 
    { fix a' b' assume  $a' < a$   $b' < b$  then have  $(a', b') \in ?M$ 
      by (auto intro!: Max_ge le_imp_less_Suc image_eqI[where  $x = \text{prod\_encode (} a', b')$ ]) }
    then have  $\text{sum } f \text{ (\{.. $a$ \}  $\times$  \{.. $b$ \})} \leq \text{sum } f \text{ ?M}$ 
      by (auto intro!: sum_mono2)
    then show  $\exists n. \text{sum } f \text{ (\{.. $a$ \}  $\times$  \{.. $b$ \})} \leq \text{sum } f \text{ (prod\_decode ' \{.. $n$ \})}$ 
      by auto
  qed
  also have  $\dots = (\text{SUP } p. \sum_{i < p} \sum_{n} f \text{ (} i, n)$ )
    unfolding suminf_sum[OF summableI, symmetric]
    by (simp add: suminf_eq_SUP SUP_pair sum.swap[of _ {.. $\text{fst } \_$ }])
  finally show ?thesis unfolding g_def
    by (simp add: suminf_eq_SUP)
  qed

```

6.9.1 Characterizations of Measures

definition *outer_measure_space* **where**

outer_measure_space $M f \longleftrightarrow \text{positive } M f \wedge \text{increasing } M f \wedge \text{countably\_subadditive } M f$

Lambda Systems

definition *lambda_system* $:: 'a \text{ set} \Rightarrow 'a \text{ set set} \Rightarrow ('a \text{ set} \Rightarrow \text{ennreal}) \Rightarrow 'a \text{ set set}$
where

lambda_system $\Omega M f = \{l \in M. \forall x \in M. f \text{ (} l \cap x) + f \text{ ((}\Omega - l) \cap x) = f x\}$

lemma (in algebra) *lambda_system_eq*:

lambda_system $\Omega M f = \{l \in M. \forall x \in M. f \text{ (} x \cap l) + f \text{ (} x - l) = f x\}$

proof –

have [simp]: $\bigwedge l x. l \in M \implies x \in M \implies (\Omega - l) \cap x = x - l$

by (metis Int_Diff Int_absorb1 Int_commute sets_into_space)

show ?thesis

by (auto simp add: lambda_system_def) (metis Int_commute)+
qed

lemma (in algebra) lambda_system_empty: positive $M f \implies \{\} \in \text{lambda\_system } \Omega M f$
by (auto simp add: positive_def lambda_system_eq)

lemma lambda_system_sets: $x \in \text{lambda\_system } \Omega M f \implies x \in M$
by (simp add: lambda_system_def)

lemma (in algebra) lambda_system_Compl:
fixes $f:: 'a \text{ set} \Rightarrow \text{ennreal}$
assumes $x: x \in \text{lambda\_system } \Omega M f$
shows $\Omega - x \in \text{lambda\_system } \Omega M f$
proof -
have $x \subseteq \Omega$
by (metis sets_into_space lambda_system_sets x)
hence $\Omega - (\Omega - x) = x$
by (metis double_diff equalityE)
with x show ?thesis
by (force simp add: lambda_system_def ac_simps)
qed

lemma (in algebra) lambda_system_Int:
fixes $f:: 'a \text{ set} \Rightarrow \text{ennreal}$
assumes $xl: x \in \text{lambda\_system } \Omega M f$ and $yl: y \in \text{lambda\_system } \Omega M f$
shows $x \cap y \in \text{lambda\_system } \Omega M f$
proof -
from xl yl show ?thesis
proof (auto simp add: positive_def lambda_system_eq Int)
fix u
assume $x: x \in M$ and $y: y \in M$ and $u: u \in M$
and $fx: \forall z \in M. f(z \cap x) + f(z - x) = f z$
and $fy: \forall z \in M. f(z \cap y) + f(z - y) = f z$
have $u - x \cap y \in M$
by (metis Diff Diff_Int Un u x y)
moreover
have $(u - (x \cap y)) \cap y = u \cap y - x$ by blast
moreover
have $u - x \cap y - y = u - y$ by blast
ultimately
have $ey: f(u - x \cap y) = f(u \cap y - x) + f(u - y)$ using fy
by force
have $f(u \cap (x \cap y)) + f(u - x \cap y)$
= $(f(u \cap (x \cap y)) + f(u \cap y - x)) + f(u - y)$
by (simp add: ey ac_simps)
also have ... = $(f((u \cap y) \cap x) + f(u \cap y - x)) + f(u - y)$
by (simp add: Int_ac)
also have ... = $f(u \cap y) + f(u - y)$

```

    using fx [THEN bspec, of u  $\cap$  y] Int y u
  by force
  also have ... = f u
    by (metis fy u)
  finally show f (u  $\cap$  (x  $\cap$  y)) + f (u - x  $\cap$  y) = f u .
qed
qed

```

```

lemma (in algebra) lambda_system_Un:
  fixes f:: 'a set  $\Rightarrow$  ennreal
  assumes xl: x  $\in$  lambda_system  $\Omega$  M f and yl: y  $\in$  lambda_system  $\Omega$  M f
  shows x  $\cup$  y  $\in$  lambda_system  $\Omega$  M f
proof -
  have ( $\Omega$  - x)  $\cap$  ( $\Omega$  - y)  $\in$  M
    by (metis Diff_Un Un compl_sets lambda_system_sets xl yl)
  moreover
  have x  $\cup$  y =  $\Omega$  - (( $\Omega$  - x)  $\cap$  ( $\Omega$  - y))
    by auto (metis subsetD lambda_system_sets sets_into_space xl yl)+
  ultimately show ?thesis
    by (metis lambda_system_Compl lambda_system_Int xl yl)
qed

```

```

lemma (in algebra) lambda_system_algebra:
  positive M f  $\implies$  algebra  $\Omega$  (lambda_system  $\Omega$  M f)
  apply (auto simp add: algebra_iff_Un)
  apply (metis lambda_system_sets subsetD sets_into_space)
  apply (metis lambda_system_empty)
  apply (metis lambda_system_Compl)
  apply (metis lambda_system_Un)
  done

```

```

lemma (in algebra) lambda_system_strong_additive:
  assumes z: z  $\in$  M and disj: x  $\cap$  y = {}
    and xl: x  $\in$  lambda_system  $\Omega$  M f and yl: y  $\in$  lambda_system  $\Omega$  M f
  shows f (z  $\cap$  (x  $\cup$  y)) = f (z  $\cap$  x) + f (z  $\cap$  y)
proof -
  have z  $\cap$  x = (z  $\cap$  (x  $\cup$  y))  $\cap$  x using disj by blast
  moreover
  have z  $\cap$  y = (z  $\cap$  (x  $\cup$  y)) - x using disj by blast
  moreover
  have (z  $\cap$  (x  $\cup$  y))  $\in$  M
    by (metis Int_Un lambda_system_sets xl yl z)
  ultimately show ?thesis using xl yl
    by (simp add: lambda_system_eq)
qed

```

```

lemma (in algebra) lambda_system_additive: additive (lambda_system  $\Omega$  M f) f
proof (auto simp add: additive_def)
  fix x and y

```

assume *disj*: $x \cap y = \{\}$
and *xl*: $x \in \text{lambda\_system } \Omega \ M \ f$ **and** *yl*: $y \in \text{lambda\_system } \Omega \ M \ f$
hence $x \in M \ y \in M$ **by** (*blast intro: lambda_system_sets*)+
thus $f \ (x \cup y) = f \ x + f \ y$
using *lambda_system_strong_additive* [*OF top disj xl yl*]
by (*simp add: Un*)
qed

lemma *lambda_system_increasing*: $\text{increasing } M \ f \implies \text{increasing } (\text{lambda\_system } \Omega \ M \ f) \ f$
by (*simp add: increasing_def lambda_system_def*)

lemma *lambda_system_positive*: $\text{positive } M \ f \implies \text{positive } (\text{lambda\_system } \Omega \ M \ f) \ f$
by (*simp add: positive_def lambda_system_def*)

lemma (*in algebra*) *lambda_system_strong_sum*:
fixes *A*:: $\text{nat} \Rightarrow 'a \ \text{set}$ **and** *f*:: $'a \ \text{set} \Rightarrow \text{ennreal}$
assumes *f*: $\text{positive } M \ f$ **and** *a*: $a \in M$
and *A*: $\text{range } A \subseteq \text{lambda\_system } \Omega \ M \ f$
and *disj*: *disjoint_family* *A*
shows $(\sum i = 0..<n. f \ (a \cap A \ i)) = f \ (a \cap (\bigcup i \in \{0..<n\}. A \ i))$
proof (*induct n*)
case 0 **show** ?*case* **using** *f* **by** (*simp add: positive_def*)
next
case (*Suc n*)
have 2: $A \ n \cap \bigcup (A \ ' \{0..<n\}) = \{\}$ **using** *disj*
by (*force simp add: disjoint_family_on_def neq_iff*)
have 3: $A \ n \in \text{lambda\_system } \Omega \ M \ f$ **using** *A*
by *blast*
interpret *l*: *algebra* $\Omega \ \text{lambda\_system } \Omega \ M \ f$
using *f* **by** (*rule lambda_system_algebra*)
have 4: $\bigcup (A \ ' \{0..<n\}) \in \text{lambda\_system } \Omega \ M \ f$
using *A l.UNION_in_sets* **by** *simp*
from *Suc.hyps* **show** ?*case*
by (*simp add: atLeastLessThanSuc lambda_system_strong_additive* [*OF a 2 3 4*])
qed

proposition (*in sigma_algebra*) *lambda_system_caratheodory*:
assumes *oms*: *outer_measure_space* $M \ f$
and *A*: $\text{range } A \subseteq \text{lambda\_system } \Omega \ M \ f$
and *disj*: *disjoint_family* *A*
shows $(\bigcup i. A \ i) \in \text{lambda\_system } \Omega \ M \ f \wedge (\sum i. f \ (A \ i)) = f \ (\bigcup i. A \ i)$
proof –
have *pos*: $\text{positive } M \ f$ **and** *inc*: $\text{increasing } M \ f$
and *csa*: $\text{countably\_subadditive } M \ f$
by (*metis oms outer_measure_space_def*)+
have *sa*: $\text{subadditive } M \ f$


```

  by (metis countably_subadditive_subadditive csa pos)
  have A':  $\bigwedge S. A'S \subseteq (\text{lambda\_system } \Omega \ M \ f)$  using A
  by auto
  interpret ls: algebra  $\Omega$  lambda_system  $\Omega \ M \ f$ 
  using pos by (rule lambda_system_algebra)
  have A'':  $\text{range } A \subseteq M$ 
  by (metis A image_subset_iff lambda_system_sets)

  have U_in:  $(\bigcup i. A \ i) \in M$ 
  by (metis A'' countable_UN)
  have U_eq:  $f (\bigcup i. A \ i) = (\sum i. f (A \ i))$ 
  proof (rule antisym)
    show  $f (\bigcup i. A \ i) \leq (\sum i. f (A \ i))$ 
    using csa[unfolded countably_subadditive_def] A'' disj U_in by auto
    have dis:  $\bigwedge N. \text{disjoint\_family\_on } A \ \{.. by (intro disjoint_family_on_mono[OF
  _ disj]) auto
    show  $(\sum i. f (A \ i)) \leq f (\bigcup i. A \ i)$ 
    using ls.additive_sum [OF lambda_system_positive[OF pos] lambda_system_additive
  _ A' dis] A''
    by (intro suminf_le_const[OF summableI]) (auto intro!: increasingD[OF inc]
  countable_UN)
  qed
  have  $f (a \cap (\bigcup i. A \ i)) + f (a - (\bigcup i. A \ i)) = f \ a$ 
  if a [iff]:  $a \in M$  for a
  proof (rule antisym)
    have  $\text{range } (\lambda i. a \cap A \ i) \subseteq M$  using A''
    by blast
    moreover
    have disjoint_family  $(\lambda i. a \cap A \ i)$  using disj
    by (auto simp add: disjoint_family_on_def)
    moreover
    have  $a \cap (\bigcup i. A \ i) \in M$ 
    by (metis Int U_in a)
    ultimately
    have  $f (a \cap (\bigcup i. A \ i)) \leq (\sum i. f (a \cap A \ i))$ 
    using csa[unfolded countably_subadditive_def, rule_format, of  $(\lambda i. a \cap A \ i)$ ]
    by (simp add: o_def)
    hence  $f (a \cap (\bigcup i. A \ i)) + f (a - (\bigcup i. A \ i)) \leq (\sum i. f (a \cap A \ i)) + f (a -$ 
   $(\bigcup i. A \ i))$ 
    by (rule add_right_mono)
    also have  $\dots \leq f \ a$ 
  proof (intro ennreal_suminf_bound_add)
    fix n
    have UNION_in:  $(\bigcup i \in \{0..<n\}. A \ i) \in M$ 
    by (metis A'' UNION_in_sets)
    have le_fa:  $f (\bigcup (A \ ' \ \{0..<n\}) \cap a) \leq f \ a$  using A''
    by (blast intro: increasingD [OF inc] A'' UNION_in_sets)
    have ls:  $(\bigcup i \in \{0..<n\}. A \ i) \in \text{lambda\_system } \Omega \ M \ f$ 
    using ls.UNION_in_sets by (simp add: A)$ 
```

1558

hence $eq\_fa: f\ a = f\ (a \cap (\bigcup_{i \in \{0..<n\}} A\ i)) + f\ (a - (\bigcup_{i \in \{0..<n\}} A\ i))$
i))
by (*simp add: lambda_system_eq UNION_in*)
have $f\ (a - (\bigcup_{i \in \{0..<n\}} A\ i)) \leq f\ (a - (\bigcup_{i \in \{0..<n\}} A\ i))$
by (*blast intro: increasingD [OF inc] UNION_in U_in*)
thus $(\sum_{i < n} f\ (a \cap A\ i)) + f\ (a - (\bigcup_{i \in \{0..<n\}} A\ i)) \leq f\ a$
by (*simp add: lambda_system_strong_sum pos A disj eq_fa add_left_mono atLeast0LessThan[symmetric]*)
qed
finally show $f\ (a \cap (\bigcup_{i \in \{0..<n\}} A\ i)) + f\ (a - (\bigcup_{i \in \{0..<n\}} A\ i)) \leq f\ a$
by *simp*
next
have $f\ a \leq f\ (a \cap (\bigcup_{i \in \{0..<n\}} A\ i) \cup (a - (\bigcup_{i \in \{0..<n\}} A\ i)))$
by (*blast intro: increasingD [OF inc] U_in*)
also have $\dots \leq f\ (a \cap (\bigcup_{i \in \{0..<n\}} A\ i)) + f\ (a - (\bigcup_{i \in \{0..<n\}} A\ i))$
by (*blast intro: subadditiveD [OF sa] U_in*)
finally show $f\ a \leq f\ (a \cap (\bigcup_{i \in \{0..<n\}} A\ i)) + f\ (a - (\bigcup_{i \in \{0..<n\}} A\ i))$.
qed
thus *?thesis*
by (*simp add: lambda_system_eq sums_iff U_eq U_in*)
qed

proposition (*in sigma_algebra*) *caratheodory_lemma*:

assumes *oms: outer_measure_space M f*

defines $L \equiv \text{lambda\_system } \Omega\ M\ f$

shows *measure_space* $\Omega\ L\ f$

proof –

have *pos: positive M f*

by (*metis oms outer_measure_space_def*)

have *alg: algebra* $\Omega\ L$

using *lambda_system_algebra [of f, OF pos]*

by (*simp add: algebra_iff_Un L_def*)

then

have *sigma_algebra* $\Omega\ L$

using *lambda_system_caratheodory [OF oms]*

by (*simp add: sigma_algebra_disjoint_iff L_def*)

moreover

have *countably_additive L f positive L f*

using *pos lambda_system_caratheodory [OF oms]*

by (*auto simp add: lambda_system_sets L_def countably_additive_def positive_def*)

ultimately

show *?thesis*

using *pos* **by** (*simp add: measure_space_def*)

qed

definition *outer_measure* :: $'a\ set\ set \Rightarrow ('a\ set \Rightarrow ennreal) \Rightarrow 'a\ set \Rightarrow ennreal$

where

outer_measure M f X =

$(\text{INF } A \in \{A. \text{range } A \subseteq M \wedge \text{disjoint\_family } A \wedge X \subseteq (\bigcup i. A \ i)\}. \sum i. f \ (A \ i))$

lemma (in ring_of_sets) outer_measure_agrees:
 assumes posf: positive M f and ca: countably_additive M f and s: $s \in M$
 shows outer_measure M f s = f s
 unfolding outer_measure_def
proof (safe intro!: antisym INF_greatest)
 fix A :: nat $\Rightarrow$ 'a set assume A: range A $\subseteq$ M and dA: disjoint_family A and
 sA: $s \subseteq (\bigcup x. A \ x)$
 have inc: increasing M f
 by (metis additive_increasing ca countably_additive_additive posf)
 have f s = f $(\bigcup i. A \ i \cap s)$
 using sA by (auto simp: Int_absorb1)
 also have ... = $(\sum i. f \ (A \ i \cap s))$
 using sA dA A s
 by (intro ca[unfolded countably_additive_def, rule_format, symmetric])
 (auto simp: Int_absorb1 disjoint_family_on_def)
 also have ... $\leq (\sum i. f \ (A \ i))$
 using A s by (auto intro!: suminf_le increasingD[OF inc])
 finally show f s $\leq (\sum i. f \ (A \ i))$.
next
 have $(\sum i. f \ (\text{if } i = 0 \text{ then } s \text{ else } \{\})) \leq f s$
 using positiveD1[OF posf] by (subst suminf_finite[of {0}]) auto
 with s show $(\text{INF } A \in \{A. \text{range } A \subseteq M \wedge \text{disjoint\_family } A \wedge s \subseteq \bigcup (A \ i) \}. \sum i. f \ (A \ i)) \leq f s$
 by (intro INF_lower2[of $\lambda i. \text{if } i = 0 \text{ then } s \text{ else } \{\}$])
 (auto simp: disjoint_family_on_def)
qed

lemma outer_measure_empty:
 positive M f $\implies \{\} \in M \implies \text{outer\_measure } M f \ \{\} = 0$
 unfolding outer_measure_def
 by (intro antisym INF_lower2[of $\lambda \_. \{\}$]) (auto simp: disjoint_family_on_def positive_def)

lemma (in ring_of_sets) positive_outer_measure:
 assumes positive M f shows positive (Pow Ω) (outer_measure M f)
 unfolding positive_def by (auto simp: assms outer_measure_empty)

lemma (in ring_of_sets) increasing_outer_measure: increasing (Pow Ω) (outer_measure M f)
 by (force simp: increasing_def outer_measure_def intro!: INF_greatest intro: INF_lower)

lemma (in ring_of_sets) outer_measure_le:
 assumes pos: positive M f and inc: increasing M f and A: range A $\subseteq$ M and
 X: $X \subseteq (\bigcup i. A \ i)$
 shows outer_measure M f X $\leq (\sum i. f \ (A \ i))$

```

unfolding outer_measure_def
proof (safe intro!: INF_lower2[of disjointed A] del: subsetI)
  show dA: range (disjointed A)  $\subseteq$  M
    by (auto intro!: A range_disjointed_sets)
  have  $\forall n. f \text{ (disjointed A } n) \leq f \text{ (A } n)$ 
    by (metis increasingD [OF inc] UNIV_I dA image_subset_iff disjointed_subset
A)
  then show  $(\sum i. f \text{ (disjointed A } i)) \leq (\sum i. f \text{ (A } i))$ 
    by (blast intro!: suminf_le)
qed (auto simp: X UN_disjointed_eq disjoint_family_disjointed)

```

```

lemma (in ring_of_sets) outer_measure_close:
  outer_measure M f X < e  $\implies \exists A. \text{range } A \subseteq M \wedge \text{disjoint\_family } A \wedge X \subseteq$ 
 $(\bigcup i. A \ i) \wedge (\sum i. f \text{ (A } i)) < e$ 
  unfolding outer_measure_def INF_less_iff by auto

```

```

lemma (in ring_of_sets) countably_subadditive_outer_measure:
  assumes posf: positive M f and inc: increasing M f
  shows countably_subadditive (Pow  $\Omega$ ) (outer_measure M f)
proof (simp add: countably_subadditive_def, safe)
  fix A :: nat  $\Rightarrow$  _ assume A: range A  $\subseteq$  Pow ( $\Omega$ ) and sb:  $(\bigcup i. A \ i) \subseteq \Omega$ 
  let ?O = outer_measure M f
  show ?O  $(\bigcup i. A \ i) \leq (\sum n. ?O \text{ (A } n))$ 
  proof (rule ennreal_le_epsilon)
    fix b and e :: real assume 0 < e  $(\sum n. \text{outer\_measure } M \ f \text{ (A } n)) < \text{top}$ 
    then have *:  $\bigwedge n. \text{outer\_measure } M \ f \text{ (A } n) < \text{outer\_measure } M \ f \text{ (A } n) + e$ 
    * (1/2) ^ Suc n
    by (auto simp add: less_top dest!: ennreal_suminf_lessD)
  obtain B
    where B:  $\bigwedge n. \text{range } (B \ n) \subseteq M$ 
    and sbB:  $\bigwedge n. A \ n \subseteq (\bigcup i. B \ n \ i)$ 
    and B1e:  $\bigwedge n. (\sum i. f \text{ (B } n \ i)) \leq ?O \text{ (A } n) + e * (1/2) ^ (\text{Suc } n)$ 
    by (metis less_imp_le outer_measure_close[OF *])

```

```

define C where C = case_prod B o prod_decode
from B have B_in_M:  $\bigwedge i \ j. B \ i \ j \in M$ 
  by (rule range_subsetD)
then have C: range C  $\subseteq M$ 
  by (auto simp add: C_def split_def)
have A_C:  $(\bigcup i. A \ i) \subseteq (\bigcup i. C \ i)$ 
  using sbB by (auto simp add: C_def subset_eq) (metis prod.case prod_encode_inverse)

have ?O  $(\bigcup i. A \ i) \leq ?O \text{ (}\bigcup i. C \ i\text{)}$ 
  using A_C A C by (intro increasing_outer_measure[THEN increasingD])
(auto dest!: sets_into_space)
also have ...  $\leq (\sum i. f \text{ (C } i))$ 
  using C by (intro outer_measure_le[OF posf inc]) auto
also have ...  $= (\sum n. \sum i. f \text{ (B } n \ i))$ 
  using B_in_M unfolding C_def comp_def by (intro suminf_ennreal_2dimen)

```

```

auto
  also have ... ≤ (∑ n. ?O (A n) + e * (1/2) ^ Suc n)
    using B_in_M by (intro suminf_le suminf_nonneg allI Ble) auto
  also have ... = (∑ n. ?O (A n)) + (∑ n. ennreal e * ennreal ((1/2) ^ Suc n))
    using ‹0 < e› by (subst suminf_add[symmetric])
      (auto simp del: ennreal_suminf_cmult simp add: en-
nreal_mult[symmetric])
  also have ... = (∑ n. ?O (A n)) + e
    unfolding ennreal_suminf_cmult
    by (subst suminf_ennreal_eq[OF zero_le_power power_half_series]) auto
  finally show ?O (⋃ i. A i) ≤ (∑ n. ?O (A n)) + e .
qed
qed

```

```

lemma (in ring_of_sets) outer_measure_space_outer_measure:
  positive M f ⇒ increasing M f ⇒ outer_measure_space (Pow Ω) (outer_measure
M f)
  by (simp add: outer_measure_space_def
    positive_outer_measure increasing_outer_measure countably_subadditive_outer_measure)

```

```

lemma (in ring_of_sets) algebra_subset_lambda_system:
  assumes posf: positive M f and inc: increasing M f
  and add: additive M f
  shows M ⊆ lambda_system Ω (Pow Ω) (outer_measure M f)
proof (auto dest: sets_into_space
  simp add: algebra_lambda_system_eq [OF algebra_Pow])
  fix x s assume x: x ∈ M and s: s ⊆ Ω
  have [simp]: ⋀ x. x ∈ M ⇒ s ∩ (Ω - x) = s - x using s
    by blast
  have outer_measure M f (s ∩ x) + outer_measure M f (s - x) ≤ outer_measure
M f s
  unfolding outer_measure_def[of M f s]
proof (safe intro!: INF_greatest)
  fix A :: nat ⇒ 'a set assume A: disjoint_family A range A ⊆ M s ⊆ (⋃ i. A
i)
  have outer_measure M f (s ∩ x) ≤ (∑ i. f (A i ∩ x))
    unfolding outer_measure_def
  proof (safe intro!: INF_lower2[of λ i. A i ∩ x])
    from A(1) show disjoint_family (λ i. A i ∩ x)
      by (rule disjoint_family_on_bisimulation) auto
    qed (insert x A, auto)
  moreover
  have outer_measure M f (s - x) ≤ (∑ i. f (A i - x))
    unfolding outer_measure_def
  proof (safe intro!: INF_lower2[of λ i. A i - x])
    from A(1) show disjoint_family (λ i. A i - x)
      by (rule disjoint_family_on_bisimulation) auto
    qed (insert x A, auto)
  ultimately have outer_measure M f (s ∩ x) + outer_measure M f (s - x) ≤

```

```

      (∑ i. f (A i ∩ x)) + (∑ i. f (A i - x)) by (rule add_mono)
also have ... = (∑ i. f (A i ∩ x) + f (A i - x))
      using A(2) x posf by (subst suminf_add) (auto simp: positive_def)
also have ... = (∑ i. f (A i))
      using A x
      by (subst add[THEN additiveD, symmetric])
      (auto intro!: arg_cong[where f=suminf] arg_cong[where f=f])
      finally show outer_measure M f (s ∩ x) + outer_measure M f (s - x) ≤
(∑ i. f (A i)) .
qed
moreover
have outer_measure M f s ≤ outer_measure M f (s ∩ x) + outer_measure M f
(s - x)
proof -
  have outer_measure M f s = outer_measure M f ((s ∩ x) ∪ (s - x))
    by (metis Un_Diff_Int Un_commute)
  also have ... ≤ outer_measure M f (s ∩ x) + outer_measure M f (s - x)
    apply (rule subadditiveD)
  apply (rule ring_of_sets.countably_subadditive_subadditive [OF ring_of_sets_Pow])
  apply (simp add: positive_def outer_measure_empty[OF posf])
  apply (rule countably_subadditive_outer_measure)
  using s by (auto intro!: posf inc)
  finally show ?thesis .
qed
ultimately
show outer_measure M f (s ∩ x) + outer_measure M f (s - x) = outer_measure
M f s
  by (rule order_antisym)
qed

```

```

lemma measure_down: measure_space Ω N μ ⇒ sigma_algebra Ω M ⇒ M ⊆
N ⇒ measure_space Ω M μ
  by (auto simp add: measure_space_def positive_def countably_additive_def sub-
set_eq)

```

6.9.2 Caratheodory's theorem

```

theorem (in ring_of_sets) caratheodory':
  assumes posf: positive M f and ca: countably_additive M f
  shows ∃ μ :: 'a set ⇒ ennreal. (∀ s ∈ M. μ s = f s) ∧ measure_space Ω
(sigma_sets Ω M) μ
proof -
  have inc: increasing M f
    by (metis additive_increasing ca countably_additive_additive posf)
  let ?O = outer_measure M f
  define ls where ls = lambda_system Ω (Pow Ω) ?O
  have mls: measure_space Ω ls ?O
    using sigma_algebra.caratheodory_lemma
      [OF sigma_algebra_Pow outer_measure_space_outer_measure [OF posf

```

```

inc]]
  by (simp add: ls_def)
  hence sls: sigma_algebra  $\Omega$  ls
  by (simp add: measure_space_def)
  have  $M \subseteq ls$ 
  by (simp add: ls_def)
  (metis ca posf inc countably_additive_additive_algebra_subset_lambda_system)
  hence sgs_sb: sigma_sets ( $\Omega$ ) ( $M$ )  $\subseteq$  ls
  using sigma_algebra.sigma_sets_subset [OF sls, of  $M$ ]
  by simp
  have measure_space  $\Omega$  (sigma_sets  $\Omega$   $M$ ) ?O
  by (rule measure_down [OF mls], rule sigma_algebra_sigma_sets)
  (simp_all add: sgs_sb space_closed)
  thus ?thesis using outer_measure_agrees [OF posf ca]
  by (intro exI[of _ ?O]) auto
qed

```

```

lemma (in ring_of_sets) caratheodory_empty_continuous:
  assumes f: positive  $M$  f additive  $M$  f and fin:  $\bigwedge A. A \in M \implies f A \neq \infty$ 
  assumes cont:  $\bigwedge A. \text{range } A \subseteq M \implies \text{decseq } A \implies (\bigcap i. A i) = \{\} \implies (\lambda i. f (A i)) \longrightarrow 0$ 
  shows  $\exists \mu :: 'a \text{ set} \Rightarrow \text{ennreal}. (\forall s \in M. \mu s = f s) \wedge \text{measure\_space } \Omega$ 
  (sigma_sets  $\Omega$   $M$ )  $\mu$ 
proof (intro caratheodory' empty_continuous_imp_countably_additive f)
  show  $\forall A \in M. f A \neq \infty$  using fin by auto
qed (rule cont)

```

6.9.3 Volumes

```

definition volume :: 'a set set  $\Rightarrow$  ('a set  $\Rightarrow$  ennreal)  $\Rightarrow$  bool where
  volume  $M$  f  $\longleftrightarrow$ 
  (f  $\{\} = 0$ )  $\wedge$  ( $\forall a \in M. 0 \leq f a$ )  $\wedge$ 
  ( $\forall C \subseteq M. \text{disjoint } C \longrightarrow \text{finite } C \longrightarrow \bigcup C \in M \longrightarrow f (\bigcup C) = (\sum c \in C. f c)$ )

```

```

lemma volumeI:
  assumes f  $\{\} = 0$ 
  assumes  $\bigwedge a. a \in M \implies 0 \leq f a$ 
  assumes  $\bigwedge C. C \subseteq M \implies \text{disjoint } C \implies \text{finite } C \implies \bigcup C \in M \implies f (\bigcup C) = (\sum c \in C. f c)$ 
  shows volume  $M$  f
  using assms by (auto simp: volume_def)

```

```

lemma volume_positive:
  volume  $M$  f  $\implies a \in M \implies 0 \leq f a$ 
  by (auto simp: volume_def)

```

```

lemma volume_empty:
  volume  $M$  f  $\implies f \{\} = 0$ 
  by (auto simp: volume_def)

```

proposition *volume_finite_additive*:

assumes *volume M f*

assumes *A*: $\bigwedge i. i \in I \implies A\ i \in M$ *disjoint_family_on A I finite I* $\bigcup (A\ 'I) \in M$

shows $f\ (\bigcup (A\ 'I)) = (\sum_{i \in I}. f\ (A\ i))$

proof –

have $A\ 'I \subseteq M$ *disjoint (A'I) finite (A'I)* $\bigcup (A\ 'I) \in M$

using *A* **by** (*auto simp: disjoint_family_on_disjoint_image*)

with $\langle \text{volume } M\ f \rangle$ **have** $f\ (\bigcup (A\ 'I)) = (\sum_{a \in A\ 'I}. f\ a)$

unfolding *volume_def* **by** *blast*

also have $\dots = (\sum_{i \in I}. f\ (A\ i))$

proof (*subst sum.reindex_nontrivial*)

fix *i j* **assume** $i \in I\ j \in I\ i \neq j\ A\ i = A\ j$

with $\langle \text{disjoint\_family\_on } A\ I \rangle$ **have** $A\ i = \{\}$

by (*auto simp: disjoint_family_on_def*)

then show $f\ (A\ i) = 0$

using *volume_empty[OF volume M f]* **by** *simp*

qed (*auto intro: finite I*)

finally show $f\ (\bigcup (A\ 'I)) = (\sum_{i \in I}. f\ (A\ i))$

by *simp*

qed

lemma (*in ring_of_sets*) *volume_additiveI*:

assumes *pos*: $\bigwedge a. a \in M \implies 0 \leq \mu\ a$

assumes [*simp*]: $\mu\ \{\} = 0$

assumes *add*: $\bigwedge a\ b. a \in M \implies b \in M \implies a \cap b = \{\} \implies \mu\ (a \cup b) = \mu\ a + \mu\ b$

shows *volume M μ*

proof (*unfold volume_def, safe*)

fix *C* **assume** *finite C C* $C \subseteq M$ *disjoint C*

then show $\mu\ (\bigcup C) = \text{sum } \mu\ C$

proof (*induct C*)

case (*insert c C*)

from *insert(1,2,4,5)* **have** $\mu\ (\bigcup (\text{insert } c\ C)) = \mu\ c + \mu\ (\bigcup C)$

by (*auto intro!: add simp: disjoint_def*)

with *insert* **show** *?case*

by (*simp add: disjoint_def*)

qed *simp*

qed *fact+*

proposition (*in semiring_of_sets*) *extend_volume*:

assumes *volume M μ*

shows $\exists \mu'. \text{volume\_generated\_ring } \mu' \wedge (\forall a \in M. \mu'\ a = \mu\ a)$

proof –

let *?R* = *generated_ring*

have $\forall a \in ?R. \exists m. \exists C \subseteq M. a = \bigcup C \wedge \text{finite } C \wedge \text{disjoint } C \wedge m = (\sum_{c \in C}. \mu\ c)$

by (*auto simp: generated_ring_def*)


```

from bchoice[OF this] obtain  $\mu'$ 
  where  $\mu'_\text{spec}$ :  $\forall x \in \text{generated\_ring}. \exists C \subseteq M. x = \bigcup C \wedge \text{finite } C \wedge \text{disjoint}$ 
 $C \wedge \mu' x = \text{sum } \mu C$ 
by blast
{ fix  $C$  assume  $C$ :  $C \subseteq M$  finite  $C$  disjoint  $C$ 
  fix  $D$  assume  $D$ :  $D \subseteq M$  finite  $D$  disjoint  $D$ 
  assume  $\bigcup C = \bigcup D$ 
  have  $(\sum_{d \in D}. \mu d) = (\sum_{d \in D}. \sum_{c \in C}. \mu (c \cap d))$ 
  proof (intro sum.cong refl)
    fix  $d$  assume  $d \in D$ 
    have  $Un\_eq\_d$ :  $(\bigcup_{c \in C}. c \cap d) = d$ 
      using  $\langle d \in D \rangle \langle \bigcup C = \bigcup D \rangle$  by auto
    moreover have  $\mu (\bigcup_{c \in C}. c \cap d) = (\sum_{c \in C}. \mu (c \cap d))$ 
    proof (rule volume_finite_additive)
      { fix  $c$  assume  $c \in C$  then show  $c \cap d \in M$ 
        using  $C D \langle d \in D \rangle$  by auto }
      show  $(\bigcup_{a \in C}. a \cap d) \in M$ 
        unfolding  $Un\_eq\_d$  using  $\langle d \in D \rangle D$  by auto
      show disjoint_family_on  $(\lambda a. a \cap d) C$ 
        using  $\langle \text{disjoint } C \rangle$  by (auto simp: disjoint_family_on_def disjoint_def)
    qed fact+
    ultimately show  $\mu d = (\sum_{c \in C}. \mu (c \cap d))$  by simp
  qed }
note split_sum = this

{ fix  $C$  assume  $C$ :  $C \subseteq M$  finite  $C$  disjoint  $C$ 
  fix  $D$  assume  $D$ :  $D \subseteq M$  finite  $D$  disjoint  $D$ 
  assume  $\bigcup C = \bigcup D$ 
  with split_sum[OF  $C D$ ] split_sum[OF  $D C$ ]
  have  $(\sum_{d \in D}. \mu d) = (\sum_{c \in C}. \mu c)$ 
    by (simp, subst sum.swap, simp add: ac_simps) }
note sum_eq = this

{ fix  $C$  assume  $C$ :  $C \subseteq M$  finite  $C$  disjoint  $C$ 
  then have  $\bigcup C \in ?R$  by (auto simp: generated_ring_def)
  with  $\mu'_\text{spec}$ [THEN bspec, of  $\bigcup C$ ]
  obtain  $D$  where
     $D$ :  $D \subseteq M$  finite  $D$  disjoint  $D \cup C = \bigcup D$  and  $\mu' (\bigcup C) = (\sum_{d \in D}. \mu d)$ 
    by auto
  with sum_eq[OF  $C D$ ] have  $\mu' (\bigcup C) = (\sum_{c \in C}. \mu c)$  by simp }
note  $\mu' = \text{this}$ 

show ?thesis
proof (intro exI conjI ring_of_sets.volume_additiveI[OF generating_ring] ballI)
  fix  $a$  assume  $a \in M$  with  $\mu'$ [of  $\{a\}$ ] show  $\mu' a = \mu a$ 
    by (simp add: disjoint_def)
next
  fix  $a$  assume  $a \in ?R$ 
  then obtain  $C a$  where  $C a$ : finite  $C a$  disjoint  $C a$   $C a \subseteq M$   $a = \bigcup C a$  ..

```

```

with  $\mu'$ [of  $Ca$ ]  $\langle$ volume  $M$   $\mu$  $\rangle$ [THEN volume_positive]
show  $0 \leq \mu' a$ 
  by (auto intro!: sum_nonneg)
next
show  $\mu' \{\} = 0$  using  $\mu'$ [of  $\{\}$ ] by auto
next
fix  $a$   $b$  assume  $a \in ?R$   $b \in ?R$ 
then obtain  $Ca$   $Cb$ 
  where  $Ca$ : finite  $Ca$  disjoint  $Ca$   $Ca \subseteq M$   $a = \bigcup Ca$ 
  and  $Cb$ : finite  $Cb$  disjoint  $Cb$   $Cb \subseteq M$   $b = \bigcup Cb$ 
  by (meson generated_ringE)
assume  $a \cap b = \{\}$ 
with  $Ca$   $Cb$  have  $Ca \cap Cb \subseteq \{\{\}\}$  by auto
then have  $C\_Int\_cases$ :  $Ca \cap Cb = \{\{\}\} \vee Ca \cap Cb = \{\}$  by auto

from  $\langle a \cap b = \{\} \rangle$  have  $\mu' (\bigcup (Ca \cup Cb)) = (\sum_{c \in Ca \cup Cb} \mu c)$ 
  using  $Ca$   $Cb$  by (intro  $\mu'$ ) (auto intro!: disjoint_union)
also have  $\dots = (\sum_{c \in Ca \cup Cb} \mu c) + (\sum_{c \in Ca \cap Cb} \mu c)$ 
  using  $C\_Int\_cases$  volume_empty[OF  $\langle$ volume  $M$   $\mu$  $\rangle$ ] by (elim disjointE)
simp_all
also have  $\dots = (\sum_{c \in Ca} \mu c) + (\sum_{c \in Cb} \mu c)$ 
  using  $Ca$   $Cb$  by (simp add: sum.union_inter)
also have  $\dots = \mu' a + \mu' b$ 
  using  $Ca$   $Cb$  by (simp add:  $\mu'$ )
finally show  $\mu' (a \cup b) = \mu' a + \mu' b$ 
  using  $Ca$   $Cb$  by simp
qed
qed

```

Caratheodory on semirings

```

theorem (in semiring_of_sets) caratheodory:
  assumes pos: positive  $M$   $\mu$  and ca: countably_additive  $M$   $\mu$ 
  shows  $\exists \mu' :: 'a \text{ set} \Rightarrow \text{ennreal}. (\forall s \in M. \mu' s = \mu s) \wedge \text{measure\_space } \Omega$ 
  (sigma_sets  $\Omega$   $M$ )  $\mu'$ 
proof -
  have volume  $M$   $\mu$ 
  proof (rule volumeI)
    { fix  $a$  assume  $a \in M$  then show  $0 \leq \mu a$ 
      using pos unfolding positive_def by auto }
    note  $p = \text{this}$ 

    fix  $C$  assume sets_C:  $C \subseteq M \bigcup C \in M$  and disjoint  $C$  finite  $C$ 
    have  $\exists F'. \text{bij\_betw } F' \{..<\text{card } C\} C$ 
      by (rule finite_same_card_bij[OF  $\langle$ finite  $C$  $\rangle$ ]) auto
    then obtain  $F'$  where bij_betw  $F' \{..<\text{card } C\} C$  ..
    then have  $F'$ :  $C = F' \setminus \{..<\text{card } C\}$  inj_on  $F' \{..<\text{card } C\}$ 
      by (auto simp: bij_betw_def)
    { fix  $i$   $j$  assume *:  $i < \text{card } C$   $j < \text{card } C$   $i \neq j$ 

```

```

with F' have F' i ∈ C F' j ∈ C F' i ≠ F' j
  unfolding inj_on_def by auto
with ⟨disjoint C⟩[THEN disjointD]
have F' i ∩ F' j = {}
  by auto }
note F'_disj = this
define F where F i = (if i < card C then F' i else {}) for i
then have disjoint_family F
  using F'_disj by (auto simp: disjoint_family_on_def)
moreover from F' have (⋃ i. F i) = ⋃ C
  by (auto simp add: F_def split: if_split_asm cong del: SUP_cong)
moreover have sets_F: ⋀ i. F i ∈ M
  using F' sets_C by (auto simp: F_def)
moreover note sets_C
ultimately have μ (⋃ C) = (∑ i. μ (F i))
  using ca[unfolded countably_additive_def, THEN spec, of F] by auto
also have ... = (∑ i < card C. μ (F' i))
proof -
  have (λ i. if i ∈ {.. $\text{card } C\}$  then μ (F' i) else 0) sums (∑ i < card C. μ (F'
i))
    by (rule sums_If_finite_set) auto
  also have (λ i. if i ∈ {.. $\text{card } C\}$  then μ (F' i) else 0) = (λ i. μ (F i))
    using pos by (auto simp: positive_def F_def)
  finally show (∑ i. μ (F i)) = (∑ i < card C. μ (F' i))
    by (simp add: sums_iff)
qed
also have ... = (∑ c ∈ C. μ c)
  using F'(2) by (subst (2) F') (simp add: sum.reindex)
finally show μ (⋃ C) = (∑ c ∈ C. μ c) .
next
show μ {} = 0
  using ⟨positive M μ⟩ by (rule positiveD1)
qed
from extend_volume[OF this] obtain μ_r where
  V: volume_generated_ring μ_r ∧ a. a ∈ M ⇒ μ a = μ_r a
  by auto

interpret G: ring_of_sets Ω generated_ring
  by (rule generating_ring)

have pos: positive_generated_ring μ_r
  using V unfolding positive_def by (auto simp: positive_def intro!: volume_positive volume_empty)

have countably_additive_generated_ring μ_r
proof (rule countably_additiveI)
  fix A' :: nat ⇒ 'a set
  assume A': range A' ⊆ generated_ring_disjoint_family A'
  and Un_A: (⋃ i. A' i) ∈ generated_ring

```

```

obtain  $C'$  where  $C'$ : finite  $C'$  disjoint  $C' C' \subseteq M \cup (\text{range } A') = \bigcup C'$ 
using generated_ringE[OF Un_A] by auto

{ fix  $c$  assume  $c \in C'$ 
  moreover define  $A$  where [abs_def]:  $A\ i = A'\ i \cap c$  for  $i$ 
  ultimately have  $A$ :  $\text{range } A \subseteq \text{generated\_ring disjoint\_family } A$ 
    and  $\text{Un\_A}$ :  $(\bigcup i. A\ i) \in \text{generated\_ring}$ 
    using  $A'\ C'$ 
    by (auto intro!:  $G.\text{Int } G.\text{finite\_Union intro: generated\_ringI\_Basic simp: disjoint\_family\_on\_def}$ )
    from  $A\ C' \langle c \in C' \rangle$  have  $\text{UN\_eq}$ :  $(\bigcup i. A\ i) = c$ 
    by (auto simp: A\_def)

  have  $\forall i::\text{nat}. \exists f::\text{nat} \Rightarrow 'a\ \text{set}. \mu\_r\ (A\ i) = (\sum j. \mu\_r\ (f\ j)) \wedge \text{disjoint\_family}$ 
 $f \wedge \bigcup (\text{range } f) = A\ i \wedge (\forall j. f\ j \in M)$ 
    (is  $\forall i. ?P\ i$ )
  proof
    fix  $i$ 
    from  $A$  have  $Ai$ :  $A\ i \in \text{generated\_ring}$  by auto
    from generated_ringE[OF this] obtain  $C$ 
      where  $C$ : finite  $C$  disjoint  $C C \subseteq M\ A\ i = \bigcup C$  by auto
    have  $\exists F'. \text{bij\_betw } F' \{..<\text{card } C\}\ C$ 
      by (rule finite\_same\_card\_bij[OF  $\langle \text{finite } C \rangle$ ]) auto
    then obtain  $F$  where  $F$ :  $\text{bij\_betw } F \{..<\text{card } C\}\ C$  ..
    define  $f$  where [abs_def]:  $f\ i = (\text{if } i < \text{card } C \text{ then } F\ i \text{ else } \{\})$  for  $i$ 
    then have  $f$ :  $\text{bij\_betw } f \{..<\text{card } C\}\ C$ 
      by (intro bij\_betw\_cong[THEN iffD1, OF  $F$ ]) auto
    with  $C$  have  $\forall j. f\ j \in M$ 
      by (auto simp: Pi\_iff f\_def dest!: bij\_betw\_imp\_funcset)
    moreover
    from  $f\ C$  have  $d\_f$ : disjoint_family_on  $f \{..<\text{card } C\}$ 
      by (intro disjoint\_image\_disjoint\_family\_on) (auto simp: bij\_betw\_def)
    then have disjoint_family  $f$ 
      by (auto simp: disjoint\_family\_on\_def f\_def)
    moreover
    have  $Ai\_eq$ :  $A\ i = (\bigcup x<\text{card } C. f\ x)$ 
      using  $f\ C\ Ai$  unfolding bij\_betw\_def by auto
    then have  $\bigcup (\text{range } f) = A\ i$ 
      using  $f$  by (auto simp add: f\_def)
    moreover
    { have  $(\sum j. \mu\_r\ (f\ j)) = (\sum j. \text{if } j \in \{..<\text{card } C\} \text{ then } \mu\_r\ (f\ j) \text{ else } 0)$ 
      using volume\_empty[OF V(1)] by (auto intro!: arg\_cong[where
 $f=\text{suminf}$ ] simp: f\_def)
      also have  $\dots = (\sum j<\text{card } C. \mu\_r\ (f\ j))$ 
        by (rule sums\_If\_finite\_set[THEN sums\_unique, symmetric]) simp
      also have  $\dots = \mu\_r\ (A\ i)$ 
        using  $C\ f$  [THEN bij\_betw\_imp\_funcset] unfolding  $Ai\_eq$ 
        by (intro volume\_finite\_additive[OF V(1)  $\_ d\_f, \text{symmetric}$ ])
```

```

      (auto simp: Pi_iff Ai_eq intro: generated_ringI_Basic)
    finally have  $\mu_r (A \ i) = (\sum j. \mu_r (f \ j)) \ .. \}$ 
  ultimately show  $?P \ i$ 
    by blast
qed
from choice[OF this] obtain f
  where f:  $\forall x. \mu_r (A \ x) = (\sum j. \mu_r (f \ x \ j)) \wedge \text{disjoint\_family } (f \ x) \wedge \bigcup (\text{range } (f \ x)) = A \ x \wedge (\forall j. f \ x \ j \in M)$ 
  ..
then have UN_f_eq:  $(\bigcup i. \text{case\_prod } f \ (\text{prod\_decode } i)) = (\bigcup i. A \ i)$ 
  unfolding UN_extend_simps surj_prod_decode by (auto simp: set_eq_iff)

have d: disjoint_family  $(\lambda i. \text{case\_prod } f \ (\text{prod\_decode } i))$ 
  unfolding disjoint_family_on_def
proof (intro ballI impI)
  fix m n :: nat assume  $m \neq n$ 
  then have neq:  $\text{prod\_decode } m \neq \text{prod\_decode } n$ 
    using inj_prod_decode[of UNIV] by (auto simp: inj_on_def)
  show  $\text{case\_prod } f \ (\text{prod\_decode } m) \cap \text{case\_prod } f \ (\text{prod\_decode } n) = \{\}$ 
  proof cases
    assume fst  $(\text{prod\_decode } m) = \text{fst } (\text{prod\_decode } n)$ 
    then show  $?thesis$ 
      using neq f by (fastforce simp: disjoint_family_on_def)
  next
    assume neq:  $\text{fst } (\text{prod\_decode } m) \neq \text{fst } (\text{prod\_decode } n)$ 
    have  $\text{case\_prod } f \ (\text{prod\_decode } m) \subseteq A \ (\text{fst } (\text{prod\_decode } m))$ 
       $\text{case\_prod } f \ (\text{prod\_decode } n) \subseteq A \ (\text{fst } (\text{prod\_decode } n))$ 
    using f[THEN spec, of fst  $(\text{prod\_decode } m)$ ]
      f[THEN spec, of fst  $(\text{prod\_decode } n)$ ]
    by (auto simp: set_eq_iff)
    with f A neq show  $?thesis$ 
      by (fastforce simp: disjoint_family_on_def subset_eq set_eq_iff)
  qed
qed
qed
from f have  $(\sum n. \mu_r (A \ n)) = (\sum n. \mu_r (\text{case\_prod } f \ (\text{prod\_decode } n)))$ 
  by (intro suminf_enreal_2dimen[symmetric] generated_ringI_Basic)
  (auto split: prod.split)
also have  $\dots = (\sum n. \mu (\text{case\_prod } f \ (\text{prod\_decode } n)))$ 
  using f V(2) by (auto intro!: arg_cong[where f=suminf] split: prod.split)
also have  $\dots = \mu (\bigcup i. \text{case\_prod } f \ (\text{prod\_decode } i))$ 
  using f  $\langle c \in C' \rangle C'$ 
  by (intro ca[unfolded countably_additive_def, rule_format])
  (auto split: prod.split simp: UN_f_eq d UN_eq)
finally have  $(\sum n. \mu_r (A' \ n \cap c)) = \mu \ c$ 
  using UN_f_eq UN_eq by (simp add: A_def) }
note eq = this

have  $(\sum n. \mu_r (A' \ n)) = (\sum n. \sum c \in C'. \mu_r (A' \ n \cap c))$ 
  using C' A'

```

```

    by (subst volume_finite_additive[symmetric, OF V(1)])
      (auto simp: disjoint_def disjoint_family_on_def
        intro!: G.Int G.finite_Union arg_cong[where f= $\lambda X. \text{suminf } (\lambda i. \mu_r$ 
( $X\ i$ ))]) ext
      intro: generated_ringI_Basic)
  also have ... =  $(\sum c \in C'. \sum n. \mu_r (A' n \cap c))$ 
    using C' A'
  by (intro suminf_sum G.Int G.finite_Union) (auto intro: generated_ringI_Basic)
  also have ... =  $(\sum c \in C'. \mu_r c)$ 
    using eq V C' by (auto intro!: sum.cong)
  also have ... =  $\mu_r (\bigcup C')$ 
    using C' Un_A
  by (subst volume_finite_additive[symmetric, OF V(1)])
    (auto simp: disjoint_family_on_def disjoint_def
      intro: generated_ringI_Basic)
  finally show  $(\sum n. \mu_r (A' n)) = \mu_r (\bigcup i. A' i)$ 
    using C' by simp
qed
  obtain  $\mu'$  where  $(\forall s \in \text{generated\_ring}. \mu' s = \mu_r s) \wedge \text{measure\_space } \Omega$ 
    (sigma_sets  $\Omega$  generated_ring)  $\mu'$ 
    using G.caratheodory'[OF pos <countably_additive generated_ring  $\mu_r$ ] by
auto
  with V show ?thesis
    unfolding sigma_sets_generated_ring_eq
    by (intro exI[of _  $\mu'$ ]) (auto intro: generated_ringI_Basic)
qed

```

lemma *extend_measure_caratheodory*:

```

  fixes  $G :: 'i \Rightarrow 'a \text{ set}$ 
  assumes  $M: M = \text{extend\_measure } \Omega\ I\ G\ \mu$ 
  assumes  $i \in I$ 
  assumes semiring_of_sets  $\Omega\ (G\ 'I)$ 
  assumes empty:  $\bigwedge i. i \in I \implies G\ i = \{\}$   $\implies \mu\ i = 0$ 
  assumes inj:  $\bigwedge i\ j. i \in I \implies j \in I \implies G\ i = G\ j \implies \mu\ i = \mu\ j$ 
  assumes nonneg:  $\bigwedge i. i \in I \implies 0 \leq \mu\ i$ 
  assumes add:  $\bigwedge A::\text{nat} \Rightarrow 'i. \bigwedge j. A \in \text{UNIV} \rightarrow I \implies j \in I \implies \text{disjoint\_family}$ 
    ( $G \circ A$ )  $\implies$ 
     $(\bigcup i. G\ (A\ i)) = G\ j \implies (\sum n. \mu\ (A\ n)) = \mu\ j$ 
  shows  $\text{emeasure } M\ (G\ i) = \mu\ i$ 

```

proof –

```

  interpret semiring_of_sets  $\Omega\ G\ 'I$ 
  by fact
  have  $\forall g \in G\ 'I. \exists i \in I. g = G\ i$ 
    by auto
  then obtain sel where sel:  $\bigwedge g. g \in G\ 'I \implies \text{sel } g \in I \wedge g. g \in G\ 'I \implies G$ 
    (sel  $g$ ) =  $g$ 
    by metis

```

```

have  $\exists \mu'. (\forall s \in G \text{ ' } I. \mu' s = \mu (sel s)) \wedge measure\_space \Omega (sigma\_sets \Omega (G \text{ ' } I)) \mu'$ 
proof (rule caratheodory)
  show positive (G ' I) ( $\lambda s. \mu (sel s)$ )
    by (auto simp: positive_def intro!: empty sel nonneg)
  show countably_additive (G ' I) ( $\lambda s. \mu (sel s)$ )
  proof (rule countably_additiveI)
    fix A :: nat  $\Rightarrow$  'a set assume range A  $\subseteq$  G ' I disjoint_family A ( $\bigcup i. A i$ )
     $\in G \text{ ' } I$ 
    then show ( $\sum i. \mu (sel (A i))$ ) =  $\mu (sel (\bigcup i. A i))$ 
      by (intro add) (auto simp: sel_image_subset_iff_funcset comp_def Pi_iff
intro!: sel)
    qed
  qed
  then obtain  $\mu'$  where  $\mu': \forall s \in G \text{ ' } I. \mu' s = \mu (sel s)$  measure_space  $\Omega (sigma\_sets \Omega (G \text{ ' } I)) \mu'$ 
    by metis

show ?thesis
proof (rule emeasure_extend_measure[OF M])
  { fix i assume i  $\in$  I then show  $\mu' (G i) = \mu i$ 
    using  $\mu'$  by (auto intro!: inj sel) }
  show G ' I  $\subseteq$  Pow  $\Omega$ 
    by (rule space_closed)
  then show positive (sets M)  $\mu'$  countably_additive (sets M)  $\mu'$ 
    using  $\mu'$  by (simp_all add: M sets_extend_measure measure_space_def)
  qed fact
qed

proposition extend_measure_caratheodory_pair:
  fixes G :: 'i  $\Rightarrow$  'j  $\Rightarrow$  'a set
  assumes M: M = extend_measure  $\Omega \{(a, b). P a b\} (\lambda(a, b). G a b) (\lambda(a, b). \mu a b)$ 
  assumes P i j
  assumes semiring: semiring_of_sets  $\Omega \{G a b \mid a b. P a b\}$ 
  assumes empty:  $\bigwedge i j. P i j \Rightarrow G i j = \{\} \Rightarrow \mu i j = 0$ 
  assumes inj:  $\bigwedge i j k l. P i j \Rightarrow P k l \Rightarrow G i j = G k l \Rightarrow \mu i j = \mu k l$ 
  assumes nonneg:  $\bigwedge i j. P i j \Rightarrow 0 \leq \mu i j$ 
  assumes add:  $\bigwedge A::nat \Rightarrow 'i. \bigwedge B::nat \Rightarrow 'j. \bigwedge j k.$ 
    ( $\bigwedge n. P (A n) (B n) \Rightarrow P j k \Rightarrow disjoint\_family (\lambda n. G (A n) (B n)) \Rightarrow$ 
    ( $\bigcup i. G (A i) (B i) = G j k \Rightarrow (\sum n. \mu (A n) (B n)) = \mu j k$ )
  shows emeasure M (G i j) =  $\mu i j$ 
proof -
  have emeasure M (( $\lambda(a, b). G a b$ ) (i, j)) = ( $\lambda(a, b). \mu a b$ ) (i, j)
  proof (rule extend_measure_caratheodory[OF M])
    show semiring_of_sets  $\Omega ((\lambda(a, b). G a b) \text{ ' } \{(a, b). P a b\})$ 
      using semiring by (simp add: image_def conj_commute)
    next
    fix A :: nat  $\Rightarrow$  ('i  $\times$  'j) and j assume A  $\in UNIV \rightarrow \{(a, b). P a b\} j \in \{(a,$ 

```

```

b). P a b}
  disjoint_family ((λ(a, b). G a b) ∘ A)
  (⋃ i. case A i of (a, b) ⇒ G a b) = (case j of (a, b) ⇒ G a b)
  then show (∑ n. case A n of (a, b) ⇒ μ a b) = (case j of (a, b) ⇒ μ a b)
    using add[of λi. fst (A i) λi. snd (A i) fst j snd j]
    by (simp add: split_beta' comp_def Pi_iff)
  qed (auto split: prod.splits intro: assms)
  then show ?thesis by simp
qed

end

```

6.10 Bochner Integration for Vector-Valued Functions

```

theory Bochner_Integration
  imports Finite_Product_Measure
begin

```

In the following development of the Bochner integral we use second countable topologies instead of separable spaces. A second countable topology is also separable.

proposition *borel_measurable_implies_sequence_metric:*

```

  fixes f :: 'a ⇒ 'b :: {metric_space, second_countable_topology}
  assumes [measurable]: f ∈ borel_measurable M
  shows ∃ F. (∀ i. simple_function M (F i)) ∧ (∀ x ∈ space M. (λi. F i x) ⟶ f
x) ∧
  (∀ i. ∀ x ∈ space M. dist (F i x) z ≤ 2 * dist (f x) z)

```

proof –

```

  obtain D :: 'b set where countable D and D: ⋀ X. open X ⟹ X ≠ {} ⟹
∃ d ∈ D. d ∈ X
  by (erule countable_dense_setE)

```

define *e* **where** *e* = *from_nat_into D*

```

{ fix n x
  obtain d where d ∈ D and d: d ∈ ball x (1 / Suc n)
  using D[of ball x (1 / Suc n)] by auto
  from ⟨d ∈ D⟩ D[of UNIV] ⟨countable D⟩ obtain i where d = e i
  unfolding e_def by (auto dest: from_nat_into_surj)
  with d have ∃ i. dist x (e i) < 1 / Suc n
  by auto }
note e = this

```

define *A* **where** [*abs_def*]: *A m n* =

```

  {x ∈ space M. dist (f x) (e n) < 1 / (Suc m) ∧ 1 / (Suc m) ≤ dist (f x) z} for
m n

```

define *B* **where** [*abs_def*]: *B m* = *disjointed (A m)* **for** *m*


```

define m where [abs_def]:  $m\ N\ x = \text{Max } \{m. m \leq N \wedge x \in (\bigcup_{n \leq N}. B\ m\ n)\}$ 
for N x
define F where [abs_def]:  $F\ N\ x =$ 
  (if  $(\exists m \leq N. x \in (\bigcup_{n \leq N}. B\ m\ n)) \wedge (\exists n \leq N. x \in B\ (m\ N\ x)\ n)$ 
    then  $e\ (LEAST\ n. x \in B\ (m\ N\ x)\ n)$  else z) for N x

have B_imp_A[intro, simp]:  $\bigwedge x\ m\ n. x \in B\ m\ n \implies x \in A\ m\ n$ 
using disjointed_subset[of A m for m] unfolding B_def by auto

{ fix m
  have  $\bigwedge n. A\ m\ n \in \text{sets } M$ 
    by (auto simp: A_def)
  then have  $\bigwedge n. B\ m\ n \in \text{sets } M$ 
    using sets.range_disjointed_sets[of A m M] by (auto simp: B_def) }
note this[measurable]

{ fix N i x assume  $\exists m \leq N. x \in (\bigcup_{n \leq N}. B\ m\ n)$ 
  then have  $m\ N\ x \in \{m::\text{nat}. m \leq N \wedge x \in (\bigcup_{n \leq N}. B\ m\ n)\}$ 
    unfolding m_def by (intro Max_in) auto
  then have  $m\ N\ x \leq N \wedge \exists n \leq N. x \in B\ (m\ N\ x)\ n$ 
    by auto }
note m = this

{ fix j N i x assume  $j \leq N \wedge i \leq N \wedge x \in B\ j\ i$ 
  then have  $j \leq m\ N\ x$ 
    unfolding m_def by (intro Max_ge) auto }
note m_upper = this

show ?thesis
  unfolding simple_function_def
proof (safe intro!: exI[of _ F])
  have [measurable]:  $\bigwedge i. F\ i \in \text{borel\_measurable } M$ 
    unfolding F_def m_def by measurable
  show  $\bigwedge x\ i. F\ i - \{x\} \cap \text{space } M \in \text{sets } M$ 
    by measurable

  { fix i
    { fix n x assume  $x \in B\ (m\ i\ x)\ n$ 
      then have  $(LEAST\ n. x \in B\ (m\ i\ x)\ n) \leq n$ 
        by (intro Least_le)
      also assume  $n \leq i$ 
      finally have  $(LEAST\ n. x \in B\ (m\ i\ x)\ n) \leq i .$  }
    then have  $F\ i - \text{space } M \subseteq \{z\} \cup e - \{.. i\}$ 
      by (auto simp: F_def)
    then show finite  $(F\ i - \text{space } M)$ 
      by (rule finite_subset) auto }

  { fix N i n x assume  $i \leq N \wedge n \leq N \wedge x \in B\ i\ n$ 
    then have  $1: \exists m \leq N. x \in (\bigcup_{n \leq N}. B\ m\ n)$  by auto

```

```

    from m[OF this] obtain n where n: m N x ≤ N n ≤ N x ∈ B (m N x) n
  by auto
  moreover
  define L where L = (LEAST n. x ∈ B (m N x) n)
  have dist (f x) (e L) < 1 / Suc (m N x)
  proof -
    have x ∈ B (m N x) L
    using n(3) unfolding L_def by (rule LeastI)
    then have x ∈ A (m N x) L
    by auto
    then show ?thesis
    unfolding A_def by simp
  qed
  ultimately have dist (f x) (F N x) < 1 / Suc (m N x)
  by (auto simp add: F_def L_def) }
note * = this

fix x assume x ∈ space M
show (λi. F i x) ⟶ f x
proof cases
  assume f x = z
  then have ⋀i n. x ∉ A i n
  unfolding A_def by auto
  then have ⋀i. F i x = z
  by (auto simp: F_def)
  then show ?thesis
  using ⟨f x = z⟩ by auto
next
  assume f x ≠ z

  show ?thesis
  proof (rule tendstoI)
    fix e :: real assume 0 < e
    with ⟨f x ≠ z⟩ obtain n where 1 / Suc n < e 1 / Suc n < dist (f x) z
    by (metis dist_nz order_less_trans neq_iff nat_approx_posE)
    with ⟨x ∈ space M⟩ ⟨f x ≠ z⟩ have x ∈ (⋃ i. B n i)
    unfolding A_def B_def UN_disjointed_eq using e by auto
    then obtain i where i: x ∈ B n i by auto

    show eventually (λi. dist (F i x) (f x) < e) sequentially
    using eventually_ge_at_top[of max n i]
  proof eventually_elim
    fix j assume j: max n i ≤ j
    with i have dist (f x) (F j x) < 1 / Suc (m j x)
    by (intro *[OF _ _ i]) auto
    also have ... ≤ 1 / Suc n
    using j m_upper[OF _ _ i]
    by (auto simp: field_simps)
    also note ⟨1 / Suc n < e⟩
  end
end

```

```

      finally show  $\text{dist } (F j x) (f x) < e$ 
      by (simp add: less_imp_le dist_commute)
    qed
  qed
qed
fix i
{ fix n m assume  $x \in A n m$ 
  then have  $\text{dist } (e m) (f x) + \text{dist } (f x) z \leq 2 * \text{dist } (f x) z$ 
    unfolding A_def by (auto simp: dist_commute)
  also have  $\text{dist } (e m) z \leq \text{dist } (e m) (f x) + \text{dist } (f x) z$ 
    by (rule dist_triangle)
  finally (xtrans) have  $\text{dist } (e m) z \leq 2 * \text{dist } (f x) z . \}$ 
then show  $\text{dist } (F i x) z \leq 2 * \text{dist } (f x) z$ 
  unfolding F_def
  apply auto
  apply (rule LeastI2)
  apply auto
  done
qed
qed

lemma
  fixes  $f :: 'a \Rightarrow 'b::\text{semiring}_1$  assumes  $\text{finite } A$ 
  shows  $\text{sum\_mult\_indicator}[simp]: (\sum x \in A. f x * \text{indicator } (B x) (g x)) =$ 
 $(\sum x \in \{x \in A. g x \in B x\}. f x)$ 
  and  $\text{sum\_indicator\_mult}[simp]: (\sum x \in A. \text{indicator } (B x) (g x) * f x) =$ 
 $(\sum x \in \{x \in A. g x \in B x\}. f x)$ 
  unfolding indicator_def
  using assms by (auto intro!: sum.mono_neutral_cong_right split: if_split_asm)

lemma borel_measurable_induct_real[consumes 2, case_names set mult add seq]:
  fixes  $P :: ('a \Rightarrow \text{real}) \Rightarrow \text{bool}$ 
  assumes  $u: u \in \text{borel\_measurable } M \wedge x. 0 \leq u x$ 
  assumes  $\text{set}: \wedge A. A \in \text{sets } M \Longrightarrow P (\text{indicator } A)$ 
  assumes  $\text{mult}: \wedge u c. 0 \leq c \Longrightarrow u \in \text{borel\_measurable } M \Longrightarrow (\wedge x. 0 \leq u x)$ 
 $\Longrightarrow P u \Longrightarrow P (\lambda x. c * u x)$ 
  assumes  $\text{add}: \wedge u v. u \in \text{borel\_measurable } M \Longrightarrow (\wedge x. 0 \leq u x) \Longrightarrow P u \Longrightarrow$ 
 $v \in \text{borel\_measurable } M \Longrightarrow (\wedge x. 0 \leq v x) \Longrightarrow (\wedge x. x \in \text{space } M \Longrightarrow u x = 0$ 
 $\vee v x = 0) \Longrightarrow P v \Longrightarrow P (\lambda x. v x + u x)$ 
  assumes  $\text{seq}: \wedge U. (\wedge i. U i \in \text{borel\_measurable } M) \Longrightarrow (\wedge i x. 0 \leq U i x) \Longrightarrow$ 
 $(\wedge i. P (U i)) \Longrightarrow \text{incseq } U \Longrightarrow (\wedge x. x \in \text{space } M \Longrightarrow (\lambda i. U i x) \longrightarrow u x)$ 
 $\Longrightarrow P u$ 
  shows  $P u$ 
proof -
  have  $(\lambda x. \text{ennreal } (u x)) \in \text{borel\_measurable } M$  using  $u$  by auto
  from borel_measurable_implies_simple_function_sequence'[OF this]
  obtain  $U$  where  $U: \wedge i. \text{simple\_function } M (U i) \text{ incseq } U \wedge i x. U i x < \text{top}$ 
  and
  sup:  $\wedge x. (\text{SUP } i. U i x) = \text{ennreal } (u x)$ 

```

```

by blast

define U' where [abs_def]: U' i x = indicator (space M) x * enn2real (U i x)
for i x
then have U'_sf[measurable]:  $\bigwedge i. \text{simple\_function } M (U' i)$ 
  using U by (auto intro!: simple_function_compose1[where g=enn2real])

show P u
proof (rule seq)
  show U': U' i  $\in \text{borel\_measurable } M \bigwedge x. 0 \leq U' i x$  for i
    using U by (auto
      intro: borel_measurable_simple_function
      intro!: borel_measurable_enn2real borel_measurable_times
      simp: U'_def zero_le_mult_iff)
  show incseq U'
    using U(2,3)
    by (auto simp: incseq_def le_fun_def image_iff eq_commute U'_def indicator_def enn2real_mono)

  fix x assume x:  $x \in \text{space } M$ 
  have  $(\lambda i. U' i x) \longrightarrow (\text{SUP } i. U' i x)$ 
    using U(2) by (intro LIMSEQ_SUP) (auto simp: incseq_def le_fun_def)
  moreover have  $(\lambda i. U' i x) = (\lambda i. \text{ennreal } (U' i x))$ 
    using x U(3) by (auto simp: fun_eq_iff U'_def image_iff eq_commute)
  moreover have  $(\text{SUP } i. U' i x) = \text{ennreal } (u x)$ 
    using sup u(2) by (simp add: max_def)
  ultimately show  $(\lambda i. U' i x) \longrightarrow u x$ 
    using u U' by simp
next
fix i
have U' i '  $\text{space } M \subseteq \text{enn2real ' } (U' i \text{ ' space } M) \text{ finite } (U' i \text{ ' space } M)$ 
  unfolding U'_def using U(1) by (auto dest: simple_functionD)
then have fin:  $\text{finite } (U' i \text{ ' space } M)$ 
  by (metis finite_subset finite_imageI)
moreover have  $\bigwedge z. \{y. U' i z = y \wedge y \in U' i \text{ ' space } M \wedge z \in \text{space } M\} =$ 
  (if  $z \in \text{space } M$  then  $\{U' i z\}$  else  $\{\}$ )
  by auto
ultimately have U':  $(\lambda z. \sum_{y \in U' i \text{ ' space } M} y * \text{indicator } \{x \in \text{space } M. U' i x = y\} z) = U' i$ 
  by (simp add: U'_def fun_eq_iff)
have  $\bigwedge x. x \in U' i \text{ ' space } M \implies 0 \leq x$ 
  by (auto simp: U'_def)
with fin have P  $(\lambda z. \sum_{y \in U' i \text{ ' space } M} y * \text{indicator } \{x \in \text{space } M. U' i x = y\} z)$ 
proof induct
  case empty from set[of  $\{\}$ ] show ?case
    by (simp add: indicator_def[abs_def])
next
  case (insert x F)

```

```

from insert.premis have nonneg:  $x \geq 0 \wedge y. y \in F \implies y \geq 0$ 
by simp_all
hence *:  $P (\lambda xa. x * \text{indicat\_real } \{x' \in \text{space } M. U' i x' = x\} xa)$ 
by (intro mult set) auto
have  $P (\lambda z. x * \text{indicat\_real } \{x' \in \text{space } M. U' i x' = x\} z +$ 
 $(\sum_{y \in F. y * \text{indicat\_real } \{x \in \text{space } M. U' i x = y\} z))$ 
using insert(1-3)
by (intro add * sum_nonneg mult_nonneg_nonneg)
 $(\text{auto simp: nonneg indicator\_def of\_bool\_def sum\_nonneg\_eq\_0\_iff})$ 
thus ?case
using insert.hyps by (subst sum.insert) auto
qed
with U' show  $P (U' i)$  by simp
qed
qed

```

```

lemma scaleR_cong_right:
fixes  $x :: 'a :: \text{real\_vector}$ 
shows  $(x \neq 0 \implies r = p) \implies r *_R x = p *_R x$ 
by (cases  $x = 0$ ) auto

```

```

inductive simple_bochner_integrable :: 'a measure  $\Rightarrow$  ('a  $\Rightarrow$  'b::real_vector)  $\Rightarrow$ 
bool for M f where
 $\text{simple\_function } M f \implies \text{emeasure } M \{y \in \text{space } M. f y \neq 0\} \neq \infty \implies$ 
 $\text{simple\_bochner\_integrable } M f$ 

```

```

lemma simple_bochner_integrable_compose2:
assumes  $p\_0: p \ 0 \ 0 = 0$ 
shows  $\text{simple\_bochner\_integrable } M f \implies \text{simple\_bochner\_integrable } M g \implies$ 
 $\text{simple\_bochner\_integrable } M (\lambda x. p (f x) (g x))$ 
proof (safe intro!: simple_bochner_integrable.intros elim!: simple_bochner_integrable.cases
del: notI)
assume sf:  $\text{simple\_function } M f \text{ simple\_function } M g$ 
then show  $\text{simple\_function } M (\lambda x. p (f x) (g x))$ 
by (rule simple_function_compose2)

```

```

from sf have [measurable]:
 $f \in \text{measurable } M (\text{count\_space } UNIV)$ 
 $g \in \text{measurable } M (\text{count\_space } UNIV)$ 
by (auto intro: measurable_simple_function)

```

```

assume fin:  $\text{emeasure } M \{y \in \text{space } M. f y \neq 0\} \neq \infty \text{ emeasure } M \{y \in \text{space } M. g y \neq 0\} \neq \infty$ 

```

```

have  $\text{emeasure } M \{x \in \text{space } M. p (f x) (g x) \neq 0\} \leq$ 
 $\text{emeasure } M (\{x \in \text{space } M. f x \neq 0\} \cup \{x \in \text{space } M. g x \neq 0\})$ 
by (intro emeasure_mono) (auto simp: p_0)
also have  $\dots \leq \text{emeasure } M \{x \in \text{space } M. f x \neq 0\} + \text{emeasure } M \{x \in \text{space } M. g x \neq 0\}$ 

```

```

    by (intro emeasure_subadditive) auto
  finally show emeasure M {y ∈ space M. p (f y) (g y) ≠ 0} ≠ ∞
    using fin by (auto simp: top_unique)
qed

```

```

lemma simple_function_finite_support:
  assumes f: simple_function M f and fin: (∫+ x. f x ∂M) < ∞ and nn: ⋀x. 0
  ≤ f x
  shows emeasure M {x ∈ space M. f x ≠ 0} ≠ ∞
proof cases
  from f have meas[measurable]: f ∈ borel_measurable M
    by (rule borel_measurable_simple_function)

  assume non_empty: ∃ x ∈ space M. f x ≠ 0

  define m where m = Min (f ` space M - {0})
  have m ∈ f ` space M - {0}
    unfolding m_def using f non_empty by (intro Min_in) (auto simp: sim-
    ple_function_def)
  then have m: 0 < m
    using nn by (auto simp: less_le)

  from m have m * emeasure M {x ∈ space M. 0 ≠ f x} =
    (∫+ x. m * indicator {x ∈ space M. 0 ≠ f x} x ∂M)
    using f by (intro nn_integral_cmult_indicator[symmetric]) auto
  also have ... ≤ (∫+ x. f x ∂M)
    using AE_space
  proof (intro nn_integral_mono_AE, eventually_elim)
    fix x assume x ∈ space M
    with nn show m * indicator {x ∈ space M. 0 ≠ f x} x ≤ f x
      using f by (auto split: split_indicator simp: simple_function_def m_def)
  qed
  also note ⟨... < ∞⟩
  finally show ?thesis
    using m by (auto simp: ennreal_mult_less_top)
next
  assume ¬ (∃ x ∈ space M. f x ≠ 0)
  with nn have *: {x ∈ space M. f x ≠ 0} = {}
    by auto
  show ?thesis unfolding * by simp
qed

```

```

lemma simple_bochner_integrableI_bounded:
  assumes f: simple_function M f and fin: (∫+ x. norm (f x) ∂M) < ∞
  shows simple_bochner_integrable M f
proof
  have emeasure M {y ∈ space M. ennreal (norm (f y)) ≠ 0} ≠ ∞
  proof (rule simple_function_finite_support)
    show simple_function M (λx. ennreal (norm (f x)))

```

```

    using f by (rule simple_function_compose1)
    show  $(\int^+ y. \text{ennreal} (\text{norm} (f y)) \, \partial M) < \infty$  by fact
  qed simp
  then show  $\text{emeasure } M \{y \in \text{space } M. f y \neq 0\} \neq \infty$  by simp
qed fact

```

definition *simple_bochner_integral* :: $'a \text{ measure} \Rightarrow ('a \Rightarrow 'b::\text{real\_vector}) \Rightarrow 'b$
where

```

  simple_bochner_integral M f =  $(\sum y \in f' \text{space } M. \text{measure } M \{x \in \text{space } M. f x = y\} *_{\mathbb{R}} y)$ 

```

proposition *simple_bochner_integral_partition*:

```

  assumes f: simple_bochner_integrable M f and g: simple_function M g
  assumes sub:  $\bigwedge x y. x \in \text{space } M \implies y \in \text{space } M \implies g x = g y \implies f x = f y$ 
  assumes v:  $\bigwedge x. x \in \text{space } M \implies f x = v (g x)$ 
  shows simple_bochner_integral M f =  $(\sum y \in g' \text{space } M. \text{measure } M \{x \in \text{space } M. g x = y\} *_{\mathbb{R}} v y)$ 
  (is _ = ?r)

```

proof –

```

  from f g have [simp]: finite (f' space M) finite (g' space M)
  by (auto simp: simple_function_def elim: simple_bochner_integrable.cases)

```

```

  from f have [measurable]:  $f \in \text{measurable } M (\text{count\_space } \text{UNIV})$ 
  by (auto intro: measurable_simple_function elim: simple_bochner_integrable.cases)

```

```

  from g have [measurable]:  $g \in \text{measurable } M (\text{count\_space } \text{UNIV})$ 
  by (auto intro: measurable_simple_function elim: simple_bochner_integrable.cases)

```

```

  { fix y assume y ∈ space M
    then have  $f' \text{space } M \cap \{i. \exists x \in \text{space } M. i = f x \wedge g y = g x\} = \{v (g y)\}$ 
    by (auto cong: sub simp: v[symmetric]) }
  note eq = this

```

```

  have simple_bochner_integral M f =
     $(\sum y \in f' \text{space } M. (\sum z \in g' \text{space } M. \text{if } \exists x \in \text{space } M. y = f x \wedge z = g x \text{ then } \text{measure } M \{x \in \text{space } M. g x = z\} \text{ else } 0) *_{\mathbb{R}} y)$ 

```

```

  unfolding simple_bochner_integral_def

```

```

  proof (safe intro!: sum.cong scaleR_cong_right)

```

```

    fix y assume y:  $y \in \text{space } M \wedge f y \neq 0$ 

```

```

    have [simp]:  $g' \text{space } M \cap \{z. \exists x \in \text{space } M. f y = f x \wedge z = g x\} = \{z. \exists x \in \text{space } M. f y = f x \wedge z = g x\}$ 

```

```

    by auto

```

```

    have eq:  $\{x \in \text{space } M. f x = f y\} =$ 

```

```

       $(\bigcup i \in \{z. \exists x \in \text{space } M. f y = f x \wedge z = g x\}. \{x \in \text{space } M. g x = i\})$ 

```

```

    by (auto simp: eq_commute cong: sub rev_conj_cong)

```

```

    have finite (g' space M) by simp

```

```

    then have finite  $\{z. \exists x \in \text{space } M. f y = f x \wedge z = g x\}$ 

```

```

    by (rule rev_finite_subset) auto

```

```

moreover
{ fix  $x$  assume  $x \in \text{space } M$   $f\ x = f\ y$ 
  then have  $x \in \text{space } M$   $f\ x \neq 0$ 
    using  $y$  by auto
  then have  $\text{emeasure } M \{y \in \text{space } M. g\ y = g\ x\} \leq \text{emeasure } M \{y \in \text{space } M. f\ y \neq 0\}$ 
    by (auto intro!: emeasure_mono cong: sub)
  then have  $\text{emeasure } M \{x \in \text{space } M. g\ x = g\ x\} < \infty$ 
    using  $f$  by (auto simp: simple_bochner_integrable.simps less_top) }
ultimately
show  $\text{measure } M \{x \in \text{space } M. f\ x = f\ y\} =$ 
   $(\sum z \in g \text{ 'space } M. \text{if } \exists x \in \text{space } M. f\ y = f\ x \wedge z = g\ x \text{ then } \text{measure } M \{x \in \text{space } M. g\ x = z\} \text{ else } 0)$ 
    apply (simp add: sum.If_cases eq)
    apply (subst measure_finite_Union[symmetric])
    apply (auto simp: disjoint_family_on_def less_top)
    done
qed
also have  $\dots = (\sum y \in f \text{ 'space } M. (\sum z \in g \text{ 'space } M. \text{if } \exists x \in \text{space } M. y = f\ x \wedge z = g\ x \text{ then } \text{measure } M \{x \in \text{space } M. g\ x = z\} *_{\mathbb{R}} y \text{ else } 0))$ 
    by (auto intro!: sum.cong simp: scaleR_sum_left)
also have  $\dots = ?r$ 
    by (subst sum.swap)
    (auto intro!: sum.cong simp: sum.If_cases scaleR_sum_right[symmetric] eq)
finally show  $\text{simple\_bochner\_integral } M\ f = ?r$  .
qed

```

lemma *simple_bochner_integral_add*:

assumes f : *simple_bochner_integrable* $M\ f$ **and** g : *simple_bochner_integrable* $M\ g$

shows $\text{simple\_bochner\_integral } M\ (\lambda x. f\ x + g\ x) =$
 $\text{simple\_bochner\_integral } M\ f + \text{simple\_bochner\_integral } M\ g$

proof –

from $f\ g$ **have** $\text{simple\_bochner\_integral } M\ (\lambda x. f\ x + g\ x) =$
 $(\sum y \in (\lambda x. (f\ x, g\ x)) \text{ 'space } M. \text{measure } M \{x \in \text{space } M. (f\ x, g\ x) = y\} *_{\mathbb{R}} (fst\ y + snd\ y))$

by (*intro simple_bochner_integral_partition*)
(auto simp: simple_bochner_integrable_compose2 elim: simple_bochner_integrable.cases)

moreover from $f\ g$ **have** $\text{simple\_bochner\_integral } M\ f =$

$(\sum y \in (\lambda x. (f\ x, g\ x)) \text{ 'space } M. \text{measure } M \{x \in \text{space } M. (f\ x, g\ x) = y\} *_{\mathbb{R}} fst\ y)$

by (*intro simple_bochner_integral_partition*)
(auto simp: simple_bochner_integrable_compose2 elim: simple_bochner_integrable.cases)

moreover from $f\ g$ **have** $\text{simple\_bochner\_integral } M\ g =$

$(\sum y \in (\lambda x. (f\ x, g\ x)) \text{ 'space } M. \text{measure } M \{x \in \text{space } M. (f\ x, g\ x) = y\} *_{\mathbb{R}} snd\ y)$

by (*intro simple_bochner_integral_partition*)
(auto simp: simple_bochner_integrable_compose2 elim: simple_bochner_integrable.cases)


```

ultimately show ?thesis
  by (simp add: sum.distrib[symmetric] scaleR_add_right)
qed

```

```

lemma simple_bochner_integral_linear:
  assumes linear f
  assumes g: simple_bochner_integrable M g
  shows simple_bochner_integral M ( $\lambda x. f (g x)$ ) = f (simple_bochner_integral M g)
proof -
  interpret linear f by fact
  from g have simple_bochner_integral M ( $\lambda x. f (g x)$ ) =
    ( $\sum y \in g \text{ ' space } M. \text{measure } M \{x \in \text{space } M. g x = y\} *_R f y$ )
  by (intro simple_bochner_integral_partition)
    (auto simp: simple_bochner_integrable_compose2 [where p= $\lambda x y. f x$ ]
      elim: simple_bochner_integrable.cases)
  also have ... = f (simple_bochner_integral M g)
    by (simp add: simple_bochner_integral_def sum scale)
  finally show ?thesis .
qed

```

```

lemma simple_bochner_integral_minus:
  assumes f: simple_bochner_integrable M f
  shows simple_bochner_integral M ( $\lambda x. - f x$ ) = - simple_bochner_integral M f
proof -
  from linear_uminus f show ?thesis
    by (rule simple_bochner_integral_linear)
qed

```

```

lemma simple_bochner_integral_diff:
  assumes f: simple_bochner_integrable M f and g: simple_bochner_integrable M g
  shows simple_bochner_integral M ( $\lambda x. f x - g x$ ) =
    simple_bochner_integral M f - simple_bochner_integral M g
  unfolding diff_conv_add_uminus using f g
  by (subst simple_bochner_integral_add)
    (auto simp: simple_bochner_integral_minus simple_bochner_integrable_compose2 [where
      p= $\lambda x y. - y$ ])

```

```

lemma simple_bochner_integral_norm_bound:
  assumes f: simple_bochner_integrable M f
  shows norm (simple_bochner_integral M f)  $\leq$  simple_bochner_integral M ( $\lambda x. \text{norm } (f x)$ )
proof -
  have norm (simple_bochner_integral M f)  $\leq$ 
    ( $\sum y \in f \text{ ' space } M. \text{norm } (\text{measure } M \{x \in \text{space } M. f x = y\} *_R y)$ )
  unfolding simple_bochner_integral_def by (rule norm_sum)
  also have ... = ( $\sum y \in f \text{ ' space } M. \text{measure } M \{x \in \text{space } M. f x = y\} *_R \text{norm } (f y)$ )

```

```

y)
  by simp
  also have ... = simple_bochner_integral M (λx. norm (f x))
  using f
  by (intro simple_bochner_integral_partition[symmetric])
  (auto intro: f simple_bochner_integrable_compose2 elim: simple_bochner_integrable.cases)
  finally show ?thesis .
qed

```

```

lemma simple_bochner_integral_nonneg[simp]:
  fixes f :: 'a ⇒ real
  shows (Λx. 0 ≤ f x) ⇒ 0 ≤ simple_bochner_integral M f
  by (force simp add: simple_bochner_integral_def intro: sum_nonneg)

```

```

lemma simple_bochner_integral_eq_nn_integral:
  assumes f: simple_bochner_integrable M f ∧ x. 0 ≤ f x
  shows simple_bochner_integral M f = (∫+x. f x ∂M)
proof -
  { fix x y z have (x ≠ 0 ⇒ y = z) ⇒ ennreal x * y = ennreal x * z
    by (cases x = 0) (auto simp: zero_ennreal_def[symmetric]) }
  note ennreal_cong_mult = this

```

```

have [measurable]: f ∈ borel_measurable M
  using f(1) by (auto intro: borel_measurable_simple_function elim: simple_bochner_integrable.cases)

```

```

{ fix y assume y: y ∈ space M f y ≠ 0
  have ennreal (measure M {x ∈ space M. f x = f y}) = emeasure M {x ∈ space
M. f x = f y}
  proof (rule emeasure_eq_ennreal_measure[symmetric])
    have emeasure M {x ∈ space M. f x = f y} ≤ emeasure M {x ∈ space M. f
x ≠ 0}
    using y by (intro emeasure_mono) auto
    with f show emeasure M {x ∈ space M. f x = f y} ≠ top
    by (auto simp: simple_bochner_integrable.simps top_unique)
  qed
  moreover have {x ∈ space M. f x = f y} = (λx. ennreal (f x)) - ' {ennreal (f
y)} ∩ space M
  using f by auto
  ultimately have ennreal (measure M {x ∈ space M. f x = f y}) =
    emeasure M ((λx. ennreal (f x)) - ' {ennreal (f y)} ∩ space M) by simp }
with f have simple_bochner_integral M f = (∫Sx. f x ∂M)
  unfolding simple_integral_def
  by (subst simple_bochner_integral_partition[OF f(1), where g=λx. ennreal (f
x) and v=enn2real])
  (auto intro: f simple_function_compose1 elim: simple_bochner_integrable.cases
  intro!: sum.cong ennreal_cong_mult
  simp: ac_simps ennreal_mult
  simp flip: sum_ennreal)
also have ... = (∫+x. f x ∂M)

```

```

    using f
  by (intro nn_integral_eq_simple_integral[symmetric])
    (auto simp: simple_function_compose1 simple_bochner_integrable.simps)
  finally show ?thesis .
qed

```

lemma *simple_bochner_integral_bounded*:

```

  fixes f :: 'a  $\Rightarrow$  'b::{real_normed_vector, second_countable_topology}
  assumes f[measurable]: f  $\in$  borel_measurable M
  assumes s: simple_bochner_integrable M s and t: simple_bochner_integrable M
  t
  shows ennreal (norm (simple_bochner_integral M s - simple_bochner_integral
  M t))  $\leq$ 
    ( $\int^+ x. \text{norm } (f x - s x) \, \partial M$ ) + ( $\int^+ x. \text{norm } (f x - t x) \, \partial M$ )
    (is ennreal (norm (?s - ?t))  $\leq$  ?S + ?T)
  proof -
    have [measurable]: s  $\in$  borel_measurable M t  $\in$  borel_measurable M
    using s t by (auto intro: borel_measurable_simple_function elim: simple_bochner_integrable.cases)

    have ennreal (norm (?s - ?t)) = norm (simple_bochner_integral M ( $\lambda x. s x -$ 
    t x))
    using s t by (subst simple_bochner_integral_diff) auto
    also have ...  $\leq$  simple_bochner_integral M ( $\lambda x. \text{norm } (s x - t x)$ )
    using simple_bochner_integrable_compose2[of (-) M s t] s t
    by (auto intro!: simple_bochner_integral_norm_bound)
    also have ... = ( $\int^+ x. \text{norm } (s x - t x) \, \partial M$ )
    using simple_bochner_integrable_compose2[of  $\lambda x y. \text{norm } (x - y)$  M s t] s t
    by (auto intro!: simple_bochner_integral_eq_nn_integral)
    also have ...  $\leq$  ( $\int^+ x. \text{ennreal } (\text{norm } (f x - s x)) + \text{ennreal } (\text{norm } (f x - t x))$ 
     $\partial M$ )
    by (auto intro!: nn_integral_mono simp flip: ennreal_plus)
    (metis (erased, opaque_lifting) add_diff_cancel_left add_diff_eq diff_add_eq
    order_trans
    norm_minus_commute norm_triangle_ineq4 order_refl)
    also have ... = ?S + ?T
    by (rule nn_integral_add) auto
    finally show ?thesis .
  qed

```

inductive *has_bochner_integral* :: 'a measure $\Rightarrow$ ('a $\Rightarrow$ 'b) $\Rightarrow$ 'b::{real_normed_vector, second_countable_topology} $\Rightarrow$ bool

```

  for M f x where
    f  $\in$  borel_measurable M  $\implies$ 
      ( $\bigwedge i. \text{simple\_bochner\_integrable } M (s i) \implies$ 
        ( $\lambda i. \int^+ x. \text{norm } (f x - s i x) \, \partial M \longrightarrow 0 \implies$ 
          ( $\lambda i. \text{simple\_bochner\_integral } M (s i) \longrightarrow x \implies$ 
            has_bochner_integral M f x

```

lemma *has_bochner_integral_cong*:

```

assumes  $M = N \wedge x. x \in \text{space } N \implies f x = g x$ 
shows  $\text{has\_bochner\_integral } M f x \longleftrightarrow \text{has\_bochner\_integral } N g y$ 
unfolding  $\text{has\_bochner\_integral.simps}$   $\text{assms}(1,3)$ 
using  $\text{assms}(2)$  by  $(\text{simp cong: measurable\_cong\_simp nn\_integral\_cong\_simp})$ 

```

```

lemma  $\text{has\_bochner\_integral\_cong\_AE}$ :
   $f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies (\text{AE } x \text{ in } M. f x = g x) \implies$ 
     $\text{has\_bochner\_integral } M f x \longleftrightarrow \text{has\_bochner\_integral } M g x$ 
unfolding  $\text{has\_bochner\_integral.simps}$ 
by  $(\text{intro arg\_cong}[\text{where } f = Ex] \text{ ext conj\_cong rev\_conj\_cong refl arg\_cong}[\text{where } f = \lambda x. x \longrightarrow 0]$ 
     $\text{nn\_integral\_cong\_AE})$ 
  auto

```

```

lemma  $\text{borel\_measurable\_has\_bochner\_integral}$ :
   $\text{has\_bochner\_integral } M f x \implies f \in \text{borel\_measurable } M$ 
by  $(\text{rule } \text{has\_bochner\_integral.cases})$ 

```

```

lemma  $\text{borel\_measurable\_has\_bochner\_integral}'[\text{measurable\_dest}]$ :
   $\text{has\_bochner\_integral } M f x \implies g \in \text{measurable } N M \implies (\lambda x. f (g x)) \in \text{borel\_measurable } N$ 
using  $\text{borel\_measurable\_has\_bochner\_integral}[\text{measurable}]$  by measurable

```

```

lemma  $\text{has\_bochner\_integral\_simple\_bochner\_integrable}$ :
   $\text{simple\_bochner\_integrable } M f \implies \text{has\_bochner\_integral } M f$   $(\text{simple\_bochner\_integral } M f)$ 
by  $(\text{rule } \text{has\_bochner\_integral.intros}[\text{where } s = \lambda_. f])$ 
   $(\text{auto intro: borel\_measurable\_simple\_function}$ 
     $\text{elim: simple\_bochner\_integrable.cases}$ 
     $\text{simp: zero\_ennreal\_def[symmetric]})$ 

```

```

lemma  $\text{has\_bochner\_integral\_real\_indicator}$ :
assumes  $[\text{measurable}]: A \in \text{sets } M$  and  $A: \text{emeasure } M A < \infty$ 
shows  $\text{has\_bochner\_integral } M (\text{indicator } A)$   $(\text{measure } M A)$ 
proof –
  have  $\text{sbi: simple\_bochner\_integrable } M (\text{indicator } A::'a \Rightarrow \text{real})$ 
proof
  have  $\{y \in \text{space } M. (\text{indicator } A y::\text{real}) \neq 0\} = A$ 
    using  $\text{sets.sets\_into\_space}[OF \langle A \in \text{sets } M \rangle]$  by  $(\text{auto split: split\_indicator})$ 
  then show  $\text{emeasure } M \{y \in \text{space } M. (\text{indicator } A y::\text{real}) \neq 0\} \neq \infty$ 
    using  $A$  by auto
qed  $(\text{rule simple\_function\_indicator assms})+$ 
moreover have  $\text{simple\_bochner\_integral } M (\text{indicator } A) = \text{measure } M A$ 
  using  $\text{simple\_bochner\_integral\_eq\_nn\_integral}[OF \text{sbi}] A$ 
  by  $(\text{simp add: ennreal\_indicator emeasure\_eq\_ennreal\_measure})$ 
ultimately show ?thesis
  by  $(\text{metis } \text{has\_bochner\_integral\_simple\_bochner\_integrable})$ 
qed

```

```

lemma has_bochner_integral_add[intro]:
  has_bochner_integral M f x  $\implies$  has_bochner_integral M g y  $\implies$ 
    has_bochner_integral M ( $\lambda x. f x + g x$ ) (x + y)
proof (safe intro!: has_bochner_integral.intros elim!: has_bochner_integral.cases)
  fix sf sg
  assume f_sf: ( $\lambda i. \int^+ x. \text{norm } (f x - sf i x) \partial M$ )  $\longrightarrow 0$ 
  assume g_sg: ( $\lambda i. \int^+ x. \text{norm } (g x - sg i x) \partial M$ )  $\longrightarrow 0$ 

  assume sf:  $\forall i. \text{simple\_bochner\_integrable } M (sf i)$ 
  and sg:  $\forall i. \text{simple\_bochner\_integrable } M (sg i)$ 
  then have [measurable]:  $\bigwedge i. sf i \in \text{borel\_measurable } M \bigwedge i. sg i \in \text{borel\_measurable } M$ 
  by (auto intro: borel_measurable_simple_function elim: simple_bochner_integrable.cases)
  assume [measurable]:  $f \in \text{borel\_measurable } M \ g \in \text{borel\_measurable } M$ 

  show  $\bigwedge i. \text{simple\_bochner\_integrable } M (\lambda x. sf i x + sg i x)$ 
  using sf sg by (simp add: simple_bochner_integrable_compose2)

  show ( $\lambda i. \int^+ x. (\text{norm } (f x + g x - (sf i x + sg i x))) \partial M$ )  $\longrightarrow 0$ 
  (is ?f  $\longrightarrow 0$ )
  proof (rule tendsto_sandwich)
    show eventually ( $\lambda n. 0 \leq ?f n$ ) sequentially ( $\lambda \_. 0$ )  $\longrightarrow 0$ 
    by auto
    show eventually ( $\lambda i. ?f i \leq (\int^+ x. (\text{norm } (f x - sf i x)) \partial M) + \int^+ x. (\text{norm } (g x - sg i x)) \partial M$ ) sequentially
    (is eventually ( $\lambda i. ?f i \leq ?g i$ ) sequentially)
    proof (intro always_eventually_allI)
      fix i have ?f i  $\leq (\int^+ x. (\text{norm } (f x - sf i x)) + \text{ennreal } (\text{norm } (g x - sg i x)) \partial M)$ 
      by (auto intro!: nn_integral_mono norm_diff_triangle_ineq simp flip: ennreal_plus)
      also have ... = ?g i
      by (intro nn_integral_add) auto
      finally show ?f i  $\leq ?g i$  .
    qed
    show ?g  $\longrightarrow 0$ 
    using tendsto_add[OF f_sf g_sg] by simp
  qed
qed (auto simp: simple_bochner_integral_add tendsto_add)

lemma has_bochner_integral_bounded_linear:
  assumes bounded_linear T
  shows has_bochner_integral M f x  $\implies$  has_bochner_integral M ( $\lambda x. T (f x)$ ) (T x)
proof (safe intro!: has_bochner_integral.intros elim!: has_bochner_integral.cases)
  interpret T: bounded_linear T by fact
  have [measurable]:  $T \in \text{borel\_measurable borel}$ 
  by (intro borel_measurable_continuous_onI T.continuous_on continuous_on_id)

```

```

assume [measurable]:  $f \in \text{borel\_measurable } M$ 
then show  $(\lambda x. T (f x)) \in \text{borel\_measurable } M$ 
  by auto

fix  $s$  assume  $f\_s: (\lambda i. \int^+ x. \text{norm } (f x - s i x) \partial M) \longrightarrow 0$ 
assume  $s: \forall i. \text{simple\_bochner\_integrable } M (s i)$ 
then show  $\bigwedge i. \text{simple\_bochner\_integrable } M (\lambda x. T (s i x))$ 
  by (auto intro: simple\_bochner\_integrable\_compose2 T.zero)

have [measurable]:  $\bigwedge i. s i \in \text{borel\_measurable } M$ 
using  $s$  by (auto intro: borel\_measurable\_simple\_function elim: simple\_bochner\_integrable.cases)

obtain  $K$  where  $K: K > 0 \bigwedge x i. \text{norm } (T (f x) - T (s i x)) \leq \text{norm } (f x - s i x) * K$ 
  using T.pos_bounded by (auto simp: T.diff[symmetric])

show  $(\lambda i. \int^+ x. \text{norm } (T (f x) - T (s i x)) \partial M) \longrightarrow 0$ 
  (is ?f  $\longrightarrow 0$ )
proof (rule tendsto_sandwich)
  show eventually  $(\lambda n. 0 \leq ?f n)$  sequentially  $(\lambda \_. 0) \longrightarrow 0$ 
    by auto

  show eventually  $(\lambda i. ?f i \leq K * (\int^+ x. \text{norm } (f x - s i x) \partial M))$  sequentially
    (is eventually  $(\lambda i. ?f i \leq ?g i)$  sequentially)
  proof (intro always_eventually allI)
    fix  $i$  have  $?f i \leq (\int^+ x. \text{ennreal } K * \text{norm } (f x - s i x) \partial M)$ 
      using  $K$  by (intro nn\_integral_mono) (auto simp: ac_simps ennreal_mult[symmetric])
    also have  $\dots = ?g i$ 
      using  $K$  by (intro nn\_integral_cmult) auto
    finally show  $?f i \leq ?g i$  .

  qed
  show  $?g \longrightarrow 0$ 
    using ennreal_tendsto_cmult[OF f_s] by simp
  qed

assume  $(\lambda i. \text{simple\_bochner\_integral } M (s i)) \longrightarrow x$ 
with  $s$  show  $(\lambda i. \text{simple\_bochner\_integral } M (\lambda x. T (s i x))) \longrightarrow T x$ 
  by (auto intro!: T.tendsto simp: simple\_bochner\_integral\_linear T.linear_axioms)
qed

lemma has_bochner_integral_zero[intro]:  $\text{has\_bochner\_integral } M (\lambda x. 0) 0$ 
  by (auto intro!: has_bochner_integral.intros[where s= $\lambda \_. 0$ ])
    simp: zero_ennreal_def[symmetric] simple_bochner_integrable.simps
    simple_bochner_integral_def image_constant_conv)

lemma has_bochner_integral_scaleR_left[intro]:
   $(c \neq 0 \implies \text{has\_bochner\_integral } M f x \implies \text{has\_bochner\_integral } M (\lambda x. f x *_{\mathbb{R}} c) (x *_{\mathbb{R}} c))$ 
  by (cases c = 0) (auto simp add: has_bochner_integral_bounded_linear[OF
```

bounded_linear_scaleR_left)

lemma *has_bochner_integral_scaleR_right*[intro]:

$(c \neq 0 \implies \text{has\_bochner\_integral } M \ f \ x) \implies \text{has\_bochner\_integral } M \ (\lambda x. c *_R f \ x) \ (c *_R x)$

by (cases $c = 0$) (auto simp add: *has_bochner_integral_bounded_linear*[OF *bounded_linear_scaleR_right*])

lemma *has_bochner_integral_mult_left*[intro]:

fixes $c :: \_ :: \{\text{real\_normed\_algebra}, \text{second\_countable\_topology}\}$

shows $(c \neq 0 \implies \text{has\_bochner\_integral } M \ f \ x) \implies \text{has\_bochner\_integral } M \ (\lambda x. f \ x * c) \ (x * c)$

by (cases $c = 0$) (auto simp add: *has_bochner_integral_bounded_linear*[OF *bounded_linear_mult_left*])

lemma *has_bochner_integral_mult_right*[intro]:

fixes $c :: \_ :: \{\text{real\_normed\_algebra}, \text{second\_countable\_topology}\}$

shows $(c \neq 0 \implies \text{has\_bochner\_integral } M \ f \ x) \implies \text{has\_bochner\_integral } M \ (\lambda x. c * f \ x) \ (c * x)$

by (cases $c = 0$) (auto simp add: *has_bochner_integral_bounded_linear*[OF *bounded_linear_mult_right*])

lemmas *has_bochner_integral_divide* =

has_bochner_integral_bounded_linear[OF *bounded_linear_divide*]

lemma *has_bochner_integral_divide_zero*[intro]:

fixes $c :: \_ :: \{\text{real\_normed\_field}, \text{field}, \text{second\_countable\_topology}\}$

shows $(c \neq 0 \implies \text{has\_bochner\_integral } M \ f \ x) \implies \text{has\_bochner\_integral } M \ (\lambda x. f \ x / c) \ (x / c)$

using *has_bochner_integral_divide* **by** (cases $c = 0$) auto

lemma *has_bochner_integral_inner_left*[intro]:

$(c \neq 0 \implies \text{has\_bochner\_integral } M \ f \ x) \implies \text{has\_bochner\_integral } M \ (\lambda x. f \ x \cdot c) \ (x \cdot c)$

by (cases $c = 0$) (auto simp add: *has_bochner_integral_bounded_linear*[OF *bounded_linear_inner_left*])

lemma *has_bochner_integral_inner_right*[intro]:

$(c \neq 0 \implies \text{has\_bochner\_integral } M \ f \ x) \implies \text{has\_bochner\_integral } M \ (\lambda x. c \cdot f \ x) \ (c \cdot x)$

by (cases $c = 0$) (auto simp add: *has_bochner_integral_bounded_linear*[OF *bounded_linear_inner_right*])

lemmas *has_bochner_integral_minus* =

has_bochner_integral_bounded_linear[OF *bounded_linear_minus*[OF *bounded_linear_ident*]]

lemmas *has_bochner_integral_Re* =

has_bochner_integral_bounded_linear[OF *bounded_linear_Re*]

lemmas *has_bochner_integral_Im* =

has_bochner_integral_bounded_linear[OF *bounded_linear_Im*]

```

lemmas has_bochner_integral_cnj =
  has_bochner_integral_bounded_linear[OF bounded_linear_cnj]
lemmas has_bochner_integral_of_real =
  has_bochner_integral_bounded_linear[OF bounded_linear_of_real]
lemmas has_bochner_integral_fst =
  has_bochner_integral_bounded_linear[OF bounded_linear_fst]
lemmas has_bochner_integral_snd =
  has_bochner_integral_bounded_linear[OF bounded_linear_snd]

```

```

lemma has_bochner_integral_indicator:
   $A \in \text{sets } M \implies \text{emeasure } M A < \infty \implies$ 
   $\text{has\_bochner\_integral } M (\lambda x. \text{indicator } A x *_{\mathbb{R}} c) (\text{measure } M A *_{\mathbb{R}} c)$ 
by (intro has_bochner_integral_scaleR_left has_bochner_integral_real_indicator)

```

```

lemma has_bochner_integral_diff:
   $\text{has\_bochner\_integral } M f x \implies \text{has\_bochner\_integral } M g y \implies$ 
   $\text{has\_bochner\_integral } M (\lambda x. f x - g x) (x - y)$ 
unfolding diff_conv_add_uminus
by (intro has_bochner_integral_add has_bochner_integral_minus)

```

```

lemma has_bochner_integral_sum:
   $(\bigwedge i. i \in I \implies \text{has\_bochner\_integral } M (f i) (x i)) \implies$ 
   $\text{has\_bochner\_integral } M (\lambda x. \sum_{i \in I} f i x) (\sum_{i \in I} x i)$ 
by (induct I rule: infinite_finite_induct auto)

```

```

proposition has_bochner_integral_implies_finite_norm:
   $\text{has\_bochner\_integral } M f x \implies (\int^+ x. \text{norm } (f x) \partial M) < \infty$ 

```

```

proof (elim has_bochner_integral.cases)

```

```

  fix s v
  assume [measurable]:  $f \in \text{borel\_measurable } M$  and  $s: \bigwedge i. \text{simple\_bochner\_integrable}$ 
   $M (s i)$  and
   $\text{lim}_0: (\lambda i. \int^+ x. \text{ennreal } (\text{norm } (f x - s i x)) \partial M) \longrightarrow 0$ 
  from order_tendstoD[OF lim_0, of_infinity]
  obtain i where f_s_fin:  $(\int^+ x. \text{ennreal } (\text{norm } (f x - s i x)) \partial M) < \infty$ 
  by (auto simp: eventually_sequentially)

```

```

  have [measurable]:  $\bigwedge i. s i \in \text{borel\_measurable } M$ 
  using s by (auto intro: borel_measurable_simple_function elim: simple_bochner_integrable.cases)

```

```

  define m where  $m = (\text{if space } M = \{\} \text{ then } 0 \text{ else } \text{Max } ((\lambda x. \text{norm } (s i x))' \text{space } M))$ 

```

```

  have finite ( $s i \text{ ' space } M$ )
  using s by (auto simp: simple_function_def simple_bochner_integrable.simps)
  then have finite ( $\text{norm } ' s i \text{ ' space } M$ )
  by (rule finite_imageI)
  then have  $\bigwedge x. x \in \text{space } M \implies \text{norm } (s i x) \leq m \ 0 \leq m$ 
  by (auto simp: m_def image_comp_def Max_ge_iff)
  then have  $(\int^+ x. \text{norm } (s i x) \partial M) \leq (\int^+ x. \text{ennreal } m * \text{indicator } \{x \in \text{space } M. s i x \neq 0\} x \partial M)$ 

```



```

    by (auto split: split_indicator intro!: Max_ge nn_integral_mono simp:)
  also have ... <  $\infty$ 
    using s by (subst nn_integral_cmult_indicator) (auto simp:  $\langle 0 \leq m \rangle$  simple_bochner_integrable.simps ennreal_mult_less_top less_top)
  finally have s_fin:  $(\int^+ x. \text{norm } (s \ i \ x) \ \partial M) < \infty$  .

  have  $(\int^+ x. \text{norm } (f \ x) \ \partial M) \leq (\int^+ x. \text{ennreal } (\text{norm } (f \ x - s \ i \ x))) + \text{ennreal } (\text{norm } (s \ i \ x) \ \partial M)$ 
    by (auto intro!: nn_integral_mono simp flip: ennreal_plus)
    (metis add_commute norm_triangle_sub)
  also have ... =  $(\int^+ x. \text{norm } (f \ x - s \ i \ x) \ \partial M) + (\int^+ x. \text{norm } (s \ i \ x) \ \partial M)$ 
    by (rule nn_integral_add) auto
  also have ... <  $\infty$ 
    using s_fin f_s_fin by auto
  finally show  $(\int^+ x. \text{ennreal } (\text{norm } (f \ x)) \ \partial M) < \infty$  .
qed

```

proposition *has_bochner_integral_norm_bound:*

```

  assumes i: has_bochner_integral M f x
  shows norm x  $\leq (\int^+ x. \text{norm } (f \ x) \ \partial M)$ 
using assms proof
  fix s assume
    x:  $(\lambda i. \text{simple\_bochner\_integral } M \ (s \ i)) \longrightarrow x$  (is ?s  $\longrightarrow x$ ) and
    s[simp]:  $\bigwedge i. \text{simple\_bochner\_integrable } M \ (s \ i)$  and
    lim:  $(\lambda i. \int^+ x. \text{ennreal } (\text{norm } (f \ x - s \ i \ x)) \ \partial M) \longrightarrow 0$  and
    f[measurable]:  $f \in \text{borel\_measurable } M$ 

```

```

  have [measurable]:  $\bigwedge i. s \ i \in \text{borel\_measurable } M$ 

```

```

  using s by (auto simp: simple_bochner_integrable.simps intro: borel_measurable_simple_function)

```

```

  show norm x  $\leq (\int^+ x. \text{norm } (f \ x) \ \partial M)$ 

```

```

  proof (rule LIMSEQ_le)

```

```

    show  $(\lambda i. \text{ennreal } (\text{norm } (?s \ i))) \longrightarrow \text{norm } x$ 

```

```

    using x by (auto simp: tendsto_ennreal_iff intro: tendsto_intros)

```

```

    show  $\exists N. \forall n \geq N. \text{norm } (?s \ n) \leq (\int^+ x. \text{norm } (f \ x - s \ n \ x) \ \partial M) + (\int^+ x. \text{norm } (f \ x) \ \partial M)$ 

```

```

    (is  $\exists N. \forall n \geq N. \_ \leq ?t \ n$ )

```

```

  proof (intro exI allI impI)

```

```

    fix n

```

```

    have  $\text{ennreal } (\text{norm } (?s \ n)) \leq \text{simple\_bochner\_integral } M \ (\lambda x. \text{norm } (s \ n \ x))$ 

```

```

    by (auto intro!: simple_bochner_integral_norm_bound)

```

```

    also have ... =  $(\int^+ x. \text{norm } (s \ n \ x) \ \partial M)$ 

```

```

    by (intro simple_bochner_integral_eq_nn_integral)

```

```

    (auto intro: s simple_bochner_integrable_compose2)

```

```

    also have ...  $\leq (\int^+ x. \text{ennreal } (\text{norm } (f \ x - s \ n \ x)) + \text{norm } (f \ x) \ \partial M)$ 

```

```

    by (auto intro!: nn_integral_mono simp flip: ennreal_plus)

```

```

    (metis add_commute norm_minus_commute norm_triangle_sub)

```

```

    also have ... = ?t n

```

```

    by (rule nn_integral_add) auto

```

```

    finally show  $\text{norm } (?s \ n) \leq ?t \ n$  .
  qed
  have  $?t \longrightarrow 0 + (\int^+ x. \text{ennreal } (\text{norm } (f \ x)) \ \partial M)$ 
    using  $\text{has\_bochner\_integral\_implies\_finite\_norm}[OF \ i]$ 
    by (intro tendsto_add tendsto_const lim)
  then show  $?t \longrightarrow \int^+ x. \text{ennreal } (\text{norm } (f \ x)) \ \partial M$ 
    by simp
  qed
qed

lemma has_bochner_integral_eq:
   $\text{has\_bochner\_integral } M \ f \ x \implies \text{has\_bochner\_integral } M \ f \ y \implies x = y$ 
proof (elim has_bochner_integral.cases)
  assume  $f[\text{measurable}]: f \in \text{borel\_measurable } M$ 

  fix  $s \ t$ 
  assume  $(\lambda i. \int^+ x. \text{norm } (f \ x - s \ i \ x) \ \partial M) \longrightarrow 0$  (is  $?S \longrightarrow 0$ )
  assume  $(\lambda i. \int^+ x. \text{norm } (f \ x - t \ i \ x) \ \partial M) \longrightarrow 0$  (is  $?T \longrightarrow 0$ )
  assume  $s: \bigwedge i. \text{simple\_bochner\_integrable } M \ (s \ i)$ 
  assume  $t: \bigwedge i. \text{simple\_bochner\_integrable } M \ (t \ i)$ 

  have  $[\text{measurable}]: \bigwedge i. s \ i \in \text{borel\_measurable } M \ \bigwedge i. t \ i \in \text{borel\_measurable } M$ 
  using  $s \ t$  by (auto intro: borel_measurable_simple_function elim: simple_bochner_integrable.cases)

  let  $?s = \lambda i. \text{simple\_bochner\_integral } M \ (s \ i)$ 
  let  $?t = \lambda i. \text{simple\_bochner\_integral } M \ (t \ i)$ 
  assume  $?s \longrightarrow x \ ?t \longrightarrow y$ 
  then have  $(\lambda i. \text{norm } (?s \ i - ?t \ i)) \longrightarrow \text{norm } (x - y)$ 
    by (intro tendsto_intros)
  moreover
  have  $(\lambda i. \text{ennreal } (\text{norm } (?s \ i - ?t \ i))) \longrightarrow \text{ennreal } 0$ 
  proof (rule tendsto_sandwich)
    show eventually  $(\lambda i. 0 \leq \text{ennreal } (\text{norm } (?s \ i - ?t \ i)))$  sequentially  $(\lambda \_. 0)$ 
     $\longrightarrow \text{ennreal } 0$ 
    by auto
  show eventually  $(\lambda i. \text{norm } (?s \ i - ?t \ i) \leq ?S \ i + ?T \ i)$  sequentially
    by (intro always_eventually_allI simple_bochner_integral_bounded s t f)
  show  $(\lambda i. ?S \ i + ?T \ i) \longrightarrow \text{ennreal } 0$ 
    using tendsto_add[OF  $\langle ?S \longrightarrow 0 \rangle \ \langle ?T \longrightarrow 0 \rangle$ ] by simp
  qed
  then have  $(\lambda i. \text{norm } (?s \ i - ?t \ i)) \longrightarrow 0$ 
    by (simp flip: ennreal_0)
  ultimately have  $\text{norm } (x - y) = 0$ 
    by (rule LIMSEQ_unique)
  then show  $x = y$  by simp
qed

lemma has_bochner_integral_AE:

```

```

    assumes f: has_bochner_integral M f x
    and g: g ∈ borel_measurable M
    and ae: AE x in M. f x = g x
  shows has_bochner_integral M g x
  using f
proof (safe intro!: has_bochner_integral.intros elim!: has_bochner_integral.cases)
  fix s assume (λi. ∫+ x. ennreal (norm (f x - s i x)) ∂M) ⟶ 0
  also have (λi. ∫+ x. ennreal (norm (f x - s i x)) ∂M) = (λi. ∫+ x. ennreal
    (norm (g x - s i x)) ∂M)
    using ae
  by (intro ext nn_integral_cong_AE, eventually_elim) simp
  finally show (λi. ∫+ x. ennreal (norm (g x - s i x)) ∂M) ⟶ 0 .
qed (auto intro: g)

```

```

lemma has_bochner_integral_eq_AE:
  assumes f: has_bochner_integral M f x
  and g: has_bochner_integral M g y
  and ae: AE x in M. f x = g x
  shows x = y
proof -
  from assms have has_bochner_integral M g x
  by (auto intro: has_bochner_integralI_AE)
  from this g show x = y
  by (rule has_bochner_integral_eq)
qed

```

```

lemma simple_bochner_integrable_restrict_space:
  fixes f :: _ ⇒ 'b::real_normed_vector
  assumes Ω: Ω ∩ space M ∈ sets M
  shows simple_bochner_integrable (restrict_space M Ω) f ⟷
    simple_bochner_integrable M (λx. indicator Ω x *R f x)
  by (simp add: simple_bochner_integrable.simps space_restrict_space
    simple_function_restrict_space[OF Ω] emeasure_restrict_space[OF Ω] Collect_restrict
    indicator_eq_0_iff conj_left_commute)

```

```

lemma simple_bochner_integral_restrict_space:
  fixes f :: _ ⇒ 'b::real_normed_vector
  assumes Ω: Ω ∩ space M ∈ sets M
  assumes f: simple_bochner_integrable (restrict_space M Ω) f
  shows simple_bochner_integral (restrict_space M Ω) f =
    simple_bochner_integral M (λx. indicator Ω x *R f x)
proof -
  have finite ((λx. indicator Ω x *R f x) `space M)
  using f simple_bochner_integrable_restrict_space[OF Ω, of f]
  by (simp add: simple_bochner_integrable.simps simple_function_def)
  then show ?thesis
  by (auto simp: space_restrict_space measure_restrict_space[OF Ω(1)] le_infI2
    simple_bochner_integral_def Collect_restrict)

```

1592

split: *split_indicator split_indicator_asm*
intro!: *sum.mono_neutral_cong_left arg_cong2*[**where** *f=measure*])

qed

context

notes [[*inductive_internals*]]

begin

inductive integrable for *M f* **where**

has_bochner_integral M f x $\implies$ *integrable M f*

end

definition *lebesgue_integral* (*integral^L*) **where**

integral^L M f = (*if* $\exists x$. *has_bochner_integral M f x* *then THE x. has_bochner_integral M f x* *else 0*)

syntax

_lebesgue_integral :: *pttrn* $\Rightarrow$ *real* $\Rightarrow$ '*a measure* $\Rightarrow$ *real* ($\int ((2 \_./ \_)/ \partial \_)$
 [60,61] 110)

translations

$\int x. f \partial M == \text{CONST } \text{lebesgue\_integral } M (\lambda x. f)$

syntax

_ascii_lebesgue_integral :: *pttrn* $\Rightarrow$ '*a measure* $\Rightarrow$ *real* $\Rightarrow$ *real* (($\text{3LINT } (1\_)/|(\_)./$
 $\_)$ [0,110,60] 60)

translations

$\text{LINT } x|M. f == \text{CONST } \text{lebesgue\_integral } M (\lambda x. f)$

lemma *has_bochner_integral_integral_eq*: *has_bochner_integral M f x* $\implies$ *integral^L M f = x*

by (*metis the_equality has_bochner_integral_eq lebesgue_integral_def*)

lemma *has_bochner_integral_integrable*:

integrable M f $\implies$ *has_bochner_integral M f* (*integral^L M f*)

by (*auto simp: has_bochner_integral_integral_eq integrable.simps*)

lemma *has_bochner_integral_iff*:

has_bochner_integral M f x $\iff$ *integrable M f* $\wedge$ *integral^L M f = x*

by (*metis has_bochner_integral_integrable has_bochner_integral_integral_eq integrable.intros*)

lemma *simple_bochner_integrable_eq_integral*:

simple_bochner_integrable M f $\implies$ *simple_bochner_integral M f = integral^L M f*

using *has_bochner_integral_simple_bochner_integrable*[*of M f*]

by (*simp add: has_bochner_integral_integral_eq*)

lemma *not_integrable_integral_eq*: $\neg \text{integrable } M f \implies \text{integral}^L M f = 0$
unfolding *integrable.simps lebesgue_integral_def* **by** (auto intro!: arg_cong[where $f=The$])

lemma *integral_eq_cases*:
 $\text{integrable } M f \longleftrightarrow \text{integrable } N g \implies$
 $(\text{integrable } M f \implies \text{integrable } N g \implies \text{integral}^L M f = \text{integral}^L N g) \implies$
 $\text{integral}^L M f = \text{integral}^L N g$
by (metis *not_integrable_integral_eq*)

lemma *borel_measurable_integrable*[*measurable_dest*]: $\text{integrable } M f \implies f \in \text{borel\_measurable } M$
by (auto elim: *integrable.cases has_bochner_integral.cases*)

lemma *borel_measurable_integrable'*[*measurable_dest*]:
 $\text{integrable } M f \implies g \in \text{measurable } N M \implies (\lambda x. f (g x)) \in \text{borel\_measurable } N$
using *borel_measurable_integrable[measurable]* **by** *measurable*

lemma *integrable_cong*:
 $M = N \implies (\bigwedge x. x \in \text{space } N \implies f x = g x) \implies \text{integrable } M f \longleftrightarrow \text{integrable } N g$
by (*simp cong: has_bochner_integral_cong add: integrable.simps*)

lemma *integrable_cong_AE*:
 $f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies AE x \text{ in } M. f x = g x \implies$
 $\text{integrable } M f \longleftrightarrow \text{integrable } M g$
unfolding *integrable.simps*
by (*intro has_bochner_integral_cong_AE arg_cong[where $f=Ex$] ext*)

lemma *integrable_cong_AE_imp*:
 $\text{integrable } M g \implies f \in \text{borel\_measurable } M \implies (AE x \text{ in } M. g x = f x) \implies \text{integrable } M f$
using *integrable_cong_AE[of $f M g$]* **by** (auto *simp: eq_commute*)

lemma *integral_cong*:
 $M = N \implies (\bigwedge x. x \in \text{space } N \implies f x = g x) \implies \text{integral}^L M f = \text{integral}^L N g$
by (*simp cong: has_bochner_integral_cong cong del: if_weak_cong add: lebesgue_integral_def*)

lemma *integral_cong_AE*:
 $f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies AE x \text{ in } M. f x = g x \implies$
 $\text{integral}^L M f = \text{integral}^L M g$
unfolding *lebesgue_integral_def*
by (*rule arg_cong[where $x=\text{has\_bochner\_integral } M f$]*) (*intro has_bochner_integral_cong_AE ext*)

lemma *integrable_add*[*simp, intro*]: $\text{integrable } M f \implies \text{integrable } M g \implies \text{inte-}$

grable $M (\lambda x. f x + g x)$
by (*auto simp: integrable.simps*)

lemma *integrable_zero*[*simp, intro*]: *integrable* $M (\lambda x. 0)$
by (*metis has_bochner_integral_zero integrable.simps*)

lemma *integrable_sum*[*simp, intro*]: $(\bigwedge i. i \in I \implies \text{integrable } M (f i)) \implies \text{integrable } M (\lambda x. \sum_{i \in I} f i x)$
by (*metis has_bochner_integral_sum integrable.simps*)

lemma *integrable_indicator*[*simp, intro*]: $A \in \text{sets } M \implies \text{emeasure } M A < \infty \implies \text{integrable } M (\lambda x. \text{indicator } A x *_R c)$
by (*metis has_bochner_integral_indicator integrable.simps*)

lemma *integrable_real_indicator*[*simp, intro*]: $A \in \text{sets } M \implies \text{emeasure } M A < \infty \implies \text{integrable } M (\text{indicator } A :: 'a \Rightarrow \text{real})$
by (*metis has_bochner_integral_real_indicator integrable.simps*)

lemma *integrable_diff*[*simp, intro*]: *integrable* $M f \implies \text{integrable } M g \implies \text{integrable } M (\lambda x. f x - g x)$
by (*auto simp: integrable.simps intro: has_bochner_integral_diff*)

lemma *integrable_bounded_linear*: *bounded_linear* $T \implies \text{integrable } M f \implies \text{integrable } M (\lambda x. T (f x))$
by (*auto simp: integrable.simps intro: has_bochner_integral_bounded_linear*)

lemma *integrable_scaleR_left*[*simp, intro*]: $(c \neq 0 \implies \text{integrable } M f) \implies \text{integrable } M (\lambda x. f x *_R c)$
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable_scaleR_right*[*simp, intro*]: $(c \neq 0 \implies \text{integrable } M f) \implies \text{integrable } M (\lambda x. c *_R f x)$
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable_mult_left*[*simp, intro*]:
fixes $c :: \_ :: \{\text{real\_normed\_algebra}, \text{second\_countable\_topology}\}$
shows $(c \neq 0 \implies \text{integrable } M f) \implies \text{integrable } M (\lambda x. f x * c)$
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable_mult_right*[*simp, intro*]:
fixes $c :: \_ :: \{\text{real\_normed\_algebra}, \text{second\_countable\_topology}\}$
shows $(c \neq 0 \implies \text{integrable } M f) \implies \text{integrable } M (\lambda x. c * f x)$
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable_divide_zero*[*simp, intro*]:
fixes $c :: \_ :: \{\text{real\_normed\_field}, \text{field}, \text{second\_countable\_topology}\}$
shows $(c \neq 0 \implies \text{integrable } M f) \implies \text{integrable } M (\lambda x. f x / c)$
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable_inner_left*[simp, intro]:
 $(c \neq 0 \implies \text{integrable } M \ f) \implies \text{integrable } M \ (\lambda x. f \ x \cdot c)$
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable_inner_right*[simp, intro]:
 $(c \neq 0 \implies \text{integrable } M \ f) \implies \text{integrable } M \ (\lambda x. c \cdot f \ x)$
unfolding *integrable.simps* **by** *fastforce*

lemmas *integrable_minus*[simp, intro] =
integrable_bounded_linear[OF *bounded_linear_minus* [OF *bounded_linear_ident*]]
lemmas *integrable_divide*[simp, intro] =
integrable_bounded_linear[OF *bounded_linear_divide*]
lemmas *integrable_Re*[simp, intro] =
integrable_bounded_linear[OF *bounded_linear_Re*]
lemmas *integrable_Im*[simp, intro] =
integrable_bounded_linear[OF *bounded_linear_Im*]
lemmas *integrable_cnj*[simp, intro] =
integrable_bounded_linear[OF *bounded_linear_cnj*]
lemmas *integrable_of_real*[simp, intro] =
integrable_bounded_linear[OF *bounded_linear_of_real*]
lemmas *integrable fst*[simp, intro] =
integrable_bounded_linear[OF *bounded_linear fst*]
lemmas *integrable_snd*[simp, intro] =
integrable_bounded_linear[OF *bounded_linear_snd*]

lemma *integral_zero*[simp]: $\text{integral}^L \ M \ (\lambda x. 0) = 0$
by (*intro* *has_bochner_integral_integral_eq* *has_bochner_integral_zero*)

lemma *integral_add*[simp]: $\text{integrable } M \ f \implies \text{integrable } M \ g \implies$
 $\text{integral}^L \ M \ (\lambda x. f \ x + g \ x) = \text{integral}^L \ M \ f + \text{integral}^L \ M \ g$
by (*intro* *has_bochner_integral_integral_eq* *has_bochner_integral_add* *has_bochner_integral_integrable*)

lemma *integral_diff*[simp]: $\text{integrable } M \ f \implies \text{integrable } M \ g \implies$
 $\text{integral}^L \ M \ (\lambda x. f \ x - g \ x) = \text{integral}^L \ M \ f - \text{integral}^L \ M \ g$
by (*intro* *has_bochner_integral_integral_eq* *has_bochner_integral_diff* *has_bochner_integral_integrable*)

lemma *integral_sum*: $(\bigwedge i. i \in I \implies \text{integrable } M \ (f \ i)) \implies$
 $\text{integral}^L \ M \ (\lambda x. \sum_{i \in I} f \ i \ x) = (\sum_{i \in I} \text{integral}^L \ M \ (f \ i))$
by (*intro* *has_bochner_integral_integral_eq* *has_bochner_integral_sum* *has_bochner_integral_integrable*)

lemma *integral_sum'*[simp]: $(\bigwedge i. i \in I \implies \text{integrable } M \ (f \ i)) \implies$
 $\text{integral}^L \ M \ (\lambda x. \sum_{i \in I} f \ i \ x) = (\sum_{i \in I} \text{integral}^L \ M \ (f \ i))$
unfolding *simp_implies_def* **by** (*rule* *integral_sum*)

lemma *integral_bounded_linear*: $\text{bounded\_linear } T \implies \text{integrable } M \ f \implies$
 $\text{integral}^L \ M \ (\lambda x. T \ (f \ x)) = T \ (\text{integral}^L \ M \ f)$
by (*metis* *has_bochner_integral_bounded_linear* *has_bochner_integral_integrable* *has_bochner_integral_integral_eq*)

```

lemma integral_bounded_linear':
  assumes T: bounded_linear T and T': bounded_linear T'
  assumes *:  $\neg (\forall x. T x = 0) \implies (\forall x. T' (T x) = x)$ 
  shows  $\text{integral}^L M (\lambda x. T (f x)) = T (\text{integral}^L M f)$ 
proof cases
  assume  $(\forall x. T x = 0)$  then show ?thesis
    by simp
next
  assume **:  $\neg (\forall x. T x = 0)$ 
  show ?thesis
  proof cases
    assume integrable M f with T show ?thesis
      by (rule integral_bounded_linear)
  next
    assume not:  $\neg \text{integrable } M f$ 
    moreover have  $\neg \text{integrable } M (\lambda x. T (f x))$ 
  proof
    assume integrable M  $(\lambda x. T (f x))$ 
    from integral_bounded_linear[OF T' this] not *[OF **]
    show False
      by auto
  qed
  ultimately show ?thesis
    using T by (simp add: not_integrable_integral_eq linear_simps)
qed
qed

```

```

lemma integral_scaleR_left[simp]:  $(c \neq 0 \implies \text{integrable } M f) \implies (\int x. f x *_{\mathbb{R}} c \, \partial M) = \text{integral}^L M f *_{\mathbb{R}} c$ 
  by (intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_mult)

```

```

lemma integral_scaleR_right[simp]:  $(\int x. c *_{\mathbb{R}} f x \, \partial M) = c *_{\mathbb{R}} \text{integral}^L M f$ 
  by (rule integral_bounded_linear'[OF bounded_linear_scaleR_right bounded_linear_scaleR_right[of 1 / c]]) simp

```

```

lemma integral_mult_left[simp]:
  fixes c ::  $\_ :: \{\text{real\_normed\_algebra}, \text{second\_countable\_topology}\}$ 
  shows  $(c \neq 0 \implies \text{integrable } M f) \implies (\int x. f x * c \, \partial M) = \text{integral}^L M f * c$ 
  by (intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_mult)

```

```

lemma integral_mult_right[simp]:
  fixes c ::  $\_ :: \{\text{real\_normed\_algebra}, \text{second\_countable\_topology}\}$ 
  shows  $(c \neq 0 \implies \text{integrable } M f) \implies (\int x. c * f x \, \partial M) = c * \text{integral}^L M f$ 
  by (intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_mult)

```

```

lemma integral_mult_left_zero[simp]:
  fixes c ::  $\_ :: \{\text{real\_normed\_field}, \text{second\_countable\_topology}\}$ 
  shows  $(\int x. f x * c \, \partial M) = \text{integral}^L M f * c$ 

```


by (rule integral_bounded_linear'[OF bounded_linear_mult_left bounded_linear_mult_left[of 1 / c]]) simp

lemma integral_mult_right_zero[simp]:

fixes $c :: \_ :: \{\text{real\_normed\_field}, \text{second\_countable\_topology}\}$

shows $(\int x. c * f x \, \partial M) = c * \text{integral}^L M f$

by (rule integral_bounded_linear'[OF bounded_linear_mult_right bounded_linear_mult_right[of 1 / c]]) simp

lemma integral_inner_left[simp]: $(c \neq 0 \implies \text{integrable } M f) \implies (\int x. f x \cdot c \, \partial M) = \text{integral}^L M f \cdot c$

by (intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_inner_left)

lemma integral_inner_right[simp]: $(c \neq 0 \implies \text{integrable } M f) \implies (\int x. c \cdot f x \, \partial M) = c \cdot \text{integral}^L M f$

by (intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_inner_right)

lemma integral_divide_zero[simp]:

fixes $c :: \_ :: \{\text{real\_normed\_field}, \text{field}, \text{second\_countable\_topology}\}$

shows $\text{integral}^L M (\lambda x. f x / c) = \text{integral}^L M f / c$

by (rule integral_bounded_linear'[OF bounded_linear_divide bounded_linear_mult_left[of c]]) simp

lemma integral_minus[simp]: $\text{integral}^L M (\lambda x. - f x) = - \text{integral}^L M f$

by (rule integral_bounded_linear'[OF bounded_linear_minus[OF bounded_linear_ident] bounded_linear_minus[OF bounded_linear_ident]]) simp

lemma integral_complex_of_real[simp]: $\text{integral}^L M (\lambda x. \text{complex\_of\_real } (f x)) = \text{of\_real } (\text{integral}^L M f)$

by (rule integral_bounded_linear'[OF bounded_linear_of_real bounded_linear_Re]) simp

lemma integral_cnj[simp]: $\text{integral}^L M (\lambda x. \text{cnj } (f x)) = \text{cnj } (\text{integral}^L M f)$

by (rule integral_bounded_linear'[OF bounded_linear_cnj bounded_linear_cnj]) simp

lemmas integral_divide[simp] =
integral_bounded_linear[OF bounded_linear_divide]

lemmas integral_Re[simp] =
integral_bounded_linear[OF bounded_linear_Re]

lemmas integral_Im[simp] =
integral_bounded_linear[OF bounded_linear_Im]

lemmas integral_of_real[simp] =
integral_bounded_linear[OF bounded_linear_of_real]

lemmas integralfst[simp] =
integral_bounded_linear[OF bounded_linearfst]

lemmas integralsnd[simp] =
integral_bounded_linear[OF bounded_linearsnd]

lemma *integral_norm_bound_enreal*:

integrable $M f \implies \text{norm}(\text{integral}^L M f) \leq (\int^+ x. \text{norm}(f x) \partial M)$

by (*metis* *has_bochner_integral_integrable* *has_bochner_integral_norm_bound*)

lemma *integrableI_sequence*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second\_countable\_topology}\}$

assumes $f[\text{measurable}]$: $f \in \text{borel\_measurable } M$

assumes s : $\bigwedge i. \text{simple\_bochner\_integrable } M (s i)$

assumes lim : $(\lambda i. \int^+ x. \text{norm}(f x - s i x) \partial M) \longrightarrow 0$ (**is** $?S \longrightarrow 0$)

shows *integrable* $M f$

proof –

let $?s = \lambda n. \text{simple\_bochner\_integral } M (s n)$

have $\exists x. ?s \longrightarrow x$

unfolding *convergent_eq_Cauchy*

proof (*rule metric_CauchyI*)

fix $e :: \text{real}$ **assume** $0 < e$

then have $0 < \text{ennreal}(e / 2)$ **by** *auto*

from *order_tendstoD(2)[OF lim this]*

obtain M **where** M : $\bigwedge n. M \leq n \implies ?S n < e / 2$

by (*auto simp: eventually_sequentially*)

show $\exists M. \forall m \geq M. \forall n \geq M. \text{dist} (?s m) (?s n) < e$

proof (*intro exI allI impI*)

fix $m n$ **assume** $m: M \leq m$ **and** $n: M \leq n$

have $?S n \neq \infty$

using $M[OF n]$ **by** *auto*

have $\text{norm} (?s n - ?s m) \leq ?S n + ?S m$

by (*intro simple_bochner_integral_bounded s f*)

also have $\dots < \text{ennreal}(e / 2) + e / 2$

by (*intro add_strict_mono M n m*)

also have $\dots = e$ **using** $\langle 0 < e \rangle$ **by** (*simp flip: ennreal_plus*)

finally show $\text{dist} (?s n) (?s m) < e$

using $\langle 0 < e \rangle$ **by** (*simp add: dist_norm ennreal_less_iff*)

qed

qed

then obtain x **where** $?s \longrightarrow x$..

show *?thesis*

by (*rule, rule*) *fact+*

qed

proposition *nn_integral_dominated_convergence_norm*:

fixes $u' :: \_ \Rightarrow \_ :: \{\text{real\_normed\_vector}, \text{second\_countable\_topology}\}$

assumes $[\text{measurable}]$:

$\bigwedge i. u i \in \text{borel\_measurable } M \ u' \in \text{borel\_measurable } M \ w \in \text{borel\_measurable}$

M

and *bound*: $\bigwedge j. \text{AE } x \text{ in } M. \text{norm}(u j x) \leq w x$

and w : $(\int^+ x. w x \partial M) < \infty$

and u' : $\text{AE } x \text{ in } M. (\lambda i. u i x) \longrightarrow u' x$

shows $(\lambda i. (\int^+ x. \text{norm}(u' x - u i x) \partial M)) \longrightarrow 0$

```

proof –
  have  $AE\ x\ in\ M. \forall j. norm\ (u\ j\ x) \leq w\ x$ 
    unfolding  $AE\_all\_countable$  by  $rule\ fact$ 
  with  $u'$  have  $bnd: AE\ x\ in\ M. \forall j. norm\ (u'\ x - u\ j\ x) \leq 2 * w\ x$ 
  proof ( $eventually\_elim, intro\ allI$ )
    fix  $i\ x$  assume  $(\lambda i. u\ i\ x) \longrightarrow u'\ x \ \forall j. norm\ (u\ j\ x) \leq w\ x \ \forall j. norm\ (u\ j\ x) \leq w\ x$ 
    then have  $norm\ (u'\ x) \leq w\ x \ \& \ norm\ (u\ i\ x) \leq w\ x$ 
      by ( $auto\ intro: LIMSEQ\_le\_const2\ tendsto\_norm$ )
    then have  $norm\ (u'\ x) + norm\ (u\ i\ x) \leq 2 * w\ x$ 
      by  $simp$ 
    also have  $norm\ (u'\ x - u\ i\ x) \leq norm\ (u'\ x) + norm\ (u\ i\ x)$ 
      by ( $rule\ norm\_triangle\_ineq4$ )
    finally ( $xtrans$ ) show  $norm\ (u'\ x - u\ i\ x) \leq 2 * w\ x$  .
  qed
  have  $w\_nonneg: AE\ x\ in\ M. 0 \leq w\ x$ 
    using  $bound[of\ 0]$  by ( $auto\ intro: order\_trans[OF\ norm\_ge\_zero]$ )

  have  $(\lambda i. (\int^+ x. norm\ (u'\ x - u\ i\ x)\ \partial M)) \longrightarrow (\int^+ x. 0\ \partial M)$ 
  proof ( $rule\ nn\_integral\_dominated\_convergence$ )
    show  $(\int^+ x. 2 * w\ x\ \partial M) < \infty$ 
      by ( $rule\ nn\_integral\_mult\_bounded\_inf[OF\ \_w, of\ 2]$ ) ( $insert\ w\_nonneg,$ 
 $auto\ simp: ennreal\_mult$ )
    show  $AE\ x\ in\ M. (\lambda i. ennreal\ (norm\ (u'\ x - u\ i\ x))) \longrightarrow 0$ 
      using  $u'$ 
    proof  $eventually\_elim$ 
      fix  $x$  assume  $(\lambda i. u\ i\ x) \longrightarrow u'\ x$ 
      from  $tendsto\_diff[OF\ tendsto\_const[of\ u'\ x]\ this]$ 
      show  $(\lambda i. ennreal\ (norm\ (u'\ x - u\ i\ x))) \longrightarrow 0$ 
        by ( $simp\ add: tendsto\_norm\_zero\_iff\ flip: ennreal\_0$ )
    qed
  qed ( $insert\ bnd\ w\_nonneg, auto$ )
  then show  $?thesis$  by  $simp$ 
qed

```

proposition $integrableI\_bounded$:

fixes $f :: 'a \Rightarrow 'b :: \{banach, second\_countable\_topology\}$
assumes $f[measurable]: f \in borel\_measurable\ M$ **and** $fin: (\int^+ x. norm\ (f\ x)\ \partial M) < \infty$

shows $integrable\ M\ f$

proof –

from $borel\_measurable\_implies\_sequence\_metric[OF\ f, of\ 0]$ **obtain** s **where**
 $s: \bigwedge i. simple\_function\ M\ (s\ i)$ **and**
 $pointwise: \bigwedge x. x \in space\ M \implies (\lambda i. s\ i\ x) \longrightarrow f\ x$ **and**
 $bound: \bigwedge i\ x. x \in space\ M \implies norm\ (s\ i\ x) \leq 2 * norm\ (f\ x)$
by $simp\ metis$

show $?thesis$

proof ($rule\ integrableI\_sequence$)

```

{ fix i
  have (∫+x. norm (s i x) ∂M) ≤ (∫+x. ennreal (2 * norm (f x)) ∂M)
    by (intro nn_integral_mono) (simp add: bound)
  also have ... = 2 * (∫+x. ennreal (norm (f x)) ∂M)
    by (simp add: ennreal_mult nn_integral_cmult)
  also have ... < top
    using fin by (simp add: ennreal_mult_less_top)
  finally have (∫+x. norm (s i x) ∂M) < ∞
    by simp }
note fin_s = this

show ∧i. simple_bochner_integrable M (s i)
  by (rule simple_bochner_integrableI_bounded) fact+

show (λi. ∫+x. ennreal (norm (f x - s i x)) ∂M) → 0
proof (rule nn_integral_dominated_convergence_norm)
  show ∧j. AE x in M. norm (s j x) ≤ 2 * norm (f x)
    using bound by auto
  show ∧i. s i ∈ borel_measurable M (λx. 2 * norm (f x)) ∈ borel_measurable
M
    using s by (auto intro: borel_measurable_simple_function)
  show (∫+x. ennreal (2 * norm (f x)) ∂M) < ∞
    using fin by (simp add: nn_integral_cmult ennreal_mult ennreal_mult_less_top)
  show AE x in M. (λi. s i x) → f x
    using pointwise by auto
qed fact
qed fact
qed

lemma integrableI_bounded_set:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  assumes [measurable]: A ∈ sets M f ∈ borel_measurable M
  assumes finite: emeasure M A < ∞
  and bnd: AE x in M. x ∈ A → norm (f x) ≤ B
  and null: AE x in M. x ∉ A → f x = 0
  shows integrable M f
proof (rule integrableI_bounded)
  { fix x :: 'b have norm x ≤ B ⇒ 0 ≤ B
    using norm_ge_zero[of x] by arith }
  with bnd null have (∫+x. ennreal (norm (f x)) ∂M) ≤ (∫+x. ennreal (max 0
B) * indicator A x ∂M)
    by (intro nn_integral_mono_AE) (auto split: split_indicator split_max)
  also have ... < ∞
    using finite by (subst nn_integral_cmult_indicator) (auto simp: ennreal_mult_less_top)
  finally show (∫+x. ennreal (norm (f x)) ∂M) < ∞ .
qed simp

lemma integrableI_bounded_set_indicator:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}

```

```

shows  $A \in \text{sets } M \implies f \in \text{borel\_measurable } M \implies$ 
 $\text{emeasure } M A < \infty \implies (\text{AE } x \text{ in } M. x \in A \longrightarrow \text{norm } (f x) \leq B) \implies$ 
 $\text{integrable } M (\lambda x. \text{indicator } A x *_R f x)$ 
by (rule integrableI_bounded_set[where A=A]) auto

```

lemma *integrableI_nonneg*:

```

fixes  $f :: 'a \Rightarrow \text{real}$ 
assumes  $f \in \text{borel\_measurable } M \text{ AE } x \text{ in } M. 0 \leq f x \text{ and } (\int^+ x. f x \, \partial M) < \infty$ 
shows  $\text{integrable } M f$ 

```

proof –

```

have  $(\int^+ x. \text{norm } (f x) \, \partial M) = (\int^+ x. f x \, \partial M)$ 
using assms by (intro nn_integral_cong_AE) auto
then show ?thesis
using assms by (intro integrableI_bounded) auto

```

qed

lemma *integrable_iff_bounded*:

```

fixes  $f :: 'a \Rightarrow 'b :: \{\text{banach, second\_countable\_topology}\}$ 
shows  $\text{integrable } M f \longleftrightarrow f \in \text{borel\_measurable } M \wedge (\int^+ x. \text{norm } (f x) \, \partial M) < \infty$ 
using integrableI_bounded[of f M] has_bochner_integral_implies_finite_norm[of M f]
unfolding integrable.simps has_bochner_integral.simps[abs_def] by auto

```

lemma (in *finite_measure*) *square_integrable_imp_integrable*:

```

fixes  $f :: 'a \Rightarrow 'b :: \{\text{second\_countable\_topology, banach, real\_normed\_div\_algebra}\}$ 
assumes [measurable]:  $f \in \text{borel\_measurable } M$ 
assumes  $\text{integrable } M (\lambda x. f x ^ 2)$ 
shows  $\text{integrable } M f$ 

```

proof –

```

have less_top:  $\text{emeasure } M (\text{space } M) < \text{top}$ 
using finite_emeasure_space by (meson top.not_eq_extremum)
have nn_integral  $M (\lambda x. \text{norm } (f x) ^ 2) \leq$ 
 $\text{nn\_integral } M (\lambda x. \text{norm } (f x) ^ 2) * \text{emeasure } M (\text{space } M)$ 
using Cauchy_Schwarz_nn_integral[of  $\lambda x. \text{norm } (f x) M \lambda_. 1$ ]
by (simp add: ennreal_power)
also have  $\dots < \infty$ 
using assms(2) less_top
by (subst (asm) integrable_iff_bounded) (auto simp: norm_power ennreal_mult_less_top)
finally have nn_integral  $M (\lambda x. \text{norm } (f x)) < \infty$ 
by (simp add: power_less_top_ennreal)
thus ?thesis
by (simp add: integrable_iff_bounded)

```

qed

lemma *integrable_bound*:

```

fixes  $f :: 'a \Rightarrow 'b :: \{\text{banach, second\_countable\_topology}\}$ 
and  $g :: 'a \Rightarrow 'c :: \{\text{banach, second\_countable\_topology}\}$ 
shows  $\text{integrable } M f \implies g \in \text{borel\_measurable } M \implies (\text{AE } x \text{ in } M. \text{norm } (g x))$ 

```

```

≤ norm (f x)) ==>
  integrable M g
unfolding integrable_iff_bounded
proof safe
  assume f ∈ borel_measurable M g ∈ borel_measurable M
  assume AE x in M. norm (g x) ≤ norm (f x)
  then have (∫+ x. ennreal (norm (g x)) ∂M) ≤ (∫+ x. ennreal (norm (f x))
∂M)
    by (intro nn_integral_mono_AE) auto
  also assume (∫+ x. ennreal (norm (f x)) ∂M) < ∞
  finally show (∫+ x. ennreal (norm (g x)) ∂M) < ∞ .
qed

```

```

lemma integrable_mult_indicator:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  shows A ∈ sets M ==> integrable M f ==> integrable M (λx. indicator A x *R f
x)
  by (rule integrable_bound[of M f]) (auto split: split_indicator)

```

```

lemma integrable_real_mult_indicator:
  fixes f :: 'a ⇒ real
  shows A ∈ sets M ==> integrable M f ==> integrable M (λx. f x * indicator A x)
  using integrable_mult_indicator[of A M f] by (simp add: mult_ac)

```

```

lemma integrable_abs[simp, intro]:
  fixes f :: 'a ⇒ real
  assumes [measurable]: integrable M f shows integrable M (λx. |f x|)
  using assms by (rule integrable_bound) auto

```

```

lemma integrable_norm[simp, intro]:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  assumes [measurable]: integrable M f shows integrable M (λx. norm (f x))
  using assms by (rule integrable_bound) auto

```

```

lemma integrable_norm_cancel:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  assumes [measurable]: integrable M (λx. norm (f x)) f ∈ borel_measurable M
shows integrable M f
  using assms by (rule integrable_bound) auto

```

```

lemma integrable_norm_iff:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  shows f ∈ borel_measurable M ==> integrable M (λx. norm (f x)) ↔ integrable
M f
  by (auto intro: integrable_norm_cancel)

```

```

lemma integrable_abs_cancel:
  fixes f :: 'a ⇒ real
  assumes [measurable]: integrable M (λx. |f x|) f ∈ borel_measurable M shows

```

```

integrable M f
  using assms by (rule integrable_bound) auto

lemma integrable_abs_iff:
  fixes f :: 'a  $\Rightarrow$  real
  shows  $f \in \text{borel\_measurable } M \implies \text{integrable } M (\lambda x. |f x|) \longleftrightarrow \text{integrable } M f$ 
  by (auto intro: integrable_abs_cancel)

lemma integrable_max[simp, intro]:
  fixes f :: 'a  $\Rightarrow$  real
  assumes fg[measurable]: integrable M f integrable M g
  shows integrable M ( $\lambda x. \max (f x) (g x)$ )
  using integrable_add[OF integrable_norm[OF fg(1)] integrable_norm[OF fg(2)]]
  by (rule integrable_bound) auto

lemma integrable_min[simp, intro]:
  fixes f :: 'a  $\Rightarrow$  real
  assumes fg[measurable]: integrable M f integrable M g
  shows integrable M ( $\lambda x. \min (f x) (g x)$ )
  using integrable_add[OF integrable_norm[OF fg(1)] integrable_norm[OF fg(2)]]
  by (rule integrable_bound) auto

lemma integral_minus_iff[simp]:
  integrable M ( $\lambda x. - f x :: 'a :: \{\text{banach, second\_countable\_topology}\}$ )  $\longleftrightarrow$  integrable
  M f
  unfolding integrable_iff_bounded
  by (auto)

lemma integrable_indicator_iff:
  integrable M (indicator A ::  $\_ \Rightarrow$  real)  $\longleftrightarrow$   $A \cap \text{space } M \in \text{sets } M \wedge \text{emeasure } M$ 
  ( $A \cap \text{space } M$ )  $< \infty$ 
  by (simp add: integrable_iff_bounded borel_measurable_indicator_iff ennreal_indicator
  nn_integral_indicator'
  cong: conj_cong)

lemma integral_indicator[simp]:  $\text{integral}^L M (\text{indicator } A) = \text{measure } M (A \cap$ 
   $\text{space } M)$ 
proof cases
  assume *:  $A \cap \text{space } M \in \text{sets } M \wedge \text{emeasure } M (A \cap \text{space } M) < \infty$ 
  have  $\text{integral}^L M (\text{indicator } A) = \text{integral}^L M (\text{indicator } (A \cap \text{space } M))$ 
  by (intro integral_cong) (auto split: split_indicator)
  also have  $\dots = \text{measure } M (A \cap \text{space } M)$ 
  using * by (intro has_bochner_integral_integral_eq has_bochner_integral_real_indicator)
  auto
  finally show ?thesis .
next
  assume *:  $\neg (A \cap \text{space } M \in \text{sets } M \wedge \text{emeasure } M (A \cap \text{space } M) < \infty)$ 
  have  $\text{integral}^L M (\text{indicator } A) = \text{integral}^L M (\text{indicator } (A \cap \text{space } M)) :: \_ \Rightarrow$ 
  real)

```

```

    by (intro integral_cong) (auto split: split_indicator)
  also have ... = 0
    using * by (subst not_integrable_integral_eq) (auto simp: integrable_indicator_iff)
  also have ... = measure M (A ∩ space M)
    using * by (auto simp: measure_def emeasure_notin_sets not_less_top_unique)
  finally show ?thesis .
qed

```

```

lemma integrable_discrete_difference:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  assumes X: countable X
  assumes null:  $\bigwedge x. x \in X \implies \text{emeasure } M \{x\} = 0$ 
  assumes sets:  $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$ 
  assumes eq:  $\bigwedge x. x \in \text{space } M \implies x \notin X \implies f x = g x$ 
  shows integrable M f  $\longleftrightarrow$  integrable M g
  unfolding integrable_iff_bounded
proof (rule conj_cong)
  { assume f ∈ borel_measurable M then have g ∈ borel_measurable M
    by (rule measurable_discrete_difference[where X=X]) (auto simp: assms) }
  moreover
  { assume g ∈ borel_measurable M then have f ∈ borel_measurable M
    by (rule measurable_discrete_difference[where X=X]) (auto simp: assms) }
  ultimately show f ∈ borel_measurable M  $\longleftrightarrow$  g ∈ borel_measurable M ..
next
  have AE x in M. x ∉ X
    by (rule AE_discrete_difference) fact+
  then have  $(\int^+ x. \text{norm } (f x) \partial M) = (\int^+ x. \text{norm } (g x) \partial M)$ 
    by (intro nn_integral_cong_AE) (auto simp: eq)
  then show  $(\int^+ x. \text{norm } (f x) \partial M) < \infty \longleftrightarrow (\int^+ x. \text{norm } (g x) \partial M) < \infty$ 
    by simp
qed

```

```

lemma integral_discrete_difference:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  assumes X: countable X
  assumes null:  $\bigwedge x. x \in X \implies \text{emeasure } M \{x\} = 0$ 
  assumes sets:  $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$ 
  assumes eq:  $\bigwedge x. x \in \text{space } M \implies x \notin X \implies f x = g x$ 
  shows  $\text{integral}^L M f = \text{integral}^L M g$ 
proof (rule integral_eq_cases)
  show eq: integrable M f  $\longleftrightarrow$  integrable M g
    by (rule integrable_discrete_difference[where X=X]) fact+

  assume f: integrable M f
  show  $\text{integral}^L M f = \text{integral}^L M g$ 
proof (rule integral_cong_AE)
  show f ∈ borel_measurable M g ∈ borel_measurable M
    using f eq by (auto intro: borel_measurable_integrable)

```



```

  have AE x in M. x  $\notin$  X
    by (rule AE_discrete_difference) fact+
  with AE_space show AE x in M. f x = g x
    by eventually_elim fact
qed
qed

```

```

lemma has_bochner_integral_discrete_difference:
  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  assumes X: countable X
  assumes null:  $\bigwedge x. x \in X \implies \text{emeasure } M \{x\} = 0$ 
  assumes sets:  $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$ 
  assumes eq:  $\bigwedge x. x \in \text{space } M \implies x \notin X \implies f x = g x$ 
  shows has_bochner_integral M f x  $\longleftrightarrow$  has_bochner_integral M g x
  using integrable_discrete_difference[of X M f g, OF assms]
  using integral_discrete_difference[of X M f g, OF assms]
  by (metis has_bochner_integral_iff)

```

```

lemma
  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology} and w :: 'a  $\Rightarrow$  real
  assumes f  $\in$  borel_measurable M  $\bigwedge i. s i \in$  borel_measurable M integrable M w
  assumes lim: AE x in M.  $(\lambda i. s i x) \longrightarrow f x$ 
  assumes bound:  $\bigwedge i. AE x in M. \text{norm } (s i x) \leq w x$ 
  shows integrable_dominated_convergence: integrable M f
    and integrable_dominated_convergence2:  $\bigwedge i. \text{integrable } M (s i)$ 
    and integral_dominated_convergence:  $(\lambda i. \text{integral}^L M (s i)) \longrightarrow \text{integral}^L M f$ 

```

```

proof -
  have w_nonneg: AE x in M.  $0 \leq w x$ 
    using bound[of 0] by eventually_elim (auto intro: norm_ge_zero order_trans)
  then have  $(\int^+ x. w x \partial M) = (\int^+ x. \text{norm } (w x) \partial M)$ 
    by (intro nn_integral_cong_AE) auto
  with  $\langle \text{integrable } M \rangle$  have w:  $w \in \text{borel\_measurable } M (\int^+ x. w x \partial M) < \infty$ 
    unfolding integrable_iff_bounded by auto

```

```

show int_s:  $\bigwedge i. \text{integrable } M (s i)$ 
  unfolding integrable_iff_bounded

```

```

proof
  fix i
  have  $(\int^+ x. \text{ennreal } (\text{norm } (s i x)) \partial M) \leq (\int^+ x. w x \partial M)$ 
    using bound[of i] w_nonneg by (intro nn_integral_mono_AE) auto
  with w show  $(\int^+ x. \text{ennreal } (\text{norm } (s i x)) \partial M) < \infty$  by auto
qed fact

```

```

have all_bound: AE x in M.  $\forall i. \text{norm } (s i x) \leq w x$ 
  using bound unfolding AE_all_countable by auto

```

```

show int_f: integrable M f
  unfolding integrable_iff_bounded

```

```

proof
  have  $(\int^+ x. \text{ennreal} (\text{norm} (f x)) \partial M) \leq (\int^+ x. w x \partial M)$ 
    using all_bound lim w_nonneg
  proof (intro nn_integral_mono_AE, eventually_elim)
    fix  $x$  assume  $\forall i. \text{norm} (s i x) \leq w x \ (\lambda i. s i x) \longrightarrow f x \ 0 \leq w x$ 
    then show  $\text{ennreal} (\text{norm} (f x)) \leq \text{ennreal} (w x)$ 
      by (intro LIMSEQ_le_const2[where X= $\lambda i. \text{ennreal} (\text{norm} (s i x))$ ]) (auto
intro: tendsto_intros)
    qed
  with  $w$  show  $(\int^+ x. \text{ennreal} (\text{norm} (f x)) \partial M) < \infty$  by auto
qed fact

  have  $(\lambda n. \text{ennreal} (\text{norm} (\text{integral}^L M (s n) - \text{integral}^L M f))) \longrightarrow \text{ennreal}$ 
   $0$  (is ?d  $\longrightarrow \text{ennreal } 0$ )
  proof (rule tendsto_sandwich)
    show eventually  $(\lambda n. \text{ennreal } 0 \leq ?d n)$  sequentially  $(\lambda \_. \text{ennreal } 0) \longrightarrow$ 
  ennreal 0 by auto
    show eventually  $(\lambda n. ?d n \leq (\int^+ x. \text{norm} (s n x - f x) \partial M))$  sequentially
  proof (intro always_eventually allI)
    fix  $n$ 
    have  $?d n = \text{norm} (\text{integral}^L M (\lambda x. s n x - f x))$ 
      using int_f int_s by simp
    also have  $\dots \leq (\int^+ x. \text{norm} (s n x - f x) \partial M)$ 
      by (intro int_f int_s integrable_diff integral_norm_bound_ennreal)
    finally show  $?d n \leq (\int^+ x. \text{norm} (s n x - f x) \partial M)$  .
  qed
  show  $(\lambda n. \int^+ x. \text{norm} (s n x - f x) \partial M) \longrightarrow \text{ennreal } 0$ 
    unfolding ennreal_0
    apply (subst norm_minus_commute)
  proof (rule nn_integral_dominated_convergence_norm[where w=w])
    show  $\bigwedge n. s n \in \text{borel\_measurable } M$ 
      using int_s unfolding integrable_iff_bounded by auto
    qed fact+
  qed
  then have  $(\lambda n. \text{integral}^L M (s n) - \text{integral}^L M f) \longrightarrow 0$ 
    by (simp add: tendsto_norm_zero_iff del: ennreal_0)
  from tendsto_add[OF this tendsto_const[of  $\text{integral}^L M f$ ]]
  show  $(\lambda i. \text{integral}^L M (s i)) \longrightarrow \text{integral}^L M f$  by simp
qed

context
  fixes  $s :: \text{real} \Rightarrow 'a \Rightarrow 'b :: \{\text{banach, second\_countable\_topology}\}$  and  $w :: 'a \Rightarrow$ 
real
    and  $f :: 'a \Rightarrow 'b$  and  $M$ 
  assumes  $f \in \text{borel\_measurable } M \bigwedge t. s t \in \text{borel\_measurable } M$  integrable M w
  assumes lim: AE x in M.  $((\lambda i. s i x) \longrightarrow f x)$  at_top
  assumes bound:  $\forall_F i$  in at_top. AE x in M.  $\text{norm} (s i x) \leq w x$ 
begin

```

lemma *integral_dominated_convergence_at_top*: $((\lambda t. \text{integral}^L M (s\ t)) \longrightarrow \text{integral}^L M f) \text{ at\_top}$

proof (rule *tendsto_at_topI_sequentially*)

fix $X :: \text{nat} \Rightarrow \text{real}$ **assume** $X: \text{filterlim } X \text{ at\_top sequentially}$

from *filterlim_iff* [THEN *iffD1*, OF *this*, rule_format, OF *bound*]

obtain N **where** $w: \bigwedge n. N \leq n \implies AE\ x \text{ in } M. \text{norm } (s\ (X\ n))\ x \leq w\ x$
by (auto simp: *eventually_sequentially*)

show $(\lambda n. \text{integral}^L M (s\ (X\ n))) \longrightarrow \text{integral}^L M f$

proof (rule *LIMSEQ_offset*, rule *integral_dominated_convergence*)

show $AE\ x \text{ in } M. \text{norm } (s\ (X\ (n + N)))\ x \leq w\ x$ **for** n

by (rule *w*) auto

show $AE\ x \text{ in } M. (\lambda n. s\ (X\ (n + N)))\ x \longrightarrow f\ x$

using *lim*

proof *eventually_elim*

fix x **assume** $((\lambda i. s\ i\ x) \longrightarrow f\ x) \text{ at\_top}$

then show $(\lambda n. s\ (X\ (n + N)))\ x \longrightarrow f\ x$

by (intro *LIMSEQ_ignore_initial_segment* *filterlim_compose* [OF X])

qed

qed fact+

qed

lemma *integrable_dominated_convergence_at_top*: *integrable* $M\ f$

proof –

from *bound* **obtain** N **where** $w: \bigwedge n. N \leq n \implies AE\ x \text{ in } M. \text{norm } (s\ n\ x) \leq w\ x$

by (auto simp: *eventually_at_top_linorder*)

show *?thesis*

proof (rule *integrable_dominated_convergence*)

show $AE\ x \text{ in } M. \text{norm } (s\ (N + i))\ x \leq w\ x$ **for** $i :: \text{nat}$

by (intro *w*) auto

show $AE\ x \text{ in } M. (\lambda i. s\ (N + \text{real } i))\ x \longrightarrow f\ x$

using *lim*

proof *eventually_elim*

fix x **assume** $((\lambda i. s\ i\ x) \longrightarrow f\ x) \text{ at\_top}$

then show $(\lambda n. s\ (N + n))\ x \longrightarrow f\ x$

by (rule *filterlim_compose*)

(auto intro!: *filterlim_tendsto_add_at_top* *filterlim_real_sequentially*)

qed

qed fact+

qed

end

lemma *integrable_mult_left_iff* [simp]:

fixes $f :: 'a \Rightarrow \text{real}$

shows *integrable* $M\ (\lambda x. c * f\ x) \longleftrightarrow c = 0 \vee \text{integrable } M\ f$

using *integrable_mult_left* [of $c\ M\ f$] *integrable_mult_left* [of $1\ /\ c\ M\ \lambda x. c * f\ x$]

by (*cases* $c = 0$) *auto*

lemma *integrable_mult_right_iff* [*simp*]:

fixes $f :: 'a \Rightarrow \text{real}$

shows $\text{integrable } M (\lambda x. f x * c) \longleftrightarrow c = 0 \vee \text{integrable } M f$

using *integrable_mult_left_iff* [*of* $M c f$] **by** (*simp add: mult.commute*)

lemma *integrableI_nn_integral_finite*:

assumes [*measurable*]: $f \in \text{borel\_measurable } M$

and *nonneg*: $AE x \text{ in } M. 0 \leq f x$

and *finite*: $(\int^+ x. f x \partial M) = \text{ennreal } x$

shows $\text{integrable } M f$

proof (*rule integrableI_bounded*)

have $(\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial M) = (\int^+ x. \text{ennreal } (f x) \partial M)$

using *nonneg* **by** (*intro nn_integral_cong_AE*) *auto*

with *finite* **show** $(\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial M) < \infty$

by *auto*

qed *simp*

lemma *integral_nonneg_AE*:

fixes $f :: 'a \Rightarrow \text{real}$

assumes *nonneg*: $AE x \text{ in } M. 0 \leq f x$

shows $0 \leq \text{integral}^L M f$

proof *cases*

assume $f: \text{integrable } M f$

then have [*measurable*]: $f \in M \rightarrow_M \text{borel}$

by *auto*

have $(\lambda x. \max 0 (f x)) \in M \rightarrow_M \text{borel} \wedge x. 0 \leq \max 0 (f x) \text{ integrable } M (\lambda x. \max 0 (f x))$

using f **by** *auto*

from this have $0 \leq \text{integral}^L M (\lambda x. \max 0 (f x))$

proof (*induction rule: borel_measurable_induct_real*)

case (*add f g*)

then have $\text{integrable } M f \text{ integrable } M g$

by (*auto intro!: integrable_bound[OF add.premis]*)

with *add* **show** *?case*

by (*simp add: nn_integral_add*)

next

case (*seq U*)

show *?case*

proof (*rule LIMSEQ_le_const*)

have $U\_le: x \in \text{space } M \implies U i x \leq \max 0 (f x) \text{ for } x i$

using *seq* **by** (*intro incseq_le*) (*auto simp: incseq_def le_fun_def*)

with *seq nonneg* **show** $(\lambda i. \text{integral}^L M (U i)) \longrightarrow \text{LINT } x | M. \max 0 (f$

$x)$

by (*intro integral_dominated_convergence*)

(*simp_all, fastforce*)

have $\text{integrable } M (U i) \text{ for } i$

using *seq.premis* **by** (*rule integrable_bound*) (*insert U_le seq, auto*)

```

    with seq show  $\exists N. \forall n \geq N. 0 \leq \text{integral}^L M (U n)$ 
    by auto
  qed
qed (auto)
also have  $\dots = \text{integral}^L M f$ 
  using nonneg by (auto intro!: integral_cong_AE)
  finally show ?thesis .
qed (simp add: not_integrable_integral_eq)

lemma integral_nonneg[simp]:
  fixes  $f :: 'a \Rightarrow \text{real}$ 
  shows  $(\bigwedge x. x \in \text{space } M \implies 0 \leq f x) \implies 0 \leq \text{integral}^L M f$ 
  by (intro integral_nonneg_AE) auto

proposition nn_integral_eq_integral:
  assumes  $f: \text{integrable } M f$ 
  assumes nonneg:  $\text{AE } x \text{ in } M. 0 \leq f x$ 
  shows  $(\int^+ x. f x \, \partial M) = \text{integral}^L M f$ 
proof -
  { fix  $f :: 'a \Rightarrow \text{real}$  assume  $f: f \in \text{borel\_measurable } M \bigwedge x. 0 \leq f x \text{ integrable } M f$ 
  then have  $(\int^+ x. f x \, \partial M) = \text{integral}^L M f$ 
  proof (induct rule: borel_measurable_induct_real)
    case (set A) then show ?case
    by (simp add: integrable_indicator_iff ennreal_indicator emeasure_eq_ennreal_measure)
  next
    case (mult f c) then show ?case
    by (auto simp add: nn_integral_cmult ennreal_mult integral_nonneg_AE)
  next
    case (add g f)
    then have  $\text{integrable } M f \text{ integrable } M g$ 
    by (auto intro!: integrable_bound[OF add.prem])
    with add show ?case
    by (simp add: nn_integral_add integral_nonneg_AE)
  next
    case (seq U)
    show ?case
    proof (rule LIMSEQ_unique)
      have  $U\_le\_f: x \in \text{space } M \implies U i x \leq f x \text{ for } x i$ 
      using seq by (intro incseq_le) (auto simp: incseq_def le_fun_def)
      have  $\text{int\_}U: \bigwedge i. \text{integrable } M (U i)$ 
      using seq  $f \, U\_le\_f$  by (intro integrable_bound[OF f(3)]) auto
      from  $U\_le\_f$  seq have  $(\lambda i. \text{integral}^L M (U i)) \longrightarrow \text{integral}^L M f$ 
      by (intro integral_dominated_convergence) auto
      then show  $(\lambda i. \text{ennreal } (\text{integral}^L M (U i))) \longrightarrow \text{ennreal } (\text{integral}^L M f)$ 
    end
  }
  using seq  $f \, \text{int\_}U$  by (simp add: f integral_nonneg_AE)
  have  $(\lambda i. \int^+ x. U i x \, \partial M) \longrightarrow \int^+ x. f x \, \partial M$ 
  using seq  $U\_le\_f f$ 

```

```

      by (intro nn_integral_dominated_convergence[where w=f]) (auto simp:
integrable_iff_bounded)
    then show  $(\lambda i. \int x. U \ i \ x \ \partial M) \longrightarrow \int^+ x. f \ x \ \partial M$ 
      using seq_int_U by simp
    qed
  qed }
  from this[of  $\lambda x. \max 0 \ (f \ x)$ ] assms have  $(\int^+ x. \max 0 \ (f \ x) \ \partial M) = \text{integral}^L$ 
  M  $(\lambda x. \max 0 \ (f \ x))$ 
    by simp
  also have  $\dots = \text{integral}^L \ M \ f$ 
    using assms by (auto intro!: integral_cong_AE simp: integral_nonneg_AE)
  also have  $(\int^+ x. \max 0 \ (f \ x) \ \partial M) = (\int^+ x. f \ x \ \partial M)$ 
    using assms by (auto intro!: nn_integral_cong_AE simp: max_def)
  finally show ?thesis .
qed

```

```

lemma nn_integral_eq_integrable:
  assumes f:  $f \in M \rightarrow_M \text{borel AE } x \text{ in } M. 0 \leq f \ x$  and  $0 \leq x$ 
  shows  $(\int^+ x. f \ x \ \partial M) = \text{ennreal } x \longleftrightarrow (\text{integrable } M \ f \wedge \text{integral}^L \ M \ f = x)$ 
proof (safe intro!: nn_integral_eq_integral assms)
  assume *:  $(\int^+ x. f \ x \ \partial M) = \text{ennreal } x$ 
  with integrableI_nn_integral_finite[OF f this] nn_integral_eq_integral[of M f,
OF _ f(2)]
  show  $\text{integrable } M \ f \ \text{integral}^L \ M \ f = x$ 
    by (simp_all add: * assms integral_nonneg_AE)
qed

```

```

lemma
  fixes f ::  $\_ \Rightarrow \_ \Rightarrow 'a :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
  assumes integrable[measurable]:  $\bigwedge i. \text{integrable } M \ (f \ i)$ 
  and summable:  $\text{AE } x \text{ in } M. \text{summable } (\lambda i. \text{norm } (f \ i \ x))$ 
  and sums:  $\text{summable } (\lambda i. (\int x. \text{norm } (f \ i \ x) \ \partial M))$ 
  shows integrable_suminf:  $\text{integrable } M \ (\lambda x. (\sum i. f \ i \ x))$  (is  $\text{integrable } M \ ?S$ )
    and sums_integral:  $(\lambda i. \text{integral}^L \ M \ (f \ i)) \text{ sums } (\int x. (\sum i. f \ i \ x) \ \partial M)$  (is  $?f$ 
sums ?x)
    and integral_suminf:  $(\int x. (\sum i. f \ i \ x) \ \partial M) = (\sum i. \text{integral}^L \ M \ (f \ i))$ 
    and summable_integral:  $\text{summable } (\lambda i. \text{integral}^L \ M \ (f \ i))$ 
proof -
  have 1:  $\text{integrable } M \ (\lambda x. \sum i. \text{norm } (f \ i \ x))$ 
  proof (rule integrableI_bounded)
    have  $(\int^+ x. \text{ennreal } (\text{norm } (\sum i. \text{norm } (f \ i \ x))) \ \partial M) = (\int^+ x. (\sum i. \text{ennreal}$ 
 $(\text{norm } (f \ i \ x))) \ \partial M)$ 
      apply (intro nn_integral_cong_AE)
      using summable
      apply eventually_elim
      apply (simp add: suminf_nonneg ennreal_suminf_neq_top)
      done
    also have  $\dots = (\sum i. \int^+ x. \text{norm } (f \ i \ x) \ \partial M)$ 
      by (intro nn_integral_suminf) auto

```

```

    also have ... = ( $\sum i. \text{ennreal } (\int x. \text{norm } (f i x) \partial M)$ )
      by (intro arg_cong[where f=suminf] ext nn_integral_eq_integral integrable_norm integrable) auto
    finally show ( $\int^+ x. \text{ennreal } (\text{norm } (\sum i. \text{norm } (f i x))) \partial M$ ) <  $\infty$ 
      by (simp add: sums_ennreal_suminf_neq_top_less_top[symmetric] integral_nonneg_AE)
  qed simp

  have 2: AE x in M. ( $\lambda n. \sum i < n. f i x$ )  $\longrightarrow$  ( $\sum i. f i x$ )
    using summable by eventually_elim (auto intro: summable_LIMSEQ summable_norm_cancel)

  have 3:  $\bigwedge j. \text{AE } x \text{ in } M. \text{norm } (\sum i < j. f i x) \leq (\sum i. \text{norm } (f i x))$ 
    using summable
  proof eventually_elim
    fix j x assume [simp]: summable ( $\lambda i. \text{norm } (f i x)$ )
    have  $\text{norm } (\sum i < j. f i x) \leq (\sum i < j. \text{norm } (f i x))$  by (rule norm_sum)
    also have ...  $\leq (\sum i. \text{norm } (f i x))$ 
      using sum_le_suminf[of  $\lambda i. \text{norm } (f i x)$ ] unfolding sums_iff by auto
    finally show  $\text{norm } (\sum i < j. f i x) \leq (\sum i. \text{norm } (f i x))$  by simp
  qed

  note ibl = integrable_dominated_convergence[OF _ _ 1 2 3]
  note int = integral_dominated_convergence[OF _ _ 1 2 3]

  show integrable M ?S
    by (rule ibl) measurable

  show ?f sums ?x unfolding sums_def
    using int by (simp add: integrable)
  then show ?x = suminf ?f summable ?f
    unfolding sums_iff by auto
  qed

  proposition integral_norm_bound [simp]:
    fixes f ::  $\_ \Rightarrow 'a :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
    shows  $\text{norm } (\text{integral}^L M f) \leq (\int x. \text{norm } (f x) \partial M)$ 
  proof (cases integrable M f)
    case True then show ?thesis
      using nn_integral_eq_integral[of M  $\lambda x. \text{norm } (f x)$ ] integral_norm_bound_ennreal[of M f]
      by (simp add: integral_nonneg_AE)
    next
    case False
      then have  $\text{norm } (\text{integral}^L M f) = 0$  by (simp add: not_integrable_integral_eq)
      moreover have  $(\int x. \text{norm } (f x) \partial M) \geq 0$  by auto
      ultimately show ?thesis by simp
  qed

  proposition integral_abs_bound [simp]:

```

fixes $f :: 'a \Rightarrow \text{real}$ **shows** $\text{abs } (\int x. f x \partial M) \leq (\int x. |f x| \partial M)$
using $\text{integral\_norm\_bound}$ [of $M f$] **by** auto

lemma $\text{integral\_eq\_nn\_integral}$:

assumes $[\text{measurable}]$: $f \in \text{borel\_measurable } M$

assumes nonneg : $\text{AE } x \text{ in } M. 0 \leq f x$

shows $\text{integral}^L M f = \text{enn2real } (\int^+ x. \text{ennreal } (f x) \partial M)$

proof cases

assume $*$: $(\int^+ x. \text{ennreal } (f x) \partial M) = \infty$

also have $(\int^+ x. \text{ennreal } (f x) \partial M) = (\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial M)$

using nonneg **by** $(\text{intro } \text{nn\_integral\_cong\_AE}) \text{ auto}$

finally have $\neg \text{integrable } M f$

by $(\text{auto simp: integrable\_iff\_bounded})$

then show $?thesis$

by $(\text{simp add: } * \text{ not\_integrable\_integral\_eq})$

next

assume $(\int^+ x. \text{ennreal } (f x) \partial M) \neq \infty$

then have $\text{integrable } M f$

by $(\text{cases } \int^+ x. \text{ennreal } (f x) \partial M \text{ rule: ennreal\_cases})$

$(\text{auto intro!: integrableI\_nn\_integral\_finite assms})$

from $\text{nn\_integral\_eq\_integral}$ [OF this] nonneg **show** $?thesis$

by $(\text{simp add: integral\_nonneg\_AE})$

qed

lemma $\text{enn2real\_nn\_integral\_eq\_integral}$:

assumes eq : $\text{AE } x \text{ in } M. f x = \text{ennreal } (g x)$ **and** nn : $\text{AE } x \text{ in } M. 0 \leq g x$

and fin : $(\int^+ x. f x \partial M) < \text{top}$

and $[\text{measurable}]$: $g \in M \rightarrow_M \text{borel}$

shows $\text{enn2real } (\int^+ x. f x \partial M) = (\int x. g x \partial M)$

proof $-$

have $\text{ennreal } (\text{enn2real } (\int^+ x. f x \partial M)) = (\int^+ x. f x \partial M)$

using fin **by** $(\text{intro } \text{ennreal\_enn2real}) \text{ auto}$

also have $\dots = (\int^+ x. g x \partial M)$

using eq **by** $(\text{rule } \text{nn\_integral\_cong\_AE})$

also have $\dots = (\int x. g x \partial M)$

proof $(\text{rule } \text{nn\_integral\_eq\_integral})$

show $\text{integrable } M g$

proof $(\text{rule } \text{integrableI\_bounded})$

have $(\int^+ x. \text{ennreal } (\text{norm } (g x)) \partial M) = (\int^+ x. f x \partial M)$

using eq nn **by** $(\text{auto intro!: nn\_integral\_cong\_AE elim!: eventually\_elim2})$

also note fin

finally show $(\int^+ x. \text{ennreal } (\text{norm } (g x)) \partial M) < \infty$

by simp

qed simp

qed fact

finally show $?thesis$

using nn **by** $(\text{simp add: integral\_nonneg\_AE})$

qed


```

lemma has_bochner_integral_nn_integral:
  assumes  $f \in \text{borel\_measurable } M \text{ AE } x \text{ in } M. 0 \leq f x 0 \leq x$ 
  assumes  $(\int^+ x. f x \partial M) = \text{ennreal } x$ 
  shows has_bochner_integral  $M f x$ 
  unfolding has_bochner_integral_iff
  using assms by (auto simp: assms integral_eq_nn_integral intro: integrableI_nn_integral_finite)

```

```

lemma integrableI_simple_bochner_integrable:
  fixes  $f :: 'a \Rightarrow 'b::\{\text{banach}, \text{second\_countable\_topology}\}$ 
  shows simple_bochner_integrable  $M f \implies \text{integrable } M f$ 
  by (intro integrableI_sequence[where  $s=\lambda_. f$ ] borel_measurable_simple_function)
    (auto simp: zero_ennreal_def[symmetric] simple_bochner_integrable.simps)

```

proposition integrable_induct[consumes 1, case_names base add lim, induct pred: integrable]:

```

  fixes  $f :: 'a \Rightarrow 'b::\{\text{banach}, \text{second\_countable\_topology}\}$ 
  assumes integrable  $M f$ 
  assumes base:  $\bigwedge A. c. A \in \text{sets } M \implies \text{emeasure } M A < \infty \implies P (\lambda x. \text{indicator } A x *_R c)$ 
  assumes add:  $\bigwedge f g. \text{integrable } M f \implies P f \implies \text{integrable } M g \implies P g \implies P (\lambda x. f x + g x)$ 
  assumes lim:  $\bigwedge f s. (\bigwedge i. \text{integrable } M (s i)) \implies (\bigwedge i. P (s i)) \implies$ 
     $(\bigwedge x. x \in \text{space } M \implies (\lambda i. s i x) \longrightarrow f x) \implies$ 
     $(\bigwedge i x. x \in \text{space } M \implies \text{norm } (s i x) \leq 2 * \text{norm } (f x)) \implies \text{integrable } M f \implies$ 
     $P f$ 
  shows  $P f$ 

```

proof –

```

  from <integrable  $M f$ > have  $f: f \in \text{borel\_measurable } M (\int^+ x. \text{norm } (f x) \partial M) < \infty$ 
  unfolding integrable_iff_bounded by auto
  from borel_measurable_implies_sequence_metric[OF  $f(1)$ ]
  obtain  $s$  where  $s: \bigwedge i. \text{simple\_function } M (s i) \bigwedge x. x \in \text{space } M \implies (\lambda i. s i x) \longrightarrow f x$ 
   $\bigwedge i x. x \in \text{space } M \implies \text{norm } (s i x) \leq 2 * \text{norm } (f x)$ 
  unfolding norm_conv_dist by metis

```

```

{ fix  $f A$ 
  have [simp]:  $P (\lambda x. 0)$ 
  using base[of {} undefined] by simp
  have  $(\bigwedge i::'b. i \in A \implies \text{integrable } M (f i::'a \Rightarrow 'b)) \implies$ 
     $(\bigwedge i. i \in A \implies P (f i)) \implies P (\lambda x. \sum_{i \in A} f i x)$ 
  by (induct A rule: infinite_finite_induct) (auto intro!: add) }
note sum = this

```

```

define  $s'$  where [abs_def]:  $s' i z = \text{indicator } (\text{space } M) z *_R s i z$  for  $i z$ 
then have  $s'_eq_s: \bigwedge i x. x \in \text{space } M \implies s' i x = s i x$ 
  by simp

```

```

have  $sf[\text{measurable}]: \bigwedge i. \text{simple\_function } M (s' i)$ 

```

```

    unfolding s'_def using s(1)
  by (intro simple_function_compose2[where h=(*_R)] simple_function_indicator)
auto

```

```

{ fix i
  have  $\bigwedge z. \{y. s' i z = y \wedge y \in s' i \text{'space } M \wedge y \neq 0 \wedge z \in \text{space } M\} =$ 
    (if  $z \in \text{space } M \wedge s' i z \neq 0$  then  $\{s' i z\}$  else  $\{\}$ )
    by (auto simp add: s'_def split: split_indicator)
  then have  $\bigwedge z. s' i = (\lambda z. \sum_{y \in s' i \text{'space } M - \{0\}} \text{indicator } \{x \in \text{space } M. s' i x = y\} z *_R y)$ 
    using sf by (auto simp: fun_eq_iff simple_function_def s'_def) }
note s'_eq = this

```

```

show P f
proof (rule lim)
  fix i

```

```

  have  $(\int^+ x. \text{norm } (s' i x) \partial M) \leq (\int^+ x. \text{ennreal } (2 * \text{norm } (f x)) \partial M)$ 
    using s by (intro nn_integral_mono) (auto simp: s'_eq_s)
  also have  $\dots < \infty$ 
    using f by (simp add: nn_integral_cmult ennreal_mult_less_top ennreal_mult)
  finally have sbi: simple_bochner_integrable M (s' i)
    using sf by (intro simple_bochner_integrableI_bounded) auto
  then show integrable M (s' i)
    by (rule integrableI_simple_bochner_integrable)

```

```

{ fix x assume  $x \in \text{space } M$  s' i x  $\neq 0$ 
  then have  $\text{emeasure } M \{y \in \text{space } M. s' i y = s' i x\} \leq \text{emeasure } M \{y \in$ 
     $\text{space } M. s' i y \neq 0\}$ 
    by (intro emeasure_mono) auto
  also have  $\dots < \infty$ 
    using sbi by (auto elim: simple_bochner_integrable.cases simp: less_top)
  finally have  $\text{emeasure } M \{y \in \text{space } M. s' i y = s' i x\} \neq \infty$  by simp }
then show P (s' i)
  by (subst s'_eq) (auto intro!: sum_base simp: less_top)

```

```

fix x assume  $x \in \text{space } M$  with s show  $(\lambda i. s' i x) \longrightarrow f x$ 
  by (simp add: s'_eq_s)
show  $\text{norm } (s' i x) \leq 2 * \text{norm } (f x)$ 
  using  $\langle x \in \text{space } M \rangle s$  by (simp add: s'_eq_s)

```

```

qed fact
qed

```

```

lemma integral_eq_zero_AE:
   $(AE x \text{ in } M. f x = 0) \implies \text{integral}^L M f = 0$ 
  using integral_cong_AE[of f M  $\lambda_. 0$ ]
  by (cases integrable M f) (simp_all add: not_integrable_integral_eq)

```

```

lemma integral_nonneg_eq_0_iff_AE:

```

```

fixes  $f :: \_ \Rightarrow \text{real}$ 
assumes  $f[\text{measurable}]$ :  $\text{integrable } M f$  and  $\text{nonneg}$ :  $\text{AE } x \text{ in } M. 0 \leq f x$ 
shows  $\text{integral}^L M f = 0 \longleftrightarrow (\text{AE } x \text{ in } M. f x = 0)$ 
proof
  assume  $\text{integral}^L M f = 0$ 
  then have  $\text{integral}^N M f = 0$ 
    using  $\text{nn\_integral\_eq\_integral}[OF f \text{ nonneg}]$  by  $\text{simp}$ 
  then have  $\text{AE } x \text{ in } M. \text{ennreal } (f x) \leq 0$ 
    by  $(\text{simp add: nn\_integral\_0\_iff\_AE})$ 
  with nonneg show  $\text{AE } x \text{ in } M. f x = 0$ 
    by  $\text{auto}$ 
qed  $(\text{auto simp add: integral\_eq\_zero\_AE})$ 

```

```

lemma integral_mono_AE:
  fixes  $f :: 'a \Rightarrow \text{real}$ 
  assumes  $\text{integrable } M f$   $\text{integrable } M g$   $\text{AE } x \text{ in } M. f x \leq g x$ 
  shows  $\text{integral}^L M f \leq \text{integral}^L M g$ 
proof -
  have  $0 \leq \text{integral}^L M (\lambda x. g x - f x)$ 
    using  $\text{assms}$  by  $(\text{intro integral\_nonneg\_AE integral\_diff assms})$   $\text{auto}$ 
  also have  $\dots = \text{integral}^L M g - \text{integral}^L M f$ 
    by  $(\text{intro integral\_diff assms})$ 
  finally show  $?thesis$  by  $\text{simp}$ 
qed

```

```

lemma integral_mono:
  fixes  $f :: 'a \Rightarrow \text{real}$ 
  shows  $\text{integrable } M f \implies \text{integrable } M g \implies (\bigwedge x. x \in \text{space } M \implies f x \leq g x)$ 
 $\implies$ 
   $\text{integral}^L M f \leq \text{integral}^L M g$ 
  by  $(\text{intro integral\_mono\_AE})$   $\text{auto}$ 

```

```

lemma integral_norm_bound_integral:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\{\text{banach}, \text{second\_countable\_topology}\}$ 
  assumes  $\text{integrable } M f$   $\text{integrable } M g$   $\bigwedge x. x \in \text{space } M \implies \text{norm}(f x) \leq g x$ 
  shows  $\text{norm} (\int x. f x \partial M) \leq (\int x. g x \partial M)$ 
proof -
  have  $\text{norm} (\int x. f x \partial M) \leq (\int x. \text{norm } (f x) \partial M)$ 
    by  $(\text{rule integral\_norm\_bound})$ 
  also have  $\dots \leq (\int x. g x \partial M)$ 
    using  $\text{assms integral\_norm integral\_mono}$  by  $\text{blast}$ 
  finally show  $?thesis$  .
qed

```

```

lemma integral_abs_bound_integral:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow \text{real}$ 
  assumes  $\text{integrable } M f$   $\text{integrable } M g$   $\bigwedge x. x \in \text{space } M \implies |f x| \leq g x$ 
  shows  $|\int x. f x \partial M| \leq (\int x. g x \partial M)$ 
  by  $(\text{metis integral\_norm\_bound\_integral assms real\_norm\_def})$ 

```

The next two statements are useful to bound Lebesgue integrals, as they avoid one integrability assumption. The price to pay is that the upper function has to be nonnegative, but this is often true and easy to check in computations.

lemma *integral_mono_AE'*:

fixes $f::\_ \Rightarrow \text{real}$

assumes *integrable* $M f AE x \text{ in } M. g x \leq f x AE x \text{ in } M. 0 \leq f x$

shows $(\int x. g x \partial M) \leq (\int x. f x \partial M)$

proof (*cases integrable* $M g$)

case *True*

show *?thesis* **by** (*rule integral_mono_AE*, *auto simp add: assms True*)

next

case *False*

then have $(\int x. g x \partial M) = 0$ **by** (*simp add: not_integrable_integral_eq*)

also have $\dots \leq (\int x. f x \partial M)$ **by** (*simp add: integral_nonneg_AE[OF assms(3)]*)

finally show *?thesis* **by** *simp*

qed

lemma *integral_mono'*:

fixes $f::\_ \Rightarrow \text{real}$

assumes *integrable* $M f \bigwedge x. x \in \text{space } M \Rightarrow g x \leq f x \bigwedge x. x \in \text{space } M \Rightarrow 0 \leq f x$

shows $(\int x. g x \partial M) \leq (\int x. f x \partial M)$

by (*rule integral_mono_AE'*, *insert assms, auto*)

lemma (*in finite_measure*) *integrable_measure*:

assumes *I: disjoint_family_on* $X I$ *countable* I

shows *integrable* (*count_space* I) $(\lambda i. \text{measure } M (X i))$

proof –

have $(\int^{+i. \text{measure } M (X i) \partial \text{count\_space } I} = (\int^{+i. \text{measure } M (if } X i \in \text{sets } M \text{ then } X i \text{ else } \{\}) \partial \text{count\_space } I)$

by (*auto intro!: nn_integral_cong_measure_notin_sets*)

also have $\dots = \text{measure } M (\bigcup i \in I. if } X i \in \text{sets } M \text{ then } X i \text{ else } \{\})$

using *I unfolding* *emeasure_eq_measure[symmetric]*

by (*subst emeasure_UN_countable*) (*auto simp: disjoint_family_on_def*)

finally show *?thesis*

by (*auto intro!: integrableI_bounded*)

qed

lemma *integrableI_real_bounded*:

assumes $f: f \in \text{borel\_measurable } M$ **and** $ae: AE x \text{ in } M. 0 \leq f x$ **and** $fin: \text{integral}^N M f < \infty$

shows *integrable* $M f$

proof (*rule integrableI_bounded*)

have $(\int^{+x. \text{ennreal } (norm (f x)) \partial M} = \int^{+x. \text{ennreal } (f x) \partial M$

using *ae* **by** (*auto intro: nn_integral_cong_AE*)

also note *fin*

finally show $(\int^{+x. \text{ennreal } (norm (f x)) \partial M} < \infty$.

qed *fact*

```

lemma nn_integral_nonneg_infinite:
  fixes f::'a  $\Rightarrow$  real
  assumes f  $\in$  borel_measurable M  $\neg$  integrable M f AE x in M. f x  $\geq$  0
  shows  $(\int^+ x. f x \partial M) = \infty$ 
using assms integrableI_real_bounded less_top by auto

lemma integral_real_bounded:
  assumes 0  $\leq$  r integralN M f  $\leq$  ennreal r
  shows integralL M f  $\leq$  r
proof cases
  assume [simp]: integrable M f

  have integralL M  $(\lambda x. \max 0 (f x)) =$  integralN M  $(\lambda x. \max 0 (f x))$ 
    by (intro nn_integral_eq_integral[symmetric]) auto
  also have ... = integralN M f
    by (intro nn_integral_cong) (simp add: max_def ennreal_neg)
  also have ...  $\leq$  r
    by fact
  finally have integralL M  $(\lambda x. \max 0 (f x)) \leq$  r
    using  $\langle 0 \leq r \rangle$  by simp

  moreover have integralL M f  $\leq$  integralL M  $(\lambda x. \max 0 (f x))$ 
    by (rule integral_mono_AE) auto
  ultimately show ?thesis
    by simp
next
  assume  $\neg$  integrable M f then show ?thesis
    using  $\langle 0 \leq r \rangle$  by (simp add: not_integrable_integral_eq)
qed

lemma integrable_MIN:
  fixes f:: _  $\Rightarrow$  _  $\Rightarrow$  real
  shows  $\llbracket$  finite I; I  $\neq$  {};  $\bigwedge i. i \in I \implies$  integrable M (f i)  $\rrbracket$ 
 $\implies$  integrable M  $(\lambda x. \text{MIN } i \in I. f i x)$ 
by (induct rule: finite_ne_induct) simp+

lemma integrable_MAX:
  fixes f:: _  $\Rightarrow$  _  $\Rightarrow$  real
  shows  $\llbracket$  finite I; I  $\neq$  {};  $\bigwedge i. i \in I \implies$  integrable M (f i)  $\rrbracket$ 
 $\implies$  integrable M  $(\lambda x. \text{MAX } i \in I. f i x)$ 
by (induct rule: finite_ne_induct) simp+

theorem integral_Markov_inequality:
  assumes [measurable]: integrable M u and AE x in M. 0  $\leq$  u x 0 < (c::real)
  shows (emeasure M)  $\{x \in \text{space } M. u x \geq c\} \leq (1/c) * (\int x. u x \partial M)$ 
proof -
  have  $(\int^+ x. \text{ennreal}(u x) * \text{indicator } (\text{space } M) x \partial M) \leq (\int^+ x. u x \partial M)$ 
    by (rule nn_integral_mono_AE, auto simp add:  $\langle c > 0 \rangle$  less_eq_real_def)

```

also have ... = $(\int x. u x \partial M)$
 by (rule nn_integral_eq_integral, auto simp add: assms)
 finally have *: $(\int^+ x. ennreal(u x) * indicator (space M) x \partial M) \leq (\int x. u x \partial M)$
 by simp

have $\{x \in space M. u x \geq c\} = \{x \in space M. ennreal(1/c) * u x \geq 1\} \cap (space M)$
 using $\langle c > 0 \rangle$ by (auto simp: ennreal_mult'[symmetric])
 then have $emeasure M \{x \in space M. u x \geq c\} = emeasure M (\{x \in space M. ennreal(1/c) * u x \geq 1\})$
 by simp
 also have $\dots \leq ennreal(1/c) * (\int^+ x. ennreal(u x) * indicator (space M) x \partial M)$
 by (rule nn_integral_Markov_inequality) (auto simp add: assms)
 also have $\dots \leq ennreal(1/c) * (\int x. u x \partial M)$
 by (rule mult_left_mono) (use * $\langle c > 0 \rangle$ in auto)
 finally show ?thesis
 using $\langle 0 < c \rangle$ by (simp add: ennreal_mult'[symmetric])
 qed

theorem integral_Markov_inequality_measure:

assumes [measurable]: integrable M u and $A \in sets M$ and $AE x in M. 0 \leq u x < (c::real)$
 shows $measure M \{x \in space M. u x \geq c\} \leq (\int x. u x \partial M) / c$
 proof -
 have $le: emeasure M \{x \in space M. u x \geq c\} \leq ennreal ((1/c) * (\int x. u x \partial M))$
 by (rule integral_Markov_inequality) (use assms in auto)
 also have $\dots < top$
 by auto
 finally have $ennreal (measure M \{x \in space M. u x \geq c\}) = emeasure M \{x \in space M. u x \geq c\}$
 by (intro emeasure_eq_ennreal_measure [symmetric]) auto
 also note le
 finally show ?thesis
 by (subst (asm) ennreal_le_iff)
 (auto intro!: divide_nonneg_pos Bochner_Integration.integral_nonneg AE assms)
 qed

theorem (in finite_measure) second_moment_method:

assumes [measurable]: $f \in M \rightarrow_M borel$
 assumes integrable M $(\lambda x. f x ^ 2)$
 defines $\mu \equiv lebesgue\_integral M f$
 assumes $a > 0$
 shows $measure M \{x \in space M. |f x| \geq a\} \leq lebesgue\_integral M (\lambda x. f x ^ 2) / a^2$
 proof -
 have integrable: integrable M f
 using assms by (blast dest: square_integrable_imp_integrable)

```

have {x∈space M. |f x| ≥ a} = {x∈space M. f x ^ 2 ≥ a^2}
  using ‹a > 0› by (simp flip: abs_le_square_iff)
hence measure M {x∈space M. |f x| ≥ a} = measure M {x∈space M. f x ^ 2 ≥
a^2}
  by simp
also have ... ≤ lebesgue_integral M (λx. f x ^ 2) / a^2
  using assms by (intro integral_Markov_inequality_measure) auto
finally show ?thesis .
qed

```

```

lemma integral_ineq_eq_0_then_AE:
  fixes f::_ ⇒ real
  assumes AE x in M. f x ≤ g x integrable M f integrable M g
    (∫ x. f x ∂M) = (∫ x. g x ∂M)
  shows AE x in M. f x = g x
proof -
  define h where h = (λx. g x - f x)
  have AE x in M. h x = 0
    apply (subst integral_nonneg_eq_0_iff_AE[symmetric])
    unfolding h_def using assms by auto
  then show ?thesis unfolding h_def by auto
qed

```

```

lemma not_AE_zero_int_E:
  fixes f::'a ⇒ real
  assumes AE x in M. f x ≥ 0 (∫ x. f x ∂M) > 0
    and [measurable]: f ∈ borel_measurable M
  shows ∃ A e. A ∈ sets M ∧ e>0 ∧ emeasure M A > 0 ∧ (∀ x ∈ A. f x ≥ e)
proof (rule not_AE_zero_E, auto simp add: assms)
  assume *: AE x in M. f x = 0
  have (∫ x. f x ∂M) = (∫ x. 0 ∂M) by (rule integral_cong_AE, auto simp add:
*)
  then have (∫ x. f x ∂M) = 0 by simp
  then show False using assms(2) by simp
qed

```

```

proposition tendsto_L1_int:
  fixes u :: _ ⇒ _ ⇒ 'b::{banach, second_countable_topology}
  assumes [measurable]: ∧ n. integrable M (u n) integrable M f
    and ((λn. (∫+x. norm(u n x - f x) ∂M)) ⟶ 0) F
  shows ((λn. (∫ x. u n x ∂M)) ⟶ (∫ x. f x ∂M)) F
proof -
  have ((λn. norm((∫ x. u n x ∂M) - (∫ x. f x ∂M))) ⟶ (0::ennreal)) F
  proof (rule tendsto_sandwich[of λ_. 0, where ?h = λn. (∫+x. norm(u n x -
f x) ∂M)], auto simp add: assms)
  {
    fix n
    have (∫ x. u n x ∂M) - (∫ x. f x ∂M) = (∫ x. u n x - f x ∂M)
      apply (rule Bochner_Integration.integral_diff[symmetric]) using assms by

```

```

auto
  then have norm(( $\int x. u \, n \, x \, \partial M$ ) - ( $\int x. f \, x \, \partial M$ )) = norm ( $\int x. u \, n \, x - f \, x \, \partial M$ )
    by auto
  also have ...  $\leq (\int x. \text{norm}(u \, n \, x - f \, x) \, \partial M)$ 
    by (rule integral_norm_bound)
  finally have ennreal(norm(( $\int x. u \, n \, x \, \partial M$ ) - ( $\int x. f \, x \, \partial M$ )))  $\leq (\int x. \text{norm}(u \, n \, x - f \, x) \, \partial M)$ 
    by simp
  also have ... = ( $\int^+ x. \text{norm}(u \, n \, x - f \, x) \, \partial M$ )
    apply (rule nn_integral_eq_integral[symmetric]) using assms by auto
  finally have norm(( $\int x. u \, n \, x \, \partial M$ ) - ( $\int x. f \, x \, \partial M$ ))  $\leq (\int^+ x. \text{norm}(u \, n \, x - f \, x) \, \partial M)$  by simp
}
then show eventually ( $\lambda n. \text{norm}((\int x. u \, n \, x \, \partial M) - (\int x. f \, x \, \partial M)) \leq (\int^+ x. \text{norm}(u \, n \, x - f \, x) \, \partial M)$ ) F
  by auto
qed
then have (( $\lambda n. \text{norm}((\int x. u \, n \, x \, \partial M) - (\int x. f \, x \, \partial M))$ )  $\longrightarrow 0$ ) F
  by (simp flip: ennreal_0)
then have (( $\lambda n. ((\int x. u \, n \, x \, \partial M) - (\int x. f \, x \, \partial M))$ )  $\longrightarrow 0$ ) F using tendsto_norm_zero_iff by blast
then show ?thesis using Lim_null by auto
qed

```

The next lemma asserts that, if a sequence of functions converges in L^1 , then it admits a subsequence that converges almost everywhere.

```

proposition tendsto_L1_AE_subseq:
  fixes u :: nat  $\Rightarrow$  'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  assumes [measurable]:  $\bigwedge n. \text{integrable } M \, (u \, n)$ 
    and ( $\lambda n. (\int x. \text{norm}(u \, n \, x) \, \partial M)$ )  $\longrightarrow 0$ 
  shows  $\exists r::\text{nat} \Rightarrow \text{nat}. \text{strict\_mono } r \wedge (AE \, x \, \text{in } M. (\lambda n. u \, (r \, n) \, x) \longrightarrow 0)$ 
proof -
{
  fix k
  have eventually ( $\lambda n. (\int x. \text{norm}(u \, n \, x) \, \partial M) < (1/2)^\wedge k$ ) sequentially
    using order_tendstoD(2)[OF assms(2)] by auto
  with eventually_elim2[OF eventually_gt_at_top[of k] this]
  have  $\exists n > k. (\int x. \text{norm}(u \, n \, x) \, \partial M) < (1/2)^\wedge k$ 
    by (metis eventually_False_sequentially)
}
then have  $\exists r. \forall n. \text{True} \wedge (r \, (\text{Suc } n) > r \, n \wedge (\int x. \text{norm}(u \, (r \, (\text{Suc } n)) \, x) \, \partial M) < (1/2)^\wedge (r \, n))$ 
  by (intro dependent_nat_choice, auto)
then obtain r0 where r0: strict_mono r0  $\bigwedge n. (\int x. \text{norm}(u \, (r0 \, (\text{Suc } n)) \, x) \, \partial M) < (1/2)^\wedge (r0 \, n)$ 
  by (auto simp: strict_mono_Suc_iff)
define r where r = ( $\lambda n. r0 \, (n+1)$ )
have strict_mono r unfolding r_def using r0(1) by (simp add: strict_mono_Suc_iff)

```



```

have I: ( $\int^+ x. \text{norm}(u (r\ n) x) \partial M$ ) < ennreal((1/2)n) for n
proof -
  have r0 n ≥ n using ‹strict_mono r0› by (simp add: seq_suble)
  have (1/2::real)(r0 n) ≤ (1/2)n by (rule power_decreasing[OF ‹r0 n ≥ n›], auto)
  then have ( $\int x. \text{norm}(u (r0 (Suc\ n)) x) \partial M$ ) < (1/2)n
    using r0(2) less_le_trans by blast
  then have ( $\int x. \text{norm}(u (r\ n) x) \partial M$ ) < (1/2)n
    unfolding r_def by auto
  moreover have ( $\int^+ x. \text{norm}(u (r\ n) x) \partial M$ ) = ( $\int x. \text{norm}(u (r\ n) x) \partial M$ )
    by (rule nn_integral_eq_integral, auto simp add: integrable_norm[OF assms(1)[of r n]])
  ultimately show ?thesis by (auto intro: ennreal_lessI)
qed

have AE x in M. limsup (λn. ennreal (norm(u (r n) x))) ≤ 0
proof (rule AE_upper_bound_inf_ennreal)
  fix e::real assume e > 0
  define A where A = (λn. {x ∈ space M. norm(u (r n) x) ≥ e})
  have A_meas [measurable]: ∧n. A n ∈ sets M unfolding A_def by auto
  have A_bound: emeasure M (A n) < (1/e) * ennreal((1/2)n) for n
  proof -
    have *: indicator (A n) x ≤ (1/e) * ennreal(norm(u (r n) x)) for x
      apply (cases x ∈ A n) unfolding A_def using ‹0 < e› by (auto simp:
ennreal_mult[symmetric])
    have emeasure M (A n) = ( $\int^+ x. \text{indicator} (A\ n) x \partial M$ ) by auto
    also have ... ≤ ( $\int^+ x. (1/e) * \text{ennreal}(\text{norm}(u (r\ n) x)) \partial M$ )
      apply (rule nn_integral_mono) using * by auto
    also have ... = (1/e) * ( $\int^+ x. \text{norm}(u (r\ n) x) \partial M$ )
      apply (rule nn_integral_cmult) using ‹e > 0› by auto
    also have ... < (1/e) * ennreal((1/2)n)
      using I[of n] ‹e > 0› by (intro ennreal_mult_strict_left_mono) auto
    finally show ?thesis by simp
  qed
  have A_fin: emeasure M (A n) < ∞ for n
    using ‹e > 0› A_bound[of n]
    by (auto simp add: ennreal_mult_less_top[symmetric])

  have A_sum: summable (λn. measure M (A n))
  proof (rule summable_comparison_test'[of λn. (1/e) * (1/2)n 0])
    have summable (λn. (1/2::real)n) by (simp add: summable_geometric)
    then show summable (λn. (1/e) * (1/2)n) using summable_mult by blast
  fix n::nat assume n ≥ 0
  have norm(measure M (A n)) = measure M (A n) by simp
  also have ... = enn2real(emeasure M (A n)) unfolding measure_def by
simp
  also have ... < enn2real((1/e) * (1/2)n)
    using A_bound[of n] ‹emeasure M (A n) < ∞› ‹0 < e›
    by (auto simp: emeasure_eq_ennreal_measure ennreal_mult[symmetric])

```

```

ennreal_less_iff)
  also have ... = (1/e) * (1/2)^n
  using ‹0 < e› by auto
  finally show norm(measure M (A n)) ≤ (1/e) * (1/2)^n by simp
qed

have AE x in M. eventually (λn. x ∈ space M - A n) sequentially
  by (rule borel_cantelli_AE1[OF A_meas A_fin A_sum])
moreover
{
  fix x assume eventually (λn. x ∈ space M - A n) sequentially
  moreover have norm(u (r n) x) ≤ ennreal e if x ∈ space M - A n for n
    using that unfolding A_def by (auto intro: ennreal_leI)
  ultimately have eventually (λn. norm(u (r n) x) ≤ ennreal e) sequentially
    by (simp add: eventually_mono)
  then have limsup (λn. ennreal (norm(u (r n) x))) ≤ e
    by (simp add: Limsup_bounded)
}
ultimately show AE x in M. limsup (λn. ennreal (norm(u (r n) x))) ≤ 0 +
ennreal e by auto
qed
moreover
{
  fix x assume limsup (λn. ennreal (norm(u (r n) x))) ≤ 0
  moreover then have liminf (λn. ennreal (norm(u (r n) x))) ≤ 0
    by (rule order_trans[rotated]) (auto intro: Liminf_le_Limsup)
  ultimately have (λn. ennreal (norm(u (r n) x))) ⟶ 0
    using tendsto_Limsup[of sequentially λn. ennreal (norm(u (r n) x))] by auto
  then have (λn. norm(u (r n) x)) ⟶ 0
    by (simp flip: ennreal_0)
  then have (λn. u (r n) x) ⟶ 0
    by (simp add: tendsto_norm_zero_iff)
}
ultimately have AE x in M. (λn. u (r n) x) ⟶ 0 by auto
then show ?thesis using ‹strict_mono r› by auto
qed

```

6.10.1 Restricted measure spaces

```

lemma integrable_restrict_space:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  assumes Ω[simp]: Ω ∩ space M ∈ sets M
  shows integrable (restrict_space M Ω) f ⟷ integrable M (λx. indicator Ω x
*_R f x)
unfolding integrable_iff_bounded
  borel_measurable_restrict_space_iff[OF Ω]
  nn_integral_restrict_space[OF Ω]
  by (simp add: ac_simps ennreal_indicator ennreal_mult)

```

```

lemma integral_restrict_space:
  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  assumes  $\Omega[simp]$ :  $\Omega \cap \text{space } M \in \text{sets } M$ 
  shows  $\text{integral}^L (\text{restrict\_space } M \ \Omega) f = \text{integral}^L M (\lambda x. \text{indicator } \Omega \ x *_R f \ x)$ 
proof (rule integral_eq_cases)
  assume integrable (restrict_space M  $\Omega$ ) f
  then show ?thesis
  proof induct
    case (base A c) then show ?case
      by (simp add: indicator_inter_arith[symmetric] sets_restrict_space_iff
        emeasure_restrict_space Int_absorb1 measure_restrict_space)
  next
    case (add g f) then show ?case
      by (simp add: scaleR_add_right integrable_restrict_space)
  next
    case (lim f s)
    show ?case
    proof (rule LIMSEQ_unique)
      show  $(\lambda i. \text{integral}^L (\text{restrict\_space } M \ \Omega) (s \ i)) \longrightarrow \text{integral}^L (\text{restrict\_space } M \ \Omega) f$ 
        using lim by (intro integral_dominated_convergence[where w= $\lambda x. 2 * \text{norm } (f \ x)$ ]) simp_all

      show  $(\lambda i. \text{integral}^L (\text{restrict\_space } M \ \Omega) (s \ i)) \longrightarrow (\int x. \text{indicator } \Omega \ x *_R f \ x \ \partial M)$ 
        unfolding lim
        using lim
        by (intro integral_dominated_convergence[where w= $\lambda x. 2 * \text{norm } (\text{indicator } \Omega \ x *_R f \ x)$ ])
          (auto simp add: space_restrict_space integrable_restrict_space simp del:
            norm_scaleR split: split_indicator)
    qed
  qed
qed (simp add: integrable_restrict_space)

```

```

lemma integral_empty:
  assumes  $\text{space } M = \{\}$ 
  shows  $\text{integral}^L M f = 0$ 
proof -
  have  $(\int x. f \ x \ \partial M) = (\int x. 0 \ \partial M)$ 
    by (rule integral_cong)(simp_all add: assms)
  thus ?thesis by simp
qed

```

6.10.2 Measure spaces with an associated density

```

lemma integrable_density:

```

```

fixes  $f :: 'a \Rightarrow 'b :: \{\text{banach, second\_countable\_topology}\}$  and  $g :: 'a \Rightarrow \text{real}$ 
assumes  $[measurable]: f \in \text{borel\_measurable } M \ g \in \text{borel\_measurable } M$ 
and  $nn: AE \ x \text{ in } M. \ 0 \leq g \ x$ 
shows  $\text{integrable } (\text{density } M \ g) \ f \longleftrightarrow \text{integrable } M \ (\lambda x. \ g \ x *_{\mathbb{R}} f \ x)$ 
unfolding  $\text{integrable\_iff\_bounded}$  using  $nn$ 
apply  $(\text{simp add: nn\_integral\_density\_less\_top}[symmetric])$ 
apply  $(\text{intro arg\_cong2}[\text{where } f=(=)] \text{ refl nn\_integral\_cong\_AE})$ 
apply  $(\text{auto simp: ennreal\_mult})$ 
done

```

lemma *integral_density*:

```

fixes  $f :: 'a \Rightarrow 'b :: \{\text{banach, second\_countable\_topology}\}$  and  $g :: 'a \Rightarrow \text{real}$ 
assumes  $f: f \in \text{borel\_measurable } M$ 
and  $g[measurable]: g \in \text{borel\_measurable } M \ AE \ x \text{ in } M. \ 0 \leq g \ x$ 
shows  $\text{integral}^L (\text{density } M \ g) \ f = \text{integral}^L M \ (\lambda x. \ g \ x *_{\mathbb{R}} f \ x)$ 
proof  $(\text{rule integral\_eq\_cases})$ 
assume  $\text{integrable } (\text{density } M \ g) \ f$ 
then show ?thesis
proof induct
case  $(\text{base } A \ c)$ 
then have  $[measurable]: A \in \text{sets } M$  by auto

have  $\text{int}: \text{integrable } M \ (\lambda x. \ g \ x * \text{indicator } A \ x)$ 
using  $g \text{ base integrable\_density}[of \text{indicator } A :: 'a \Rightarrow \text{real } M \ g]$  by simp
then have  $\text{integral}^L M \ (\lambda x. \ g \ x * \text{indicator } A \ x) = (\int^+ x. \text{ennreal } (g \ x * \text{indicator } A \ x) \ dM)$ 
using  $g$  by  $(\text{subst nn\_integral\_eq\_integral}) \text{ auto}$ 
also have  $\dots = (\int^+ x. \text{ennreal } (g \ x) * \text{indicator } A \ x \ dM)$ 
by  $(\text{intro nn\_integral\_cong}) (\text{auto split: split\_indicator})$ 
also have  $\dots = \text{emeasure } (\text{density } M \ g) \ A$ 
by  $(\text{rule emeasure\_density}[symmetric]) \text{ auto}$ 
also have  $\dots = \text{ennreal } (\text{measure } (\text{density } M \ g) \ A)$ 
using base by  $(\text{auto intro: emeasure\_eq\_ennreal\_measure})$ 
also have  $\dots = \text{integral}^L (\text{density } M \ g) (\text{indicator } A)$ 
using base by simp
finally show ?case
using base g
apply  $(\text{simp add: int integral\_nonneg\_AE})$ 
apply  $(\text{subst (asm) ennreal\_inj})$ 
apply  $(\text{auto intro!: integral\_nonneg\_AE})$ 
done
next
case  $(\text{add } f \ h)$ 
then have  $[measurable]: f \in \text{borel\_measurable } M \ h \in \text{borel\_measurable } M$ 
by  $(\text{auto dest!: borel\_measurable\_integrable})$ 
from add g show ?case
by  $(\text{simp add: scaleR\_add\_right integrable\_density})$ 
next
case  $(\text{lim } f \ s)$ 

```

```

have [measurable]: f ∈ borel_measurable M ∧ i. s i ∈ borel_measurable M
  using lim(1,5)[THEN borel_measurable_integrable] by auto

show ?case
proof (rule LIMSEQ_unique)
  show (λi. integralL M (λx. g x *R s i x)) ⟶ integralL M (λx. g x *R f
x)
  proof (rule integral_dominated_convergence)
    show integrable M (λx. 2 * norm (g x *R f x))
    by (intro integrable_mult_right integrable_norm integrable_density[THEN
iffD1] lim g) auto
    show AE x in M. (λi. g x *R s i x) ⟶ g x *R f x
    using lim(3) by (auto intro!: tendsto_scaleR AE_I2[of M])
    show ∧i. AE x in M. norm (g x *R s i x) ≤ 2 * norm (g x *R f x)
    using lim(4) g by (auto intro!: AE_I2[of M] mult_left_mono simp:
field_simps)
    qed auto
    show (λi. integralL M (λx. g x *R s i x)) ⟶ integralL (density M g) f
    unfolding lim(2)[symmetric]
    by (rule integral_dominated_convergence[where w=λx. 2 * norm (f x)])
    (insert lim(3-5), auto)
  qed
qed
qed (simp add: f g integrable_density)

```

lemma

```

fixes g :: 'a ⇒ real
assumes f ∈ borel_measurable M AE x in M. 0 ≤ f x g ∈ borel_measurable M
shows integral_real_density: integralL (density M f) g = (∫ x. f x * g x ∂M)
  and integrable_real_density: integrable (density M f) g ⟷ integrable M (λx.
f x * g x)
using assms integral_density[of g M f] integrable_density[of g M f] by auto

```

lemma has_bochner_integral_density:

```

fixes f :: 'a ⇒ 'b::{banach, second_countable_topology} and g :: 'a ⇒ real
shows f ∈ borel_measurable M ⟹ g ∈ borel_measurable M ⟹ (AE x in M.
0 ≤ g x) ⟹
  has_bochner_integral M (λx. g x *R f x) x ⟹ has_bochner_integral (density
M g) f x
by (simp add: has_bochner_integral_iff integrable_density integral_density)

```

6.10.3 Distributions

lemma integrable_distr_eq:

```

fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
assumes [measurable]: g ∈ measurable M N f ∈ borel_measurable N
shows integrable (distr M N g) f ⟷ integrable M (λx. f (g x))
unfolding integrable_iff_bounded by (simp_all add: nn_integral_distr)

```

```

lemma integrable_distr:
  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  shows  $T \in \text{measurable } M \ M' \implies \text{integrable } (\text{distr } M \ M' \ T) \ f \implies \text{integrable } M$ 
  ( $\lambda x. f \ (T \ x)$ )
  by (subst integrable_distr_eq[symmetric, where g=T])
  (auto dest: borel_measurable_integrable)

```

```

lemma integral_distr:
  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  assumes g[measurable]:  $g \in \text{measurable } M \ N$  and f:  $f \in \text{borel\_measurable } N$ 
  shows  $\text{integral}^L (\text{distr } M \ N \ g) \ f = \text{integral}^L M \ (\lambda x. f \ (g \ x))$ 
proof (rule integral_eq_cases)
  assume integrable (distr M N g) f
  then show ?thesis
  proof induct
    case (base A c)
    then have [measurable]:  $A \in \text{sets } N$  by auto
    from base have int:  $\text{integrable } (\text{distr } M \ N \ g) \ (\lambda a. \text{indicator } A \ a \ *_R \ c)$ 
    by (intro integrable_indicator)

    have  $\text{integral}^L (\text{distr } M \ N \ g) \ (\lambda a. \text{indicator } A \ a \ *_R \ c) = \text{measure } (\text{distr } M \ N$ 
  g)  $A \ *_R \ c$ 
    using base by auto
    also have  $\dots = \text{measure } M \ (g - ' A \cap \text{space } M) \ *_R \ c$ 
    by (subst measure_distr) auto
    also have  $\dots = \text{integral}^L M \ (\lambda a. \text{indicator } (g - ' A \cap \text{space } M) \ a \ *_R \ c)$ 
    using base by (auto simp: emeasure_distr)
    also have  $\dots = \text{integral}^L M \ (\lambda a. \text{indicator } A \ (g \ a) \ *_R \ c)$ 
    using int base by (intro integral_cong_AE) (auto simp: emeasure_distr split:
  split_indicator)
    finally show ?case .
  next
    case (add f h)
    then have [measurable]:  $f \in \text{borel\_measurable } N \ h \in \text{borel\_measurable } N$ 
    by (auto dest!: borel_measurable_integrable)
    from add g show ?case
    by (simp add: scaleR_add_right integrable_distr_eq)
  next
    case (lim f s)
    have [measurable]:  $f \in \text{borel\_measurable } N \ \bigwedge i. s \ i \in \text{borel\_measurable } N$ 
    using  $\text{lim}(1,5)[\text{THEN borel\_measurable\_integrable}]$  by auto

    show ?case
  proof (rule LIMSEQ_unique)
    show  $(\lambda i. \text{integral}^L M \ (\lambda x. s \ i \ (g \ x))) \longrightarrow \text{integral}^L M \ (\lambda x. f \ (g \ x))$ 
    proof (rule integral_dominated_convergence)
      show  $\text{integrable } M \ (\lambda x. 2 \ * \ \text{norm } (f \ (g \ x)))$ 
      using lim by (auto simp: integrable_distr_eq)
      show  $\text{AE } x \text{ in } M. (\lambda i. s \ i \ (g \ x)) \longrightarrow f \ (g \ x)$ 

```

```

      using lim(3) g[THEN measurable_space] by auto
    show  $\bigwedge i. \text{AE } x \text{ in } M. \text{norm } (s \ i \ (g \ x)) \leq 2 * \text{norm } (f \ (g \ x))$ 
      using lim(4) g[THEN measurable_space] by auto
  qed auto
  show  $(\lambda i. \text{integral}^L M (\lambda x. s \ i \ (g \ x))) \longrightarrow \text{integral}^L (\text{distr } M \ N \ g) f$ 
    unfolding lim(2)[symmetric]
    by (rule integral_dominated_convergence[where w= $\lambda x. 2 * \text{norm } (f \ x)$ ])
      (insert lim(3-5), auto)
  qed
  qed
  qed (simp add: f g integrable_distr_eq)

lemma has_bochner_integral_distr:
  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  shows  $f \in \text{borel\_measurable } N \implies g \in \text{measurable } M \ N \implies$ 
     $\text{has\_bochner\_integral } M (\lambda x. f \ (g \ x)) \ x \implies \text{has\_bochner\_integral } (\text{distr } M \ N$ 
  g) f x
  by (simp add: has_bochner_integral_iff integrable_distr_eq integral_distr)

```

6.10.4 Lebesgue integration on count_space

```

lemma integrable_count_space:
  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  shows  $\text{finite } X \implies \text{integrable } (\text{count\_space } X) f$ 
  by (auto simp: nn_integral_count_space integrable_iff_bounded)

lemma measure_count_space[simp]:
   $B \subseteq A \implies \text{finite } B \implies \text{measure } (\text{count\_space } A) B = \text{card } B$ 
  unfolding measure_def by (subst emeasure_count_space) auto

lemma lebesgue_integral_count_space_finite_support:
  assumes f:  $\text{finite } \{a \in A. f \ a \neq 0\}$ 
  shows  $(\int x. f \ x \ \partial \text{count\_space } A) = (\sum a \mid a \in A \wedge f \ a \neq 0. f \ a)$ 
proof -
  have eq:  $\bigwedge x. x \in A \implies (\sum a \mid x = a \wedge a \in A \wedge f \ a \neq 0. f \ a) = (\sum x \in \{x\}. f \ x)$ 
    by (intro sum.mono_neutral_cong_left) auto

  have  $(\int x. f \ x \ \partial \text{count\_space } A) = (\int x. (\sum a \mid a \in A \wedge f \ a \neq 0. \text{indicator } \{a\}$ 
  x *R f a)  $\partial \text{count\_space } A)$ 
    by (intro integral_cong_refl) (simp add: f eq)
  also have  $\dots = (\sum a \mid a \in A \wedge f \ a \neq 0. \text{measure } (\text{count\_space } A) \{a\} *_{\text{R}} f \ a)$ 
    by (subst integral_sum) (auto intro!: sum.cong)
  finally show ?thesis
    by auto
  qed

lemma lebesgue_integral_count_space_finite:  $\text{finite } A \implies (\int x. f \ x \ \partial \text{count\_space } A) = (\sum a \in A. f \ a)$ 
  by (subst lebesgue_integral_count_space_finite_support)

```

(auto intro!: sum.mono_neutral_cong_left)

lemma *integrable_count_space_nat_iff*:
fixes $f :: \text{nat} \Rightarrow \_ :: \{\text{banach}, \text{second\_countable\_topology}\}$
shows $\text{integrable} (\text{count\_space } \text{UNIV}) f \longleftrightarrow \text{summable } (\lambda x. \text{norm } (f x))$
by (auto simp add: integrable_iff_bounded nn_integral_count_space_nat_en-
nreal_suminf_neq_top
intro: summable_suminf_not_top)

lemma *sums_integral_count_space_nat*:
fixes $f :: \text{nat} \Rightarrow \_ :: \{\text{banach}, \text{second\_countable\_topology}\}$
assumes $*$: $\text{integrable} (\text{count\_space } \text{UNIV}) f$
shows $f \text{ sums } (\text{integral}^L (\text{count\_space } \text{UNIV}) f)$
proof –
let $?f = \lambda n i. \text{indicator } \{n\} i *_{\mathbb{R}} f i$
have $f': \bigwedge n i. ?f n i = \text{indicator } \{n\} i *_{\mathbb{R}} f n$
by (auto simp: fun_eq_iff split: split_indicator)

have $(\lambda i. \int n. ?f i n \partial \text{count\_space } \text{UNIV}) \text{ sums } \int n. (\sum i. ?f i n) \partial \text{count\_space } \text{UNIV}$
proof (rule sums_integral)
show $\bigwedge i. \text{integrable} (\text{count\_space } \text{UNIV}) (?f i)$
using $*$ **by** (intro integrable_mult_indicator) auto
show $\text{AE } n \text{ in } \text{count\_space } \text{UNIV}. \text{summable } (\lambda i. \text{norm } (?f i n))$
using summable_finite[of $\{n\} \lambda i. \text{norm } (?f i n)$ **for** n] **by** simp
show $\text{summable } (\lambda i. \int n. \text{norm } (?f i n) \partial \text{count\_space } \text{UNIV})$
using $*$ **by** (subst f') (simp add: integrable_count_space_nat_iff)
qed
also have $(\int n. (\sum i. ?f i n) \partial \text{count\_space } \text{UNIV}) = (\int n. f n \partial \text{count\_space } \text{UNIV})$
using suminf_finite[of $\{n\} \lambda i. ?f i n$ **for** n] **by** (auto intro!: integral_cong)
also have $(\lambda i. \int n. ?f i n \partial \text{count\_space } \text{UNIV}) = f$
by (subst f') simp
finally show ?thesis .
qed

lemma *integral_count_space_nat*:
fixes $f :: \text{nat} \Rightarrow \_ :: \{\text{banach}, \text{second\_countable\_topology}\}$
shows $\text{integrable} (\text{count\_space } \text{UNIV}) f \implies \text{integral}^L (\text{count\_space } \text{UNIV}) f = (\sum x. f x)$
using sums_integral_count_space_nat **by** (rule sums_unique)

lemma *integrable_bij_count_space*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second\_countable\_topology}\}$
assumes g : $\text{bij\_betw } g \text{ } A \text{ } B$
shows $\text{integrable} (\text{count\_space } A) (\lambda x. f (g x)) \longleftrightarrow \text{integrable} (\text{count\_space } B) f$
unfolding integrable_iff_bounded **by** (subst nn_integral_bij_count_space[OF g]) auto


```

lemma integral_bij_count_space:
  fixes  $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
  assumes  $g: \text{bij\_betw } g \ A \ B$ 
  shows  $\text{integral}^L (\text{count\_space } A) (\lambda x. f (g \ x)) = \text{integral}^L (\text{count\_space } B) f$ 
  using  $g[\text{THEN } \text{bij\_betw\_imp\_funcset}]$ 
  apply (subst distr_bij_count_space[OF  $g$ , symmetric])
  apply (intro integral_distr[symmetric])
  apply auto
  done

lemma has_bochner_integral_count_space_nat:
  fixes  $f :: \text{nat} \Rightarrow \_ :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
  shows  $\text{has\_bochner\_integral } (\text{count\_space } \text{UNIV}) \ f \ x \implies f \ \text{sums } x$ 
  unfolding has_bochner_integral_iff by (auto intro!: sums_integral_count_space_nat)

```

6.10.5 Point measure

```

lemma lebesgue_integral_point_measure_finite:
  fixes  $g :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
  shows  $\text{finite } A \implies (\bigwedge a. a \in A \implies 0 \leq f \ a) \implies$ 
     $\text{integral}^L (\text{point\_measure } A \ f) \ g = (\sum_{a \in A}. f \ a \ *_{\mathbb{R}} \ g \ a)$ 
  by (simp add: lebesgue_integral_count_space_finite AE_count_space integral_density
point_measure_def)

```

```

proposition integrable_point_measure_finite:
  fixes  $g :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second\_countable\_topology}\}$  and  $f :: 'a \Rightarrow \text{real}$ 
  shows  $\text{finite } A \implies \text{integrable } (\text{point\_measure } A \ f) \ g$ 
  unfolding point_measure_def
  apply (subst density_cong[where  $f' = \lambda x. \text{ennreal } (\max \ 0 \ (f \ x))$ ])
  apply (auto split: split_max simp: ennreal_neg)
  apply (subst integrable_density)
  apply (auto simp: AE_count_space integrable_count_space)
  done

```

6.10.6 Lebesgue integration on null measure

```

lemma has_bochner_integral_null_measure_iff[iff]:
   $\text{has\_bochner\_integral } (\text{null\_measure } M) \ f \ 0 \longleftrightarrow f \in \text{borel\_measurable } M$ 
  by (auto simp add: has_bochner_integral.simps simple_bochner_integral_def[abs_def]
intro!: exI[of  $\_ \ \lambda n \ x. \ 0$ ] simple_bochner_integrable.intros)

```

```

lemma integrable_null_measure_iff[iff]:  $\text{integrable } (\text{null\_measure } M) \ f \longleftrightarrow f \in$ 
borel\_measurable  $M$ 
  by (auto simp add: integrable.simps)

```

```

lemma integral_null_measure[simp]:  $\text{integral}^L (\text{null\_measure } M) \ f = 0$ 
  by (cases integrable  $(\text{null\_measure } M) \ f$ )
  (auto simp add: not_integrable_integral_eq has_bochner_integral_integral_eq)

```

6.10.7 Legacy lemmas for the real-valued Lebesgue integral

theorem *real_lebesgue_integral_def*:

assumes $f[\text{measurable}]$: *integrable* M f

shows $\text{integral}^L M f = \text{enn2real} (\int^+ x. f x \partial M) - \text{enn2real} (\int^+ x. \text{ennreal} (- f x) \partial M)$

proof –

have $\text{integral}^L M f = \text{integral}^L M (\lambda x. \max 0 (f x) - \max 0 (- f x))$

by (*auto intro!*: *arg_cong*[**where** $f = \text{integral}^L M$])

also have $\dots = \text{integral}^L M (\lambda x. \max 0 (f x)) - \text{integral}^L M (\lambda x. \max 0 (- f x))$

by (*intro integral_diff integrable_max integrable_minus integrable_zero* f)

also have $\text{integral}^L M (\lambda x. \max 0 (f x)) = \text{enn2real} (\int^+ x. \text{ennreal} (f x) \partial M)$

by (*subst integral_eq_nn_integral*) (*auto intro!*: *arg_cong*[**where** $f = \text{enn2real}$]
nn_integral_cong simp: max_def ennreal_neg)

also have $\text{integral}^L M (\lambda x. \max 0 (- f x)) = \text{enn2real} (\int^+ x. \text{ennreal} (- f x) \partial M)$

by (*subst integral_eq_nn_integral*) (*auto intro!*: *arg_cong*[**where** $f = \text{enn2real}$]
nn_integral_cong simp: max_def ennreal_neg)

finally show *?thesis* .

qed

theorem *real_integrable_def*:

$\text{integrable } M f \longleftrightarrow f \in \text{borel\_measurable } M \wedge$

$(\int^+ x. \text{ennreal} (f x) \partial M) \neq \infty \wedge (\int^+ x. \text{ennreal} (- f x) \partial M) \neq \infty$

unfolding *integrable_iff_bounded*

proof (*safe del: notI*)

assume *: $(\int^+ x. \text{ennreal} (\text{norm} (f x)) \partial M) < \infty$

have $(\int^+ x. \text{ennreal} (f x) \partial M) \leq (\int^+ x. \text{ennreal} (\text{norm} (f x)) \partial M)$

by (*intro nn_integral_mono*) *auto*

also note *

finally show $(\int^+ x. \text{ennreal} (f x) \partial M) \neq \infty$

by *simp*

have $(\int^+ x. \text{ennreal} (- f x) \partial M) \leq (\int^+ x. \text{ennreal} (\text{norm} (f x)) \partial M)$

by (*intro nn_integral_mono*) *auto*

also note *

finally show $(\int^+ x. \text{ennreal} (- f x) \partial M) \neq \infty$

by *simp*

next

assume [*measurable*]: $f \in \text{borel\_measurable } M$

assume *fin*: $(\int^+ x. \text{ennreal} (f x) \partial M) \neq \infty \wedge (\int^+ x. \text{ennreal} (- f x) \partial M) \neq \infty$

have $(\int^+ x. \text{norm} (f x) \partial M) = (\int^+ x. \text{ennreal} (f x) + \text{ennreal} (- f x) \partial M)$

by (*intro nn_integral_cong*) (*auto simp: abs_real_def ennreal_neg*)

also have $\dots = (\int^+ x. \text{ennreal} (f x) \partial M) + (\int^+ x. \text{ennreal} (- f x) \partial M)$

by (*intro nn_integral_add*) *auto*

also have $\dots < \infty$

using *fin* **by** (*auto simp: less_top*)

finally show $(\int^+ x. \text{norm} (f x) \partial M) < \infty$.

qed

```

lemma integrableD[dest]:
  assumes integrable M f
  shows  $f \in \text{borel\_measurable } M \ (\int^+ x. \text{ennreal } (f x) \partial M) \neq \infty \ (\int^+ x. \text{ennreal } (-f x) \partial M) \neq \infty$ 
  using assms unfolding real_integrable_def by auto

```

```

lemma integrableE:
  assumes integrable M f
  obtains  $r \ q$  where  $0 \leq r \ 0 \leq q$ 
     $(\int^+ x. \text{ennreal } (f x) \partial M) = \text{ennreal } r$ 
     $(\int^+ x. \text{ennreal } (-f x) \partial M) = \text{ennreal } q$ 
     $f \in \text{borel\_measurable } M \ \text{integral}^L M f = r - q$ 
  using assms unfolding real_integrable_def real_lebesgue_integral_def[OF assms]
  by (cases rule: ennreal2_cases[of  $(\int^+ x. \text{ennreal } (-f x) \partial M) \ (\int^+ x. \text{ennreal } (f x) \partial M)$ ]) auto

```

```

lemma integral_monotone_convergence_nonneg:
  fixes  $f :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$ 
  assumes  $i: \bigwedge i. \text{integrable } M (f i)$  and  $\text{mono}: \text{AE } x \text{ in } M. \text{mono } (\lambda n. f n x)$ 
    and  $\text{pos}: \bigwedge i. \text{AE } x \text{ in } M. 0 \leq f i x$ 
    and  $\text{lim}: \text{AE } x \text{ in } M. (\lambda i. f i x) \longrightarrow u x$ 
    and  $\text{ilim}: (\lambda i. \text{integral}^L M (f i)) \longrightarrow x$ 
    and  $u: u \in \text{borel\_measurable } M$ 
  shows  $\text{integrable } M u$ 
    and  $\text{integral}^L M u = x$ 
proof -
  have  $nn: \text{AE } x \text{ in } M. \forall i. 0 \leq f i x$ 
    using pos unfolding AE_all_countable by auto
  with  $\text{lim}$  have  $u\_nn: \text{AE } x \text{ in } M. 0 \leq u x$ 
    by eventually_elim (auto intro: LIMSEQ_le_const)
  have  $[simp]: 0 \leq x$ 
    by (intro LIMSEQ_le_const[OF ilim] allI exI impI integral_nonneg_AE pos)
  have  $(\int^+ x. \text{ennreal } (u x) \partial M) = (\text{SUP } n. (\int^+ x. \text{ennreal } (f n x) \partial M))$ 
  proof (subst nn_integral_monotone_convergence_SUP_AE[symmetric])
    fix  $i$ 
    from  $\text{mono } nn$  show  $\text{AE } x \text{ in } M. \text{ennreal } (f i x) \leq \text{ennreal } (f (\text{Suc } i) x)$ 
      by eventually_elim (auto simp: mono_def)
    show  $(\lambda x. \text{ennreal } (f i x)) \in \text{borel\_measurable } M$ 
      using  $i$  by auto
  next
    show  $(\int^+ x. \text{ennreal } (u x) \partial M) = \int^+ x. (\text{SUP } i. \text{ennreal } (f i x)) \partial M$ 
      apply (rule nn_integral_cong_AE)
      using  $\text{lim } \text{mono } nn \ u\_nn$ 
      apply eventually_elim
      apply (simp add: LIMSEQ_unique[OF LIMSEQ_SUP] incseq_def)
      done
  qed
  also have  $\dots = \text{ennreal } x$ 
    using  $\text{mono } i \ nn$  unfolding nn_integral_eq_integral[OF  $i \ pos$ ]

```

by (subst LIMSEQ_unique[OF LIMSEQ_SUP]) (auto simp: mono_def integral_nonneg_AE pos intro!: integral_mono_AE ilim)
 finally have $(\int^+ x. \text{ennreal } (u \ x) \ \partial M) = \text{ennreal } x$.
 moreover have $(\int^+ x. \text{ennreal } (- \ u \ x) \ \partial M) = 0$
 using $u \ u\_nn$ by (subst nn_integral_0_iff_AE) (auto simp add: ennreal_neg)
 ultimately show $\text{integrable } M \ u \ \text{integral}^L \ M \ u = x$
 by (auto simp: real_integrable_def real_lebesgue_integral_def u)
 qed

lemma

fixes $f :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$
 assumes $f: \bigwedge i. \text{integrable } M \ (f \ i)$ and $\text{mono}: AE \ x \ \text{in } M. \text{mono } (\lambda n. f \ n \ x)$
 and $\text{lim}: AE \ x \ \text{in } M. (\lambda i. f \ i \ x) \longrightarrow u \ x$
 and $\text{ilim}: (\lambda i. \text{integral}^L \ M \ (f \ i)) \longrightarrow x$
 and $u: u \in \text{borel\_measurable } M$
 shows $\text{integrable\_monotone\_convergence}: \text{integrable } M \ u$
 and $\text{integral\_monotone\_convergence}: \text{integral}^L \ M \ u = x$
 and $\text{has\_bochner\_integral\_monotone\_convergence}: \text{has\_bochner\_integral } M \ u$

x

proof -

have 1: $\bigwedge i. \text{integrable } M \ (\lambda x. f \ i \ x - f \ 0 \ x)$
 using f by auto
 have 2: $AE \ x \ \text{in } M. \text{mono } (\lambda n. f \ n \ x - f \ 0 \ x)$
 using mono by (auto simp: mono_def le_fun_def)
 have 3: $\bigwedge n. AE \ x \ \text{in } M. 0 \leq f \ n \ x - f \ 0 \ x$
 using mono by (auto simp: field_simps mono_def le_fun_def)
 have 4: $AE \ x \ \text{in } M. (\lambda i. f \ i \ x - f \ 0 \ x) \longrightarrow u \ x - f \ 0 \ x$
 using lim by (auto intro!: tendsto_diff)
 have 5: $(\lambda i. (\int x. f \ i \ x - f \ 0 \ x \ \partial M)) \longrightarrow x - \text{integral}^L \ M \ (f \ 0)$
 using $f \ \text{ilim}$ by (auto intro!: tendsto_diff)
 have 6: $(\lambda x. u \ x - f \ 0 \ x) \in \text{borel\_measurable } M$
 using $f[0]$ by auto
 note $\text{diff} = \text{integral\_monotone\_convergence\_nonneg}[OF \ 1 \ 2 \ 3 \ 4 \ 5 \ 6]$
 have $\text{integrable } M \ (\lambda x. (u \ x - f \ 0 \ x) + f \ 0 \ x)$
 using $\text{diff}(1) \ f$ by (rule integrable_add)
 with $\text{diff}(2) \ f$ show $\text{integrable } M \ u \ \text{integral}^L \ M \ u = x$
 by auto
 then show $\text{has\_bochner\_integral } M \ u \ x$
 by (metis $\text{has\_bochner\_integral\_integrable}$)

qed

lemma $\text{integral\_norm\_eq\_0\_iff}$:

fixes $f :: 'a \Rightarrow 'b::\{\text{banach, second\_countable\_topology}\}$
 assumes $f[\text{measurable}]: \text{integrable } M \ f$
 shows $(\int x. \text{norm } (f \ x) \ \partial M) = 0 \iff \text{emeasure } M \ \{x \in \text{space } M. f \ x \neq 0\} = 0$

proof -

have $(\int^+ x. \text{norm } (f \ x) \ \partial M) = (\int x. \text{norm } (f \ x) \ \partial M)$
 using f by (intro nn_integral_eq_integral integrable_norm) auto
 then have $(\int x. \text{norm } (f \ x) \ \partial M) = 0 \iff (\int^+ x. \text{norm } (f \ x) \ \partial M) = 0$

```

    by simp
  also have ...  $\longleftrightarrow$   $\text{emeasure } M \{x \in \text{space } M. \text{ennreal } (\text{norm } (f x)) \neq 0\} = 0$ 
    by (intro nn_integral_0_iff) auto
  finally show ?thesis
    by simp
qed

lemma integral_0_iff:
  fixes  $f :: 'a \Rightarrow \text{real}$ 
  shows  $\text{integrable } M f \implies (\int x. |f x| \partial M) = 0 \longleftrightarrow \text{emeasure } M \{x \in \text{space } M. f x \neq 0\} = 0$ 
  using integral_norm_eq_0_iff[of  $M f$ ] by simp

lemma (in finite_measure) integrable_const[intro!, simp]:  $\text{integrable } M (\lambda x. a)$ 
  using integrable_indicator[of  $\text{space } M M a$ ] by (simp cong: integrable_cong add: less_top[symmetric])

lemma lebesgue_integral_const[simp]:
  fixes  $a :: 'a :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
  shows  $(\int x. a \partial M) = \text{measure } M (\text{space } M) *_R a$ 
proof -
  { assume  $\text{emeasure } M (\text{space } M) = \infty$   $a \neq 0$ 
    then have ?thesis
      by (auto simp add: not_integrable_integral_eq ennreal_mult_less_top measure_def integrable_iff_bounded) }
  moreover
  { assume  $a = 0$  then have ?thesis by simp }
  moreover
  { assume  $\text{emeasure } M (\text{space } M) \neq \infty$ 
    interpret finite_measure M
    proof qed fact
    have  $(\int x. a \partial M) = (\int x. \text{indicator } (\text{space } M) x *_R a \partial M)$ 
      by (intro integral_cong) auto
    also have ... =  $\text{measure } M (\text{space } M) *_R a$ 
      by (simp add: less_top[symmetric])
    finally have ?thesis . }
  ultimately show ?thesis by blast
qed

lemma (in finite_measure) integrable_const_bound:
  fixes  $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
  shows  $\text{AE } x \text{ in } M. \text{norm } (f x) \leq B \implies f \in \text{borel\_measurable } M \implies \text{integrable } M f$ 
  apply (rule integrable_bound[OF integrable_const[of  $B$ ], of  $f$ ])
  apply assumption
  apply (cases  $0 \leq B$ )
  apply auto
  done

```

lemma (in *finite_measure*) *integral_bounded_eq_bound_then_AE*:
assumes *AE x in M. f x ≤ (c::real)*
*integrable M f (∫ x. f x ∂M) = c * measure M (space M)*
shows *AE x in M. f x = c*
apply (rule *integral_ineq_eq_0_then_AE*) **using** *assms* **by** *auto*

lemma *integral_indicator_finite_real*:
fixes *f :: 'a ⇒ real*
assumes [*simp*]: *finite A*
assumes [*measurable*]: $\bigwedge a. a \in A \implies \{a\} \in \text{sets } M$
assumes *finite*: $\bigwedge a. a \in A \implies \text{emeasure } M \{a\} < \infty$
shows $(\int x. f x * \text{indicator } A x \partial M) = (\sum_{a \in A}. f a * \text{measure } M \{a\})$
proof –
have $(\int x. f x * \text{indicator } A x \partial M) = (\int x. (\sum_{a \in A}. f a * \text{indicator } \{a\} x) \partial M)$
proof (intro *integral_cong_refl*)
fix *x* **show** $f x * \text{indicator } A x = (\sum_{a \in A}. f a * \text{indicator } \{a\} x)$
by (auto *split: split_indicator simp: eq_commute[of x] cong: conj_cong*)
qed
also have $\dots = (\sum_{a \in A}. f a * \text{measure } M \{a\})$
using *finite* **by** (*subst integral_sum*) (*auto*)
finally show ?*thesis* .
qed

lemma (in *finite_measure*) *ennreal_integral_real*:
assumes [*measurable*]: $f \in \text{borel\_measurable } M$
assumes *ae*: *AE x in M. f x ≤ ennreal B 0 ≤ B*
shows *ennreal* $(\int x. \text{enn2real } (f x) \partial M) = (\int^+ x. f x \partial M)$
proof (*subst nn_integral_eq_integral[symmetric]*)
show *integrable M (λx. enn2real (f x))*
using *ae* **by** (intro *integrable_const_bound* [**where** *B=B*]) (*auto simp: enn2real_leI*)
show $(\int^+ x. \text{ennreal } (\text{enn2real } (f x)) \partial M) = \text{integral}^N M f$
using *ae* **by** (intro *nn_integral_cong_AE*) (*auto simp: le_less_trans[OF _ ennreal_less_top]*)
qed *auto*

lemma (in *finite_measure*) *integral_less_AE*:
fixes *X Y :: 'a ⇒ real*
assumes *int*: *integrable M X integrable M Y*
assumes *A*: $(\text{emeasure } M) A \neq 0 \ A \in \text{sets } M \ AE x \text{ in } M. x \in A \longrightarrow X x \neq Y x$
assumes *gt*: *AE x in M. X x ≤ Y x*
shows $\text{integral}^L M X < \text{integral}^L M Y$
proof –
have $\text{integral}^L M X \leq \text{integral}^L M Y$
using *gt int* **by** (intro *integral_mono_AE*) *auto*
moreover
have $\text{integral}^L M X \neq \text{integral}^L M Y$
proof
assume *eq*: $\text{integral}^L M X = \text{integral}^L M Y$
have $\text{integral}^L M (\lambda x. |Y x - X x|) = \text{integral}^L M (\lambda x. Y x - X x)$

```

    using gt int by (intro integral_cong_AE) auto
  also have ... = 0
    using eq int by simp
  finally have (emeasure M) {x ∈ space M. Y x - X x ≠ 0} = 0
    using int by (simp add: integral_0_iff)
  moreover
  have (∫+ x. indicator A x ∂M) ≤ (∫+ x. indicator {x ∈ space M. Y x - X x
    ≠ 0} x ∂M)
    using A by (intro nn_integral_mono_AE) auto
  then have (emeasure M) A ≤ (emeasure M) {x ∈ space M. Y x - X x ≠ 0}
    using int A by (simp add: integrable_def)
  ultimately have emeasure M A = 0
    by simp
  with ⟨(emeasure M) A ≠ 0⟩ show False by auto
qed
ultimately show ?thesis by auto
qed

```

```

lemma (in finite_measure) integral_less_AE_space:
  fixes X Y :: 'a ⇒ real
  assumes int: integrable M X integrable M Y
  assumes gt: AE x in M. X x < Y x emeasure M (space M) ≠ 0
  shows integralL M X < integralL M Y
  using gt by (intro integral_less_AE[OF int, where A=space M]) auto

```

```

lemma tendsto_integral_at_top:
  fixes f :: real ⇒ 'a::{banach, second_countable_topology}
  assumes [measurable_cong]: sets M = sets borel and f[measurable]: integrable
    M f
  shows ((λy. ∫ x. indicator {.. y} x *R f x ∂M) ⟶ ∫ x. f x ∂M) at_top
proof (rule tendsto_at_topI_sequentially)
  fix X :: nat ⇒ real assume filterlim X at_top sequentially
  show (λn. ∫ x. indicator {..X n} x *R f x ∂M) ⟶ integralL M f
proof (rule integral_dominated_convergence)
  show integrable M (λx. norm (f x))
    by (rule integrable_norm) fact
  show AE x in M. (λn. indicator {..X n} x *R f x) ⟶ f x
proof
  fix x
  from ⟨filterlim X at_top sequentially⟩
  have eventually (λn. x ≤ X n) sequentially
    unfolding filterlim_at_top_ge[where c=x] by auto
  then show (λn. indicator {..X n} x *R f x) ⟶ f x
    by (intro tendsto_eventually) (auto split: split_indicator elim!: eventu-
ally_mono)
qed
  fix n show AE x in M. norm (indicator {..X n} x *R f x) ≤ norm (f x)
    by (auto split: split_indicator)
qed auto

```

qed

lemma

```

  fixes f :: real ⇒ real
  assumes M: sets M = sets borel
  assumes nonneg: AE x in M. 0 ≤ f x
  assumes borel: f ∈ borel_measurable borel
  assumes int:  $\bigwedge y. \text{integrable } M (\lambda x. f x * \text{indicator } \{.. y\} x)$ 
  assumes conv:  $((\lambda y. \int x. f x * \text{indicator } \{.. y\} x \partial M) \longrightarrow x) \text{ at\_top}$ 
  shows has_bochner_integral_monotone_convergence_at_top: has_bochner_integral
    M f x
    and integrable_monotone_convergence_at_top: integrable M f
    and integral_monotone_convergence_at_top:  $\text{integral}^L M f = x$ 
proof -
  from nonneg have AE x in M. mono  $(\lambda n::\text{nat}. f x * \text{indicator } \{.. \text{real } n\} x)$ 
  by (auto split: split_indicator intro!: monoI)
  { fix x have eventually  $(\lambda n. f x * \text{indicator } \{.. \text{real } n\} x = f x)$  sequentially
    by (rule eventually_sequentiallyI[of nat  $\lceil x \rceil$ ])
    (auto split: split_indicator simp: nat_le_iff ceiling_le_iff) }
  from filterlim_cong[OF refl refl this]
  have AE x in M.  $(\lambda i. f x * \text{indicator } \{.. \text{real } i\} x) \longrightarrow f x$ 
  by simp
  have  $(\lambda i. \int x. f x * \text{indicator } \{.. \text{real } i\} x \partial M) \longrightarrow x$ 
  using conv filterlim_real_sequentially by (rule filterlim_compose)
  have M_measure[simp]: borel_measurable M = borel_measurable borel
  using M by (simp add: sets_eq_imp_space_eq measurable_def)
  have f ∈ borel_measurable M
  using borel by simp
  show has_bochner_integral M f x
  by (rule has_bochner_integral_monotone_convergence) fact+
  then show integrable M f  $\text{integral}^L M f = x$ 
  by (auto simp: _has_bochner_integral_iff)
qed

```

6.10.8 Product measure

```

lemma (in sigma_finite_measure) borel_measurable_lebesgue_integrable[measurable
  (raw)]:
  fixes f ::  $\_ \Rightarrow \_ \Rightarrow \_::\{\text{banach}, \text{second\_countable\_topology}\}$ 
  assumes [measurable]: case_prod f ∈ borel_measurable  $(N \otimes_M M)$ 
  shows Measurable.pred N  $(\lambda x. \text{integrable } M (f x))$ 
proof -
  have [simp]:  $\bigwedge x. x \in \text{space } N \implies \text{integrable } M (f x) \longleftrightarrow (\int^+ y. \text{norm } (f x y) \partial M) < \infty$ 
  unfolding integrable_iff_bounded by simp
  show ?thesis
  by (simp cong: measurable_cong)
qed

```


lemma (in *sigma_finite_measure*) *measurable_measure*[*measurable (raw)*]:

($\bigwedge x. x \in \text{space } N \implies A \ x \subseteq \text{space } M \implies$
 $\{x \in \text{space } (N \otimes_M M). \text{snd } x \in A \ (\text{fst } x)\} \in \text{sets } (N \otimes_M M) \implies$
 $(\lambda x. \text{measure } M \ (A \ x)) \in \text{borel\_measurable } N$

unfolding *measure_def* **by** (intro *measurable_emeasure borel_measurable_enn2real*)
auto

proposition (in *sigma_finite_measure*) *borel_measurable_lebesgue_integral*[*measurable (raw)*]:

fixes *f* :: $\_ \Rightarrow \_ \Rightarrow \_ :: \{\text{banach}, \text{second\_countable\_topology}\}$
assumes *f*[*measurable*]: *case_prod f* $\in \text{borel\_measurable } (N \otimes_M M)$
shows $(\lambda x. \int y. f \ x \ y \ \partial M) \in \text{borel\_measurable } N$

proof –

from *borel_measurable_implies_sequence_metric*[*OF f, of 0*]

obtain *s* **where** *s*: $\bigwedge i. \text{simple\_function } (N \otimes_M M) \ (s \ i)$

and $\forall x \in \text{space } (N \otimes_M M). (\lambda i. s \ i \ x) \longrightarrow (\text{case } x \text{ of } (x, y) \Rightarrow f \ x \ y)$

and $\forall i. \forall x \in \text{space } (N \otimes_M M). \text{dist } (s \ i \ x) \ 0 \leq 2 * \text{dist } (\text{case } x \text{ of } (x, xa) \Rightarrow$

f x xa) *0*

by *auto*

then have *:

$\bigwedge x \ y. x \in \text{space } N \implies y \in \text{space } M \implies (\lambda i. s \ i \ (x, y)) \longrightarrow f \ x \ y$

$\bigwedge i \ x \ y. x \in \text{space } N \implies y \in \text{space } M \implies \text{norm } (s \ i \ (x, y)) \leq 2 * \text{norm } (f \ x \ y)$

by (*auto simp: space_pair_measure*)

have [*measurable*]: $\bigwedge i. s \ i \in \text{borel\_measurable } (N \otimes_M M)$

by (*rule borel_measurable_simple_function*) *fact*

have $\bigwedge i. s \ i \in \text{measurable } (N \otimes_M M) \ (\text{count\_space } UNIV)$

by (*rule measurable_simple_function*) *fact*

define *f'* **where** [*abs_def*]: *f' i x* =

(*if integrable M (f x) then simple_bochner_integral M* ($\lambda y. s \ i \ (x, y)$) *else 0*)

for *i x*

{ **fix** *i x* **assume** *x* $\in \text{space } N$

then have *simple_bochner_integral M* ($\lambda y. s \ i \ (x, y)$) =

($\sum z \in s \ i \ ' (\text{space } N \times \text{space } M). \text{measure } M \ \{y \in \text{space } M. s \ i \ (x, y) = z\}$

$*_R \ z)$

using *s[THEN simple_functionD(1)]*

unfolding *simple_bochner_integral_def*

by (*intro sum.mono_neutral_cong_left*)

(*auto simp: eq_commute space_pair_measure image_iff cong: conj_cong*)

}

note *eq = this*

show *?thesis*

proof (*rule borel_measurable_LIMSEQ_metric*)

fix *i* **show** *f' i* $\in \text{borel\_measurable } N$

unfolding *f'_def* **by** (*simp_all add: eq cong: measurable_cong if_cong*)

```

next
  fix x assume x: x ∈ space N
  { assume int_f: integrable M (f x)
    have int_2f: integrable M (λy. 2 * norm (f x y))
      by (intro integrable_norm integrable_mult_right int_f)
    have (λi. integralL M (λy. s i (x, y))) ⟶ integralL M (f x)
    proof (rule integral_dominated_convergence)
      from int_f show f x ∈ borel_measurable M by auto
      show ∧i. (λy. s i (x, y)) ∈ borel_measurable M
        using x by simp
      show AE xa in M. (λi. s i (x, xa)) ⟶ f x xa
        using x * by auto
      show ∧i. AE xa in M. norm (s i (x, xa)) ≤ 2 * norm (f x xa)
        using x * by auto
    qed fact
  moreover
  { fix i
    have simple_bochner_integrable M (λy. s i (x, y))
    proof (rule simple_bochner_integrableI_bounded)
      have (λy. s i (x, y)) ' space M ⊆ s i ' (space N × space M)
        using x by auto
      then show simple_function M (λy. s i (x, y))
        using simple_functionD(1)[OF s(1), of i] x
        by (intro simple_function_borel_measurable)
          (auto simp: space_pair_measure dest: finite_subset)
      have (∫+ y. ennreal (norm (s i (x, y))) ∂M) ≤ (∫+ y. 2 * norm (f x y)
∂M)
        using x * by (intro nn_integral_mono) auto
      also have (∫+ y. 2 * norm (f x y) ∂M) < ∞
        using int_2f unfolding integrable_iff_bounded by simp
      finally show (∫+ xa. ennreal (norm (s i (x, xa))) ∂M) < ∞ .
    qed
    then have integralL M (λy. s i (x, y)) = simple_bochner_integral M (λy.
s i (x, y))
      by (rule simple_bochner_integrable_eq_integral[symmetric]) }
    ultimately have (λi. simple_bochner_integral M (λy. s i (x, y))) ⟶
integralL M (f x)
      by simp }
    then
    show (λi. f' i x) ⟶ integralL M (f x)
      unfolding f'_def
      by (cases integrable M (f x)) (simp_all add: not_integrable_integral_eq)
    qed
  qed

lemma (in pair_sigma_finite) integrable_product_swap:
  fixes f :: _ ⇒ _::{banach, second_countable_topology}
  assumes integrable (M1 ⊗M M2) f
  shows integrable (M2 ⊗M M1) (λ(x,y). f (y,x))

```

proof –

```

  interpret Q: pair_sigma_finite M2 M1 ..
  have *: (λ(x,y). f (y,x)) = (λx. f (case x of (x,y)⇒(y,x))) by (auto simp:
fun_eq_iff)
  show ?thesis unfolding *
    by (rule integrable_distr[OF measurable_pair_swap])
      (simp add: distr_pair_swap[symmetric] assms)
qed

```

lemma (in pair_sigma_finite) integrable_product_swap_iff:
 fixes $f :: \_ \Rightarrow \_ :: \{\text{banach}, \text{second\_countable\_topology}\}$
 shows $\text{integrable } (M2 \otimes_M M1) (\lambda(x,y). f (y,x)) \longleftrightarrow \text{integrable } (M1 \otimes_M M2) f$

proof –

```

  interpret Q: pair_sigma_finite M2 M1 ..
  from Q.integrable_product_swap[of λ(x,y). f (y,x)] integrable_product_swap[of
f]
  show ?thesis by auto
qed

```

lemma (in pair_sigma_finite) integral_product_swap:
 fixes $f :: \_ \Rightarrow \_ :: \{\text{banach}, \text{second\_countable\_topology}\}$
 assumes $f: f \in \text{borel\_measurable } (M1 \otimes_M M2)$
 shows $(\int (x,y). f (y,x) \partial(M2 \otimes_M M1)) = \text{integral}^L (M1 \otimes_M M2) f$

proof –

```

  have *: (λ(x,y). f (y,x)) = (λx. f (case x of (x,y)⇒(y,x))) by (auto simp:
fun_eq_iff)
  show ?thesis unfolding *
    by (simp add: integral_distr[symmetric, OF measurable_pair_swap'] distr_pair_swap[symmetric])
qed

```

theorem (in pair_sigma_finite) Fubini_integrable:

```

  fixes  $f :: \_ \Rightarrow \_ :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
  assumes  $f[\text{measurable}]: f \in \text{borel\_measurable } (M1 \otimes_M M2)$ 
    and  $\text{integ1}: \text{integrable } M1 (\lambda x. \int y. \text{norm } (f (x, y)) \partial M2)$ 
    and  $\text{integ2}: \text{AE } x \text{ in } M1. \text{integrable } M2 (\lambda y. f (x, y))$ 
  shows  $\text{integrable } (M1 \otimes_M M2) f$ 
proof (rule integrableI_bounded)
  have  $(\int^+ p. \text{norm } (f p) \partial(M1 \otimes_M M2)) = (\int^+ x. (\int^+ y. \text{norm } (f (x, y)) \partial M2) \partial M1)$ 
  by (simp add: M2.nn_integral_fst [symmetric])
  also have  $\dots = (\int^+ x. |\int y. \text{norm } (f (x, y)) \partial M2| \partial M1)$ 
  apply (intro nn_integral_cong_AE)
  using integ2
proof eventually_elim
  fix  $x$  assume  $\text{integrable } M2 (\lambda y. f (x, y))$ 
  then have  $f: \text{integrable } M2 (\lambda y. \text{norm } (f (x, y)))$ 
    by simp
  then have  $(\int^+ y. \text{ennreal } (\text{norm } (f (x, y))) \partial M2) = \text{ennreal } (LINT y | M2.$ 

```

```

norm (f (x, y)))
  by (rule nn_integral_eq_integral) simp
  also have ... = ennreal |LINT y|M2. norm (f (x, y))|
    using f by simp
  finally show (∫+y. ennreal (norm (f (x, y))) ∂M2) = ennreal |LINT y|M2.
norm (f (x, y))| .
qed
also have ... < ∞
  using integ1 by (simp add: integrable_iff_bounded integral_nonneg_AE)
  finally show (∫+p. norm (f p) ∂(M1 ⊗M M2)) < ∞ .
qed fact

```

```

lemma (in pair_sigma_finite) emeasure_pair_measure_finite:
  assumes A: A ∈ sets (M1 ⊗M M2) and finite: emeasure (M1 ⊗M M2) A <
  ∞
  shows AE x in M1. emeasure M2 {y ∈ space M2. (x, y) ∈ A} < ∞
proof -
  from M2.emeasure_pair_measure_alt[OF A] finite
  have (∫+x. emeasure M2 (Pair x - 'A) ∂M1) ≠ ∞
    by simp
  then have AE x in M1. emeasure M2 (Pair x - 'A) ≠ ∞
    by (rule nn_integral_PInf_AE[rotated]) (intro M2.measurable_emeasure_Pair
  A)
  moreover have ∧x. x ∈ space M1 ⇒ Pair x - 'A = {y ∈ space M2. (x, y) ∈
  A}
    using sets.sets_into_space[OF A] by (auto simp: space_pair_measure)
  ultimately show ?thesis by (auto simp: less_top)
qed

```

```

lemma (in pair_sigma_finite) AE_integrable_fst':
  fixes f :: _ ⇒ _::{banach, second_countable_topology}
  assumes f[measurable]: integrable (M1 ⊗M M2) f
  shows AE x in M1. integrable M2 (λy. f (x, y))
proof -
  have (∫+x. (∫+y. norm (f (x, y)) ∂M2) ∂M1) = (∫+x. norm (f x) ∂(M1 ⊗M
  M2))
    by (rule M2.nn_integral_fst) simp
  also have (∫+x. norm (f x) ∂(M1 ⊗M M2)) ≠ ∞
    using f unfolding integrable_iff_bounded by simp
  finally have AE x in M1. (∫+y. norm (f (x, y)) ∂M2) ≠ ∞
    by (intro nn_integral_PInf_AE M2.borel_measurable_nn_integral )
    (auto simp: measurable_split_conv)
  with AE_space show ?thesis
    by eventually_elim
    (auto simp: integrable_iff_bounded measurable_compose[OF _ borel_measurable_integrable[OF
  f]] less_top)
qed

```

```

lemma (in pair_sigma_finite) integrable_fst':

```

```

fixes  $f :: \_ \Rightarrow \_ :: \{\text{banach, second\_countable\_topology}\}$ 
assumes  $f[\text{measurable}]: \text{integrable } (M1 \otimes_M M2) f$ 
shows  $\text{integrable } M1 \ (\lambda x. \int y. f(x, y) \partial M2)$ 
unfolding  $\text{integrable\_iff\_bounded}$ 
proof
  show  $(\lambda x. \int y. f(x, y) \partial M2) \in \text{borel\_measurable } M1$ 
    by  $(\text{rule } M2.\text{borel\_measurable\_lebesgue\_integral}) \text{ simp}$ 
  have  $(\int^+ x. \text{ennreal } (\text{norm } (\int y. f(x, y) \partial M2)) \partial M1) \leq (\int^+ x. (\int^+ y. \text{norm } (f(x, y)) \partial M2) \partial M1)$ 
    using  $AE.\text{integrable\_fst}'[OF f] \text{ by } (\text{auto intro!: nn\_integral\_mono\_AE integral\_norm\_bound\_ennreal})$ 
  also have  $(\int^+ x. (\int^+ y. \text{norm } (f(x, y)) \partial M2) \partial M1) = (\int^+ x. \text{norm } (f x) \partial (M1 \otimes_M M2))$ 
    by  $(\text{rule } M2.\text{nn\_integral\_fst}) \text{ simp}$ 
  also have  $(\int^+ x. \text{norm } (f x) \partial (M1 \otimes_M M2)) < \infty$ 
    using  $f \text{ unfolding integrable\_iff\_bounded by simp}$ 
  finally show  $(\int^+ x. \text{ennreal } (\text{norm } (\int y. f(x, y) \partial M2)) \partial M1) < \infty .$ 
qed

proposition  $(\text{in pair\_sigma\_finite}) \text{ integral\_fst}'$ :
  fixes  $f :: \_ \Rightarrow \_ :: \{\text{banach, second\_countable\_topology}\}$ 
  assumes  $f: \text{integrable } (M1 \otimes_M M2) f$ 
  shows  $(\int x. (\int y. f(x, y) \partial M2) \partial M1) = \text{integral}^L (M1 \otimes_M M2) f$ 
using  $f \text{ proof induct}$ 
  case  $(\text{base } A \ c)$ 
  have  $A[\text{measurable}]: A \in \text{sets } (M1 \otimes_M M2) \text{ by fact}$ 

  have  $\text{eq}: \bigwedge x y. x \in \text{space } M1 \implies \text{indicator } A(x, y) = \text{indicator } \{y \in \text{space } M2. (x, y) \in A\} y$ 
    using  $\text{sets.sets\_into\_space}[OF A] \text{ by } (\text{auto split: split\_indicator simp: space\_pair\_measure})$ 

  have  $\text{int\_A}: \text{integrable } (M1 \otimes_M M2) (\text{indicator } A :: \_ \Rightarrow \text{real})$ 
    using  $\text{base by } (\text{rule integrable\_real\_indicator})$ 

  have  $(\int x. \int y. \text{indicator } A(x, y) *_R c \partial M2 \partial M1) = (\int x. \text{measure } M2 \{y \in \text{space } M2. (x, y) \in A\} *_R c \partial M1)$ 
    proof  $(\text{intro integral\_cong\_AE, simp, simp})$ 
    from  $AE.\text{integrable\_fst}'[OF \text{int\_A}] AE.\text{space}$ 
    show  $AE \ x \ \text{in } M1. (\int y. \text{indicator } A(x, y) *_R c \partial M2) = \text{measure } M2 \{y \in \text{space } M2. (x, y) \in A\} *_R c$ 
    by  $\text{eventually\_elim } (\text{simp add: eq integrable\_indicator\_iff})$ 
  qed
  also have  $\dots = \text{measure } (M1 \otimes_M M2) A *_R c$ 
proof  $(\text{subst integral\_scaleR\_left})$ 
  have  $(\int^+ x. \text{ennreal } (\text{measure } M2 \{y \in \text{space } M2. (x, y) \in A\}) \partial M1) = (\int^+ x. \text{emeasure } M2 \{y \in \text{space } M2. (x, y) \in A\} \partial M1)$ 
    using  $\text{emeasure\_pair\_measure\_finite}[OF \text{base}]$ 
  by  $(\text{intro nn\_integral\_cong\_AE, eventually\_elim}) \text{ (simp add: emeasure\_eq\_ennreal\_measure)}$ 
  also have  $\dots = \text{emeasure } (M1 \otimes_M M2) A$ 

```

```

    using sets.sets_into_space[OF A]
    by (subst M2.emeasure_pair_measure_alt)
      (auto intro!: nn_integral_cong arg_cong[where f=emeasure M2] simp:
space_pair_measure)
    finally have *: ( $\int^+ x. \text{ennreal} (\text{measure } M2 \{y \in \text{space } M2. (x, y) \in A\}) \partial M1$ )
= emeasure (M1  $\otimes_M$  M2) A .

    from base * show integrable M1 ( $\lambda x. \text{measure } M2 \{y \in \text{space } M2. (x, y) \in A\}$ )
A})
    by (simp add: integrable_iff_bounded)
    then have ( $\int x. \text{measure } M2 \{y \in \text{space } M2. (x, y) \in A\} \partial M1$ ) =
( $\int^+ x. \text{ennreal} (\text{measure } M2 \{y \in \text{space } M2. (x, y) \in A\}) \partial M1$ )
    by (rule nn_integral_eq_integral[symmetric]) simp
    also note *
    finally show ( $\int x. \text{measure } M2 \{y \in \text{space } M2. (x, y) \in A\} \partial M1$ )  $*_R c$  =
measure (M1  $\otimes_M$  M2) A  $*_R c$ 
    using base by (simp add: emeasure_eq_ennreal_measure)
qed
    also have ... = ( $\int a. \text{indicator } A a *_R c \partial(M1 \otimes_M M2)$ )
    using base by simp
    finally show ?case .
next
    case (add f g)
    then have [measurable]:  $f \in \text{borel\_measurable} (M1 \otimes_M M2) \wedge g \in \text{borel\_measurable} (M1 \otimes_M M2)$ 
    by auto
    have ( $\int x. \int y. f(x, y) + g(x, y) \partial M2 \partial M1$ ) =
( $\int x. (\int y. f(x, y) \partial M2) + (\int y. g(x, y) \partial M2) \partial M1$ )
    apply (rule integral_cong_AE)
    apply simp_all
    using AE_integrable_fst'[OF add(1)] AE_integrable_fst'[OF add(3)]
    apply eventually_elim
    apply simp
    done
    also have ... = ( $\int x. f x \partial(M1 \otimes_M M2)$ ) + ( $\int x. g x \partial(M1 \otimes_M M2)$ )
    using integrable_fst'[OF add(1)] integrable_fst'[OF add(3)] add(2,4) by simp
    finally show ?case
    using add by simp
next
    case (lim f s)
    then have [measurable]:  $f \in \text{borel\_measurable} (M1 \otimes_M M2) \wedge i. s i \in \text{borel\_measurable} (M1 \otimes_M M2)$ 
    by auto
    show ?case
    proof (rule LIMSEQ_unique)
    show ( $\lambda i. \text{integral}^L (M1 \otimes_M M2) (s i)$ )  $\longrightarrow \text{integral}^L (M1 \otimes_M M2) f$ 
    proof (rule integral_dominated_convergence)
    show integrable (M1  $\otimes_M$  M2) ( $\lambda x. 2 * \text{norm} (f x)$ )

```

```

    using lim(5) by auto
  qed (insert lim, auto)
  have  $(\lambda i. \int x. \int y. s \ i \ (x, y) \ \partial M2 \ \partial M1) \longrightarrow \int x. \int y. f \ (x, y) \ \partial M2 \ \partial M1$ 
  proof (rule integral_dominated_convergence)
    have  $AE \ x \ in \ M1. \ \forall i. \ integrable \ M2 \ (\lambda y. \ s \ i \ (x, y))$ 
    unfolding AE_all_countable using AE_integrable_fst'[OF lim(1)] ..
    with AE_space AE_integrable_fst'[OF lim(5)]
    show  $AE \ x \ in \ M1. \ (\lambda i. \int y. s \ i \ (x, y) \ \partial M2) \longrightarrow \int y. f \ (x, y) \ \partial M2$ 
  proof eventually_elim
    fix x assume x:  $x \in space \ M1$  and
      s:  $\forall i. \ integrable \ M2 \ (\lambda y. \ s \ i \ (x, y))$  and f:  $integrable \ M2 \ (\lambda y. \ f \ (x, y))$ 
    show  $(\lambda i. \int y. s \ i \ (x, y) \ \partial M2) \longrightarrow \int y. f \ (x, y) \ \partial M2$ 
    proof (rule integral_dominated_convergence)
      show  $integrable \ M2 \ (\lambda y. \ 2 * norm \ (f \ (x, y)))$ 
      using f by auto
      show  $AE \ xa \ in \ M2. \ (\lambda i. \ s \ i \ (x, xa)) \longrightarrow f \ (x, xa)$ 
      using x lim(3) by (auto simp: space_pair_measure)
      show  $\bigwedge i. \ AE \ xa \ in \ M2. \ norm \ (s \ i \ (x, xa)) \leq 2 * norm \ (f \ (x, xa))$ 
      using x lim(4) by (auto simp: space_pair_measure)
    qed (insert x, measurable)
  qed
  show  $integrable \ M1 \ (\lambda x. \ (\int y. \ 2 * norm \ (f \ (x, y)) \ \partial M2))$ 
  by (intro integrable_mult_right integrable_norm integrable_fst' lim)
  fix i show  $AE \ x \ in \ M1. \ norm \ (\int y. s \ i \ (x, y) \ \partial M2) \leq (\int y. \ 2 * norm \ (f \ (x, y)) \ \partial M2)$ 
  using AE_space AE_integrable_fst'[OF lim(1), of i] AE_integrable_fst'[OF lim(5)]
  proof eventually_elim
    fix x assume x:  $x \in space \ M1$ 
    and s:  $integrable \ M2 \ (\lambda y. \ s \ i \ (x, y))$  and f:  $integrable \ M2 \ (\lambda y. \ f \ (x, y))$ 
    from s have  $norm \ (\int y. s \ i \ (x, y) \ \partial M2) \leq (\int^+ y. norm \ (s \ i \ (x, y)) \ \partial M2)$ 
    by (rule integral_norm_bound_enreal)
    also have  $\dots \leq (\int^+ y. \ 2 * norm \ (f \ (x, y)) \ \partial M2)$ 
    using x lim by (auto intro!: nn_integral_mono simp: space_pair_measure)
    also have  $\dots = (\int y. \ 2 * norm \ (f \ (x, y)) \ \partial M2)$ 
    using f by (intro nn_integral_eq_integral) auto
    finally show  $norm \ (\int y. s \ i \ (x, y) \ \partial M2) \leq (\int y. \ 2 * norm \ (f \ (x, y)) \ \partial M2)$ 
  by simp
  qed
  qed simp_all
  then show  $(\lambda i. \ integral^L \ (M1 \ \otimes_M \ M2) \ (s \ i)) \longrightarrow \int x. \int y. f \ (x, y) \ \partial M2 \ \partial M1$ 
  using lim by simp
  qed
  qed

```

lemma (in pair_sigma_finite)

fixes $f :: \_ \Rightarrow \_ \Rightarrow \_ :: \{\text{banach, second\_countable\_topology}\}$

```

assumes  $f$ : integrable  $(M1 \otimes_M M2)$  (case_prod  $f$ )
shows  $AE\_integrable\_fst$ :  $AE\ x\ in\ M1.$  integrable  $M2\ (\lambda y. f\ x\ y)$  (is ? $AE$ )
  and  $integrable\_fst$ : integrable  $M1\ (\lambda x. \int y. f\ x\ y\ \partial M2)$  (is ? $INT$ )
  and  $integral\_fst$ :  $(\int x. (\int y. f\ x\ y\ \partial M2)\ \partial M1) = integral^L\ (M1 \otimes_M M2)\ (\lambda(x,$ 
 $y). f\ x\ y)$  (is ? $EQ$ )
using  $AE\_integrable\_fst'[OF\ f]$   $integrable\_fst'[OF\ f]$   $integral\_fst'[OF\ f]$  by auto

```

```

lemma (in pair_sigma_finite)
  fixes  $f :: \_ \Rightarrow \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$ 
  assumes  $f[measurable]$ : integrable  $(M1 \otimes_M M2)$  (case_prod  $f$ )
  shows  $AE\_integrable\_snd$ :  $AE\ y\ in\ M2.$  integrable  $M1\ (\lambda x. f\ x\ y)$  (is ? $AE$ )
    and  $integrable\_snd$ : integrable  $M2\ (\lambda y. \int x. f\ x\ y\ \partial M1)$  (is ? $INT$ )
    and  $integral\_snd$ :  $(\int y. (\int x. f\ x\ y\ \partial M1)\ \partial M2) = integral^L\ (M1 \otimes_M M2)$ 
    (case_prod  $f$ ) (is ? $EQ$ )
proof –
  interpret  $Q$ : pair_sigma_finite  $M2\ M1\ ..$ 
  have  $Q\_int$ : integrable  $(M2 \otimes_M M1)\ (\lambda(x, y). f\ y\ x)$ 
    using  $f$  unfolding integrable_product_swap_iff[symmetric] by simp
  show ? $AE$  using  $Q.AE\_integrable\_fst'[OF\ Q\_int]$  by simp
  show ? $INT$  using  $Q.integrable\_fst'[OF\ Q\_int]$  by simp
  show ? $EQ$  using  $Q.integral\_fst'[OF\ Q\_int]$ 
    using integral_product_swap[of case_prod  $f$ ] by simp
qed

```

```

proposition (in pair_sigma_finite) Fubini_integral:
  fixes  $f :: \_ \Rightarrow \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$ 
  assumes  $f$ : integrable  $(M1 \otimes_M M2)$  (case_prod  $f$ )
  shows  $(\int y. (\int x. f\ x\ y\ \partial M1)\ \partial M2) = (\int x. (\int y. f\ x\ y\ \partial M2)\ \partial M1)$ 
  unfolding integral_snd[OF assms] integral_fst[OF assms] ..

```

```

lemma (in product_sigma_finite) product_integral_singleton:
  fixes  $f :: \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$ 
  shows  $f \in borel\_measurable\ (M\ i) \implies (\int x. f\ (x\ i)\ \partial Pi_M\ \{i\}\ M) = integral^L$ 
 $(M\ i)\ f$ 
  apply (subst distr_singleton[symmetric])
  apply (subst integral_distr)
  apply simp_all
done

```

```

lemma (in product_sigma_finite) product_integral_fold:
  fixes  $f :: \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$ 
  assumes  $IJ[simp]$ :  $I \cap J = \{\}$  and  $fin$ : finite  $I$  finite  $J$ 
  and  $f$ : integrable  $(Pi_M\ (I \cup J)\ M)\ f$ 
  shows  $integral^L\ (Pi_M\ (I \cup J)\ M)\ f = (\int x. (\int y. f\ (merge\ I\ J\ (x, y))\ \partial Pi_M\ J$ 
 $M)\ \partial Pi_M\ I\ M)$ 
proof –
  interpret  $I$ : finite_product_sigma_finite  $M\ I$  by standard fact
  interpret  $J$ : finite_product_sigma_finite  $M\ J$  by standard fact
  have finite  $(I \cup J)$  using fin by auto

```



```

interpret IJ: finite_product_sigma_finite M I  $\cup$  J by standard fact
interpret P: pair_sigma_finite Pi_M I M Pi_M J M ..
let ?M = merge I J
let ?f =  $\lambda x. f$  (?M x)
from f have f_borel:  $f \in \text{borel\_measurable } (Pi_M (I \cup J) M)$ 
  by auto
have P_borel:  $(\lambda x. f (merge I J x)) \in \text{borel\_measurable } (Pi_M I M \otimes_M Pi_M J M)$ 
  using measurable_comp[OF measurable_merge f_borel] by (simp add: comp_def)
have f_int: integrable  $(Pi_M I M \otimes_M Pi_M J M)$  ?f
  by (rule integrable_distr[OF measurable_merge]) (simp add: distr_merge[OF IJ fin] f)
show ?thesis
  apply (subst distr_merge[symmetric, OF IJ fin])
  apply (subst integral_distr[OF measurable_merge f_borel])
  apply (subst P.integral_fst'[symmetric, OF f_int])
  apply simp
  done
qed

```

```

lemma (in product_sigma_finite) product_integral_insert:
  fixes f ::  $\_ \Rightarrow \_::\{\text{banach}, \text{second\_countable\_topology}\}$ 
  assumes I: finite I i  $\notin$  I
  and f: integrable  $(Pi_M (\text{insert } i I) M)$  f
  shows  $\text{integral}^L (Pi_M (\text{insert } i I) M) f = (\int x. (\int y. f (x(i:=y))) \partial M i) \partial Pi_M I M)$ 
proof -
  have  $\text{integral}^L (Pi_M (\text{insert } i I) M) f = \text{integral}^L (Pi_M (I \cup \{i\}) M) f$ 
    by simp
  also have  $\dots = (\int x. (\int y. f (merge I \{i\} (x,y)) \partial Pi_M \{i\} M) \partial Pi_M I M)$ 
    using f I by (intro product_integral_fold) auto
  also have  $\dots = (\int x. (\int y. f (x(i:=y)) \partial M i) \partial Pi_M I M)$ 
  proof (rule integral_cong[OF refl], subst product_integral_singleton[symmetric])
    fix x assume x:  $x \in \text{space } (Pi_M I M)$ 
    have f_borel:  $f \in \text{borel\_measurable } (Pi_M (\text{insert } i I) M)$ 
      using f by auto
    show  $(\lambda y. f (x(i:=y))) \in \text{borel\_measurable } (M i)$ 
      using measurable_comp[OF measurable_component_update f_borel, OF x <i  $\notin$  I>]
    unfolding comp_def .
    from x I show  $(\int y. f (merge I \{i\} (x,y)) \partial Pi_M \{i\} M) = (\int xa. f (x(i:=xa i)) \partial Pi_M \{i\} M)$ 
      by (auto intro!: integral_cong arg_cong[where f=f] simp: merge_def space_PiM extensional_def PiE_def)
    qed
  finally show ?thesis .
qed

```

```

lemma (in product_sigma_finite) product_integrable_prod:

```

fixes $f :: 'i \Rightarrow 'a \Rightarrow \_ :: \{\text{real\_normed\_field}, \text{banach}, \text{second\_countable\_topology}\}$
 assumes $[simp]: \text{finite } I \text{ and integrable: } \bigwedge i. i \in I \implies \text{integrable } (M\ i) (f\ i)$
 shows $\text{integrable } (Pi_M\ I\ M) (\lambda x. (\prod i \in I. f\ i\ (x\ i)))$ (is integrable $\_ ?f$)
proof (unfold integrable_iff_bounded, intro conjI)
 interpret finite_product_sigma_finite $M\ I$ by standard fact

show $?f \in \text{borel\_measurable } (Pi_M\ I\ M)$
 using assms by simp
 have $(\int^+ x. \text{ennreal } (\text{norm } (\prod i \in I. f\ i\ (x\ i))))\ \partial Pi_M\ I\ M =$
 $(\int^+ x. (\prod i \in I. \text{ennreal } (\text{norm } (f\ i\ (x\ i))))\ \partial Pi_M\ I\ M)$
 by (simp add: prod_norm_prod_ennreal)
 also have $\dots = (\prod i \in I. \int^+ x. \text{ennreal } (\text{norm } (f\ i\ x))\ \partial M\ i)$
 using assms by (intro product_nn_integral_prod) auto
 also have $\dots < \infty$
 using integrable by (simp add: less_top[symmetric] ennreal_prod_eq_top integrable_iff_bounded)
 finally show $(\int^+ x. \text{ennreal } (\text{norm } (\prod i \in I. f\ i\ (x\ i))))\ \partial Pi_M\ I\ M < \infty$.
qed

lemma (in product_sigma_finite) product_integral_prod:
 fixes $f :: 'i \Rightarrow 'a \Rightarrow \_ :: \{\text{real\_normed\_field}, \text{banach}, \text{second\_countable\_topology}\}$
 assumes $\text{finite } I \text{ and integrable: } \bigwedge i. i \in I \implies \text{integrable } (M\ i) (f\ i)$
 shows $(\int x. (\prod i \in I. f\ i\ (x\ i))\ \partial Pi_M\ I\ M) = (\prod i \in I. \text{integral}^L (M\ i) (f\ i))$
using assms **proof** induct
 case empty
 interpret finite_measure $Pi_M\ \{\}\ M$
 by rule (simp add: space_PiM)
 show ?case by (simp add: space_PiM measure_def)
next
 case (insert $i\ I$)
 then have $iI: \text{finite } (\text{insert } i\ I)$ by auto
 then have $\text{prod: } \bigwedge J. J \subseteq \text{insert } i\ I \implies$
 $\text{integrable } (Pi_M\ J\ M) (\lambda x. (\prod i \in J. f\ i\ (x\ i)))$
 by (intro product_integrable_prod insert(4)) (auto intro: finite_subset)
 interpret $I: \text{finite\_product\_sigma\_finite } M\ I$ by standard fact
 have $*$: $\bigwedge x\ y. (\prod j \in I. f\ j\ (\text{if } j = i \text{ then } y \text{ else } x\ j)) = (\prod j \in I. f\ j\ (x\ j))$
 using $\langle i \notin I \rangle$ by (auto intro!: prod.cong)
 show ?case
 unfolding product_integral_insert[OF insert(1,2) prod[OF subset_refl]]
 by (simp add: * insert_prod_subset_insertI)
qed

lemma integrable_subalgebra:
 fixes $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second\_countable\_topology}\}$
 assumes $\text{borel: } f \in \text{borel\_measurable } N$
 and $N: \text{sets } N \subseteq \text{sets } M \text{ space } N = \text{space } M \bigwedge A. A \in \text{sets } N \implies \text{emeasure } N$
 $A = \text{emeasure } M\ A$
 shows $\text{integrable } N\ f \longleftrightarrow \text{integrable } M\ f$ (is ?P)
proof –

```

have  $f \in \text{borel\_measurable } M$ 
  using assms by (auto simp: measurable_def)
with assms show ?thesis
  using assms by (auto simp: integrable_iff_bounded_nn_integral_subalgebra)
qed

lemma integral_subalgebra:
  fixes  $f :: 'a \Rightarrow 'b::\{\text{banach}, \text{second\_countable\_topology}\}$ 
  assumes borel:  $f \in \text{borel\_measurable } N$ 
  and  $N: \text{sets } N \subseteq \text{sets } M \text{ space } N = \text{space } M \wedge A. A \in \text{sets } N \implies \text{emeasure } N$ 
 $A = \text{emeasure } M A$ 
  shows  $\text{integral}^L N f = \text{integral}^L M f$ 
proof cases
  assume integrable  $N f$ 
  then show ?thesis
  proof induct
    case base with assms show ?case by (auto simp: subset_eq measure_def)
  next
    case (add f g)
    then have  $(\int a. f a + g a \, \partial N) = \text{integral}^L M f + \text{integral}^L M g$ 
      by simp
    also have  $\dots = (\int a. f a + g a \, \partial M)$ 
      using add integrable_subalgebra[OF  $N$ , of  $f$ ] integrable_subalgebra[OF  $N$ , of  $g$ ] by simp
    finally show ?case .
  next
    case (lim f s)
    then have  $M: \bigwedge i. \text{integrable } M (s \, i) \text{ integrable } M f$ 
      using integrable_subalgebra[OF  $N$ , of  $f$ ] integrable_subalgebra[OF  $N$ , of  $s \, i$  for  $i$ ] by simp_all
    show ?case
    proof (intro LIMSEQ_unique)
      show  $(\lambda i. \text{integral}^L N (s \, i)) \longrightarrow \text{integral}^L N f$ 
        apply (rule integral_dominated_convergence[where  $w = \lambda x. 2 * \text{norm } (f \, x)$ ])
          using lim
          apply auto
          done
      show  $(\lambda i. \text{integral}^L N (s \, i)) \longrightarrow \text{integral}^L M f$ 
        unfolding lim
        apply (rule integral_dominated_convergence[where  $w = \lambda x. 2 * \text{norm } (f \, x)$ ])
          using lim  $M \, N(2)$ 
          apply auto
          done
    qed
  qed
qed (simp add: not_integrable_integral_eq integrable_subalgebra[OF assms])

```

```

hide_const (open) simple_bochner_integral
hide_const (open) simple_bochner_integrable

end

```

6.11 Complete Measures

```

theory Complete_Measure
  imports Bochner_Integration
begin

locale complete_measure =
  fixes M :: 'a measure
  assumes complete:  $\bigwedge A B. B \subseteq A \implies A \in \text{null\_sets } M \implies B \in \text{sets } M$ 

definition
  split_completion M A p = (if A  $\in$  sets M then p = (A, {}) else
     $\exists N'. A = \text{fst } p \cup \text{snd } p \wedge \text{fst } p \cap \text{snd } p = \{\}$   $\wedge$   $\text{fst } p \in \text{sets } M \wedge \text{snd } p \subseteq N' \wedge$ 
     $N' \in \text{null\_sets } M$ )

definition
  main_part M A = fst (Eps (split_completion M A))

definition
  null_part M A = snd (Eps (split_completion M A))

definition completion :: 'a measure  $\Rightarrow$  'a measure where
  completion M = measure_of (space M) { S  $\cup$  N | S N N'. S  $\in$  sets M  $\wedge$  N'  $\in$ 
  null_sets M  $\wedge$  N  $\subseteq$  N' }
  (emeasure M  $\circ$  main_part M)

lemma completion_into_space:
  { S  $\cup$  N | S N N'. S  $\in$  sets M  $\wedge$  N'  $\in$  null_sets M  $\wedge$  N  $\subseteq$  N' }  $\subseteq$  Pow (space
  M)
  using sets.sets_into_space by auto

lemma space_completion[simp]: space (completion M) = space M
  unfolding completion_def using space_measure_of[OF completion_into_space]
  by simp

lemma completionI:
  assumes A = S  $\cup$  N N  $\subseteq$  N' N'  $\in$  null_sets M S  $\in$  sets M
  shows A  $\in$  { S  $\cup$  N | S N N'. S  $\in$  sets M  $\wedge$  N'  $\in$  null_sets M  $\wedge$  N  $\subseteq$  N' }
  using assms by auto

lemma completionE:
  assumes A  $\in$  { S  $\cup$  N | S N N'. S  $\in$  sets M  $\wedge$  N'  $\in$  null_sets M  $\wedge$  N  $\subseteq$  N' }
  obtains S N N' where A = S  $\cup$  N N  $\subseteq$  N' N'  $\in$  null_sets M S  $\in$  sets M
  using assms by auto

```

```

lemma sigma_algebra_completion:
  sigma_algebra (space M) { S ∪ N | S N N'. S ∈ sets M ∧ N' ∈ null_sets M ∧
N ⊆ N' }
  (is sigma_algebra _ ?A)
  unfolding sigma_algebra_iff2
proof (intro conjI ballI allI impI)
  show ?A ⊆ Pow (space M)
    using sets.sets_into_space by auto
next
  show {} ∈ ?A by auto
next
  let ?C = space M
  fix A assume A ∈ ?A
  then obtain S N N'
    where A = S ∪ N N ⊆ N' N' ∈ null_sets M S ∈ sets M
    by (rule completionE)
  then show space M − A ∈ ?A
    by (intro completionI[of _ ( ?C − S) ∩ ( ?C − N') ( ?C − S) ∩ N' ∩ ( ?C −
N)]) auto
next
  fix A :: nat ⇒ 'a set assume A: range A ⊆ ?A
  then have ∀ n. ∃ S N N'. A n = S ∪ N ∧ S ∈ sets M ∧ N' ∈ null_sets M ∧
N ⊆ N'
    by (auto simp: image_subset_iff)
  then obtain S N N' where ∀ x. A x = S x ∪ N x ∧ S x ∈ sets M ∧ N' x ∈
null_sets M ∧ N x ⊆ N' x
    by metis
  then show ∪(A ' UNIV) ∈ ?A
    using null_sets_UN[of N']
    by (intro completionI[of _ ∪(S ' UNIV) ∪(N ' UNIV) ∪(N' ' UNIV)]) auto
qed

```

```

lemma sets_completion:
  sets (completion M) = { S ∪ N | S N N'. S ∈ sets M ∧ N' ∈ null_sets M ∧ N
⊆ N' }
  using sigma_algebra.sets_measure_of_eq[OF sigma_algebra_completion]
  by (simp add: completion_def)

```

```

lemma sets_completionE:
  assumes A ∈ sets (completion M)
  obtains S N N' where A = S ∪ N N ⊆ N' N' ∈ null_sets M S ∈ sets M
  using assms unfolding sets_completion by auto

```

```

lemma sets_completionI:
  assumes A = S ∪ N N ⊆ N' N' ∈ null_sets M S ∈ sets M
  shows A ∈ sets (completion M)
  using assms unfolding sets_completion by auto

```

lemma *sets_completionI_sets*[intro, simp]:
 $A \in \text{sets } M \implies A \in \text{sets } (\text{completion } M)$
unfolding *sets_completion* **by** force

lemma *measurable_completion*: $f \in M \rightarrow_M N \implies f \in \text{completion } M \rightarrow_M N$
by (auto simp: measurable_def)

lemma *null_sets_completion*:
assumes $N' \in \text{null\_sets } M$ $N \subseteq N'$ **shows** $N \in \text{sets } (\text{completion } M)$
using *assms* **by** (intro sets_completionI[of N $\{\}$ N N']) auto

lemma *split_completion*:
assumes $A \in \text{sets } (\text{completion } M)$
shows *split_completion* M A (*main_part* M A , *null_part* M A)
proof cases
assume $A \in \text{sets } M$ **then show** ?thesis
by (simp add: split_completion_def[abs_def] *main_part_def* *null_part_def*)
next
assume $nA: A \notin \text{sets } M$
show ?thesis
unfolding *main_part_def* *null_part_def* *if_not_P*[OF nA]
proof (rule someI2_ex)
from *assms* **obtain** S N N' **where** $A = S \cup N$ $N \subseteq N'$ $N' \in \text{null\_sets } M$
 $S \in \text{sets } M$
by (blast intro: sets_completionE)
let $?P = (S, N - S)$
show $\exists p. \text{split\_completion } M$ A p
unfolding *split_completion_def* *if_not_P*[OF nA] **using** A
proof (intro exI conjI)
show $A = \text{fst } ?P \cup \text{snd } ?P$ **using** A **by** auto
show $\text{snd } ?P \subseteq N'$ **using** A **by** auto
qed auto
qed auto
qed

lemma *sets_restrict_space_subset*:
assumes $S: S \in \text{sets } (\text{completion } M)$
shows $\text{sets } (\text{restrict\_space } (\text{completion } M))$ $S \subseteq \text{sets } (\text{completion } M)$
by (metis *assms* sets.Int_space_eq2 sets_restrict_space_iff subsetI)

lemma
assumes $S \in \text{sets } (\text{completion } M)$
shows *main_part_sets*[intro, simp]: *main_part* M $S \in \text{sets } M$
and *main_part_null_part_Un*[simp]: *main_part* M $S \cup \text{null\_part } M$ $S = S$
and *main_part_null_part_Int*[simp]: *main_part* M $S \cap \text{null\_part } M$ $S = \{\}$
using *split_completion*[OF *assms*]
by (auto simp: split_completion_def *split*: *if_split_asm*)

lemma *main_part*[simp]: $S \in \text{sets } M \implies \text{main\_part } M$ $S = S$

```

using split_completion[of S M]
by (auto simp: split_completion_def split: if_split_asm)

```

lemma null_part:

```

assumes S ∈ sets (completion M) shows ∃ N. N ∈ null_sets M ∧ null_part M
S ⊆ N
using split_completion[OF assms] by (auto simp: split_completion_def split:
if_split_asm)

```

lemma null_part_sets[intro, simp]:

```

assumes S ∈ sets M shows null_part M S ∈ sets M emeasure M (null_part M
S) = 0
proof –
  have S: S ∈ sets (completion M) using assms by auto
  have *: S – main_part M S ∈ sets M using assms by auto
  from main_part_null_part_Un[OF S] main_part_null_part_Int[OF S]
  have S – main_part M S = null_part M S by auto
  with * show sets: null_part M S ∈ sets M by auto
  from null_part[OF S] obtain N where N ∈ null_sets M ∧ null_part M S ⊆
N ..
  with emeasure_eq_0[of N _ null_part M S] sets
  show emeasure M (null_part M S) = 0 by auto
qed

```

lemma emeasure_main_part_UN:

```

fixes S :: nat ⇒ 'a set
assumes range S ⊆ sets (completion M)
shows emeasure M (main_part M (⋃ i. (S i))) = emeasure M (⋃ i. main_part
M (S i))
proof –
  have S: ⋀ i. S i ∈ sets (completion M) using assms by auto
  then have UN: (⋃ i. S i) ∈ sets (completion M) by auto
  have ∀ i. ∃ N. N ∈ null_sets M ∧ null_part M (S i) ⊆ N
    using null_part[OF S] by auto
  then obtain N where N: ∀ x. N x ∈ null_sets M ∧ null_part M (S x) ⊆ N x
by metis
  then have UN_N: (⋃ i. N i) ∈ null_sets M by (intro null_sets_UN) auto
  from S have (⋃ i. S i) ∈ sets (completion M) by auto
  from null_part[OF this] obtain N' where N': N' ∈ null_sets M ∧ null_part
M (⋃ (range S)) ⊆ N' ..
  let ?N = (⋃ i. N i) ∪ N'
  have null_set: ?N ∈ null_sets M using N' UN_N by (intro null_sets.Un)
auto
  have main_part M (⋃ i. S i) ∪ ?N = (main_part M (⋃ i. S i) ∪ null_part M
(⋃ i. S i)) ∪ ?N
    using N' by auto
  also have ... = (⋃ i. main_part M (S i) ∪ null_part M (S i)) ∪ ?N
    unfolding main_part_null_part_Un[OF S] main_part_null_part_Un[OF
UN] by auto

```

```

    also have ... = ( $\bigcup i. \text{main\_part } M (S i)$ )  $\cup$  ?N
    using N by auto
    finally have *:  $\text{main\_part } M (\bigcup i. S i) \cup ?N = (\bigcup i. \text{main\_part } M (S i)) \cup ?N$ 
  .
  have  $\text{emeasure } M (\text{main\_part } M (\bigcup i. S i)) = \text{emeasure } M (\text{main\_part } M (\bigcup i. S i) \cup ?N)$ 
  using null_set UN by (intro  $\text{emeasure\_Un\_null\_set[symmetric]}$ ) auto
  also have ... =  $\text{emeasure } M ((\bigcup i. \text{main\_part } M (S i)) \cup ?N)$ 
  unfolding * ..
  also have ... =  $\text{emeasure } M (\bigcup i. \text{main\_part } M (S i))$ 
  using null_set S by (intro  $\text{emeasure\_Un\_null\_set}$ ) auto
  finally show ?thesis .
qed

```

```

lemma  $\text{emeasure\_completion[simp]}$ :
  assumes  $S: S \in \text{sets } (\text{completion } M)$ 
  shows  $\text{emeasure } (\text{completion } M) S = \text{emeasure } M (\text{main\_part } M S)$ 
proof (subst  $\text{emeasure\_measure\_of[OF completion\_def completion\_into\_space]}$ )
  let ? $\mu = \text{emeasure } M \circ \text{main\_part } M$ 
  show  $S \in \text{sets } (\text{completion } M) \text{ ?}\mu S = \text{emeasure } M (\text{main\_part } M S)$  using S
by simp_all
  show positive ( $\text{sets } (\text{completion } M)$ ) ? $\mu$ 
  by (simp add: positive_def)
  show countably_additive ( $\text{sets } (\text{completion } M)$ ) ? $\mu$ 
proof (intro countably_additiveI)
  fix  $A :: \text{nat} \Rightarrow 'a \text{ set}$  assume  $A: \text{range } A \subseteq \text{sets } (\text{completion } M) \text{ disjoint\_family } A$ 
  have disjoint_family ( $\lambda i. \text{main\_part } M (A i)$ )
  proof (intro disjoint_family_on_bisimulation[OF  $A(2)$ ])
    fix  $n m$  assume  $A n \cap A m = \{\}$ 
    then have  $(\text{main\_part } M (A n) \cup \text{null\_part } M (A n)) \cap (\text{main\_part } M (A m) \cup \text{null\_part } M (A m)) = \{\}$ 
    using A by (subst (1 2)  $\text{main\_part\_null\_part\_Un}$ ) auto
    then show  $\text{main\_part } M (A n) \cap \text{main\_part } M (A m) = \{\}$  by auto
  qed
  then have  $(\sum n. \text{emeasure } M (\text{main\_part } M (A n))) = \text{emeasure } M (\bigcup i. \text{main\_part } M (A i))$ 
  using A by (auto intro!:  $\text{suminf\_emeasure}$ )
  then show  $(\sum n. ?\mu (A n)) = ?\mu (\bigcup (A \text{ ' } UNIV))$ 
  by (simp add:  $\text{completion\_def emeasure\_main\_part\_UN[OF } A(1)]$ )
qed
qed

```

```

lemma  $\text{measure\_completion[simp]}$ :  $S \in \text{sets } M \implies \text{measure } (\text{completion } M) S = \text{measure } M S$ 
  unfolding  $\text{measure\_def}$  by auto

```

```

lemma  $\text{emeasure\_completion\_UN}$ :
   $\text{range } S \subseteq \text{sets } (\text{completion } M) \implies$ 

```


$\text{emeasure } (\text{completion } M) (\bigcup i::\text{nat. } (S \ i)) = \text{emeasure } M (\bigcup i. \text{main\_part } M (S \ i))$

by (subst *emeasure_completion*) (auto simp add: *emeasure_main_part_UN*)

lemma *emeasure_completion_Un*:

assumes *S*: $S \in \text{sets } (\text{completion } M)$ **and** *T*: $T \in \text{sets } (\text{completion } M)$

shows $\text{emeasure } (\text{completion } M) (S \cup T) = \text{emeasure } M (\text{main\_part } M S \cup \text{main\_part } M T)$

proof (subst *emeasure_completion*)

have *UN*: $(\bigcup i. \text{binary } (\text{main\_part } M S) (\text{main\_part } M T) \ i) = (\bigcup i. \text{main\_part } M (\text{binary } S \ T \ i))$

unfolding *binary_def* **by** (auto split: *if_split_asm*)

show $\text{emeasure } M (\text{main\_part } M (S \cup T)) = \text{emeasure } M (\text{main\_part } M S \cup \text{main\_part } M T)$

using *emeasure_main_part_UN*[of *binary S T M*] *assms*

by (simp add: *range_binary_eq*, simp add: *Un_range_binary UN*)

qed (auto intro: *S T*)

lemma *sets_completionI_sub*:

assumes *N*: $N' \in \text{null\_sets } M \ N \subseteq N'$

shows $N \in \text{sets } (\text{completion } M)$

using *assms* **by** (intro *sets_completionI*[of _ $\{\}$ *N N'*]) auto

lemma *completion_ex_simple_function*:

assumes *f*: *simple_function* (*completion M*) *f*

shows $\exists f'. \text{simple\_function } M \ f' \wedge (\forall x \in M. f \ x = f' \ x)$

proof –

let $?F = \lambda x. f - \{x\} \cap \text{space } M$

have *F*: $\bigwedge x. ?F \ x \in \text{sets } (\text{completion } M)$ **and** *fin*: *finite* (*f*'*space M*)

using *simple_functionD*[*OF f*] *simple_functionD*[*OF f*] **by** *simp_all*

have $\forall x. \exists N. N \in \text{null\_sets } M \wedge \text{null\_part } M \ (?F \ x) \subseteq N$

using *F null_part* **by** auto

from *choice*[*OF this*] **obtain** *N* **where**

N: $\bigwedge x. \text{null\_part } M \ (?F \ x) \subseteq N \ x \wedge x \in \text{null\_sets } M$ **by** auto

let $?N = \bigcup x \in f' \text{space } M. N \ x$

let $?f' = \lambda x. \text{if } x \in ?N \text{ then undefined else } f \ x$

have *sets*: $?N \in \text{null\_sets } M$ **using** *N fin* **by** (intro *null_sets.finite_UN*) auto

show *thesis* **unfolding** *simple_function_def*

proof (safe intro!: *exI*[of _ $?f'$])

have $?f' - \{f' \text{space } M \cup \{\text{undefined}\}\}$ **by** auto

from *finite_subset*[*OF this*] *simple_functionD*(1)[*OF f*]

show *finite* ($?f' - \{f' \text{space } M\}$) **by** auto

next

fix *x* **assume** $x \in \text{space } M$

have $?f' - \{?f' \ x\} \cap \text{space } M =$

(if $x \in ?N$ then $?F \ \text{undefined} \cup ?N$

else if $f \ x = \text{undefined}$ then $?F \ (f \ x) \cup ?N$

else $?F \ (f \ x) - ?N$)

using *N*(2) *sets.sets_into_space* **by** (auto split: *if_split_asm* simp: *null_sets_def*)

```

moreover { fix y have ?F y  $\cup$  ?N  $\in$  sets M
proof cases
  assume y: y  $\in$  f'space M
  have ?F y  $\cup$  ?N = (main_part M (?F y)  $\cup$  null_part M (?F y))  $\cup$  ?N
    using main_part_null_part_Un[OF F] by auto
  also have ... = main_part M (?F y)  $\cup$  ?N
    using y N by auto
  finally show ?thesis
    using F sets by auto
next
  assume y  $\notin$  f'space M then have ?F y = {} by auto
  then show ?thesis using sets by auto
qed }
moreover {
  have ?F (f x) - ?N = main_part M (?F (f x))  $\cup$  null_part M (?F (f x)) -
?N
    using main_part_null_part_Un[OF F] by auto
  also have ... = main_part M (?F (f x)) - ?N
    using N  $\langle$  x  $\in$  space M  $\rangle$  by auto
  finally have ?F (f x) - ?N  $\in$  sets M
    using F sets by auto }
ultimately show ?f' - ' { ?f' x }  $\cap$  space M  $\in$  sets M by auto
next
  show AE x in M. f x = ?f' x
    by (rule AE_I', rule sets) auto
qed
qed

lemma completion_ex_borel_measurable:
  fixes g :: 'a  $\Rightarrow$  ennreal
  assumes g: g  $\in$  borel_measurable (completion M)
  shows  $\exists$  g'  $\in$  borel_measurable M. (AE x in M. g x = g' x)
proof -
  obtain f :: nat  $\Rightarrow$  'a  $\Rightarrow$  ennreal
    where *:  $\bigwedge$  i. simple_function (completion M) (f i)
    and **:  $\bigwedge$  x. (SUP i. f i x) = g x
  using g[THEN borel_measurable_implies_simple_function_sequence] by metis
  from *[THEN completion_ex_simple_function]
  have  $\forall$  i.  $\exists$  f'. simple_function M f'  $\wedge$  (AE x in M. f i x = f' x) ..
  then obtain f'
    where sf:  $\bigwedge$  i. simple_function M (f' i)
    and AE:  $\forall$  i. AE x in M. f i x = f' i x
  by metis
  show ?thesis
proof (intro bexI)
  from AE[unfolded AE_all_countable[symmetric]]
  show AE x in M. g x = (SUP i. f' i x) (is AE x in M. g x = ?f x)
proof (elim AE_mp, safe intro!: AE_I2)
  fix x assume eq:  $\forall$  i. f i x = f' i x

```

```

      have  $g\ x = (\text{SUP } i. f\ i\ x)$  by (auto simp: ** split: split_max)
      with eq show  $g\ x = ?f\ x$  by auto
    qed
  show  $?f \in \text{borel\_measurable } M$ 
    using sf[THEN borel_measurable_simple_function] by auto
  qed
qed

lemma null_sets_completionI:  $N \in \text{null\_sets } M \implies N \in \text{null\_sets } (\text{completion } M)$ 
  by (auto simp: null_sets_def)

lemma AE_completion:  $(AE\ x\ in\ M. P\ x) \implies (AE\ x\ in\ \text{completion } M. P\ x)$ 
  unfolding eventually_ae_filter by (auto intro: null_sets_completionI)

lemma null_sets_completion_iff:  $N \in \text{sets } M \implies N \in \text{null\_sets } (\text{completion } M) \iff N \in \text{null\_sets } M$ 
  by (auto simp: null_sets_def)

lemma sets_completion_AE:  $(AE\ x\ in\ M. \neg P\ x) \implies \text{Measurable.pred } (\text{completion } M)\ P$ 
  unfolding pred_def sets_completion eventually_ae_filter
  by auto

lemma null_sets_completion_iff2:
   $A \in \text{null\_sets } (\text{completion } M) \iff (\exists N' \in \text{null\_sets } M. A \subseteq N')$ 
proof safe
  assume  $A \in \text{null\_sets } (\text{completion } M)$ 
  then have  $A: A \in \text{sets } (\text{completion } M)$  and main_part  $M\ A \in \text{null\_sets } M$ 
    by (auto simp: null_sets_def)
  moreover obtain  $N$  where  $N \in \text{null\_sets } M$  null_part  $M\ A \subseteq N$ 
    using null_part[OF A] by auto
  ultimately show  $\exists N' \in \text{null\_sets } M. A \subseteq N'$ 
  proof (intro bexI)
    show  $A \subseteq N \cup \text{main\_part } M\ A$ 
      using  $\langle \text{null\_part } M\ A \subseteq N \rangle$  by (subst main_part_null_part_Un[OF A, symmetric]) auto
    qed auto
  next
    fix  $N$  assume  $N \in \text{null\_sets } M\ A \subseteq N$ 
    then have  $A \in \text{sets } (\text{completion } M)$  and  $N: N \in \text{sets } M\ A \subseteq N \text{ emeasure } M\ N = 0$ 
      by (auto intro: null_sets_completion)
    moreover have  $\text{emeasure } (\text{completion } M)\ A = 0$ 
      using  $N$  by (intro emeasure_eq_0[of  $N\ A$ ]) auto
    ultimately show  $A \in \text{null\_sets } (\text{completion } M)$ 
      by auto
  qed

```

lemma *null_sets_completion_subset*:

$B \subseteq A \implies A \in \text{null\_sets}(\text{completion } M) \implies B \in \text{null\_sets}(\text{completion } M)$

unfolding *null_sets_completion_iff2* **by** *auto*

interpretation *completion*: *complete_measure completion M for M*

proof

show $B \subseteq A \implies A \in \text{null\_sets}(\text{completion } M) \implies B \in \text{sets}(\text{completion } M)$

for *B A*

using *null_sets_completion_subset[of B A M]* **by** (*simp add: null_sets_def*)

qed

lemma *null_sets_restrict_space*:

$\Omega \in \text{sets } M \implies A \in \text{null\_sets}(\text{restrict\_space } M \ \Omega) \longleftrightarrow A \subseteq \Omega \wedge A \in \text{null\_sets } M$

by (*auto simp: null_sets_def emeasure_restrict_space sets_restrict_space*)

lemma *completion_ex_borel_measurable_real*:

fixes *g :: 'a $\Rightarrow$ real*

assumes *g*: $g \in \text{borel\_measurable}(\text{completion } M)$

shows $\exists g' \in \text{borel\_measurable } M. (AE \ x \text{ in } M. g \ x = g' \ x)$

proof –

have $(\lambda x. \text{ennreal}(g \ x)) \in \text{completion } M \rightarrow_M \text{borel} \ (\lambda x. \text{ennreal}(-g \ x)) \in \text{completion } M \rightarrow_M \text{borel}$

using *g* **by** *auto*

from *this[THEN completion_ex_borel_measurable]*

obtain *pf nf :: 'a $\Rightarrow$ ennreal*

where [*measurable*]: $nf \in M \rightarrow_M \text{borel}$ $pf \in M \rightarrow_M \text{borel}$

and *ae*: $AE \ x \text{ in } M. pf \ x = \text{ennreal}(g \ x) \ AE \ x \text{ in } M. nf \ x = \text{ennreal}(-g \ x)$

by (*auto simp: eq_commute*)

then have $AE \ x \text{ in } M. pf \ x = \text{ennreal}(g \ x) \wedge nf \ x = \text{ennreal}(-g \ x)$

by *auto*

then obtain *N* **where** $N \in \text{null\_sets } M \ \{x \in \text{space } M. pf \ x \neq \text{ennreal}(g \ x) \wedge nf \ x \neq \text{ennreal}(-g \ x)\} \subseteq N$

by (*auto elim!: AE_E*)

show *?thesis*

proof

let *?F* = $\lambda x. \text{indicator}(\text{space } M - N) \ x * (\text{enn2real}(pf \ x) - \text{enn2real}(nf \ x))$

show *?F* $\in M \rightarrow_M \text{borel}$

using $\langle N \in \text{null\_sets } M \rangle$ **by** *auto*

show $AE \ x \text{ in } M. g \ x = ?F \ x$

using $\langle N \in \text{null\_sets } M \rangle$ [*THEN AE_not_in*] *ae AE_space*

apply *eventually_elim*

subgoal for *x*

by (*cases 0::real g x rule: linorder_le_cases*) (*auto simp: ennreal_neg*)

done

qed

qed

lemma *simple_function_completion*: $\text{simple\_function } M \ f \implies \text{simple\_function}$

(completion M) f
 by (simp add: simple_function_def)

lemma simple_integral_completion:
 simple_function M f $\implies$ simple_integral (completion M) f = simple_integral M f
 unfolding simple_integral_def by simp

lemma nn_integral_completion: nn_integral (completion M) f = nn_integral M f
 unfolding nn_integral_def
proof (safe intro!: SUP_eq)
 fix s assume s: simple_function (completion M) s and s $\leq$ f
 then obtain s' where s': simple_function M s' AE x in M. s x = s' x
 by (auto dest: completion_ex_simple_function)
 then obtain N where N: N $\in$ null_sets M {x $\in$ space M. s x $\neq$ s' x} $\subseteq$ N
 by (auto elim!: AE_E)
 then have ae_N: AE x in M. (s x $\neq$ s' x $\longrightarrow$ x $\in$ N) $\wedge$ x $\notin$ N
 by (auto dest: AE_not_in)
 define s'' where s'' x = (if x $\in$ N then 0 else s x) for x
 then have ae_s_eq_s'': AE x in completion M. s x = s'' x
 using s' ae_N by (intro AE_completion) auto
 have s'': simple_function M s''
proof (subst simple_function_cong)
 show t $\in$ space M $\implies$ s'' t = (if t $\in$ N then 0 else s' t) for t
 using N by (auto simp: s''_def dest: sets.sets_into_space)
 show simple_function M (λt . if t $\in$ N then 0 else s' t)
 unfolding s''_def[abs_def] using N by (auto intro!: simple_function_if s')
qed

show $\exists j \in \{g. \text{simple\_function } M \ g \wedge g \leq f\}. \text{integral}^S (\text{completion } M) \ s \leq \text{integral}^S M \ j$
proof (safe intro!: bexI[of _ s''])
 have integral^S (completion M) s = integral^S (completion M) s''
 by (intro simple_integral_cong_AE s simple_function_completion s'' ae_s_eq_s'')
 then show integral^S (completion M) s $\leq$ integral^S M s''
 using s'' by (simp add: simple_integral_completion)
 from $\langle s \leq f \rangle$ show s'' $\leq$ f
 unfolding s''_def le_fun_def by auto
qed fact

next
 fix s assume simple_function M s s $\leq$ f
 then show $\exists j \in \{g. \text{simple\_function } (\text{completion } M) \ g \wedge g \leq f\}. \text{integral}^S M \ s \leq \text{integral}^S (\text{completion } M) \ j$
 by (intro bexI[of _ s]) (auto simp: simple_integral_completion simple_function_completion)
qed

lemma integrable_completion:
 fixes f :: 'a $\Rightarrow$ 'b::{banach, second_countable_topology}

shows $f \in M \rightarrow_M \text{borel} \implies \text{integrable } (\text{completion } M) f \longleftrightarrow \text{integrable } M f$
by (rule integrable_subalgebra[symmetric]) auto

lemma integral_completion:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second\_countable\_topology}\}$

assumes $f: f \in M \rightarrow_M \text{borel}$ **shows** $\text{integral}^L (\text{completion } M) f = \text{integral}^L M f$

by (rule integrable_subalgebra[symmetric]) (auto intro: f)

locale semifinite_measure =

fixes $M :: 'a \text{ measure}$

assumes semifinite:

$\bigwedge A. A \in \text{sets } M \implies \text{emeasure } M A = \infty \implies \exists B \in \text{sets } M. B \subseteq A \wedge \text{emeasure } M B < \infty$

locale locally_determined_measure = semifinite_measure +

assumes locally_determined:

$\bigwedge A. A \subseteq \text{space } M \implies (\bigwedge B. B \in \text{sets } M \implies \text{emeasure } M B < \infty \implies A \cap B \in \text{sets } M) \implies A \in \text{sets } M$

locale cld_measure =

$\text{complete\_measure } M + \text{locally\_determined\_measure } M$ **for** $M :: 'a \text{ measure}$

definition outer_measure_of :: $'a \text{ measure} \Rightarrow 'a \text{ set} \Rightarrow \text{ennreal}$

where $\text{outer\_measure\_of } M A = (\text{INF } B \in \{B \in \text{sets } M. A \subseteq B\}. \text{emeasure } M B)$

lemma outer_measure_of_eq[simp]: $A \in \text{sets } M \implies \text{outer\_measure\_of } M A = \text{emeasure } M A$

by (auto simp: outer_measure_of_def intro!: INF_eqI emeasure_mono)

lemma outer_measure_of_mono: $A \subseteq B \implies \text{outer\_measure\_of } M A \leq \text{outer\_measure\_of } M B$

unfolding outer_measure_of_def **by** (intro INF_superset_mono) auto

lemma outer_measure_of_attain:

assumes $A \subseteq \text{space } M$

shows $\exists E \in \text{sets } M. A \subseteq E \wedge \text{outer\_measure\_of } M A = \text{emeasure } M E$

proof –

have $\text{emeasure } M \text{ ‘ } \{B \in \text{sets } M. A \subseteq B\} \neq \{\}$

using $\langle A \subseteq \text{space } M \rangle$ **by** auto

from ennreal_Inf_countable_INF[OF this]

obtain f

where $f: \text{range } f \subseteq \text{emeasure } M \text{ ‘ } \{B \in \text{sets } M. A \subseteq B\}$ decseq f

and $\text{outer\_measure\_of } M A = (\text{INF } i. f i)$

unfolding outer_measure_of_def **by** auto

have $\exists E. \forall n. (E n \in \text{sets } M \wedge A \subseteq E n \wedge \text{emeasure } M (E n) \leq f n) \wedge E (\text{Suc } n) \subseteq E n$

proof (rule dependent_nat_choice)

```

show  $\exists x. x \in \text{sets } M \wedge A \subseteq x \wedge \text{emeasure } M x \leq f 0$ 
  using  $f(1)$  by (fastforce simp: image_subset_iff image_iff intro: eq_refl[OF
sym])
next
  fix  $E n$  assume  $E \in \text{sets } M \wedge A \subseteq E \wedge \text{emeasure } M E \leq f n$ 
  moreover obtain  $F$  where  $F \in \text{sets } M \wedge A \subseteq F \wedge f(\text{Suc } n) = \text{emeasure } M F$ 
    using  $f(1)$  by (auto simp: image_subset_iff image_iff)
  ultimately show  $\exists y. (y \in \text{sets } M \wedge A \subseteq y \wedge \text{emeasure } M y \leq f(\text{Suc } n)) \wedge$ 
 $y \subseteq E$ 
    by (auto intro!: exI[of _  $F \cap E$ ] emeasure_mono)
qed
then obtain  $E$ 
  where [simp]:  $\bigwedge n. E n \in \text{sets } M$ 
    and  $\bigwedge n. A \subseteq E n$ 
    and  $\text{le}_f: \bigwedge n. \text{emeasure } M (E n) \leq f n$ 
    and  $\text{decseq } E$ 
    by (auto simp: decseq_Suc_iff)
show ?thesis
proof cases
  assume  $\text{fin}: \exists i. \text{emeasure } M (E i) < \infty$ 
  show ?thesis
  proof (intro beI[of _  $\bigcap i. E i$ ] conjI)
    show  $A \subseteq (\bigcap i. E i) \wedge (\bigcap i. E i) \in \text{sets } M$ 
      using  $\langle \bigwedge n. A \subseteq E n \rangle$  by auto
    have  $(\text{INF } i. \text{emeasure } M (E i)) \leq \text{outer\_measure\_of } M A$ 
      unfolding  $\langle \text{outer\_measure\_of } M A = (\text{INF } n. f n) \rangle$ 
      by (intro INF_superset_mono le_f) auto
    moreover have  $\text{outer\_measure\_of } M A \leq (\text{INF } i. \text{outer\_measure\_of } M (E$ 
 $i))$ 
      by (intro INF_greatest outer_measure_of_mono  $\langle \bigwedge n. A \subseteq E n \rangle$ )
    ultimately have  $\text{outer\_measure\_of } M A = (\text{INF } i. \text{emeasure } M (E i))$ 
      by auto
    also have  $\dots = \text{emeasure } M (\bigcap i. E i)$ 
      using fin by (intro INF_emeasure_decseq'  $\langle \text{decseq } E \rangle$ ) (auto simp: less_top)
    finally show  $\text{outer\_measure\_of } M A = \text{emeasure } M (\bigcap i. E i)$  .
  qed
next
  assume  $\nexists i. \text{emeasure } M (E i) < \infty$ 
  then have  $f n = \infty$  for  $n$ 
    using le_f by (auto simp: not_less top_unique)
  moreover have  $\exists E \in \text{sets } M. A \subseteq E \wedge f 0 = \text{emeasure } M E$ 
    using f by auto
  ultimately show ?thesis
    unfolding  $\langle \text{outer\_measure\_of } M A = (\text{INF } n. f n) \rangle$  by simp
qed
qed

```

lemma SUP_outer_measure_of_incseq:

assumes $A: \bigwedge n. A\ n \subseteq \text{space } M$ **and** $\text{incseq } A$
shows $(\text{SUP } n. \text{outer\_measure\_of } M\ (A\ n)) = \text{outer\_measure\_of } M\ (\bigcup i. A\ i)$
proof (*rule antisym*)
obtain E
where $E: \bigwedge n. E\ n \in \text{sets } M \bigwedge n. A\ n \subseteq E\ n \bigwedge n. \text{outer\_measure\_of } M\ (A\ n)$
 $= \text{emeasure } M\ (E\ n)$
using $\text{outer\_measure\_of\_attain}[OF\ A]$ **by** *metis*

define F **where** $F\ n = (\bigcap i \in \{n \dots\}. E\ i)$ **for** n
with E **have** $F: \text{incseq } F \bigwedge n. F\ n \in \text{sets } M$
by (*auto simp: incseq_def*)
have $A\ n \subseteq F\ n$ **for** n
using $\text{incseqD}[OF\ \langle \text{incseq } A \rangle, \text{ of } n] \langle \bigwedge n. A\ n \subseteq E\ n \rangle$ **by** (*auto simp: F_def*)

have $\text{eq: outer\_measure\_of } M\ (A\ n) = \text{outer\_measure\_of } M\ (F\ n)$ **for** n
proof (*intro antisym*)
have $\text{outer\_measure\_of } M\ (F\ n) \leq \text{outer\_measure\_of } M\ (E\ n)$
by (*intro outer_measure_of_mono*) (*auto simp add: F_def*)
with E **show** $\text{outer\_measure\_of } M\ (F\ n) \leq \text{outer\_measure\_of } M\ (A\ n)$
by *auto*
show $\text{outer\_measure\_of } M\ (A\ n) \leq \text{outer\_measure\_of } M\ (F\ n)$
by (*intro outer_measure_of_mono*) ($\langle A\ n \subseteq F\ n \rangle$)
qed

have $\text{outer\_measure\_of } M\ (\bigcup n. A\ n) \leq \text{outer\_measure\_of } M\ (\bigcup n. F\ n)$
using $\langle \bigwedge n. A\ n \subseteq F\ n \rangle$ **by** (*intro outer_measure_of_mono*) *auto*
also have $\dots = (\text{SUP } n. \text{emeasure } M\ (F\ n))$
using F **by** (*simp add: SUP_emeasure_incseq_subset_eq*)
finally show $\text{outer\_measure\_of } M\ (\bigcup n. A\ n) \leq (\text{SUP } n. \text{outer\_measure\_of } M\ (A\ n))$
by (*simp add: eq F*)
qed (*auto intro: SUP_least outer_measure_of_mono*)

definition $\text{measurable\_envelope} :: 'a\ \text{measure} \Rightarrow 'a\ \text{set} \Rightarrow 'a\ \text{set} \Rightarrow \text{bool}$
where $\text{measurable\_envelope } M\ A\ E \longleftrightarrow$
 $(A \subseteq E \wedge E \in \text{sets } M \wedge (\forall F \in \text{sets } M. \text{emeasure } M\ (F \cap E) = \text{outer\_measure\_of } M\ (F \cap A)))$

lemma $\text{measurable\_envelopeD}$:
assumes $\text{measurable\_envelope } M\ A\ E$
shows $A \subseteq E$
and $E \in \text{sets } M$
and $\bigwedge F. F \in \text{sets } M \implies \text{emeasure } M\ (F \cap E) = \text{outer\_measure\_of } M\ (F \cap A)$
and $A \subseteq \text{space } M$
using $\text{assms sets.sets\_into\_space}[of\ E]$ **by** (*auto simp: measurable_envelope_def*)

lemma $\text{measurable\_envelopeD1}$:
assumes $E: \text{measurable\_envelope } M\ A\ E$ **and** $F: F \in \text{sets } M\ F \subseteq E - A$


```

  shows  $\text{emeasure } M F = 0$ 
proof -
  have  $\text{emeasure } M F = \text{emeasure } M (F \cap E)$ 
    using  $F$  by (intro arg_cong2[where f= $\text{emeasure}$ ]) auto
  also have  $\dots = \text{outer\_measure\_of } M (F \cap A)$ 
    using measurable_envelopeD[OF  $E$ ]  $\langle F \in \text{sets } M \rangle$  by (auto simp: measurable_envelope_def)
  also have  $\dots = \text{outer\_measure\_of } M \{\}$ 
    using  $\langle F \subseteq E - A \rangle$  by (intro arg_cong2[where f= $\text{outer\_measure\_of}$ ]) auto
  finally show  $\text{emeasure } M F = 0$ 
    by simp
qed

lemma measurable_envelope_eq1:
  assumes  $A \subseteq E$   $E \in \text{sets } M$ 
  shows  $\text{measurable\_envelope } M A E \longleftrightarrow (\forall F \in \text{sets } M. F \subseteq E - A \longrightarrow \text{emeasure } M F = 0)$ 
proof safe
  assume *:  $\forall F \in \text{sets } M. F \subseteq E - A \longrightarrow \text{emeasure } M F = 0$ 
  show  $\text{measurable\_envelope } M A E$ 
    unfolding measurable_envelope_def
  proof (rule ccontr, auto simp add:  $\langle E \in \text{sets } M \rangle \langle A \subseteq E \rangle$ )
    fix  $F$  assume  $F \in \text{sets } M$   $\text{emeasure } M (F \cap E) \neq \text{outer\_measure\_of } M (F \cap A)$ 
    then have  $\text{outer\_measure\_of } M (F \cap A) < \text{emeasure } M (F \cap E)$ 
      using outer_measure_of_mono[of  $F \cap A$   $F \cap E$   $M$ ]  $\langle A \subseteq E \rangle \langle E \in \text{sets } M \rangle$ 
    by (auto simp: less_le)
    then obtain  $G$  where  $G: G \in \text{sets } M$   $F \cap A \subseteq G$  and less:  $\text{emeasure } M G < \text{emeasure } M (E \cap F)$ 
      unfolding outer_measure_of_def INF_less_iff by (auto simp: ac_simps)
    have le:  $\text{emeasure } M (G \cap E \cap F) \leq \text{emeasure } M G$ 
      using  $\langle E \in \text{sets } M \rangle \langle G \in \text{sets } M \rangle \langle F \in \text{sets } M \rangle$  by (auto intro!: measure_mono)

    from  $G$  have  $E \cap F - G \in \text{sets } M$   $E \cap F - G \subseteq E - A$ 
      using  $\langle F \in \text{sets } M \rangle \langle E \in \text{sets } M \rangle$  by auto
    with * have  $0 = \text{emeasure } M (E \cap F - G)$ 
      by auto
    also have  $E \cap F - G = E \cap F - (G \cap E \cap F)$ 
      by auto
    also have  $\text{emeasure } M (E \cap F - (G \cap E \cap F)) = \text{emeasure } M (E \cap F) - \text{emeasure } M (G \cap E \cap F)$ 
      using  $\langle E \in \text{sets } M \rangle \langle F \in \text{sets } M \rangle$  le less  $G$  by (intro emeasure_Diff) (auto simp: top_unique)
    also have  $\dots > 0$ 
      using le less by (intro diff_gr0_enreal) auto
    finally show False by auto
  qed
qed (rule measurable_envelopeD1)

```

lemma *measurable_envelopeD2*:

assumes E : *measurable_envelope* M A E **shows** $\text{emeasure } M \ E = \text{outer\_measure\_of } M \ A$

proof –

from $\langle \text{measurable\_envelope } M \ A \ E \rangle$ **have** $\text{emeasure } M \ (E \cap E) = \text{outer\_measure\_of } M \ (E \cap A)$

by (*auto simp: measurable_envelope_def*)

with *measurable_envelopeD*[*OF E*] **show** $\text{emeasure } M \ E = \text{outer\_measure\_of } M \ A$

by (*auto simp: Int_absorb1*)

qed

lemma *measurable_envelope_eq2*:

assumes $A \subseteq E$ $E \in \text{sets } M$ $\text{emeasure } M \ E < \infty$

shows *measurable_envelope* M A $E \longleftrightarrow (\text{emeasure } M \ E = \text{outer\_measure\_of } M \ A)$

proof *safe*

assume $*$: $\text{emeasure } M \ E = \text{outer\_measure\_of } M \ A$

show *measurable_envelope* M A E

unfolding *measurable_envelope_eq1*[*OF* $\langle A \subseteq E \rangle \langle E \in \text{sets } M \rangle$]

proof (*intro conjI ballI impI assms*)

fix F **assume** F : $F \in \text{sets } M$ $F \subseteq E - A$

with $\langle E \in \text{sets } M \rangle$ **have** le : $\text{emeasure } M \ F \leq \text{emeasure } M \ E$

by (*intro emeasure_mono*) *auto*

from $F \langle A \subseteq E \rangle$ **have** $\text{outer\_measure\_of } M \ A \leq \text{outer\_measure\_of } M \ (E - F)$

by (*intro outer_measure_of_mono*) *auto*

then have $\text{emeasure } M \ E - 0 \leq \text{emeasure } M \ (E - F)$

using $*$ $\langle E \in \text{sets } M \rangle \langle F \in \text{sets } M \rangle$ **by** *simp*

also have $\dots = \text{emeasure } M \ E - \text{emeasure } M \ F$

using $\langle E \in \text{sets } M \rangle \langle \text{emeasure } M \ E < \infty \rangle F \ le$ **by** (*intro emeasure_Diff*)
(*auto simp: top_unique*)

finally show $\text{emeasure } M \ F = 0$

using *ennreal_mono_minus_cancel*[*of* $\text{emeasure } M \ E \ 0 \ \text{emeasure } M \ F$] le
assms **by** *auto*

qed

qed (*auto intro: measurable_envelopeD2*)

lemma *measurable_envelopeI_countable*:

fixes $A :: \text{nat} \Rightarrow 'a \text{ set}$

assumes E : $\bigwedge n. \text{measurable\_envelope } M \ (A \ n) \ (E \ n)$

shows *measurable_envelope* $M \ (\bigcup n. A \ n) \ (\bigcup n. E \ n)$

proof (*subst measurable_envelope_eq1*)

show $(\bigcup n. A \ n) \subseteq (\bigcup n. E \ n) \ (\bigcup n. E \ n) \in \text{sets } M$

using *measurable_envelopeD*(1,2)[*OF E*] **by** *auto*

show $\forall F \in \text{sets } M. F \subseteq (\bigcup n. E \ n) - (\bigcup n. A \ n) \longrightarrow \text{emeasure } M \ F = 0$

proof *safe*

fix F **assume** F : $F \in \text{sets } M$ $F \subseteq (\bigcup n. E \ n) - (\bigcup n. A \ n)$

```

    then have  $F \cap E\ n \in \text{sets } M$   $F \cap E\ n \subseteq E\ n - A\ n$   $F \subseteq (\bigcup n. E\ n)$  for  $n$ 
    using measurable_envelopeD(1,2)[OF E] by auto
    then have  $\text{emeasure } M (\bigcup n. F \cap E\ n) = 0$ 
    by (intro emeasure_UN_eq_0 measurable_envelopeD1[OF E]) auto
    then show  $\text{emeasure } M\ F = 0$ 
    using  $\langle F \subseteq (\bigcup n. E\ n) \rangle$  by (auto simp: Int_absorb2)
qed
qed

lemma measurable_envelopeI_countable_cover:
  fixes  $A$  and  $C :: \text{nat} \Rightarrow 'a \text{ set}$ 
  assumes  $C: A \subseteq (\bigcup n. C\ n) \wedge n. C\ n \in \text{sets } M \wedge n. \text{emeasure } M (C\ n) < \infty$ 
  shows  $\exists E \subseteq (\bigcup n. C\ n). \text{measurable\_envelope } M\ A\ E$ 
proof -
  have  $A \cap C\ n \subseteq \text{space } M$  for  $n$ 
  using  $\langle C\ n \in \text{sets } M \rangle$  by (auto dest: sets.sets_into_space)
  then have  $\forall n. \exists E \in \text{sets } M. A \cap C\ n \subseteq E \wedge \text{outer\_measure\_of } M (A \cap C\ n)$ 
  =  $\text{emeasure } M\ E$ 
  using outer_measure_of_attain[of  $A \cap C\ n$   $M$  for  $n$ ] by auto
  then obtain  $E$ 
  where  $E: \wedge n. E\ n \in \text{sets } M \wedge n. A \cap C\ n \subseteq E\ n$ 
  and  $\text{eq}: \wedge n. \text{outer\_measure\_of } M (A \cap C\ n) = \text{emeasure } M (E\ n)$ 
  by metis

  have  $\text{outer\_measure\_of } M (A \cap C\ n) \leq \text{outer\_measure\_of } M (E\ n \cap C\ n)$  for
   $n$ 
  using  $E$  by (intro outer_measure_of_mono) auto
  moreover have  $\text{outer\_measure\_of } M (E\ n \cap C\ n) \leq \text{outer\_measure\_of } M (E\ n)$  for  $n$ 
  by (intro outer_measure_of_mono) auto
  ultimately have  $\text{eq}: \text{outer\_measure\_of } M (A \cap C\ n) = \text{emeasure } M (E\ n \cap C\ n)$  for  $n$ 
  using  $E\ C$  by (intro antisym) (auto simp: eq)

  { fix  $n$ 
    have  $\text{outer\_measure\_of } M (A \cap C\ n) \leq \text{outer\_measure\_of } M (C\ n)$ 
    by (intro outer_measure_of_mono) simp
    also have  $\dots < \infty$ 
    using  $\text{assms}$  by auto
    finally have  $\text{emeasure } M (E\ n \cap C\ n) < \infty$ 
    using  $\text{eq}$  by simp }
  then have measurable_envelope  $M (\bigcup n. A \cap C\ n) (\bigcup n. E\ n \cap C\ n)$ 
  using  $E\ C$  by (intro measurable_envelopeI_countable measurable_envelope_eq2[THEN
  iffD2]) (auto simp: eq)
  with  $\langle A \subseteq (\bigcup n. C\ n) \rangle$  show ?thesis
  by (intro exI[of _  $(\bigcup n. E\ n \cap C\ n)$ ]) (auto simp add: Int_absorb2)
qed

```

lemma (in complete_measure) complete_sets_sandwich:

```

    assumes [measurable]:  $A \in \text{sets } M$   $C \in \text{sets } M$  and  $\text{subset}: A \subseteq B$   $B \subseteq C$ 
    and  $\text{measure}: \text{emeasure } M A = \text{emeasure } M C$   $\text{emeasure } M A < \infty$ 
    shows  $B \in \text{sets } M$ 
  proof -
    have  $B - A \in \text{sets } M$ 
    proof (rule complete)
      show  $B - A \subseteq C - A$ 
      using subset by auto
      show  $C - A \in \text{null\_sets } M$ 
      using measure subset by (simp add: emeasure_Diff null_setsI)
    qed
    then have  $A \cup (B - A) \in \text{sets } M$ 
    by measurable
    also have  $A \cup (B - A) = B$ 
    using  $\langle A \subseteq B \rangle$  by auto
    finally show ?thesis .
  qed

lemma (in complete_measure) complete_sets_sandwich_fmeasurable:
  assumes [measurable]:  $A \in \text{fmeasurable } M$   $C \in \text{fmeasurable } M$  and  $\text{subset}: A \subseteq B$   $B \subseteq C$ 
  and  $\text{measure}: \text{measure } M A = \text{measure } M C$ 
  shows  $B \in \text{fmeasurable } M$ 
proof (rule fmeasurableI2)
  show  $B \subseteq C$   $C \in \text{fmeasurable } M$  by fact+
  show  $B \in \text{sets } M$ 
  proof (rule complete_sets_sandwich)
    show  $A \in \text{sets } M$   $C \in \text{sets } M$   $A \subseteq B$   $B \subseteq C$ 
    using assms by auto
    show  $\text{emeasure } M A < \infty$ 
    using  $\langle A \in \text{fmeasurable } M \rangle$  by (auto simp: fmeasurable_def)
    show  $\text{emeasure } M A = \text{emeasure } M C$ 
    using assms by (simp add: emeasure_eq_measure2)
  qed
qed

lemma AE_completion_iff:  $(\text{AE } x \text{ in completion } M. P x) \longleftrightarrow (\text{AE } x \text{ in } M. P x)$ 
proof
  assume  $\text{AE } x \text{ in completion } M. P x$ 
  then obtain  $N$  where  $N \in \text{null\_sets } (\text{completion } M)$  and  $P: \{x \in \text{space } M. \neg P x\} \subseteq N$ 
  unfolding eventually_ae_filter by auto
  then obtain  $N'$  where  $N': N' \in \text{null\_sets } M$  and  $N \subseteq N'$ 
  unfolding null_sets_completion_iff2 by auto
  from  $P$   $\langle N \subseteq N' \rangle$  have  $\{x \in \text{space } M. \neg P x\} \subseteq N'$ 
  by auto
  with  $N'$  show  $\text{AE } x \text{ in } M. P x$ 
  unfolding eventually_ae_filter by auto
qed (rule AE_completion)

```

lemma *null_part_null_sets*: $S \in \text{completion } M \implies \text{null\_part } M S \in \text{null\_sets}$
 (*completion M*)
by (*auto dest!: null_part intro: null_sets_completionI null_sets_completion_subset*)

lemma *AE_notin_null_part*: $S \in \text{completion } M \implies (AE\ x\ \text{in } M. x \notin \text{null\_part } M S)$
by (*auto dest!: null_part_null_sets AE_not_in simp: AE_completion_iff*)

lemma *completion_upper*:
assumes *A*: $A \in \text{sets } (\text{completion } M)$
shows $\exists A' \in \text{sets } M. A \subseteq A' \wedge \text{emeasure } (\text{completion } M) A = \text{emeasure } M A'$
proof –
from *AE_notin_null_part*[*OF A*] **obtain** *N* **where** *N*: $N \in \text{null\_sets } M$ *null_part*
 $M A \subseteq N$
unfolding *eventually_ae_filter* **using** *null_part_null_sets*[*OF A*, *THEN null_setsD2*,
THEN sets.sets_into_space] **by** *auto*
show ?thesis
proof (*intro bexI conjI*)
show $A \subseteq \text{main\_part } M A \cup N$
using $\langle \text{null\_part } M A \subseteq N \rangle$ **by** (*subst main_part_null_part_Un[symmetric,*
OF A]) *auto*
show $\text{emeasure } (\text{completion } M) A = \text{emeasure } M (\text{main\_part } M A \cup N)$
using $A \in \text{null\_sets } M$ **by** (*simp add: emeasure_Un_null_set*)
qed (*use A N in auto*)
qed

lemma *AE_in_main_part*:
assumes *A*: $A \in \text{completion } M$ **shows** $AE\ x\ \text{in } M. x \in \text{main\_part } M A \longleftrightarrow x \in A$
using *AE_notin_null_part*[*OF A*]
by (*subst* (2) *main_part_null_part_Un[symmetric, OF A]*) *auto*

lemma *completion_density_eq*:
assumes *ae*: $AE\ x\ \text{in } M. 0 < f\ x$ **and** [*measurable*]: $f \in M \rightarrow_M \text{borel}$
shows $\text{completion } (\text{density } M f) = \text{density } (\text{completion } M) f$
proof (*intro measure_eqI*)
have $N' \in \text{sets } M \wedge (AE\ x \in N' \text{ in } M. f\ x = 0) \longleftrightarrow N' \in \text{null\_sets } M$ **for** *N'*
proof *safe*
assume *N'*: $N' \in \text{sets } M$ **and** *ae_N'*: $AE\ x \in N' \text{ in } M. f\ x = 0$
from *ae_N'* *ae* **have** $AE\ x\ \text{in } M. x \notin N'$
by *eventually_elim auto*
then show $N' \in \text{null\_sets } M$
using *N'* **by** (*simp add: AE_iff_null_sets*)
next
assume *N'*: $N' \in \text{null\_sets } M$ **then show** $N' \in \text{sets } M$ $AE\ x \in N' \text{ in } M. f\ x = 0$
using *ae AE_not_in*[*OF N'*] **by** (*auto simp: less_le*)
qed

```

then show sets_eq: sets (completion (density M f)) = sets (density (completion
M) f)
  by (simp add: sets_completion null_sets_density_iff)

fix A assume A:  $\langle A \in \text{completion } (\text{density } M f) \rangle$ 
moreover
  have A  $\in \text{completion } M$ 
    using A unfolding sets_eq by simp
moreover
  have main_part (density M f) A  $\in M$ 
    using A main_part_sets[of A density M f] unfolding sets_density sets_eq by
simp
  moreover have AE x in M.  $x \in \text{main\_part } (\text{density } M f) A \longleftrightarrow x \in A$ 
    using AE_in_main_part[OF  $\langle A \in \text{completion } (\text{density } M f) \rangle$ ] ae by (auto
simp add: AE_density)
  ultimately show emeasure (completion (density M f)) A = emeasure (density
(completion M) f) A
    by (auto simp add: emeasure_density measurable_completion nn_integral_completion
intro!: nn_integral_cong_AE)
qed

lemma null_sets_subset:  $B \in \text{null\_sets } M \implies A \in \text{sets } M \implies A \subseteq B \implies A \in$ 
null_sets M
  using emeasure_mono[of A B M] by (simp add: null_sets_def)

lemma (in complete_measure) complete2:  $A \subseteq B \implies B \in \text{null\_sets } M \implies A \in$ 
null_sets M
  using complete[of A B] null_sets_subset[of B M A] by simp

lemma (in complete_measure) AE_iff_null_sets:  $(AE x \text{ in } M. P x) \longleftrightarrow \{x \in \text{space}$ 
M.  $\neg P x\} \in \text{null\_sets } M$ 
  unfolding eventually_ae_filter by (auto intro: complete2)

lemma (in complete_measure) null_sets_iff_AE:  $A \in \text{null\_sets } M \longleftrightarrow ((AE x$ 
in M.  $x \notin A) \wedge A \subseteq \text{space } M)$ 
  unfolding AE_iff_null_sets by (auto cong: rev_conj_cong dest: sets_sets_into_space
simp: subset_eq)

lemma (in complete_measure) in_sets_AE:
  assumes ae:  $AE x \text{ in } M. x \in A \longleftrightarrow x \in B$  and A:  $A \in \text{sets } M$  and B:  $B \subseteq$ 
space M
  shows B  $\in \text{sets } M$ 
proof –
  have  $(AE x \text{ in } M. x \notin B - A \wedge x \notin A - B)$ 
    using ae by eventually_elim auto
  then have B - A  $\in \text{null\_sets } M$   $A - B \in \text{null\_sets } M$ 
    using A B unfolding null_sets_iff_AE by (auto dest: sets_sets_into_space)
  then have  $A \cup (B - A) - (A - B) \in \text{sets } M$ 
    using A by blast

```

also have $A \cup (B - A) - (A - B) = B$
 by auto
 finally show $B \in \text{sets } M$.
 qed

lemma (in complete_measure) vimage_null_part_null_sets:
 assumes $f: f \in M \rightarrow_M N$ and $\text{eq}: \text{null\_sets } N \subseteq \text{null\_sets } (\text{distr } M \ N \ f)$
 and $A \in \text{completion } N$
 shows $f - ' \text{null\_part } N \ A \cap \text{space } M \in \text{null\_sets } M$
 proof -
 obtain N' where $N' \in \text{null\_sets } N$ $\text{null\_part } N \ A \subseteq N'$
 using $\text{null\_part}[OF \ A]$ by auto
 then have $N': N' \in \text{null\_sets } (\text{distr } M \ N \ f)$
 using eq by auto
 show ?thesis
 proof (rule complete2)
 show $f - ' \text{null\_part } N \ A \cap \text{space } M \subseteq f - ' N' \cap \text{space } M$
 using $\langle \text{null\_part } N \ A \subseteq N' \rangle$ by auto
 show $f - ' N' \cap \text{space } M \in \text{null\_sets } M$
 using $f \ N'$ by (auto simp: null_sets_def emeasure_distr)
 qed
 qed

lemma (in complete_measure) vimage_null_part_sets:
 $f \in M \rightarrow_M N \implies \text{null\_sets } N \subseteq \text{null\_sets } (\text{distr } M \ N \ f) \implies A \in \text{completion } N \implies$
 $f - ' \text{null\_part } N \ A \cap \text{space } M \in \text{sets } M$
 using vimage_null_part_null_sets[of $f \ N \ A$] by auto

lemma (in complete_measure) measurable_completion2:
 assumes $f: f \in M \rightarrow_M N$ and $\text{null\_sets}: \text{null\_sets } N \subseteq \text{null\_sets } (\text{distr } M \ N \ f)$
 shows $f \in M \rightarrow_M \text{completion } N$
 proof (rule measurableI)
 show $x \in \text{space } M \implies f \ x \in \text{space } (\text{completion } N)$ for x
 using $f[THEN \text{measurable\_space}]$ by auto
 fix A assume $A: A \in \text{sets } (\text{completion } N)$
 have $f - ' A \cap \text{space } M = (f - ' \text{main\_part } N \ A \cap \text{space } M) \cup (f - ' \text{null\_part } N \ A \cap \text{space } M)$
 using $\text{main\_part\_null\_part\_Un}[OF \ A]$ by auto
 then show $f - ' A \cap \text{space } M \in \text{sets } M$
 using $f \ A \ \text{null\_sets}$ by (auto intro: vimage_null_part_sets measurable_sets)
 qed

lemma (in complete_measure) completion_distr_eq:
 assumes $X: X \in M \rightarrow_M N$ and $\text{null\_sets}: \text{null\_sets } (\text{distr } M \ N \ X) = \text{null\_sets } N$
 shows $\text{completion } (\text{distr } M \ N \ X) = \text{distr } M \ (\text{completion } N) \ X$
 proof (rule measure_eqI)

```

show eq: sets (completion (distr M N X)) = sets (distr M (completion N) X)
  by (simp add: sets_completion null_sets)

fix A assume A: A ∈ completion (distr M N X)
moreover have A': A ∈ completion N
  using A by (simp add: eq)
moreover have main_part (distr M N X) A ∈ sets N
  using main_part_sets[OF A] by simp
moreover have X - ' A ∩ space M = (X - ' main_part (distr M N X) A ∩
space M) ∪ (X - ' null_part (distr M N X) A ∩ space M)
  using main_part_null_part_Un[OF A] by auto
moreover have X - ' null_part (distr M N X) A ∩ space M ∈ null_sets M
  using X A by (intro vimage_null_part_null_sets) (auto cong: distr_cong)
ultimately show emeasure (completion (distr M N X)) A = emeasure (distr M
(completion N) X) A
  using X by (auto simp: emeasure_distr measurable_completion null_sets mea-
surable_completion2
    intro!: emeasure_Un_null_set[symmetric])
qed

lemma distr_completion: X ∈ M →M N ⇒ distr (completion M) N X = distr
M N X
  by (intro measure_eqI) (auto simp: emeasure_distr measurable_completion)

proposition (in complete_measure) fmeasurable_inner_outer:
  S ∈ fmeasurable M ⟷
    (∀ e > 0. ∃ T ∈ fmeasurable M. ∃ U ∈ fmeasurable M. T ⊆ S ∧ S ⊆ U ∧ |measure
M T - measure M U| < e)
  (is _ ⟷ ?approx)
proof safe
  let ?μ = measure M let ?D = λ T U . |?μ T - ?μ U|
  assume ?approx
  have ∃ A. ∀ n. (fst (A n) ∈ fmeasurable M ∧ snd (A n) ∈ fmeasurable M ∧ fst
(A n) ⊆ S ∧ S ⊆ snd (A n) ∧
    ?D (fst (A n)) (snd (A n)) < 1 / Suc n) ∧ (fst (A n) ⊆ fst (A (Suc n)) ∧ snd
(A (Suc n)) ⊆ snd (A n))
    (is ∃ A. ∀ n. ?P n (A n) ∧ ?Q (A n) (A (Suc n)))
  proof (intro dependent_nat_choice)
    show ∃ A. ?P 0 A
    using ‹?approx›[THEN spec, of 1] by auto
  next
    fix A n assume ?P n A
    moreover
      from ‹?approx›[THEN spec, of 1 / Suc (Suc n)]
      obtain T U where *: T ∈ fmeasurable M U ∈ fmeasurable M T ⊆ S S ⊆ U
      ?D T U < 1 / Suc (Suc n)
      by auto
    ultimately have ?μ T ≤ ?μ (T ∪ fst A) ?μ (U ∩ snd A) ≤ ?μ U
      ?μ T ≤ ?μ U ?μ (T ∪ fst A) ≤ ?μ (U ∩ snd A)

```



```

    by (auto intro!: measure_mono_fmeasurable)
  then have ?D (T ∪ fst A) (U ∩ snd A) ≤ ?D T U
    by auto
  also have ?D T U < 1/Suc (Suc n) by fact
  finally show ∃ B. ?P (Suc n) B ∧ ?Q A B
    using ‹?P n A› *
    by (intro exI[of _ (T ∪ fst A, U ∩ snd A)] conjI) auto
qed
then obtain A
  where lm: ⋀ n. fst (A n) ∈ fmeasurable M ∧ n. snd (A n) ∈ fmeasurable M
    and set_bound: ⋀ n. fst (A n) ⊆ S ∧ n. S ⊆ snd (A n)
    and mono: ⋀ n. fst (A n) ⊆ fst (A (Suc n)) ∧ n. snd (A (Suc n)) ⊆ snd (A
n)
    and bound: ⋀ n. ?D (fst (A n)) (snd (A n)) < 1/Suc n
  by metis

have INT_sA: (⋂ n. snd (A n)) ∈ fmeasurable M
  using lm by (intro fmeasurable_INT[of _ 0]) auto
have UN_fA: (⋃ n. fst (A n)) ∈ fmeasurable M
  using lm order_trans[OF set_bound(1) set_bound(2)[of 0]] by (intro fmea-
surable_UN[of _ _ snd (A 0)]) auto

have (λ n. ?μ (fst (A n)) - ?μ (snd (A n))) ⟶ 0
  using bound
  by (subst tendsto_rabs_zero_iff[symmetric])
    (intro tendsto_sandwich[OF _ tendsto_const LIMSEQ_inverse_real_of_nat];
      auto intro!: always_eventually_less_imp_le simp: divide_inverse)
moreover
have (λ n. ?μ (fst (A n)) - ?μ (snd (A n))) ⟶ ?μ (⋃ n. fst (A n)) - ?μ
(⋂ n. snd (A n))
  proof (intro tendsto_diff Lim_measure_incseq Lim_measure_decseq)
    show range (λ i. fst (A i)) ⊆ sets M range (λ i. snd (A i)) ⊆ sets M
      incseq (λ i. fst (A i)) decseq (λ n. snd (A n))
      using mono lm by (auto simp: incseq_Suc_iff decseq_Suc_iff intro!: mea-
sure_mono_fmeasurable)
    show emeasure M (⋃ x. fst (A x)) ≠ ∞ emeasure M (snd (A n)) ≠ ∞ for n
      using lm(2)[of n] UN_fA by (auto simp: fmeasurable_def)
  qed
ultimately have eq: 0 = ?μ (⋃ n. fst (A n)) - ?μ (⋂ n. snd (A n))
  by (rule LIMSEQ_unique)

show S ∈ fmeasurable M
  using UN_fA INT_sA
proof (rule complete_sets_sandwich_fmeasurable)
  show (⋃ n. fst (A n)) ⊆ S S ⊆ (⋂ n. snd (A n))
    using set_bound by auto
  show ?μ (⋃ n. fst (A n)) = ?μ (⋂ n. snd (A n))
    using eq by auto
qed

```

1670

qed (auto intro!: bexI[of _ S])

lemma (in complete_measure) fmeasurable_measure_inner_outer:

$(S \in \text{fmeasurable } M \wedge m = \text{measure } M S) \longleftrightarrow$
 $(\forall e > 0. \exists T \in \text{fmeasurable } M. T \subseteq S \wedge m - e < \text{measure } M T) \wedge$
 $(\forall e > 0. \exists U \in \text{fmeasurable } M. S \subseteq U \wedge \text{measure } M U < m + e)$
(is ?lhs = ?rhs)

proof

assume RHS: ?rhs

then have $T: \bigwedge e. 0 < e \longrightarrow (\exists T \in \text{fmeasurable } M. T \subseteq S \wedge m - e < \text{measure } M T)$

and $U: \bigwedge e. 0 < e \longrightarrow (\exists U \in \text{fmeasurable } M. S \subseteq U \wedge \text{measure } M U < m + e)$

by auto

have $S \in \text{fmeasurable } M$

proof (subst fmeasurable_inner_outer, safe)

fix $e::\text{real}$ **assume** $0 < e$

with RHS **obtain** $T U$ **where** $T: T \in \text{fmeasurable } M \ T \subseteq S \ m - e/2 < \text{measure } M T$

and $U: U \in \text{fmeasurable } M \ S \subseteq U \ \text{measure } M U < m + e/2$

by (meson half_gt_zero)+

moreover have $\text{measure } M U - \text{measure } M T < (m + e/2) - (m - e/2)$

by (intro diff_strict_mono) fact+

moreover have $\text{measure } M T \leq \text{measure } M U$

using $T U$ **by** (intro measure_mono_fmeasurable) auto

ultimately show $\exists T \in \text{fmeasurable } M. \exists U \in \text{fmeasurable } M. T \subseteq S \wedge S \subseteq U$
 $\wedge |\text{measure } M T - \text{measure } M U| < e$

apply (rule_tac bexI[OF _ $\langle T \in \text{fmeasurable } M \rangle$])

apply (rule_tac bexI[OF _ $\langle U \in \text{fmeasurable } M \rangle$])

by auto

qed

moreover have $m = \text{measure } M S$

using $\langle S \in \text{fmeasurable } M \rangle \ U[\text{of } \text{measure } M S - m] \ T[\text{of } m - \text{measure } M S]$

by (cases m_measure M S rule: linorder_cases)

(auto simp: not_le[symmetric] measure_mono_fmeasurable)

ultimately show ?lhs

by simp

qed (auto intro!: bexI[of _ S])

lemma (in complete_measure) null_sets_outer:

$S \in \text{null\_sets } M \longleftrightarrow (\forall e > 0. \exists T \in \text{fmeasurable } M. S \subseteq T \wedge \text{measure } M T < e)$

proof -

have $S \in \text{null\_sets } M \longleftrightarrow (S \in \text{fmeasurable } M \wedge 0 = \text{measure } M S)$

by (auto simp: null_sets_def emeasure_eq_measure2 intro: fmeasurableI) (simp add: measure_def)

also have $\dots = (\forall e > 0. \exists T \in \text{fmeasurable } M. S \subseteq T \wedge \text{measure } M T < e)$

unfolding fmeasurable_measure_inner_outer **by** auto

finally show ?thesis .

qed

lemma (in complete_measure) fmeasurable_measure_inner_outer_le:
 $(S \in \text{fmeasurable } M \wedge m = \text{measure } M S) \longleftrightarrow$
 $(\forall e > 0. \exists T \in \text{fmeasurable } M. T \subseteq S \wedge m - e \leq \text{measure } M T) \wedge$
 $(\forall e > 0. \exists U \in \text{fmeasurable } M. S \subseteq U \wedge \text{measure } M U \leq m + e)$ (is ?T1)
and null_sets_outer_le:
 $S \in \text{null\_sets } M \longleftrightarrow (\forall e > 0. \exists T \in \text{fmeasurable } M. S \subseteq T \wedge \text{measure } M T \leq e)$ (is ?T2)
proof –
have $e > 0 \wedge m - e/2 \leq t \implies m - e < t$
 $e > 0 \wedge t \leq m + e/2 \implies t < m + e$
 $e > 0 \longleftrightarrow e/2 > 0$
for $e \ t$
by auto
then show ?T1 ?T2
unfolding fmeasurable_measure_inner_outer null_sets_outer
by (meson dense_le_less_trans less_imp_le)+
qed

lemma (in cld_measure) notin_sets_outer_measure_of_cover:
assumes $E: E \subseteq \text{space } M \ E \notin \text{sets } M$
shows $\exists B \in \text{sets } M. 0 < \text{emeasure } M B \wedge \text{emeasure } M B < \infty \wedge$
 $\text{outer\_measure\_of } M (B \cap E) = \text{emeasure } M B \wedge \text{outer\_measure\_of } M (B - E) = \text{emeasure } M B$
proof –
from locally_determined[OF $\langle E \subseteq \text{space } M \rangle \langle E \notin \text{sets } M \rangle$]
obtain F
where [measurable]: $F \in \text{sets } M$ **and** $\text{emeasure } M F < \infty \ E \cap F \notin \text{sets } M$
by blast
then obtain $H \ H'$
where $H: \text{measurable\_envelope } M (F \cap E) \ H$ **and** $H': \text{measurable\_envelope } M (F - E) \ H'$
using measurable_envelopeI_countable_cover[of $F \cap E \ \lambda_. \ F \ M$]
measurable_envelopeI_countable_cover[of $F - E \ \lambda_. \ F \ M$]
by auto
note measurable_envelopeD(2)[OF $H', \text{measurable}$] measurable_envelopeD(2)[OF $H, \text{measurable}$]
from measurable_envelopeD(1)[OF H] measurable_envelopeD(1)[OF H]
have subset: $F - H' \subseteq F \cap E \ F \cap E \subseteq F \cap H$
by auto
moreover define G **where** $G = (F \cap H) - (F - H')$
ultimately have $G: G = F \cap H \cap H'$
by auto
have $\text{emeasure } M (F \cap H) \neq 0$
proof
assume $\text{emeasure } M (F \cap H) = 0$
then have $F \cap H \in \text{null\_sets } M$
by auto

```

    with  $\langle E \cap F \notin \text{sets } M \rangle$  show False
      using complete[OF  $\langle F \cap E \subseteq F \cap H \rangle$ ] by (auto simp: Int_commute)
  qed
  moreover
  have emeasure M ( $F - H'$ )  $\neq$  emeasure M ( $F \cap H$ )
  proof
    assume emeasure M ( $F - H'$ ) = emeasure M ( $F \cap H$ )
    with  $\langle E \cap F \notin \text{sets } M \rangle$  emeasure_mono[of  $F \cap H$  F M]  $\langle \text{emeasure } M \text{ } F < \infty \rangle$ 
    have  $F \cap E \in \text{sets } M$ 
      by (intro complete_sets_sandwich[OF  $\_ \_ \text{subset}$ ]) auto
    with  $\langle E \cap F \notin \text{sets } M \rangle$  show False
      by (simp add: Int_commute)
  qed
  moreover have emeasure M ( $F - H'$ )  $\leq$  emeasure M ( $F \cap H$ )
    using subset by (intro emeasure_mono) auto
  ultimately have emeasure M G  $\neq 0$ 
    unfolding G_def using subset
    by (subst emeasure_Diff) (auto simp: top_unique diff_eq_0_iff_enreal)
  show ?thesis
  proof (intro bexI conjI)
    have emeasure M G  $\leq$  emeasure M F
      unfolding G by (auto intro!: emeasure_mono)
    with  $\langle \text{emeasure } M \text{ } F < \infty \rangle$  show  $0 < \text{emeasure } M \text{ } G$  emeasure M G  $< \infty$ 
      using  $\langle \text{emeasure } M \text{ } G \neq 0 \rangle$  by (auto simp: zero_less_iff_neq_zero)
    show [measurable]:  $G \in \text{sets } M$ 
      unfolding G by auto

    have emeasure M G = outer_measure_of M ( $F \cap H' \cap (F \cap E)$ )
      using measurable_envelopeD(3)[OF H, of  $F \cap H$ ] unfolding G by (simp add: ac_simps)
    also have  $\dots \leq \text{outer\_measure\_of } M \text{ } (G \cap E)$ 
      using measurable_envelopeD(1)[OF H] by (intro outer_measure_of_mono)
    (auto simp: G)
    finally show outer_measure_of M ( $G \cap E$ ) = emeasure M G
      using outer_measure_of_mono[of  $G \cap E$  G M] by auto

    have emeasure M G = outer_measure_of M ( $F \cap H \cap (F - E)$ )
      using measurable_envelopeD(3)[OF  $H'$ , of  $F \cap H$ ] unfolding G by (simp add: ac_simps)
    also have  $\dots \leq \text{outer\_measure\_of } M \text{ } (G - E)$ 
      using measurable_envelopeD(1)[OF  $H'$ ] by (intro outer_measure_of_mono)
    (auto simp: G)
    finally show outer_measure_of M ( $G - E$ ) = emeasure M G
      using outer_measure_of_mono[of  $G - E$  G M] by auto
  qed
  qed

```

The following theorem is a specialization of D.H. Fremlin, Measure Theory vol 4I (413G). We only show one direction and do not use a inner regular

family K .

lemma (in *cld_measure*) *borel_measurable_cld*:

fixes $f :: 'a \Rightarrow \text{real}$

assumes $\bigwedge A \ a \ b. A \in \text{sets } M \implies 0 < \text{emeasure } M \ A \implies \text{emeasure } M \ A < \infty$
 $\implies a < b \implies$

$\min (\text{outer\_measure\_of } M \ \{x \in A. f \ x \leq a\}) (\text{outer\_measure\_of } M \ \{x \in A. b \leq f \ x\}) < \text{emeasure } M \ A$

shows $f \in M \rightarrow_M \text{borel}$

proof (rule *ccontr*)

let $?E = \lambda a. \{x \in \text{space } M. f \ x \leq a\}$ **and** $?F = \lambda a. \{x \in \text{space } M. a \leq f \ x\}$

assume $f \notin M \rightarrow_M \text{borel}$

then obtain a **where** $?E \ a \notin \text{sets } M$

unfolding *borel_measurable_iff_le* **by** *blast*

from *notin_sets_outer_measure_of_cover[OF _ this]*

obtain K

where $K: K \in \text{sets } M \ 0 < \text{emeasure } M \ K \ \text{emeasure } M \ K < \infty$

and *eq1*: $\text{outer\_measure\_of } M \ (K \cap ?E \ a) = \text{emeasure } M \ K$

and *eq2*: $\text{outer\_measure\_of } M \ (K - ?E \ a) = \text{emeasure } M \ K$

by *auto*

then have *me_K*: $\text{measurable\_envelope } M \ (K \cap ?E \ a) \ K$

by (*subst measurable_envelope_eq2*) *auto*

define b **where** $b \ n = a + \text{inverse } (\text{real } (\text{Suc } n))$ **for** n

have $(\text{SUP } n. \text{outer\_measure\_of } M \ (K \cap ?F \ (b \ n))) = \text{outer\_measure\_of } M$
 $(\bigcup n. K \cap ?F \ (b \ n))$

proof (*intro SUP_outer_measure_of_incseq*)

have $x \leq y \implies b \ y \leq b \ x$ **for** $x \ y$

by (*auto simp: b_def field_simps*)

then show *incseq* $(\lambda n. K \cap \{x \in \text{space } M. b \ n \leq f \ x\})$

by (*auto simp: incseq_def intro: order_trans*)

qed *auto*

also have $(\bigcup n. K \cap ?F \ (b \ n)) = K - ?E \ a$

proof $-$

have $b \longrightarrow a$

unfolding *b_def* **by** (*rule LIMSEQ_inverse_real_of_nat_add*)

then have $\forall n. \neg b \ n \leq f \ x \implies f \ x \leq a$ **for** x

by (*rule LIMSEQ_le_const*) (*auto intro: less_imp_le simp: not_le*)

moreover have $\neg b \ n \leq a$ **for** n

by (*auto simp: b_def*)

ultimately show *?thesis*

using $\langle K \in \text{sets } M \rangle [\text{THEN sets.sets\_into\_space}]$ **by** (*auto simp: subset_eq intro: order_trans*)

qed

finally have $0 < (\text{SUP } n. \text{outer\_measure\_of } M \ (K \cap ?F \ (b \ n)))$

using K **by** (*simp add: eq2*)

then obtain n **where** *pos_b*: $0 < \text{outer\_measure\_of } M \ (K \cap ?F \ (b \ n))$ **and** $a < b \ n$

unfolding *less_SUP_iff* **by** (*auto simp: b_def*)

```

from measurable_envelopeI_countable_cover[of  $K \cap ?F (b \ n) \ \lambda \_.$   $K \ M$ ]  $K$ 
obtain  $K'$  where  $K' \subseteq K$  and  $me\_K'$ : measurable_envelope  $M \ (K \cap ?F (b \ n)) \ K'$ 
by auto
then have  $K'\_le\_K$ :  $emeasure \ M \ K' \leq emeasure \ M \ K$ 
by (intro  $emeasure\_mono \ K$ )
have  $K' \in sets \ M$ 
using  $me\_K'$  by (rule measurable_envelopeD)

have  $min \ (outer\_measure\_of \ M \ \{x \in K'. \ f \ x \leq a\}) \ (outer\_measure\_of \ M \ \{x \in K'. \ b \ n \leq f \ x\}) < emeasure \ M \ K'$ 
proof (rule  $assms$ )
  show  $0 < emeasure \ M \ K' \ emeasure \ M \ K' < \infty$ 
  using measurable_envelopeD2[OF  $me\_K'$ ]  $pos\_b \ K \ K'\_le\_K$  by auto
qed fact+
also have  $\{x \in K'. \ f \ x \leq a\} = K' \cap (K \cap ?E \ a)$ 
  using  $\langle K' \in sets \ M \rangle [THEN \ sets.sets\_into\_space] \ \langle K' \subseteq K \rangle$  by auto
also have  $\{x \in K'. \ b \ n \leq f \ x\} = K' \cap (K \cap ?F (b \ n))$ 
  using  $\langle K' \in sets \ M \rangle [THEN \ sets.sets\_into\_space] \ \langle K' \subseteq K \rangle$  by auto
finally have  $min \ (emeasure \ M \ K) \ (emeasure \ M \ K') < emeasure \ M \ K'$ 
unfolding
  measurable_envelopeD(3)[OF  $me\_K \ \langle K' \in sets \ M \rangle$ ,  $symmetric$ ]
  measurable_envelopeD(3)[OF  $me\_K' \ \langle K' \in sets \ M \rangle$ ,  $symmetric$ ]
using  $\langle K' \subseteq K \rangle$  by (simp add:  $Int\_absorb1 \ Int\_absorb2$ )
with  $K'\_le\_K$  show  $False$ 
by (auto simp:  $min\_def \ split: \ if\_split\_asm$ )
qed

end

```

6.12 Regularity of Measures

```

theory Regularity
imports Measure_Space Borel_Space
begin

theorem
  fixes  $M::'a::\{second\_countable\_topology, \ complete\_space\} \ measure$ 
  assumes  $sb$ :  $sets \ M = sets \ borel$ 
  assumes  $emeasure \ M \ (space \ M) \neq \infty$ 
  assumes  $B \in sets \ borel$ 
  shows  $inner\_regular$ :  $emeasure \ M \ B =$ 
     $(SUP \ K \in \{K. \ K \subseteq B \wedge compact \ K\}. \ emeasure \ M \ K)$  (is ?inner B)
  and  $outer\_regular$ :  $emeasure \ M \ B =$ 
     $(INF \ U \in \{U. \ B \subseteq U \wedge open \ U\}. \ emeasure \ M \ U)$  (is ?outer B)
proof –
  have  $Us$ :  $UNIV = space \ M$  by (metis  $assms(1) \ sets\_eq \ imp\_space\_eq \ space\_borel$ )
  hence  $sU$ :  $space \ M = UNIV$  by simp
  interpret  $finite\_measure \ M$  by rule fact

```

```

have approx_inner:  $\bigwedge A. A \in \text{sets } M \implies$ 
  ( $\bigwedge e. e > 0 \implies \exists K. K \subseteq A \wedge \text{compact } K \wedge \text{emeasure } M A \leq \text{emeasure } M K$ 
+ ennreal e)  $\implies ?\text{inner } A$ 
  by (rule ennreal_approx_SUP)
  (force intro!: emeasure_mono simp: compact_imp_closed emeasure_eq_measure)+
have approx_outer:  $\bigwedge A. A \in \text{sets } M \implies$ 
  ( $\bigwedge e. e > 0 \implies \exists B. A \subseteq B \wedge \text{open } B \wedge \text{emeasure } M B \leq \text{emeasure } M A +$ 
ennreal e)  $\implies ?\text{outer } A$ 
  by (rule ennreal_approx_INF)
  (force intro!: emeasure_mono simp: emeasure_eq_measure sb)+
from countable_dense_setE obtain X :: 'a set
  where X: countable X  $\bigwedge Y :: 'a \text{ set. open } Y \implies Y \neq \{\} \implies \exists d \in X. d \in Y$ 
  by auto
  {
    fix r::real assume r > 0 hence  $\bigwedge y. \text{open } (\text{ball } y \text{ } r) \wedge y. \text{ball } y \text{ } r \neq \{\}$  by auto
    with X(2)[OF this]
    have x: space M =  $(\bigcup x \in X. \text{cball } x \text{ } r)$ 
      by (auto simp add: sU) (metis dist_commute order_less_imp_le)
    let ?U =  $\bigcup k. (\bigcup n \in \{0..k\}. \text{cball } (\text{from\_nat\_into } X \text{ } n) \text{ } r)$ 
    have  $(\lambda k. \text{emeasure } M (\bigcup n \in \{0..k\}. \text{cball } (\text{from\_nat\_into } X \text{ } n) \text{ } r)) \longrightarrow M$ 
    ?U
      by (rule Lim_emeasure_incseq) (auto intro!: borel_closed bexI simp: incseq_def Us sb)
    also have ?U = space M
    proof safe
      fix x from X(2)[OF open_ball[of x r]]  $\langle r > 0 \rangle$  obtain d where d: d  $\in X$  d
       $\in \text{ball } x \text{ } r$  by auto
      show x  $\in ?U$ 
      using X(1) d
      by simp (auto intro!: exI [where x = to_nat_on X d] simp: dist_commute Bex_def)
    qed (simp add: sU)
    finally have  $(\lambda k. M (\bigcup n \in \{0..k\}. \text{cball } (\text{from\_nat\_into } X \text{ } n) \text{ } r)) \longrightarrow M$ 
    (space M) .
  } note M_space = this
  {
    fix e ::real and n :: nat assume e > 0 n > 0
    hence  $1/n > 0 \text{ } e * 2^{\text{powr } -n} > 0$  by (auto)
    from M_space[OF  $\langle 1/n > 0 \rangle$ ]
    have  $(\lambda k. \text{measure } M (\bigcup i \in \{0..k\}. \text{cball } (\text{from\_nat\_into } X \text{ } i) (1/\text{real } n)))$ 
     $\longrightarrow \text{measure } M (\text{space } M)$ 
    unfolding emeasure_eq_measure by (auto)
    from metric_LIMSEQ_D[OF this  $\langle 0 < e * 2^{\text{powr } -n} \rangle$ ]
    obtain k where dist (measure M  $(\bigcup i \in \{0..k\}. \text{cball } (\text{from\_nat\_into } X \text{ } i)$ 
    (1/real n))) (measure M (space M)) <
    e * 2powr -n
    by auto
    hence measure M  $(\bigcup i \in \{0..k\}. \text{cball } (\text{from\_nat\_into } X \text{ } i) (1/\text{real } n)) \geq$ 
    measure M (space M) - e * 2powr -real n
  }

```

```

    by (auto simp: dist_real_def)
  hence  $\exists k. \text{measure } M (\bigcup i \in \{0..k\}. \text{cball } (\text{from\_nat\_into } X \ i) \ (1/\text{real } n)) \geq$ 
     $\text{measure } M (\text{space } M) - e * 2^{\text{powr } - \text{real } n} ..$ 
} note k=this
  hence  $\forall e \in \{0 < ..\}. \forall (n::\text{nat}) \in \{0 < ..\}. \exists k.$ 
     $\text{measure } M (\bigcup i \in \{0..k\}. \text{cball } (\text{from\_nat\_into } X \ i) \ (1/\text{real } n)) \geq \text{measure } M$ 
     $(\text{space } M) - e * 2^{\text{powr } - \text{real } n}$ 
  by blast
  then obtain k where k:  $\forall e \in \{0 < ..\}. \forall n \in \{0 < ..\}. \text{measure } M (\text{space } M) - e * 2^{\text{powr } - \text{real } (n::\text{nat})}$ 
     $\leq \text{measure } M (\bigcup i \in \{0..k \ e \ n\}. \text{cball } (\text{from\_nat\_into } X \ i) \ (1 / n))$ 
  by metis
  hence k:  $\bigwedge e \ n. e > 0 \implies n > 0 \implies \text{measure } M (\text{space } M) - e * 2^{\text{powr } - n}$ 
     $\leq \text{measure } M (\bigcup i \in \{0..k \ e \ n\}. \text{cball } (\text{from\_nat\_into } X \ i) \ (1 / n))$ 
  unfolding Ball_def by blast
  have approx_space:
     $\exists K \in \{K. K \subseteq \text{space } M \wedge \text{compact } K\}. \text{emeasure } M (\text{space } M) \leq \text{emeasure}$ 
     $M \ K + \text{ennreal } e$ 
    (is ?thesis e) if  $0 < e$  for  $e :: \text{real}$ 
  proof -
    define B where [abs_def]:
       $B \ n = (\bigcup i \in \{0..k \ e \ (\text{Suc } n)\}. \text{cball } (\text{from\_nat\_into } X \ i) \ (1 / \text{Suc } n))$  for n
    have  $\bigwedge n. \text{closed } (B \ n)$  by (auto simp: B_def)
    hence [simp]:  $\bigwedge n. B \ n \in \text{sets } M$  by (simp add: sb)
    from k[OF  $\langle e > 0 \rangle$  zero_less_Suc]
    have  $\bigwedge n. \text{measure } M (\text{space } M) - \text{measure } M (B \ n) \leq e * 2^{\text{powr } - \text{real } (\text{Suc } n)}$ 
    by (simp add: algebra_simps B_def finite_measure_compl)
    hence B_compl_le:  $\bigwedge n::\text{nat}. \text{measure } M (\text{space } M - B \ n) \leq e * 2^{\text{powr } - \text{real } (\text{Suc } n)}$ 
    by (simp add: finite_measure_compl)
    define K where  $K = (\bigcap n. B \ n)$ 
    from  $\langle \text{closed } (B \ _) \rangle$  have closed K by (auto simp: K_def)
    hence [simp]:  $K \in \text{sets } M$  by (simp add: sb)
    have  $\text{measure } M (\text{space } M) - \text{measure } M \ K = \text{measure } M (\text{space } M - K)$ 
    by (simp add: finite_measure_compl)
    also have  $\dots = \text{emeasure } M (\bigcup n. \text{space } M - B \ n)$  by (auto simp: K_def
    emeasure_eq_measure)
    also have  $\dots \leq (\sum n. \text{emeasure } M (\text{space } M - B \ n))$ 
    by (rule emeasure_subadditive_countably) (auto simp: summable_def)
    also have  $\dots \leq (\sum n. \text{ennreal } (e * 2^{\text{powr } - \text{real } (\text{Suc } n)}))$ 
    using B_compl_le by (intro suminf_le) (simp_all add: emeasure_eq_measure
    ennreal_leI)
    also have  $\dots \leq (\sum n. \text{ennreal } (e * (1 / 2)^{\wedge \text{Suc } n}))$ 
    by (simp add: powr_minus powr_realpow field_simps del: of_nat_Suc)
    also have  $\dots = \text{ennreal } e * (\sum n. \text{ennreal } ((1 / 2)^{\wedge \text{Suc } n}))$ 
    unfolding ennreal_power[symmetric]
    using  $\langle 0 < e \rangle$ 
    by (simp add: ac_simps ennreal_mult' divide_ennreal[symmetric] divide_ennreal_def)

```



```

      ennreal_power[symmetric])
    also have ... = e
      by (subst suminf_ennreal_eq[OF zero_le_power power_half_series]) auto
    finally have measure M (space M) ≤ measure M K + e
      using ⟨0 < e⟩ by simp
    hence emeasure M (space M) ≤ emeasure M K + e
      using ⟨0 < e⟩ by (simp add: emeasure_eq_measure flip: ennreal_plus)
    moreover have compact K
      unfolding compact_eq_totally_bounded
    proof safe
      show complete K using ⟨closed K⟩ by (simp add: complete_eq_closed)
      fix e'::real assume 0 < e'
      then obtain n where n: 1 / real (Suc n) < e' by (rule nat_approx_posE)
      let ?k = from_nat_into X ' {0..k e (Suc n)}
      have finite ?k by simp
      moreover have K ⊆ (⋃ x∈?k. ball x e') unfolding K_def B_def using n
    by force
      ultimately show ∃ k. finite k ∧ K ⊆ (⋃ x∈k. ball x e') by blast
    qed
    ultimately
    show ?thesis by (auto simp: sU)
  qed
  { fix A::'a set assume closed A hence A ∈ sets borel by (simp add: compact_imp_closed)
    hence [simp]: A ∈ sets M by (simp add: sb)
    have ?inner A
    proof (rule approx_inner)
      fix e::real assume e > 0
      from approx_space[OF this] obtain K where
        K: K ⊆ space M compact K emeasure M (space M) ≤ emeasure M K + e
        by (auto simp: emeasure_eq_measure)
      hence [simp]: K ∈ sets M by (simp add: sb compact_imp_closed)
      have measure M A - measure M (A ∩ K) = measure M (A - A ∩ K)
        by (subst finite_measure_Diff) auto
      also have A - A ∩ K = A ∪ K - K by auto
      also have measure M ... = measure M (A ∪ K) - measure M K
        by (subst finite_measure_Diff) auto
      also have ... ≤ measure M (space M) - measure M K
        by (simp add: emeasure_eq_measure sU sb finite_measure_mono)
      also have ... ≤ e
        using K ⟨0 < e⟩ by (simp add: emeasure_eq_measure flip: ennreal_plus)
      finally have emeasure M A ≤ emeasure M (A ∩ K) + ennreal e
        using ⟨0 < e⟩ by (simp add: emeasure_eq_measure algebra_simps flip:
ennreal_plus)
      moreover have A ∩ K ⊆ A compact (A ∩ K) using ⟨closed A⟩ ⟨compact
K⟩ by auto
      ultimately show ∃ K ⊆ A. compact K ∧ emeasure M A ≤ emeasure M K +
ennreal e
        by blast

```

```

qed simp
have ?outer A
proof cases
  assume A ≠ {}
  let ?G = λd. {x. infdist x A < d}
  {
    fix d
    have ?G d = (λx. infdist x A) - ' {..<d} by auto
    also have open ...
      by (intro continuous_open_vimage) (auto intro!: continuous_infdist
continuous_ident)
    finally have open (?G d) .
  } note open_G = this
  from in_closed_iff_infdist_zero[OF ‹closed A› ‹A ≠ {}›]
  have A = {x. infdist x A = 0} by auto
  also have ... = (⋂ i. ?G (1 / real (Suc i)))
  proof (auto simp del: of_nat_Suc, rule ccontr)
    fix x
    assume infdist x A ≠ 0
    then have pos: infdist x A > 0 using infdist_nonneg[of x A] by simp
    then obtain n where n: 1 / real (Suc n) < infdist x A by (rule
nat_approx_posE)
    assume ∀ i. infdist x A < 1 / real (Suc i)
    then have infdist x A < 1 / real (Suc n) by auto
    with n show False by simp
  qed
  also have M ... = (INF n. emeasure M (?G (1 / real (Suc n))))
  proof (rule INF_emeasure_decseq[symmetric], safe)
    fix i::nat
    from open_G[of 1 / real (Suc i)]
    show ?G (1 / real (Suc i)) ∈ sets M by (simp add: sb_borel_open)
  next
    show decseq (λi. {x. infdist x A < 1 / real (Suc i)})
      by (auto intro: less_trans intro!: divide_strict_left_mono
simp: decseq_def le_eq_less_or_eq)
  qed simp
  finally
  have emeasure M A = (INF n. emeasure M {x. infdist x A < 1 / real (Suc
n)}) .
  moreover
  have ... ≥ (INF U∈{U. A ⊆ U ∧ open U}. emeasure M U)
  proof (intro INF_mono)
    fix m
    have ?G (1 / real (Suc m)) ∈ {U. A ⊆ U ∧ open U} using open_G by
auto
    moreover have M (?G (1 / real (Suc m))) ≤ M (?G (1 / real (Suc m)))
  by simp
    ultimately show ∃ U∈{U. A ⊆ U ∧ open U}.
      emeasure M U ≤ emeasure M {x. infdist x A < 1 / real (Suc m)}

```

```

      by blast
    qed
  moreover
    have  $\text{emeasure } M \ A \leq (\text{INF } U \in \{U. A \subseteq U \wedge \text{open } U\}. \text{emeasure } M \ U)$ 
      by (rule INF_greatest) (auto intro!: emeasure_mono simp: sb)
    ultimately show ?thesis by simp
  qed (auto intro!: INF_eqI)
  note ⟨?inner A⟩ ⟨?outer A⟩ }
  note closed_in_D = this
  from ⟨B ∈ sets borel⟩
  have Int_stable (Collect closed) Collect closed ⊆ Pow UNIV B ∈ sigma_sets
    UNIV (Collect closed)
    by (auto simp: Int_stable_def borel_eq_closed)
  then show ?inner B ?outer B
  proof (induct B rule: sigma_sets_induct_disjoint)
    case empty
    { case 1 show ?case by (intro SUP_eqI[symmetric]) auto }
    { case 2 show ?case by (intro INF_eqI[symmetric]) (auto elim!: meta_allE[of
      _ {}]) }
  next
    case (basic B)
    { case 1 from basic closed_in_D show ?case by auto }
    { case 2 from basic closed_in_D show ?case by auto }
  next
    case (compl B)
    note inner = compl(2) and outer = compl(3)
    from compl have [simp]: B ∈ sets M by (auto simp: sb borel_eq_closed)
    case 2
    have  $M \ (\text{space } M - B) = M \ (\text{space } M) - \text{emeasure } M \ B$  by (auto simp:
      emeasure_compl)
    also have ... =  $(\text{INF } K \in \{K. K \subseteq B \wedge \text{compact } K\}. M \ (\text{space } M) - M \ K)$ 
      by (subst ennreal_SUP_const_minus) (auto simp: less_top[symmetric] inner)
    also have ... =  $(\text{INF } U \in \{U. U \subseteq B \wedge \text{compact } U\}. M \ (\text{space } M - U))$ 
      by (auto simp add: emeasure_compl sb compact_imp_closed)
    also have ... ≥  $(\text{INF } U \in \{U. U \subseteq B \wedge \text{closed } U\}. M \ (\text{space } M - U))$ 
      by (rule INF_superset_mono) (auto simp add: compact_imp_closed)
    also have  $(\text{INF } U \in \{U. U \subseteq B \wedge \text{closed } U\}. M \ (\text{space } M - U)) =$ 
       $(\text{INF } U \in \{U. \text{space } M - B \subseteq U \wedge \text{open } U\}. \text{emeasure } M \ U)$ 
      apply (rule arg_cong [of _ _ Inf])
      using sU
      apply (auto simp add: image_iff)
      apply (rule exI [of _ UNIV - y for y])
      apply safe
      apply (auto simp add: double_diff)
    done
  finally have
     $(\text{INF } U \in \{U. \text{space } M - B \subseteq U \wedge \text{open } U\}. \text{emeasure } M \ U) \leq \text{emeasure } M$ 
     $(\text{space } M - B) .$ 
  moreover have

```

```

      (INF U ∈ {U. space M - B ⊆ U ∧ open U}. emeasure M U) ≥ emeasure M
(space M - B)
    by (auto simp: sb sU intro!: INF_greatest emeasure_mono)
    ultimately show ?case by (auto intro!: antisym simp: sets_eq_imp_space_eq[OF
sb])

  case 1
    have M (space M - B) = M (space M) - emeasure M B by (auto simp:
emeasure_compl)
    also have ... = (SUP U ∈ {U. B ⊆ U ∧ open U}. M (space M) - M U)
      unfolding outer by (subst ennreal_INF_const_minus) auto
    also have ... = (SUP U ∈ {U. B ⊆ U ∧ open U}. M (space M - U))
      by (auto simp add: emeasure_compl sb compact_imp_closed)
    also have ... = (SUP K ∈ {K. K ⊆ space M - B ∧ closed K}. emeasure M
K)
      unfolding SUP_image [of _ λu. space M - u _, symmetric, unfolded
comp_def]
      apply (rule arg_cong [of _ _ Sup])
      using sU apply (auto intro!: imageI)
      done
    also have ... = (SUP K ∈ {K. K ⊆ space M - B ∧ compact K}. emeasure M
K)
      proof (safe intro!: antisym SUP_least)
        fix K assume closed K K ⊆ space M - B
        from closed_in_D[OF <closed K>]
        have K_inner: emeasure M K = (SUP K ∈ {Ka. Ka ⊆ K ∧ compact Ka}.
emeasure M K) by simp
        show emeasure M K ≤ (SUP K ∈ {K. K ⊆ space M - B ∧ compact K}.
emeasure M K)
          unfolding K_inner using <K ⊆ space M - B>
          by (auto intro!: SUP_upper SUP_least)
        qed (fastforce intro!: SUP_least SUP_upper simp: compact_imp_closed)
        finally show ?case by (auto intro!: antisym simp: sets_eq_imp_space_eq[OF
sb])
      next
        case (union D)
        then have range D ⊆ sets M by (auto simp: sb borel_eq_closed)
        with union have M[symmetric]: (∑ i. M (D i)) = M (⋃ i. D i) by (intro
suminf_emeasure)
        also have (λn. ∑ i < n. M (D i)) ⟶ (∑ i. M (D i))
          by (intro summable_LIMSEQ) auto
        finally have measure_LIMSEQ: (λn. ∑ i < n. measure M (D i)) ⟶ measure
M (⋃ i. D i)
          by (simp add: emeasure_eq_measure sum_nonneg)
        have (⋃ i. D i) ∈ sets M using <range D ⊆ sets M> by auto
      case 1
      show ?case
      proof (rule approx_inner)

```

```

fix e::real assume e > 0
with measure_LIMSEQ
have  $\exists n_0. \forall n \geq n_0. |(\sum_{i < n. \text{measure } M (D i)) - \text{measure } M (\bigcup x. D x)| < e/2$ 
by (auto simp: lim_sequentially dist_real_def simp del: less_divide_eq numeral1)
hence  $\exists n_0. |(\sum_{i < n_0. \text{measure } M (D i)) - \text{measure } M (\bigcup x. D x)| < e/2$ 
by auto
then obtain n0 where n0:  $|(\sum_{i < n_0. \text{measure } M (D i)) - \text{measure } M (\bigcup i. D i)| < e/2$ 
unfolding choice_iff by blast
have ennreal  $(\sum_{i < n_0. \text{measure } M (D i)) = (\sum_{i < n_0. M (D i))$ 
by (auto simp add: emeasure_eq_measure)
also have  $\dots \leq (\sum i. M (D i))$  by (rule sum_le_suminf) auto
also have  $\dots = M (\bigcup i. D i)$  by (simp add: M)
also have  $\dots = \text{measure } M (\bigcup i. D i)$  by (simp add: emeasure_eq_measure)
finally have n0:  $\text{measure } M (\bigcup i. D i) - (\sum_{i < n_0. \text{measure } M (D i)) < e/2$ 
using n0 by (auto simp: sum_nonneg)
have  $\forall i. \exists K. K \subseteq D i \wedge \text{compact } K \wedge \text{emeasure } M (D i) \leq \text{emeasure } M K + e/(2 * \text{Suc } n_0)$ 
proof
fix i
from  $\langle 0 < e \rangle$  have  $0 < e/(2 * \text{Suc } n_0)$  by simp
have  $\text{emeasure } M (D i) = (\text{SUP } K \in \{K. K \subseteq (D i) \wedge \text{compact } K\}. \text{emeasure } M K)$ 
using union by blast
from SUP_approx_ennreal[OF  $\langle 0 < e/(2 * \text{Suc } n_0) \rangle$  _ this]
show  $\exists K. K \subseteq D i \wedge \text{compact } K \wedge \text{emeasure } M (D i) \leq \text{emeasure } M K + e/(2 * \text{Suc } n_0)$ 
by (auto simp: emeasure_eq_measure intro: less_imp_le compact_empty)
qed
then obtain K where K:  $\bigwedge i. K i \subseteq D i \wedge i. \text{compact } (K i) \wedge i. \text{emeasure } M (D i) \leq \text{emeasure } M (K i) + e/(2 * \text{Suc } n_0)$ 
unfolding choice_iff by blast
let ?K =  $\bigcup i \in \{.. < n_0\}. K i$ 
have disjoint_family_on K  $\{.. < n_0\}$  using K  $\langle \text{disjoint\_family } D \rangle$ 
unfolding disjoint_family_on_def by blast
hence mK:  $\text{measure } M ?K = (\sum_{i < n_0. \text{measure } M (K i))$  using K
by (intro finite_measure_finite_Union) (auto simp: sb_compact_imp_closed)
have  $\text{measure } M (\bigcup i. D i) < (\sum_{i < n_0. \text{measure } M (D i)) + e/2$  using n0
by simp
also have  $(\sum_{i < n_0. \text{measure } M (D i)) \leq (\sum_{i < n_0. \text{measure } M (K i) + e/(2 * \text{Suc } n_0))$ 
using K  $\langle 0 < e \rangle$ 
by (auto intro: sum_mono simp: emeasure_eq_measure simp flip: ennreal_plus)
also have  $\dots = (\sum_{i < n_0. \text{measure } M (K i)) + (\sum_{i < n_0. e/(2 * \text{Suc } n_0))$ 
by (simp add: sum.distrib)
also have  $\dots \leq (\sum_{i < n_0. \text{measure } M (K i)) + e/2$  using  $\langle 0 < e \rangle$ 
by (auto simp: field_simps intro!: mult_left_mono)

```

```

finally
have  $\text{measure } M (\bigcup i. D i) < (\sum i < n0. \text{measure } M (K i)) + e / 2 + e / 2$ 
by auto
hence  $M (\bigcup i. D i) < M ?K + e$ 
using  $\langle 0 < e \rangle$  by (auto simp: mK emeasure_eq_measure sum_nonneg ennreal_less_iff simp flip: ennreal_plus)
moreover
have  $?K \subseteq (\bigcup i. D i)$  using  $K$  by auto
moreover
have compact  $?K$  using  $K$  by auto
ultimately
have  $?K \subseteq (\bigcup i. D i) \wedge \text{compact } ?K \wedge \text{emeasure } M (\bigcup i. D i) \leq \text{emeasure } M$ 
 $?K + \text{ennreal } e$  by simp
thus  $\exists K \subseteq \bigcup i. D i. \text{compact } K \wedge \text{emeasure } M (\bigcup i. D i) \leq \text{emeasure } M K +$ 
 $\text{ennreal } e$  ..
qed fact
case 2
show  $?case$ 
proof (rule approx_outer[OF  $\langle (\bigcup i. D i) \in \text{sets } M \rangle$ ])
fix  $e::\text{real}$  assume  $e > 0$ 
have  $\forall i::\text{nat}. \exists U. D i \subseteq U \wedge \text{open } U \wedge e/(2 \text{ powr } \text{Suc } i) > \text{emeasure } M U$ 
 $- \text{emeasure } M (D i)$ 
proof
fix  $i::\text{nat}$ 
from  $\langle 0 < e \rangle$  have  $0 < e/(2 \text{ powr } \text{Suc } i)$  by simp
have  $\text{emeasure } M (D i) = (\text{INF } U \in \{U. (D i) \subseteq U \wedge \text{open } U\}. \text{emeasure } M U$ 
 $- \text{emeasure } M (D i))$ 
using union by blast
from INF_approx_ennreal[OF  $\langle 0 < e/(2 \text{ powr } \text{Suc } i) \rangle$  this]
show  $\exists U. D i \subseteq U \wedge \text{open } U \wedge e/(2 \text{ powr } \text{Suc } i) > \text{emeasure } M U -$ 
 $\text{emeasure } M (D i)$ 
using  $\langle 0 < e \rangle$ 
by (auto simp: emeasure_eq_measure sum_nonneg ennreal_less_iff ennreal_minus
 $\text{finite\_measure\_mono } sb$ 
 $\text{simp flip: ennreal\_plus}$ )
qed
then obtain  $U$  where  $U: \bigwedge i. D i \subseteq U i \wedge i. \text{open } (U i)$ 
 $\bigwedge i. e/(2 \text{ powr } \text{Suc } i) > \text{emeasure } M (U i) - \text{emeasure } M (D i)$ 
unfolding choice_iff by blast
let  $?U = \bigcup i. U i$ 
have  $\text{ennreal } (\text{measure } M ?U - \text{measure } M (\bigcup i. D i)) = M ?U - M (\bigcup i.$ 
 $D i)$ 
using  $U(1,2)$ 
by (subst ennreal_minus[symmetric])
 $(\text{auto intro!: finite\_measure\_mono simp: sb emeasure_eq_measure})$ 
also have  $\dots = M (?U - (\bigcup i. D i))$  using  $U \langle (\bigcup i. D i) \in \text{sets } M \rangle$ 
by (subst emeasure_Diff) (auto simp: sb)
also have  $\dots \leq M (\bigcup i. U i - D i)$  using  $U \langle \text{range } D \subseteq \text{sets } M \rangle$ 

```

```

    by (intro emeasure_mono) (auto simp: sb intro!: sets.countable_nat_UN
sets.Diff)
  also have ... ≤ (∑ i. M (U i - D i)) using U ⟨range D ⊆ sets M⟩
    by (intro emeasure_subadditive_countably) (auto intro!: sets.Diff simp: sb)
  also have ... ≤ (∑ i. ennreal e / (2 ^ Suc i)) using U ⟨range D ⊆ sets
M⟩
    using ⟨0 < e⟩
    by (intro suminf_le, subst emeasure_Diff)
      (auto simp: emeasure_Diff emeasure_eq_measure sb ennreal_minus
finite_measure_mono divide_ennreal ennreal_less_iff
intro: less_imp_le)
  also have ... ≤ (∑ n. ennreal (e * (1 / 2) ^ Suc n))
    using ⟨0 < e⟩
    by (simp add: powr_minus powr_realpow field_simps divide_ennreal del:
of_nat_Suc)
  also have ... = ennreal e * (∑ n. ennreal ((1 / 2) ^ Suc n))
    unfolding ennreal_power[symmetric]
    using ⟨0 < e⟩
    by (simp add: ac_simps ennreal_mult' divide_ennreal[symmetric] di-
vide_ennreal_def
ennreal_power[symmetric])
  also have ... = ennreal e
    by (subst suminf_ennreal_eq[OF zero_le_power power_half_series]) auto
  finally have emeasure M ?U ≤ emeasure M (⋃ i. D i) + ennreal e
    using ⟨0 < e⟩ by (simp add: emeasure_eq_measure flip: ennreal_plus)
  moreover
  have (⋃ i. D i) ⊆ ?U using U by auto
  moreover
  have open ?U using U by auto
  ultimately
  have (⋃ i. D i) ⊆ ?U ∧ open ?U ∧ emeasure M ?U ≤ emeasure M (⋃ i. D
i) + ennreal e by simp
  thus ∃ B. (⋃ i. D i) ⊆ B ∧ open B ∧ emeasure M B ≤ emeasure M (⋃ i. D
i) + ennreal e ..
qed
qed
qed
end

```

6.13 Lebesgue Measure

```

theory Lebesgue_Measure
imports
  Finite_Product_Measure
  Caratheodory
  Complete_Measure
  Summation_Tests
  Regularity

```

begin

```

lemma measure_eqI_lessThan:
  fixes M N :: real measure
  assumes sets: sets M = sets borel sets N = sets borel
  assumes fin:  $\bigwedge x. \text{emeasure } M \{x <..\} < \infty$ 
  assumes  $\bigwedge x. \text{emeasure } M \{x <..\} = \text{emeasure } N \{x <..\}$ 
  shows M = N
proof (rule measure_eqI_generator_eq_countable)
  let ?LT =  $\lambda a::\text{real}. \{a <..\}$  let ?E = range ?LT
  show Int_stable ?E
    by (auto simp: Int_stable_def lessThan_Int_lessThan)

  show ?E  $\subseteq \text{Pow UNIV sets } M = \text{sigma\_sets UNIV ?E sets } N = \text{sigma\_sets UNIV ?E}$ 
    unfolding sets borel_Ioi by auto

  show  $?LT'Rats \subseteq ?E (\bigcup_{i \in Rats. ?LT i} = UNIV \bigwedge a. a \in ?LT'Rats \implies \text{emeasure } M a \neq \infty$ 
    using fin by (auto intro: Rats_no_bot_less simp: less_top)
qed (auto intro: assms countable_rat)

```

6.13.1 Measures defined by monotonous functions

Every right-continuous and nondecreasing function gives rise to a measure on the reals:

```

definition interval_measure ::  $(\text{real} \Rightarrow \text{real}) \Rightarrow \text{real measure}$  where
  interval_measure F =
    extend_measure UNIV  $\{(a, b). a \leq b\} (\lambda(a, b). \{a <..b\}) (\lambda(a, b). \text{ennreal } (F b - F a))$ 

```

```

lemma emeasure_interval_measure_Ioc:
  assumes  $a \leq b$ 
  assumes mono_F:  $\bigwedge x y. x \leq y \implies F x \leq F y$ 
  assumes right_cont_F:  $\bigwedge a. \text{continuous (at\_right } a) F$ 
  shows  $\text{emeasure (interval\_measure } F) \{a <..b\} = F b - F a$ 
proof (rule extend_measure_caratheodory_pair[OF interval_measure_def ‹a ≤ b›])
  show semiring_of_sets UNIV  $\{\{a <..b\} \mid a b :: \text{real}. a \leq b\}$ 
  proof (unfold locales, safe)
    fix a b c d :: real assume  $*$ :  $a \leq b \leq c \leq d$ 
    then show  $\exists C \subseteq \{\{a <..b\} \mid a b. a \leq b\}. \text{finite } C \wedge \text{disjoint } C \wedge \{a <..b\} - \{c <..d\} = \bigcup C$ 
      proof cases
        let ?C =  $\{\{a <..b\}\}$ 
        assume  $b < c \vee d \leq a \vee d \leq c$ 
        with  $*$  have  $?C \subseteq \{\{a <..b\} \mid a b. a \leq b\} \wedge \text{finite } ?C \wedge \text{disjoint } ?C \wedge \{a <..b\} - \{c <..d\} = \bigcup ?C$ 
          by (auto simp add: disjoint_def)
      qed
  qed

```



```

    thus ?thesis ..
  next
    let ?C = {{a<..c}, {d<..b}}
    assume ¬ (b < c ∨ d ≤ a ∨ d ≤ c)
    with * have ?C ⊆ {{a<..b} | a b. a ≤ b} ∧ finite ?C ∧ disjoint ?C ∧ {a<..b}
    - {c<..d} = ⋃ ?C
      by (auto simp add: disjoint_def Ioc_inj) (metis linear)+
    thus ?thesis ..
  qed
qed (auto simp: Ioc_inj, metis linear)
next
  fix l r :: nat ⇒ real and a b :: real
  assume l_r[simp]: ⋀n. l n ≤ r n and a ≤ b and disj: disjoint_family (λn. {l
n<..r n})
  assume lr_eq_ab: (⋃ i. {l i<..r i}) = {a<..b}

  have [intro, simp]: ⋀a b. a ≤ b ⇒ F a ≤ F b
    by (auto intro!: l_r mono_F)

  { fix S :: nat set assume finite S
    moreover note ⟨a ≤ b⟩
    moreover have ⋀i. i ∈ S ⇒ {l i <.. r i} ⊆ {a <.. b}
      unfolding lr_eq_ab[symmetric] by auto
    ultimately have (∑ i∈S. F (r i) - F (l i)) ≤ F b - F a
      proof (induction S arbitrary: a rule: finite_psubset_induct)
        case (psubset S)
        show ?case
        proof cases
          assume ∃ i∈S. l i < r i
          with ⟨finite S⟩ have Min (l ‘ {i∈S. l i < r i}) ∈ l ‘ {i∈S. l i < r i}
            by (intro Min_in) auto
          then obtain m where m: m ∈ S l m < r m l m = Min (l ‘ {i∈S. l i < r
i})
            by fastforce

          have (∑ i∈S. F (r i) - F (l i)) = (F (r m) - F (l m)) + (∑ i∈S - {m}.
F (r i) - F (l i))
            using m psubset by (intro sum.remove) auto
          also have (∑ i∈S - {m}. F (r i) - F (l i)) ≤ F b - F (r m)
            proof (intro psubset.IH)
              show S - {m} ⊂ S
                using ⟨m∈S⟩ by auto
              show r m ≤ b
                using psubset.premis(2)[OF ⟨m∈S⟩] ⟨l m < r m⟩ by auto
            next
              fix i assume i ∈ S - {m}
              then have i: i ∈ S i ≠ m by auto
              { assume i': l i < r i l i < r m
                with ⟨finite S⟩ i m have l m ≤ l i

```

```

    by auto
  with i' have {l i <.. r i} ∩ {l m <.. r m} ≠ {}
    by auto
  then have False
    using disjoint_family_onD[OF disj, of i m] i by auto }
  then have l i ≠ r i ⇒ r m ≤ l i
    unfolding not_less[symmetric] using l_r[of i] by auto
  then show {l i <.. r i} ⊆ {r m <.. b}
    using psubset.premis(2)[OF ⟨i ∈ S⟩] by auto
qed
also have F (r m) − F (l m) ≤ F (r m) − F a
  using psubset.premis(2)[OF ⟨m ∈ S⟩] ⟨l m < r m⟩
  by (auto simp add: Ioc_subset_iff intro!: mono_F)
finally show ?case
  by (auto intro: add_mono)
qed (auto simp add: ⟨a ≤ b⟩ less_le)
qed }
note claim1 = this

```

```

{ fix S u v and l r :: nat ⇒ real
  assume finite S ∧ i. i ∈ S ⇒ l i < r i {u..v} ⊆ (⋃ i ∈ S. {l i <.. < r i})
  then have F v − F u ≤ (∑ i ∈ S. F (r i) − F (l i))
  proof (induction arbitrary: v u rule: finite_psubset_induct)
    case (psubset S)
    show ?case
    proof cases
      assume S = {} then show ?case
        using psubset by (simp add: mono_F)
    next
      assume S ≠ {}
      then obtain j where j ∈ S
        by auto

    let ?R = r j < u ∨ l j > v ∨ (∃ i ∈ S − {j}. l i ≤ l j ∧ r j ≤ r i)
    show ?case
    proof cases
      assume ?R
      with ⟨j ∈ S⟩ psubset.premis have {u..v} ⊆ (⋃ i ∈ S − {j}. {l i <.. < r i})
      apply (auto simp: subset_eq Ball_def)
      apply (metis Diff_iff less_le_trans leD linear singletonD)
      apply (metis Diff_iff less_le_trans leD linear singletonD)
      apply (metis order_trans less_le_not_le linear)
      done
    with ⟨j ∈ S⟩ have F v − F u ≤ (∑ i ∈ S − {j}. F (r i) − F (l i))
      by (intro psubset) auto
    also have ... ≤ (∑ i ∈ S. F (r i) − F (l i))
      using psubset.premis

```

```

    by (intro sum_mono2 psubset) (auto intro: less_imp_le)
  finally show ?thesis .
next
  assume  $\neg ?R$ 
  then have  $j: u \leq r\ j\ l\ j \leq v \wedge i. i \in S - \{j\} \implies r\ i < r\ j \vee l\ i > l\ j$ 
    by (auto simp: not_less)
  let ?S1 =  $\{i \in S. l\ i < l\ j\}$ 
  let ?S2 =  $\{i \in S. r\ i > r\ j\}$ 

  have  $(\sum_{i \in S. F\ (r\ i) - F\ (l\ i)}) \geq (\sum_{i \in ?S1 \cup ?S2 \cup \{j\}. F\ (r\ i) - F\ (l\ i)})$ 
(l i))
    using  $\langle j \in S \rangle \langle \text{finite } S \rangle$  psubset.prem j
    by (intro sum_mono2) (auto intro: less_imp_le)
  also have  $(\sum_{i \in ?S1 \cup ?S2 \cup \{j\}. F\ (r\ i) - F\ (l\ i)}) =$ 
     $(\sum_{i \in ?S1. F\ (r\ i) - F\ (l\ i)}) + (\sum_{i \in ?S2. F\ (r\ i) - F\ (l\ i)}) + (F\ (r\ j) - F\ (l\ j))$ 
    using psubset(1) psubset.prem(1) j
    apply (subst sum.union_disjoint)
    apply simp_all
    apply (subst sum.union_disjoint)
    apply auto
    apply (metis less_le_not_le)
    done
  also (xtrans) have  $(\sum_{i \in ?S1. F\ (r\ i) - F\ (l\ i)}) \geq F\ (l\ j) - F\ u$ 
    using  $\langle j \in S \rangle \langle \text{finite } S \rangle$  psubset.prem j
    apply (intro psubset.IH psubset)
    apply (auto simp: subset_eq Ball_def)
    apply (metis less_le_trans not_le)
    done
  also (xtrans) have  $(\sum_{i \in ?S2. F\ (r\ i) - F\ (l\ i)}) \geq F\ v - F\ (r\ j)$ 
    using  $\langle j \in S \rangle \langle \text{finite } S \rangle$  psubset.prem j
    apply (intro psubset.IH psubset)
    apply (auto simp: subset_eq Ball_def)
    apply (metis le_less_trans not_le)
    done
  finally (xtrans) show ?case
    by (auto simp: add_mono)
qed
qed
qed }
note claim2 = this

have ennreal  $(F\ b - F\ a) \leq (\sum i. \text{ennreal } (F\ (r\ i) - F\ (l\ i)))$ 
proof (rule ennreal_le_epsilon)
  fix epsilon :: real assume egt0:  $\epsilon > 0$ 
  have  $\forall i. \exists d > 0. F\ (r\ i + d) < F\ (r\ i) + \epsilon / 2^{(i+2)}$ 
  proof
    fix i

```

```

note right_cont_F [of r i]
thus  $\exists d > 0. F (r\ i + d) < F (r\ i) + \text{epsilon} / 2^{i+2}$ 
  apply -
  apply (subst (asm) continuous_at_right_real_increasing)
  apply (rule mono_F, assumption)
  apply (drule_tac  $x = \text{epsilon} / 2^{i+2}$  in spec)
  apply (erule impE)
  using egt0 by (auto simp add: field_simps)
qed
then obtain delta where
  deltai_gt0:  $\bigwedge i. \text{delta } i > 0$  and
  deltai_prop:  $\bigwedge i. F (r\ i + \text{delta } i) < F (r\ i) + \text{epsilon} / 2^{i+2}$ 
  by metis
have  $\exists a' > a. F\ a' - F\ a < \text{epsilon} / 2$ 
  apply (insert right_cont_F [of a])
  apply (subst (asm) continuous_at_right_real_increasing)
  using mono_F apply force
  apply (drule_tac  $x = \text{epsilon} / 2$  in spec)
  using egt0 unfolding mult.commute [of 2] by force
then obtain a' where a'_lea [arith]:  $a' > a$  and
  a_prop:  $F\ a' - F\ a < \text{epsilon} / 2$ 
  by auto
define S' where  $S' = \{i. l\ i < r\ i\}$ 
obtain S :: nat set where
   $S \subseteq S'$  and finS: finite S and
  Sprop:  $\{a'..b\} \subseteq (\bigcup i \in S. \{l\ i <..< r\ i + \text{delta } i\})$ 
proof (rule compactE_image)
  show compact  $\{a'..b\}$ 
  by (rule compact_Icc)
  show  $\bigwedge i. i \in S' \implies \text{open } (\{l\ i <..< r\ i + \text{delta } i\})$  by auto
  have  $\{a'..b\} \subseteq \{a <.. b\}$ 
  by auto
  also have  $\{a <.. b\} = (\bigcup i \in S'. \{l\ i <..< r\ i\})$ 
    unfolding lr_eq_ab[symmetric] by (fastforce simp add: S'_def intro:
less_le_trans)
  also have  $\dots \subseteq (\bigcup i \in S'. \{l\ i <..< r\ i + \text{delta } i\})$ 
    apply (intro UN_mono)
    apply (auto simp: S'_def)
    apply (cut_tac i=i in deltai_gt0)
    apply simp
    done
  finally show  $\{a'..b\} \subseteq (\bigcup i \in S'. \{l\ i <..< r\ i + \text{delta } i\})$  .
qed
with S'_def have Sprop2:  $\bigwedge i. i \in S \implies l\ i < r\ i$  by auto
from finS have  $\exists n. \forall i \in S. i \leq n$ 
  by (subst finite_nat_set_iff_bounded_le [symmetric])
then obtain n where Sbound [rule_format]:  $\forall i \in S. i \leq n$  ..
have  $F\ b - F\ a' \leq (\sum i \in S. F (r\ i + \text{delta } i) - F (l\ i))$ 
  apply (rule claim2 [rule_format])

```

```

    using finS Sprop apply auto
    apply (frule Sprop2)
    apply (subgoal_tac delta i > 0)
    apply arith
    by (rule deltai_gt0)
  also have ... ≤ (∑ i ∈ S. F(r i) - F(l i) + epsilon / 2^(i+2))
    apply (rule sum_mono)
    apply simp
    apply (rule order_trans)
    apply (rule less_imp_le)
    apply (rule deltai_prop)
    by auto
  also have ... = (∑ i ∈ S. F(r i) - F(l i)) +
    (epsilon / 4) * (∑ i ∈ S. (1 / 2)^i) (is _ = ?t + _)
    by (subst sum.distrib) (simp add: field_simps sum_distrib_left)
  also have ... ≤ ?t + (epsilon / 4) * (∑ i < Suc n. (1 / 2)^i)
    apply (rule add_left_mono)
    apply (rule mult_left_mono)
    apply (rule sum_mono2)
    using egt0 apply auto
    by (frule Sbound, auto)
  also have ... ≤ ?t + (epsilon / 2)
    apply (rule add_left_mono)
    apply (subst geometric_sum)
    apply auto
    apply (rule mult_left_mono)
    using egt0 apply auto
  done
  finally have aux2: F b - F a' ≤ (∑ i ∈ S. F(r i) - F(l i)) + epsilon / 2
    by simp

  have F b - F a = (F b - F a') + (F a' - F a)
    by auto
  also have ... ≤ (F b - F a') + epsilon / 2
    using a_prop by (intro add_left_mono) simp
  also have ... ≤ (∑ i ∈ S. F(r i) - F(l i)) + epsilon / 2 + epsilon / 2
    apply (intro add_right_mono)
    apply (rule aux2)
  done
  also have ... = (∑ i ∈ S. F(r i) - F(l i)) + epsilon
    by auto
  also have ... ≤ (∑ i ≤ n. F(r i) - F(l i)) + epsilon
    using finS Sbound Sprop by (auto intro!: add_right_mono sum_mono2)
  finally have ennreal (F b - F a) ≤ (∑ i ≤ n. ennreal (F(r i) - F(l i))) +
    epsilon
    using egt0 by (simp add: sum_nonneg flip: ennreal_plus)
  then show ennreal (F b - F a) ≤ (∑ i. ennreal (F(r i) - F(l i))) + (epsilon
    :: real)
    by (rule order_trans) (auto intro!: add_mono sum_le_suminf simp del:

```

```

sum_ennreal)
qed
moreover have ( $\sum i. \text{ennreal } (F (r\ i) - F (l\ i))$ )  $\leq \text{ennreal } (F\ b - F\ a)$ 
  using  $\langle a \leq b \rangle$  by (auto intro!: suminf_le_const ennreal_le_iff[THEN iffD2]
claim1)
ultimately show ( $\sum n. \text{ennreal } (F (r\ n) - F (l\ n))$ )  $= \text{ennreal } (F\ b - F\ a)$ 
  by (rule antisym[rotated])
qed (auto simp: Ioc_inj mono_F)

```

```

lemma measure_interval_measure_Ioc:
  assumes  $a \leq b$  and  $\bigwedge x\ y. x \leq y \implies F\ x \leq F\ y$  and  $\bigwedge a. \text{continuous } (\text{at\_right } a) F$ 
  shows  $\text{measure } (\text{interval\_measure } F) \{a <.. b\} = F\ b - F\ a$ 
  unfolding measure_def
  by (simp add: asms emeasure_interval_measure_Ioc)

```

```

lemma emeasure_interval_measure_Ioc_eq:
  ( $\bigwedge x\ y. x \leq y \implies F\ x \leq F\ y$ )  $\implies (\bigwedge a. \text{continuous } (\text{at\_right } a) F) \implies$ 
     $\text{emeasure } (\text{interval\_measure } F) \{a <.. b\} = (\text{if } a \leq b \text{ then } F\ b - F\ a \text{ else } 0)$ 
  using emeasure_interval_measure_Ioc[of  $a\ b\ F$ ] by auto

```

```

lemma sets_interval_measure [simp, measurable_cong]:
   $\text{sets } (\text{interval\_measure } F) = \text{sets borel}$ 
  apply (simp add: sets_extend_measure interval_measure_def borel_sigma_sets_Ioc)
  apply (rule sigma_sets_eqI)
  apply auto
  apply (case_tac  $a \leq b$ )
  apply (auto intro: sigma_sets.Empty)
  done

```

```

lemma space_interval_measure [simp]:  $\text{space } (\text{interval\_measure } F) = \text{UNIV}$ 
  by (simp add: interval_measure_def space_extend_measure)

```

```

lemma emeasure_interval_measure_Icc:
  assumes  $a \leq b$ 
  assumes mono_F:  $\bigwedge x\ y. x \leq y \implies F\ x \leq F\ y$ 
  assumes cont_F:  $\text{continuous\_on } \text{UNIV } F$ 
  shows  $\text{emeasure } (\text{interval\_measure } F) \{a .. b\} = F\ b - F\ a$ 
  proof (rule tendsto_unique)
    { fix  $a\ b :: \text{real}$  assume  $a \leq b$  then have  $\text{emeasure } (\text{interval\_measure } F) \{a$ 
       $<.. b\} = F\ b - F\ a$ 
      using cont_F
      by (subst emeasure_interval_measure_Ioc)
      (auto intro: mono_F continuous_within_subset simp: continuous_on_eq_continuous_within)
    }
  note * = this

```

```

let ?F = interval_measure F
show (( $\lambda a. F\ b - F\ a$ )  $\longrightarrow \text{emeasure } ?F \{a..b\}$ ) (at_left a)

```

```

proof (rule tendsto_at_left_sequentially)
  show  $a - 1 < a$  by simp
  fix  $X$  assume  $\bigwedge n. X\ n < a$   $\text{incseq } X\ X \longrightarrow a$ 
  with  $\langle a \leq b \rangle$  have  $(\lambda n. \text{emeasure } ?F \{X\ n <.. b\}) \longrightarrow \text{emeasure } ?F (\bigcap n. \{X\ n <.. b\})$ 
  apply (intro Lim_emeasure_decseq)
  apply (auto simp: decseq_def incseq_def emeasure_interval_measure_Ioc *)
  apply force
  apply (subst (asm) *)
  apply (auto intro: less_le_trans less_imp_le)
  done
  also have  $(\bigcap n. \{X\ n <.. b\}) = \{a..b\}$ 
  using  $\langle \bigwedge n. X\ n < a \rangle$ 
  apply auto
  apply (rule LIMSEQ_le_const2[OF  $\langle X \longrightarrow a \rangle$ ])
  apply (auto intro: less_imp_le)
  apply (auto intro: less_le_trans)
  done
  also have  $(\lambda n. \text{emeasure } ?F \{X\ n <.. b\}) = (\lambda n. F\ b - F\ (X\ n))$ 
  using  $\langle \bigwedge n. X\ n < a \rangle$   $\langle a \leq b \rangle$  by (subst *) (auto intro: less_imp_le less_le_trans)
  finally show  $(\lambda n. F\ b - F\ (X\ n)) \longrightarrow \text{emeasure } ?F \{a..b\}$  .
qed
show  $((\lambda a. \text{ennreal } (F\ b - F\ a)) \longrightarrow F\ b - F\ a)$  (at_left a)
  by (rule continuous_on_tendsto_compose[where  $g = \lambda x. x$  and  $s = \text{UNIV}$ ])
  (auto simp: continuous_on_ennreal continuous_on_diff cont_F)
qed (rule trivial_limit_at_left_real)

```

```

lemma sigma_finite_interval_measure:
  assumes  $\text{mono}_F: \bigwedge x\ y. x \leq y \implies F\ x \leq F\ y$ 
  assumes  $\text{right\_cont}_F: \bigwedge a. \text{continuous } (\text{at\_right } a)\ F$ 
  shows  $\text{sigma\_finite\_measure } (\text{interval\_measure } F)$ 
  apply unfold_locales
  apply (intro exI[of _  $(\lambda(a, b). \{a <.. b\})$  '  $(\mathbb{Q} \times \mathbb{Q})$ ])
  apply (auto intro!: Rats_no_top_le Rats_no_bot_less countable_rat simp: emeasure_interval_measure_Ioc_eq[OF assms])
  done

```

6.13.2 Lebesgue-Borel measure

```

definition lborel :: ('a :: euclidean_space) measure where
  lborel = distr  $(\Pi_M\ b \in \text{Basis}. \text{interval\_measure } (\lambda x. x))$  borel  $(\lambda f. \sum_{b \in \text{Basis}} f\ b *_{\mathbb{R}} b)$ 

```

```

abbreviation lebesgue :: 'a::euclidean_space measure
where lebesgue  $\equiv$  completion lborel

```

```

abbreviation lebesgue_on :: 'a set  $\Rightarrow$  'a::euclidean_space measure
where lebesgue_on  $\Omega \equiv$  restrict_space (completion lborel)  $\Omega$ 

```

lemma *lebesgue_on_mono*:

assumes *major*: $\text{AE } x \text{ in lebesgue\_on } S. P \ x$ **and** *minor*: $\bigwedge x. \llbracket P \ x; x \in S \rrbracket \implies Q \ x$
shows $\text{AE } x \text{ in lebesgue\_on } S. Q \ x$
proof –
have $\text{AE } a \text{ in lebesgue\_on } S. P \ a \longrightarrow Q \ a$
using *minor space_restrict_space* **by** *fastforce*
then show *?thesis*
using *major* **by** *auto*
qed

lemma *integral_eq_zero_null_sets*:

assumes $S \in \text{null\_sets lebesgue}$
shows $\text{integral}^L (\text{lebesgue\_on } S) f = 0$
proof (*rule integral_eq_zero_AE*)
show $\text{AE } x \text{ in lebesgue\_on } S. f \ x = 0$
by (*metis (no_types, lifting) assms AE_not_in lebesgue_on_mono null_setsD2 null_sets_restrict_space order_refl*)
qed

lemma

shows $\text{sets\_lborel}[simp, measurable\_cong]: \text{sets lborel} = \text{sets borel}$
and $\text{space\_lborel}[simp]: \text{space lborel} = \text{space borel}$
and $\text{measurable\_lborel1}[simp]: \text{measurable } M \text{ lborel} = \text{measurable } M \text{ borel}$
and $\text{measurable\_lborel2}[simp]: \text{measurable lborel } M = \text{measurable borel } M$
by (*simp_all add: lborel_def*)

lemma *space_lebesgue_on [simp]*: $\text{space} (\text{lebesgue\_on } S) = S$

by (*simp add: space_restrict_space*)

lemma *sets_lebesgue_on_refl [iff]*: $S \in \text{sets} (\text{lebesgue\_on } S)$

by (*metis inf_top.right_neutral sets.top space_borel space_completion space_lborel space_restrict_space*)

lemma *Compl_in_sets_lebesgue*: $-A \in \text{sets lebesgue} \longleftrightarrow A \in \text{sets lebesgue}$

by (*metis Compl_eq_Diff_UNIV double_compl space_borel space_completion space_lborel Sigma_Algebra.sets.compl_sets*)

lemma *measurable_lebesgue_cong*:

assumes $\bigwedge x. x \in S \implies f \ x = g \ x$

shows $f \in \text{measurable} (\text{lebesgue\_on } S) \ M \longleftrightarrow g \in \text{measurable} (\text{lebesgue\_on } S) \ M$

by (*metis (mono_tags, lifting) IntD1 assms measurable_cong_simp space_restrict_space*)

lemma *lebesgue_on_UNIV_eq*: $\text{lebesgue\_on UNIV} = \text{lebesgue}$

proof –

have $\text{measure\_of UNIV} (\text{sets lebesgue}) (\text{emeasure lebesgue}) = \text{lebesgue}$

by (*metis measure_of_of_measure space_borel space_completion space_lborel*)


```

  then show ?thesis
    by (auto simp: restrict_space_def)
qed

```

```

lemma integral_restrict_Int:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes  $S \in \text{sets lebesgue}$   $T \in \text{sets lebesgue}$ 
  shows  $\text{integral}^L (\text{lebesgue\_on } T) (\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) = \text{integral}^L$ 
 $(\text{lebesgue\_on } (S \cap T)) f$ 
proof -
  have  $(\lambda x. \text{indicat\_real } T x *_{\mathbb{R}} (\text{if } x \in S \text{ then } f x \text{ else } 0)) = (\lambda x. \text{indicat\_real } (S$ 
 $\cap T) x *_{\mathbb{R}} f x)$ 
  by (force simp: indicator_def)
  then show ?thesis
    by (simp add: assms sets.Int Bochner_Integration.integral_restrict_space)
qed

```

```

lemma integral_restrict:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes  $S \subseteq T$   $S \in \text{sets lebesgue}$   $T \in \text{sets lebesgue}$ 
  shows  $\text{integral}^L (\text{lebesgue\_on } T) (\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) = \text{integral}^L$ 
 $(\text{lebesgue\_on } S) f$ 
  using integral_restrict_Int [of  $S T f$ ] assms
  by (simp add: Int_absorb2)

```

```

lemma integral_restrict_UNIV:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes  $S \in \text{sets lebesgue}$ 
  shows  $\text{integral}^L \text{lebesgue } (\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) = \text{integral}^L (\text{lebesgue\_on } S) f$ 
  using integral_restrict_Int [of  $S \text{UNIV } f$ ] assms
  by (simp add: lebesgue_on_UNIV_eq)

```

```

lemma integrable_lebesgue_on_empty [iff]:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::{second_countable_topology,banach}
  shows integrable (lebesgue_on {}) f
  by (simp add: integrable_restrict_space)

```

```

lemma integral_lebesgue_on_empty [simp]:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::{second_countable_topology,banach}
  shows  $\text{integral}^L (\text{lebesgue\_on } \{\}) f = 0$ 
  by (simp add: Bochner_Integration.integral_empty)

```

```

lemma has_bochner_integral_restrict_space:
  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  assumes  $\Omega: \Omega \cap \text{space } M \in \text{sets } M$ 
  shows  $\text{has\_bochner\_integral } (\text{restrict\_space } M \Omega) f i$ 
 $\longleftrightarrow \text{has\_bochner\_integral } M (\lambda x. \text{indicator } \Omega x *_{\mathbb{R}} f x) i$ 
  by (simp add: integrable_restrict_space [OF assms] integral_restrict_space [OF
  assms] has_bochner_integral_iff)

```

```

lemma integrable_restrict_UNIV:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::{banach, second_countable_topology}
  assumes S: S  $\in$  sets lebesgue
  shows integrable lebesgue ( $\lambda x.$  if  $x \in S$  then  $f x$  else 0)  $\longleftrightarrow$  integrable (lebesgue_on
S) f
  using has_bochner_integral_restrict_space [of S lebesgue f] assms
  by (simp add: integrable.simps indicator_scaleR_eq_if)

```

```

lemma integral_mono_lebesgue_on_AE:
  fixes f::_  $\Rightarrow$  real
  assumes f: integrable (lebesgue_on T) f
    and gf: AE x in (lebesgue_on S).  $g x \leq f x$ 
    and f0: AE x in (lebesgue_on T).  $0 \leq f x$ 
    and S  $\subseteq$  T and S: S  $\in$  sets lebesgue and T: T  $\in$  sets lebesgue
  shows ( $\int x. g x \partial(\text{lebesgue\_on } S)$ )  $\leq$  ( $\int x. f x \partial(\text{lebesgue\_on } T)$ )
proof -
  have ( $\int x. g x \partial(\text{lebesgue\_on } S)$ ) = ( $\int x.$  (if  $x \in S$  then  $g x$  else 0)  $\partial$ lebesgue)
    by (simp add: Lebesgue_Measure.integral_restrict_UNIV S)
  also have ...  $\leq$  ( $\int x.$  (if  $x \in T$  then  $f x$  else 0)  $\partial$ lebesgue)
  proof (rule Bochner_Integration.integral_mono_AE')
    show integrable lebesgue ( $\lambda x.$  if  $x \in T$  then  $f x$  else 0)
      by (simp add: integrable_restrict_UNIV T f)
    show AE x in lebesgue. (if  $x \in S$  then  $g x$  else 0)  $\leq$  (if  $x \in T$  then  $f x$  else 0)
      using assms by (auto simp: AE_restrict_space_iff)
    show AE x in lebesgue.  $0 \leq$  (if  $x \in T$  then  $f x$  else 0)
      using f0 by (simp add: AE_restrict_space_iff T)
  qed
  also have ... = ( $\int x. f x \partial(\text{lebesgue\_on } T)$ )
    using Lebesgue_Measure.integral_restrict_UNIV T by blast
  finally show ?thesis .
qed

```

6.13.3 Borel measurability

```

lemma borel_measurable_if_I:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes f: f  $\in$  borel_measurable (lebesgue_on S) and S: S  $\in$  sets lebesgue
  shows ( $\lambda x.$  if  $x \in S$  then  $f x$  else 0)  $\in$  borel_measurable lebesgue
proof -
  have eq:  $\{x. x \notin S\} \cup \{x. f x \in Y\} = \{x. x \notin S\} \cup \{x. f x \in Y\} \cap S$  for Y
    by blast
  show ?thesis
  using f S
  apply (simp add: vimage_def in_borel_measurable_borel Ball_def)
  apply (elim_all_forward imp_forward asm_rl)
  apply (simp only: Collect_conj_eq Collect_disj_eq imp_conv_disj_eq)
  apply (auto simp: Compl_eq [symmetric] Compl_in_sets_lebesgue sets_restrict_space_iff)
  done

```

qed

lemma *borel_measurable_if_D*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $(\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) \in \text{borel\_measurable lebesgue}$
shows $f \in \text{borel\_measurable (lebesgue\_on } S)$
using *assms*
apply (*simp add: in_borel_measurable_borel Ball_def*)
apply (*elim all_forward imp_forward asm_rl*)
apply (*force simp: space_restrict_space sets_restrict_space image_iff intro: rev_bexI*)
done

lemma *borel_measurable_if*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $S \in \text{sets lebesgue}$
shows $(\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) \in \text{borel\_measurable lebesgue} \longleftrightarrow f \in \text{borel\_measurable (lebesgue\_on } S)$
using *assms borel_measurable_if_D borel_measurable_if_I* **by** *blast*

lemma *borel_measurable_if_lebesgue_on*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $S \in \text{sets lebesgue } T \in \text{sets lebesgue } S \subseteq T$
shows $(\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) \in \text{borel\_measurable (lebesgue\_on } T) \longleftrightarrow f \in \text{borel\_measurable (lebesgue\_on } S)$
(is ?lhs = ?rhs)

proof

assume *?lhs* **then show** *?rhs*
using *measurable_restrict_mono [OF _ ‹S ⊆ T›]*
by (*subst measurable_lebesgue_cong [where g = (λx. if x ∈ S then f x else 0)]*) *auto*
next
assume *?rhs* **then show** *?lhs*
by (*simp add: ‹S ∈ sets lebesgue› borel_measurable_if_I measurable_restrict_space1*)
qed

lemma *borel_measurable_vimage_halfspace_component_lt*:
 $f \in \text{borel\_measurable (lebesgue\_on } S) \longleftrightarrow$
 $(\forall a\ i. i \in \text{Basis} \longrightarrow \{x \in S. f\ x \cdot i < a\} \in \text{sets (lebesgue\_on } S))$
apply (*rule trans [OF borel_measurable_iff_halfspace_less]*)
apply (*fastforce simp add: space_restrict_space*)
done

lemma *borel_measurable_vimage_halfspace_component_ge*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows $f \in \text{borel\_measurable (lebesgue\_on } S) \longleftrightarrow$
 $(\forall a\ i. i \in \text{Basis} \longrightarrow \{x \in S. f\ x \cdot i \geq a\} \in \text{sets (lebesgue\_on } S))$
apply (*rule trans [OF borel_measurable_iff_halfspace_ge]*)
apply (*fastforce simp add: space_restrict_space*)

done

lemma *borel_measurable_vimage_halfspace_component_gt*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
shows $f \in \text{borel\_measurable } (\text{lebesgue\_on } S) \longleftrightarrow$
 $(\forall a \ i. \ i \in \text{Basis} \longrightarrow \{x \in S. f \ x \cdot i > a\} \in \text{sets } (\text{lebesgue\_on } S))$
apply (rule trans [OF *borel_measurable_iff_halfspace_greater*])
apply (fastforce simp add: *space_restrict_space*)
done

lemma *borel_measurable_vimage_halfspace_component_le*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
shows $f \in \text{borel\_measurable } (\text{lebesgue\_on } S) \longleftrightarrow$
 $(\forall a \ i. \ i \in \text{Basis} \longrightarrow \{x \in S. f \ x \cdot i \leq a\} \in \text{sets } (\text{lebesgue\_on } S))$
apply (rule trans [OF *borel_measurable_iff_halfspace_le*])
apply (fastforce simp add: *space_restrict_space*)
done

lemma
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
shows *borel_measurable_vimage_open_interval*:
 $f \in \text{borel\_measurable } (\text{lebesgue\_on } S) \longleftrightarrow$
 $(\forall a \ b. \ \{x \in S. f \ x \in \text{box } a \ b\} \in \text{sets } (\text{lebesgue\_on } S))$ (**is** ?thesis1)
and *borel_measurable_vimage_open*:
 $f \in \text{borel\_measurable } (\text{lebesgue\_on } S) \longleftrightarrow$
 $(\forall T. \ \text{open } T \longrightarrow \{x \in S. f \ x \in T\} \in \text{sets } (\text{lebesgue\_on } S))$ (**is** ?thesis2)
proof –
have $\{x \in S. f \ x \in \text{box } a \ b\} \in \text{sets } (\text{lebesgue\_on } S)$ **if** $f \in \text{borel\_measurable } (\text{lebesgue\_on } S)$ **for** $a \ b$
proof –
have $S = S \cap \text{space lebesgue}$
by *simp*
then have $S \cap (f - ' \text{box } a \ b) \in \text{sets } (\text{lebesgue\_on } S)$
by (*metis* (*no_types*) *box_borel_in_borel_measurable_borel_inf_sup_aci*(1) *space_restrict_space* *that*)
then show ?thesis
by (*simp* add: *Collect_conj_eq_vimage_def*)
qed
moreover
have $\{x \in S. f \ x \in T\} \in \text{sets } (\text{lebesgue\_on } S)$
if $T: \bigwedge a \ b. \ \{x \in S. f \ x \in \text{box } a \ b\} \in \text{sets } (\text{lebesgue\_on } S)$ **open** T **for** T
proof –
obtain $\mathcal{D}$ **where** *countable* $\mathcal{D}$ **and** $\mathcal{D}: \bigwedge X. \ X \in \mathcal{D} \Longrightarrow \exists a \ b. \ X = \text{box } a \ b \cup \mathcal{D}$
 $= T$
using *open_countable_Union_open_box* **that** $\langle \text{open } T \rangle$ **by** *metis*
then have *eq*: $\{x \in S. f \ x \in T\} = (\bigcup U \in \mathcal{D}. \ \{x \in S. f \ x \in U\})$
by *blast*
have $\{x \in S. f \ x \in U\} \in \text{sets } (\text{lebesgue\_on } S)$ **if** $U \in \mathcal{D}$ **for** U
using *that* $T \ \mathcal{D}$ **by** *blast*

```

    then show ?thesis
      by (auto simp: eq intro: Sigma_Algebra.sets.countable_UN' [OF ‹countable
 $\mathcal{D}$ ›])
    qed
  moreover
  have eq:  $\{x \in S. f\ x \cdot i < a\} = \{x \in S. f\ x \in \{y. y \cdot i < a\}\}$  for  $i\ a$ 
    by auto
  have  $f \in \text{borel\_measurable } (\text{lebesgue\_on } S)$ 
    if  $\bigwedge T. \text{open } T \implies \{x \in S. f\ x \in T\} \in \text{sets } (\text{lebesgue\_on } S)$ 
    by (metis (no_types) eq borel_measurable_vimage_halfspace_component_lt
open_halfspace_component_lt that)
  ultimately show ?thesis1 ?thesis2
    by blast+
  qed

```

```

lemma borel_measurable_vimage_closed:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  shows  $f \in \text{borel\_measurable } (\text{lebesgue\_on } S) \longleftrightarrow$ 
    ( $\forall T. \text{closed } T \longrightarrow \{x \in S. f\ x \in T\} \in \text{sets } (\text{lebesgue\_on } S)$ )
    (is ?lhs = ?rhs)
proof -
  have eq:  $\{x \in S. f\ x \in T\} = S - \{x \in S. f\ x \in (-\ T)\}$  for  $T$ 
    by auto
  show ?thesis
    apply (simp add: borel_measurable_vimage_open, safe)
    apply (simp_all (no_asm) add: eq)
    apply (intro sets.Diff sets_lebesgue_on_refl, force simp: closed_open)
    apply (intro sets.Diff sets_lebesgue_on_refl, force simp: open_closed)
    done
qed

```

```

lemma borel_measurable_vimage_closed_interval:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  shows  $f \in \text{borel\_measurable } (\text{lebesgue\_on } S) \longleftrightarrow$ 
    ( $\forall a\ b. \{x \in S. f\ x \in \text{cbox } a\ b\} \in \text{sets } (\text{lebesgue\_on } S)$ )
    (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
    using borel_measurable_vimage_closed by blast
  next
  assume RHS: ?rhs
  have  $\{x \in S. f\ x \in T\} \in \text{sets } (\text{lebesgue\_on } S)$  if open  $T$  for  $T$ 
  proof -
    obtain  $\mathcal{D}$  where countable  $\mathcal{D}$  and  $\mathcal{D}: \mathcal{D} \subseteq \text{Pow } T \wedge X. X \in \mathcal{D} \implies \exists a\ b. X$ 
    =  $\text{cbox } a\ b \cup \mathcal{D} = T$ 
    using open_countable_Union_open_cbox that ‹open  $T$ › by metis
    then have eq:  $\{x \in S. f\ x \in T\} = (\bigcup U \in \mathcal{D}. \{x \in S. f\ x \in U\})$ 
    by blast
    have  $\{x \in S. f\ x \in U\} \in \text{sets } (\text{lebesgue\_on } S)$  if  $U \in \mathcal{D}$  for  $U$ 

```

```

    using that  $\mathcal{D}$  by (metis RHS)
  then show ?thesis
    by (auto simp: eq intro: Sigma_Algebra.sets.countable_UN' [OF ‹countable
 $\mathcal{D}$ ›])
  qed
  then show ?lhs
    by (simp add: borel_measurable_vimage_open)
  qed

```

```

lemma borel_measurable_vimage_borel:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  shows  $f \in \text{borel\_measurable } (\text{lebesgue\_on } S) \longleftrightarrow$ 
    ( $\forall T. T \in \text{sets borel} \longrightarrow \{x \in S. f\ x \in T\} \in \text{sets } (\text{lebesgue\_on } S)$ )
    (is ?lhs = ?rhs)
proof
  assume  $f: ?lhs$ 
  then show ?rhs
    using measurable_sets [OF  $f$ ]
    by (simp add: Collect_conj_eq inf_sup_aci(1) space_restrict_space vimage_def)
  qed (simp add: borel_measurable_vimage_open_interval)

```

```

lemma lebesgue_measurable_vimage_borel:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $f \in \text{borel\_measurable lebesgue } T \in \text{sets borel}$ 
  shows  $\{x. f\ x \in T\} \in \text{sets lebesgue}$ 
  using assms borel_measurable_vimage_borel [of  $f$  UNIV] by auto

```

```

lemma borel_measurable_lebesgue_preimage_borel:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  shows  $f \in \text{borel\_measurable lebesgue} \longleftrightarrow$ 
    ( $\forall T. T \in \text{sets borel} \longrightarrow \{x. f\ x \in T\} \in \text{sets lebesgue}$ )
  apply (intro iffI allI impI lebesgue_measurable_vimage_borel)
  apply (auto simp: in_borel_measurable_borel vimage_def)
  done

```

6.13.4 Measurability of continuous functions

```

lemma continuous_imp_measurable_on_sets_lebesgue:
  assumes  $f: \text{continuous\_on } S$  and  $S: S \in \text{sets lebesgue}$ 
  shows  $f \in \text{borel\_measurable } (\text{lebesgue\_on } S)$ 
proof -
  have  $\text{sets } (\text{restrict\_space borel } S) \subseteq \text{sets } (\text{lebesgue\_on } S)$ 
    by (simp add: mono_restrict_space subsetI)
  then show ?thesis
    by (simp add: borel_measurable_continuous_on_restrict [OF  $f$ ] borel_measurable_subalgebra
        space_restrict_space)
  qed

```

lemma *id_borel_measurable_lebesgue* [iff]: $id \in \text{borel\_measurable lebesgue}$
by (simp add: measurable_completion)

lemma *id_borel_measurable_lebesgue_on* [iff]: $id \in \text{borel\_measurable (lebesgue\_on } S)$
by (simp add: measurable_completion measurable_restrict_space1)

context
begin

interpretation *sigma_finite_measure_interval_measure* ($\lambda x. x$)
by (rule sigma_finite_interval_measure) auto
interpretation *finite_product_sigma_finite* $\lambda_. \text{interval\_measure } (\lambda x. x) \text{Basis}$
proof qed simp

lemma *lborel_eq_real*: $\text{lborel} = \text{interval\_measure } (\lambda x. x)$
unfolding *lborel_def Basis_real_def*
using *distr_id*[of *interval_measure* ($\lambda x. x$)]
by (subst *distr_component*[symmetric])
(simp_all add: *distr_distr comp_def* del: *distr_id cong: distr_cong*)

lemma *lborel_eq*: $\text{lborel} = \text{distr } (\Pi_M b \in \text{Basis}. \text{lborel}) \text{borel } (\lambda f. \sum b \in \text{Basis}. f b *_{\mathbb{R}} b)$
by (subst *lborel_def*) (simp add: *lborel_eq_real*)

lemma *nn_integral_lborel_prod*:
assumes [measurable]: $\bigwedge b. b \in \text{Basis} \implies f b \in \text{borel\_measurable borel}$
assumes *nn[simp]*: $\bigwedge b x. b \in \text{Basis} \implies 0 \leq f b x$
shows $(\int^+ x. (\prod b \in \text{Basis}. f b (x \cdot b)) \partial \text{lborel}) = (\prod b \in \text{Basis}. (\int^+ x. f b x \partial \text{lborel}))$
by (simp add: *lborel_def nn_integral_distr product_nn_integral_prod product_nn_integral_singleton*)

lemma *emeasure_lborel_Icc[simp]*:
fixes $l u :: \text{real}$
assumes [simp]: $l \leq u$
shows $\text{emeasure lborel } \{l .. u\} = u - l$
proof –
have $((\lambda f. f 1) - ' \{l..u\} \cap \text{space } (\Pi_M \{1\} (\lambda b. \text{interval\_measure } (\lambda x. x)))) = \{1 :: \text{real}\} \rightarrow_E \{l..u\}$
by (auto simp: *space_PiM*)
then show ?thesis
by (simp add: *lborel_def emeasure_distr emeasure_PiM emeasure_interval_measure_Icc*)
qed

lemma *emeasure_lborel_Icc_eq*: $\text{emeasure lborel } \{l .. u\} = \text{ennreal } (if l \leq u \text{ then } u - l \text{ else } 0)$
by simp

lemma *emeasure_lborel_cbox[simp]*:
assumes *[simp]*: $\bigwedge b. b \in \text{Basis} \implies l \cdot b \leq u \cdot b$
shows *emeasure_lborel (cbox l u) = $(\prod_{b \in \text{Basis}} (u - l) \cdot b)$*
proof –
have $(\lambda x. \prod_{b \in \text{Basis}} \text{indicator } \{l \cdot b .. u \cdot b\} (x \cdot b) :: \text{ennreal}) = \text{indicator } (\text{cbox } l \ u)$
by (*auto simp: fun_eq_iff cbox_def split: split_indicator*)
then have *emeasure_lborel (cbox l u) = $(\int^+ x. (\prod_{b \in \text{Basis}} \text{indicator } \{l \cdot b .. u \cdot b\} (x \cdot b)) \partial \text{lborel})$*
by *simp*
also have $\dots = (\prod_{b \in \text{Basis}} (u - l) \cdot b)$
by (*subst nn_integral_lborel_prod*) (*simp_all add: prod_enreal inner_diff_left*)
finally show *?thesis* .
qed

lemma *AE_lborel_singleton*: *AE x in lborel::'a::euclidean_space measure. $x \neq c$*
using *SOME_Basis AE_discrete_difference [of {c} lborel] emeasure_lborel_cbox [of c c]*
by (*auto simp add: power_0_left*)

lemma *emeasure_lborel_Ioo[simp]*:
assumes *[simp]*: $l \leq u$
shows *emeasure_lborel $\{l <..< u\} = \text{ennreal } (u - l)$*
proof –
have *emeasure_lborel $\{l <..< u\} = \text{emeasure_lborel } \{l .. u\}$*
using *AE_lborel_singleton[of u] AE_lborel_singleton[of l] by (intro emeasure_eq_AE) auto*
then show *?thesis*
by *simp*
qed

lemma *emeasure_lborel_Ioc[simp]*:
assumes *[simp]*: $l \leq u$
shows *emeasure_lborel $\{l <.. u\} = \text{ennreal } (u - l)$*
proof –
have *emeasure_lborel $\{l <.. u\} = \text{emeasure_lborel } \{l .. u\}$*
using *AE_lborel_singleton[of u] AE_lborel_singleton[of l] by (intro emeasure_eq_AE) auto*
then show *?thesis*
by *simp*
qed

lemma *emeasure_lborel_Ico[simp]*:
assumes *[simp]*: $l \leq u$
shows *emeasure_lborel $\{l ..< u\} = \text{ennreal } (u - l)$*
proof –
have *emeasure_lborel $\{l ..< u\} = \text{emeasure_lborel } \{l .. u\}$*
using *AE_lborel_singleton[of u] AE_lborel_singleton[of l] by (intro emeasure_eq_AE) auto*

then show ?thesis
 by simp
 qed

lemma *emeasure_lborel_box*[simp]:
 assumes [simp]: $\bigwedge b. b \in \text{Basis} \implies l \cdot b \leq u \cdot b$
 shows *emeasure lborel* (box l u) = $(\prod_{b \in \text{Basis}} (u - l) \cdot b)$
 proof -
 have $(\lambda x. \prod_{b \in \text{Basis}} \text{indicator } \{l \cdot b <..< u \cdot b\} (x \cdot b) :: \text{ennreal}) = \text{indicator}$
 (box l u)
 by (auto simp: fun_eq_iff box_def split: split_indicator)
 then have *emeasure lborel* (box l u) = $(\int^+ x. (\prod_{b \in \text{Basis}} \text{indicator } \{l \cdot b <..< u \cdot b\} (x \cdot b))) \partial \text{lborel}$
 by simp
 also have $\dots = (\prod_{b \in \text{Basis}} (u - l) \cdot b)$
 by (subst nn_integral_lborel_prod) (simp_all add: prod_ennreal inner_diff_left)
 finally show ?thesis .
 qed

lemma *emeasure_lborel_cbox_eq*:
emeasure lborel (cbox l u) = (if $\forall b \in \text{Basis}. l \cdot b \leq u \cdot b$ then $\prod_{b \in \text{Basis}} (u - l) \cdot b$ else 0)
 using box_eq_empty(2)[THEN iffD2, of u l] by (auto simp: not_le)

lemma *emeasure_lborel_box_eq*:
emeasure lborel (box l u) = (if $\forall b \in \text{Basis}. l \cdot b \leq u \cdot b$ then $\prod_{b \in \text{Basis}} (u - l) \cdot b$ else 0)
 using box_eq_empty(1)[THEN iffD2, of u l] by (auto simp: not_le dest!: less_imp_le)
 force

lemma *emeasure_lborel_singleton*[simp]: *emeasure lborel* {x} = 0
 using *emeasure_lborel_cbox*[of x x] nonempty_Basis
 by (auto simp del: *emeasure_lborel_cbox* nonempty_Basis)

lemma *emeasure_lborel_cbox_finite*: *emeasure lborel* (cbox a b) < ∞
 by (auto simp: *emeasure_lborel_cbox_eq*)

lemma *emeasure_lborel_box_finite*: *emeasure lborel* (box a b) < ∞
 by (auto simp: *emeasure_lborel_box_eq*)

lemma *emeasure_lborel_ball_finite*: *emeasure lborel* (ball c r) < ∞

proof -
 have bounded (ball c r) by simp
 from bounded_subset_cbox_symmetric[OF this] obtain a where $a: \text{ball } c \ r \subseteq \text{cbox } (-a) \ a$
 by auto
 hence *emeasure lborel* (ball c r) $\leq$ *emeasure lborel* (cbox (-a) a)
 by (intro *emeasure_mono*) auto
 also have $\dots < \infty$ by (simp add: *emeasure_lborel_cbox_eq*)
 qed

1702

finally show *?thesis* .
qed

lemma *emeasure_lborel_cball_finite*: *emeasure lborel (cball c r) < ∞*
proof –
 have *bounded (cball c r)* **by** *simp*
 from *bounded_subset_cbox_symmetric[OF this]* **obtain a where** *a: cball c r ⊆ cbox (−a) a*
 by *auto*
 hence *emeasure lborel (cball c r) ≤ emeasure lborel (cbox (−a) a)*
 by (*intro emeasure_mono*) *auto*
 also have *... < ∞* **by** (*simp add: emeasure_lborel_cbox_eq*)
 finally show *?thesis* .
qed

lemma *fmeasurable_cbox [iff]*: *cbox a b ∈ fmeasurable lborel*
and *fmeasurable_box [iff]*: *box a b ∈ fmeasurable lborel*
by (*auto simp: fmeasurable_def emeasure_lborel_box_eq emeasure_lborel_cbox_eq*)

lemma
 fixes *l u :: real*
 assumes [*simp*]: *l ≤ u*
 shows *measure_lborel_Icc[simp]: measure lborel {l .. u} = u − l*
 and *measure_lborel_Ico[simp]: measure lborel {l ..< u} = u − l*
 and *measure_lborel_Ioc[simp]: measure lborel {l <.. u} = u − l*
 and *measure_lborel_Ioo[simp]: measure lborel {l <..} = u − l*
by (*simp_all add: measure_def*)

lemma
 assumes [*simp*]: $\bigwedge b. b \in \text{Basis} \implies l \cdot b \leq u \cdot b$
 shows *measure_lborel_box[simp]: measure lborel (box l u) = $(\prod_{b \in \text{Basis}. (u - l) \cdot b}$*
 and *measure_lborel_cbox[simp]: measure lborel (cbox l u) = $(\prod_{b \in \text{Basis}. (u - l) \cdot b}$*
by (*simp_all add: measure_def inner_diff_left prod_nonneg*)

lemma *measure_lborel_cbox_eq*:
 measure lborel (cbox l u) = (if $\forall b \in \text{Basis}. l \cdot b \leq u \cdot b$ then $\prod_{b \in \text{Basis}. (u - l) \cdot b$ else 0)
using *box_eq_empty(2)[THEN iffD2, of u l]* **by** (*auto simp: not_le*)

lemma *measure_lborel_box_eq*:
 measure lborel (box l u) = (if $\forall b \in \text{Basis}. l \cdot b \leq u \cdot b$ then $\prod_{b \in \text{Basis}. (u - l) \cdot b$ else 0)
using *box_eq_empty(1)[THEN iffD2, of u l]* **by** (*auto simp: not_le dest!: less_imp_le*)
force

lemma *measure_lborel_singleton[simp]*: *measure lborel {x} = 0*
by (*simp add: measure_def*)

```

lemma sigma_finite_lborel: sigma_finite_measure lborel
proof
  show  $\exists A::'a \text{ set set. countable } A \wedge A \subseteq \text{sets lborel} \wedge \bigcup A = \text{space lborel} \wedge$ 
   $(\forall a \in A. \text{emeasure lborel } a \neq \infty)$ 
  by (intro exI[of _ range ( $\lambda n::\text{nat. box } (- \text{real } n *_{\mathbb{R}} \text{One}) (\text{real } n *_{\mathbb{R}} \text{One})$ )]])
  (auto simp: emeasure_lborel_cbox_eq UN_box_eq UNIV)
qed

end

lemma emeasure_lborel_UNIV [simp]: emeasure lborel (UNIV::'a::euclidean_space
set) =  $\infty$ 
proof -
  { fix  $n::\text{nat}$ 
    let ?Ba = Basis :: 'a set
    have  $\text{real } n \leq (2::\text{real}) \wedge \text{card } ?Ba * \text{real } n$ 
    by (simp add: mult_le_cancel_right1)
    also
    have  $\dots \leq (2::\text{real}) \wedge \text{card } ?Ba * \text{real } (\text{Suc } n) \wedge \text{card } ?Ba$ 
    apply (rule mult_left_mono)
    apply (metis DIM_positive One_nat_def less_eq_Suc_le less_imp_le of_nat_le_iff
of_nat_power_self_le_power_zero_less_Suc)
    apply (simp)
    done
    finally have  $\text{real } n \leq (2::\text{real}) \wedge \text{card } ?Ba * \text{real } (\text{Suc } n) \wedge \text{card } ?Ba .$ 
  } note [intro!] = this
show ?thesis
  unfolding UN_box_eq_UNIV[symmetric]
  apply (subst SUP_emeasure_incseq[symmetric])
  apply (auto simp: incseq_def subset_box inner_add_left
simp del: Sup_eq_top_iff SUP_eq_top_iff
intro!: ennreal_SUP_eq_top)
  done
qed

lemma emeasure_lborel_countable:
  fixes  $A::'a::\text{euclidean\_space set}$ 
  assumes countable A
  shows emeasure lborel A = 0
proof -
  have  $A \subseteq (\bigcup i. \{\text{from\_nat\_into } A \ i\})$  using from_nat_into_surj assms by
  force
  then have emeasure lborel A  $\leq$  emeasure lborel  $(\bigcup i. \{\text{from\_nat\_into } A \ i\})$ 
  by (intro emeasure_mono) auto
  also have emeasure lborel  $(\bigcup i. \{\text{from\_nat\_into } A \ i\}) = 0$ 
  by (rule emeasure_UN_eq_0) auto
  finally show ?thesis
  by (auto simp add: )

```

1704

qed

lemma *countable_imp_null_set_lborel*: *countable A $\implies A \in \text{null\_sets lborel}$*
by (*simp add: null_sets_def emeasure_lborel_countable sets.countable*)

lemma *finite_imp_null_set_lborel*: *finite A $\implies A \in \text{null\_sets lborel}$*
by (*intro countable_imp_null_set_lborel countable_finite*)

lemma *insert_null_sets_iff [simp]*: *insert a N $\in \text{null\_sets lebesgue} \longleftrightarrow N \in \text{null\_sets lebesgue}$*
(is ?lhs = ?rhs)

proof

assume *?lhs then show ?rhs*

by (*meson completion.complete2 subset_insertI*)

next

assume *?rhs then show ?lhs*

by (*simp add: null_sets.insert_in_sets null_setsI*)

qed

lemma *insert_null_sets_lebesgue_on_iff [simp]*:
assumes *a $\in S$ S $\in \text{sets lebesgue}$*
shows *insert a N $\in \text{null\_sets (lebesgue\_on S)}$ $\longleftrightarrow N \in \text{null\_sets (lebesgue\_on S)}$*
by (*simp add: assms null_sets_restrict_space*)

lemma *lborel_neq_count_space[simp]*: *lborel $\neq \text{count\_space } (A::('a::\text{ordered\_euclidean\_space}) \text{ set})$*

proof

assume *asm: lborel = count_space A*

have *space lborel = UNIV* **by** *simp*

hence [*simp*]: *A = UNIV* **by** (*subst (asm) asm*) (*simp only: space_count_space*)

have *emeasure lborel {undefined::'a} = 1*

by (*subst asm, subst emeasure_count_space_finite*) *auto*

moreover **have** *emeasure lborel {undefined} $\neq 1$* **by** *simp*

ultimately show *False* **by** *contradiction*

qed

lemma *mem_closed_if_AE_lebesgue_open*:

assumes *open S closed C*

assumes *AE x $\in S$ in lebesgue. x $\in C$*

assumes *x $\in S$*

shows *x $\in C$*

proof (*rule ccontr*)

assume *xC: x $\notin C$*

with *openE[of S - C] assms*

obtain *e* **where** *e: 0 < e ball x e $\subseteq S - C$*

by *blast*

then obtain *a b* **where** *box: x $\in \text{box } a b$ box a b $\subseteq S - C$*

by (*metis rational_boxes order_trans*)

```

then have  $0 < \text{emeasure lebesgue } (\text{box } a \ b)$ 
by (auto simp: emeasure_lborel_box_eq mem_box algebra_simps intro!: prod_pos)
also have  $\dots \leq \text{emeasure lebesgue } (S - C)$ 
using assms box
by (auto intro!: emeasure_mono)
also have  $\dots = 0$ 
using assms
by (auto simp: eventually_ae_filter completion.complete2 set_diff_eq null_setsD1)
finally show False by simp
qed

```

```

lemma mem_closed_if_AE_lebesgue:  $\text{closed } C \implies (\text{AE } x \text{ in lebesgue. } x \in C) \implies x \in C$ 
using mem_closed_if_AE_lebesgue_open[OF open_UNIV] by simp

```

6.13.5 Affine transformation on the Lebesgue-Borel

```

lemma lborel_eqI:
  fixes  $M :: 'a::\text{euclidean\_space} \text{ measure}$ 
  assumes emeasure_eq:  $\bigwedge l \ u. (\bigwedge b. b \in \text{Basis} \implies l \cdot b \leq u \cdot b) \implies \text{emeasure } M$ 
   $(\text{box } l \ u) = (\prod_{b \in \text{Basis}} (u - l) \cdot b)$ 
  assumes sets_eq:  $\text{sets } M = \text{sets lborel}$ 
  shows lborel = M
proof (rule measure_eqI_generator_eq)
  let ?E =  $\text{range } (\lambda(a, b). \text{box } a \ b :: 'a \text{ set})$ 
  show Int_stable ?E
    by (auto simp: Int_stable_def box_Int_box)

  show  $?E \subseteq \text{Pow UNIV}$  sets lborel = sigma_sets UNIV ?E  $\text{sets } M = \text{sigma_sets UNIV ?E}$ 
    by (simp_all add: lborel_eq_box sets_eq)

  let ?A =  $\lambda n :: \text{nat}. \text{box } (- (\text{real } n *_{\mathbb{R}} \text{One})) (\text{real } n *_{\mathbb{R}} \text{One}) :: 'a \text{ set}$ 
  show  $\text{range } ?A \subseteq ?E$   $(\bigcup i. ?A \ i) = \text{UNIV}$ 
    unfolding UN_box_eq_UNIV by auto

  { fix  $i$  show  $\text{emeasure lborel } (?A \ i) \neq \infty$  by auto }
  { fix  $X$  assume  $X \in ?E$  then show  $\text{emeasure lborel } X = \text{emeasure } M \ X$ 
    apply (auto simp: emeasure_eq emeasure_lborel_box_eq)
    apply (subst box_eq_empty(1)[THEN iffD2])
    apply (auto intro: less_imp_le simp: not_le)
    done }
qed

```

```

lemma lborel_affine_euclidean:
  fixes  $c :: 'a::\text{euclidean\_space} \Rightarrow \text{real}$  and  $t$ 
  defines  $T \ x \equiv t + (\sum_{j \in \text{Basis}} (c \ j * (x \cdot j)) *_{\mathbb{R}} j)$ 
  assumes  $c: \bigwedge j. j \in \text{Basis} \implies c \ j \neq 0$ 
  shows  $\text{lborel} = \text{density } (\text{distr lborel borel } T) (\lambda \_. (\prod_{j \in \text{Basis}} |c \ j|))$  (is _ = ?D)

```

```

proof (rule lborel_eqI)
  let ?B = Basis :: 'a set
  fix l u assume le:  $\bigwedge b. b \in ?B \implies l \cdot b \leq u \cdot b$ 
  have [measurable]:  $T \in \text{borel} \rightarrow_M \text{borel}$ 
    by (simp add: T_def[abs_def])
  have eq:  $T - ' \text{box } l \ u = \text{box}$ 
    ( $\sum j \in \text{Basis}. (((\text{if } 0 < c \ j \text{ then } l - t \text{ else } u - t) \cdot j) / c \ j) *_{\mathbb{R}} j$ )
    ( $\sum j \in \text{Basis}. (((\text{if } 0 < c \ j \text{ then } u - t \text{ else } l - t) \cdot j) / c \ j) *_{\mathbb{R}} j$ )
  using c by (auto simp: box_def T_def field_simps inner_simps divide_less_eq)
  with le c show emeasure ?D (box l u) = ( $\prod b \in ?B. (u - l) \cdot b$ )
    by (auto simp: emeasure_density emeasure_distr nn_integral_multc emeasure_lborel_box_eq inner_simps
      field_split_simps ennreal_mult'[symmetric] prod_nonneg
      prod.distrib[symmetric]
      intro!: prod.cong)
qed simp

lemma lborel_affine:
  fixes t :: 'a::euclidean_space
  shows  $c \neq 0 \implies \text{lborel} = \text{density} (\text{distr lborel borel } (\lambda x. t + c *_{\mathbb{R}} x)) (\lambda \_.$ 
 $|c|^{\text{DIM('a)}})$ 
  using lborel_affine_euclidean[where  $c = \lambda \_::'a. c$  and  $t = t$ ]
  unfolding scaleR_scaleR[symmetric] scaleR_sum_right[symmetric] euclidean_representation
  prod_constant by simp

lemma lborel_real_affine:
   $c \neq 0 \implies \text{lborel} = \text{density} (\text{distr lborel borel } (\lambda x. t + c * x)) (\lambda \_. \text{ennreal } (\text{abs } c))$ 
  using lborel_affine[of c t] by simp

lemma AE_lborel_affine:
  fixes P :: real  $\Rightarrow$  bool
  shows  $c \neq 0 \implies \text{Measurable.pred borel } P \implies \text{AE } x \text{ in lborel}. P \ x \implies \text{AE } x \text{ in lborel}. P \ (t + c * x)$ 
  by (subst lborel_real_affine[where  $t = - t / c$  and  $c = 1 / c$ ])
    (simp_all add: AE_density AE_distr_iff field_simps)

lemma nn_integral_real_affine:
  fixes c :: real assumes [measurable]:  $f \in \text{borel\_measurable borel}$  and  $c: c \neq 0$ 
  shows  $(\int^+ x. f \ x \ \partial \text{lborel}) = |c| * (\int^+ x. f \ (t + c * x) \ \partial \text{lborel})$ 
  by (subst lborel_real_affine[OF c, of t])
    (simp add: nn_integral_density nn_integral_distr nn_integral_cmult)

lemma lborel_integrable_real_affine:
  fixes f :: real  $\Rightarrow$  'a :: {banach, second_countable_topology}
  assumes f: integrable lborel f
  shows  $c \neq 0 \implies \text{integrable lborel } (\lambda x. f \ (t + c * x))$ 
  using f [THEN borel_measurable_integrable] unfolding integrable_iff_bounded
  by (subst (asm) nn_integral_real_affine[where  $c = c$  and  $t = t$ ]) (auto simp: en-

```

nreal_mult_less_top)

lemma *lborel_integrable_real_affine_iff*:

fixes $f :: \text{real} \Rightarrow 'a :: \{\text{banach}, \text{second\_countable\_topology}\}$

shows $c \neq 0 \implies \text{integrable lborel } (\lambda x. f (t + c * x)) \longleftrightarrow \text{integrable lborel } f$

using

lborel_integrable_real_affine[of f c t]

lborel_integrable_real_affine[of $\lambda x. f (t + c * x) \ 1/c - t/c$]

by (*auto simp add: field_simps*)

lemma *lborel_integral_real_affine*:

fixes $f :: \text{real} \Rightarrow 'a :: \{\text{banach}, \text{second\_countable\_topology}\}$ **and** $c :: \text{real}$

assumes $c: c \neq 0$ **shows** $(\int x. f x \ \partial \text{lborel}) = |c| *_R (\int x. f (t + c * x) \ \partial \text{lborel})$

proof *cases*

assume $f[\text{measurable}]: \text{integrable lborel } f$ **then show** *?thesis*

using $c \ f \ f[\text{THEN borel\_measurable\_integrable}] \ f[\text{THEN lborel\_integrable\_real\_affine, of } c \ t]$

by (*subst lborel_real_affine[OF c, of t]*)

(*simp add: integral_density integral_distr*)

next

assume $\neg \text{integrable lborel } f$ **with** c **show** *?thesis*

by (*simp add: lborel_integrable_real_affine_iff not_integrable_integral_eq*)

qed

lemma

fixes $c :: 'a :: \text{euclidean\_space} \Rightarrow \text{real}$ **and** t

assumes $c: \bigwedge j. j \in \text{Basis} \implies c \ j \neq 0$

defines $T == (\lambda x. t + (\sum j \in \text{Basis}. (c \ j * (x \cdot j)) *_R j))$

shows *lebesgue_affine_euclidean*: $\text{lebesgue} = \text{density } (\text{distr lebesgue lebesgue } T)$

($\lambda \_. (\prod j \in \text{Basis}. |c \ j|)$) (**is** $\_ = ?D$)

and *lebesgue_affine_measurable*: $T \in \text{lebesgue} \rightarrow_M \text{lebesgue}$

proof –

have $T\_borel[\text{measurable}]: T \in \text{borel} \rightarrow_M \text{borel}$

by (*auto simp: T_def[abs_def]*)

{ fix $A :: 'a \text{ set}$ **assume** $A: A \in \text{sets borel}$

then have $\text{emeasure lborel } A = 0 \longleftrightarrow \text{emeasure } (\text{density } (\text{distr lborel borel } T)$

($\lambda \_. (\prod j \in \text{Basis}. |c \ j|)$) $A = 0$

unfolding T_def **using** c **by** (*subst lborel_affine_euclidean[symmetric]*) *auto*

also have $\dots \longleftrightarrow \text{emeasure } (\text{distr lebesgue lborel } T) \ A = 0$

using $A \ c$ **by** (*simp add: distr_completion emeasure_density nn_integral_cmult*

prod_nonneg cong: distr_cong)

finally have $\text{emeasure lborel } A = 0 \longleftrightarrow \text{emeasure } (\text{distr lebesgue lborel } T) \ A$

$= 0 \ . \}$

then have $\text{eq: null\_sets lborel} = \text{null\_sets } (\text{distr lebesgue lborel } T)$

by (*auto simp: null_sets_def*)

show $T \in \text{lebesgue} \rightarrow_M \text{lebesgue}$

by (*rule completion.measurable_completion2*) (*auto simp: eq measurable_completion*)

```

have lebesgue = completion (density (distr lborel borel T) (λ_. (∏ j ∈ Basis. |c
j|)))
using c by (subst lborel_affine_euclidean[of c t]) (simp_all add: T_def[abs_def])
also have ... = density (completion (distr lebesgue lborel T)) (λ_. (∏ j ∈ Basis.
|c j|))
using c by (auto intro!: always_eventually_prod_pos completion_density_eq
simp: distr_completion cong: distr_cong)
also have ... = density (distr lebesgue lebesgue T) (λ_. (∏ j ∈ Basis. |c j|))
by (subst completion.completion_distr_eq) (auto simp: eq measurable_completion)
finally show lebesgue = density (distr lebesgue lebesgue T) (λ_. (∏ j ∈ Basis. |c
j|)) .
qed

```

corollary lebesgue_real_affine:

```

c ≠ 0 ⇒ lebesgue = density (distr lebesgue lebesgue (λx. t + c * x)) (λ_.
ennreal (abs c))
using lebesgue_real_affine_euclidean [where c = λx::real. c] by simp

```

lemma nn_integral_real_affine_lebesgue:

```

fixes c :: real assumes f[measurable]: f ∈ borel_measurable lebesgue and c: c ≠
0
shows (∫+x. f x ∂lebesgue) = ennreal |c| * (∫+x. f(t + c * x) ∂lebesgue)
proof -
have (∫+x. f x ∂lebesgue) = (∫+x. f x ∂density (distr lebesgue lebesgue (λx. t
+ c * x)) (λx. ennreal |c|))
using lebesgue_real_affine c by auto
also have ... = ∫+x. ennreal |c| * f x ∂distr lebesgue lebesgue (λx. t + c * x)
by (subst nn_integral_density) auto
also have ... = ennreal |c| * integralN (distr lebesgue lebesgue (λx. t + c * x))
f
using f measurable_distr_eq1 nn_integral_cmult by blast
also have ... = |c| * (∫+x. f(t + c * x) ∂lebesgue)
using lebesgue_real_affine_measurable[where c = λx::real. c]
by (subst nn_integral_distr) (force+)
finally show ?thesis .
qed

```

lemma lebesgue_measurable_scaling[measurable]: (*_R) x ∈ lebesgue →_M lebesgue

proof cases

```

assume x = 0
then have (*R) x = (λx. 0::'a)
by (auto simp: fun_eq_iff)
then show ?thesis by auto

```

next

```

assume x ≠ 0 then show ?thesis

```

```

using lebesgue_real_affine_measurable[of λ_. x 0]

```

```

unfolding scaleR_scaleR[symmetric] scaleR_sum_right[symmetric] euclidean_representation
by (auto simp add: ac_simps)

```

qed


```

lemma
  fixes  $m :: \text{real}$  and  $\delta :: 'a::\text{euclidean\_space}$ 
  defines  $T\ r\ d\ x \equiv r *_{\mathbb{R}} x + d$ 
  shows  $\text{emeasure\_lebesgue\_affine}: \text{emeasure lebesgue } (T\ m\ \delta\ 'S) = |m| \wedge \text{DIM}('a)$ 
  *  $\text{emeasure lebesgue } S$  (is ?e)
  and  $\text{measure\_lebesgue\_affine}: \text{measure lebesgue } (T\ m\ \delta\ 'S) = |m| \wedge \text{DIM}('a)$ 
  *  $\text{measure lebesgue } S$  (is ?m)
proof -
  show ?e
  proof cases
    assume  $m = 0$  then show ?thesis
      by (simp add: image_constant_conv T_def[abs_def])
    next
      let  $?T = T\ m\ \delta$  and  $?T' = T\ (1 / m)\ (-((1/m) *_{\mathbb{R}} \delta))$ 
      assume  $m \neq 0$ 
      then have  $s\_comp\_s: ?T' \circ ?T = id\ ?T \circ ?T' = id$ 
      by (auto simp: T_def[abs_def] fun_eq_iff scaleR_add_right scaleR_diff_right)
      then have  $inv\ ?T' = ?T\ bij\ ?T'$ 
      by (auto intro: inv_unique_comp o_bij)
      then have  $eq: T\ m\ \delta\ 'S = T\ (1 / m)\ ((-1/m) *_{\mathbb{R}} \delta)\ -' S \cap \text{space lebesgue}$ 
      using  $bij\_vimage\_eq\_inv\_image[OF \langle bij\ ?T' \rangle, of S]$  by auto

      have  $trans\_eq\ T: (\lambda x. \delta + (\sum_{j \in \text{Basis}. (m * (x \cdot j)) *_{\mathbb{R}} j)) = T\ m\ \delta$  for  $m\ \delta$ 
      unfolding T_def[abs_def] scaleR_scaleR[symmetric] scaleR_sum_right[symmetric]
      by (auto simp add: euclidean_representation ac_simps)

      have  $T[\text{measurable}]: T\ r\ d \in \text{lebesgue} \rightarrow_M \text{lebesgue}$  for  $r\ d$ 
      using  $\text{lebesgue\_affine\_measurable}[of \lambda_. r\ d]$ 
      by (cases  $r = 0$ ) (auto simp: trans_eq T_def[abs_def])

      show ?thesis
    proof cases
      assume  $S \in \text{sets lebesgue}$  with  $\langle m \neq 0 \rangle$  show ?thesis
        unfolding eq
        apply (subst lebesgue_affine_euclidean[ $of\ \lambda_. m\ \delta$ ])
        apply (simp_all add: emeasure_density trans_eq T nn_integral_cmult
emeasure_distr
          del: space_completion emeasure_completion)
        apply (simp add: vimage_comp s_comp_s)
        done
      next
        assume  $S \notin \text{sets lebesgue}$ 
        moreover have  $?T\ 'S \notin \text{sets lebesgue}$ 
        proof
          assume  $?T\ 'S \in \text{sets lebesgue}$ 
          then have  $?T\ -' (?T\ 'S) \cap \text{space lebesgue} \in \text{sets lebesgue}$ 
          by (rule measurable_sets[OF T])
          also have  $?T\ -' (?T\ 'S) \cap \text{space lebesgue} = S$ 

```

1710

```

      by (simp add: vimage_comp s_comp_s eq)
    finally show False using ⟨S ∉ sets lebesgue⟩ by auto
  qed
  ultimately show ?thesis
    by (simp add: emeasure_notin_sets)
  qed
  qed
  show ?m
    unfolding measure_def ⟨?e⟩ by (simp add: enn2real_mult prod_nonneg)
  qed

```

```

lemma lebesgue_real_scale:
  assumes c ≠ 0
  shows lebesgue = density (distr lebesgue lebesgue (λx. c * x)) (λx. ennreal |c|)
  using assms by (subst lebesgue_affine_euclidean[of λ_. c 0]) simp_all

```

```

lemma divideR_right:
  fixes x y :: 'a::real_normed_vector
  shows r ≠ 0 ⟹ y = x /R r ⟷ r *R y = x
  using scaleR_cancel_left[of r y x /R r] by simp

```

```

lemma lborel_has_bochner_integral_real_affine_iff:
  fixes x :: 'a :: {banach, second_countable_topology}
  shows c ≠ 0 ⟹
    has_bochner_integral lborel f x ⟷
    has_bochner_integral lborel (λx. f (t + c * x)) (x /R |c|)
  unfolding has_bochner_integral_iff lborel_integrable_real_affine_iff
  by (simp_all add: lborel_integral_real_affine[symmetric] divideR_right cong:
  conj_cong)

```

```

lemma lborel_distr_uminus: distr lborel borel uminus = (lborel :: real measure)
  by (subst lborel_real_affine[of -1 0])
  (auto simp: density_1 one_ennreal_def[symmetric])

```

```

lemma lborel_distr_mult:
  assumes (c::real) ≠ 0
  shows distr lborel borel ((* c) = density lborel (λ_. inverse |c|)
proof-
  have distr lborel borel ((* c) = distr lborel lborel ((* c) by (simp cong: distr_cong)
  also from assms have ... = density lborel (λ_. inverse |c|)
    by (subst lborel_real_affine[of inverse c 0]) (auto simp: o_def distr_density_distr)
  finally show ?thesis .
  qed

```

```

lemma lborel_distr_mult':
  assumes (c::real) ≠ 0
  shows lborel = density (distr lborel borel ((* c)) (λ_. |c|)
proof-
  have lborel = density lborel (λ_. 1) by (rule density_1[symmetric])

```

also from *assms* have $(\lambda_. 1 :: \text{ennreal}) = (\lambda_. \text{inverse } |c| * |c|)$ by (intro ext) simp
 also have $\text{density lborel } \dots = \text{density } (\text{density lborel } (\lambda_. \text{inverse } |c|)) (\lambda_. |c|)$
 by (subst density_density_eq) (auto simp: ennreal_mult)
 also from *assms* have $\text{density lborel } (\lambda_. \text{inverse } |c|) = \text{distr lborel borel } ((*) c)$
 by (rule lborel_distr_mult[symmetric])
 finally show ?thesis .
 qed

lemma lborel_distr_plus:
 fixes $c :: 'a::\text{euclidean\_space}$
 shows $\text{distr lborel borel } ((+) c) = \text{lborel}$
 by (subst lborel_affine[of 1 c], auto simp: density_1)

interpretation lborel: sigma_finite_measure lborel
 by (rule sigma_finite_lborel)

interpretation lborel_pair: pair_sigma_finite lborel lborel ..

lemma lborel_prod:
 $\text{lborel } \otimes_M \text{ lborel} = (\text{lborel} :: ('a::\text{euclidean\_space} \times 'b::\text{euclidean\_space}) \text{ measure})$
 proof (rule lborel_eqI[symmetric], clarify)
 fix $la\ ua :: 'a$ and $lb\ ub :: 'b$
 assume $lu: \bigwedge a\ b. (a, b) \in \text{Basis} \implies (la, lb) \cdot (a, b) \leq (ua, ub) \cdot (a, b)$
 have [simp]:
 $\bigwedge b. b \in \text{Basis} \implies la \cdot b \leq ua \cdot b$
 $\bigwedge b. b \in \text{Basis} \implies lb \cdot b \leq ub \cdot b$
 $\text{inj\_on } (\lambda u. (u, 0)) \text{ Basis inj\_on } (\lambda u. (0, u)) \text{ Basis}$
 $(\lambda u. (u, 0)) ' \text{Basis} \cap (\lambda u. (0, u)) ' \text{Basis} = \{\}$
 $\text{box } (la, lb) (ua, ub) = \text{box } la\ ua \times \text{box } lb\ ub$
 using $lu[\text{of } \_ 0]$ $lu[\text{of } 0]$ by (auto intro!: inj_onI simp add: Basis_prod_def ball_Un box_def)
 show $\text{emeasure } (\text{lborel } \otimes_M \text{ lborel}) (\text{box } (la, lb) (ua, ub)) =$
 $\text{ennreal } (\text{prod } ((\cdot) ((ua, ub) - (la, lb))) \text{ Basis})$
 by (simp add: lborel_emeasure_pair_measure_Times Basis_prod_def prod.union_disjoint prod.reindex ennreal_mult inner_diff_left prod_nonneg)
 qed (simp add: borel_prod[symmetric])

lemma lborelD_Collect[measurable (raw)]: $\{x \in \text{space lborel}. P\ x\} \in \text{sets lborel} \implies$
 $\{x \in \text{space lborel}. P\ x\} \in \text{sets lborel}$
 by simp

lemma lborelD[measurable (raw)]: $A \in \text{sets lborel} \implies A \in \text{sets lborel}$
 by simp

lemma emeasure_bounded_finite:
 assumes *bounded A* shows $\text{emeasure lborel } A < \infty$

```

proof –
  obtain  $a\ b$  where  $A \subseteq \text{cbox } a\ b$ 
    by (meson bounded_subset_cbox_symmetric  $\langle \text{bounded } A \rangle$ )
  then have  $\text{emeasure lborel } A \leq \text{emeasure lborel } (\text{cbox } a\ b)$ 
    by (intro emeasure_mono) auto
  then show ?thesis
    by (auto simp: emeasure_lborel_cbox_eq prod_nonneg less_top[symmetric]
top_unique split: if_split_asm)
qed

```

```

lemma emeasure_compact_finite:  $\text{compact } A \implies \text{emeasure lborel } A < \infty$ 
  using emeasure_bounded_finite[of  $A$ ] by (auto intro: compact_imp_bounded)

```

```

lemma borel_integrable_compact:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\{\text{banach, second\_countable\_topology}\}$ 
  assumes compact S continuous_on S f
  shows  $\text{integrable lborel } (\lambda x. \text{indicator } S\ x *_{\mathbb{R}} f\ x)$ 
proof cases
  assume  $S \neq \{\}$ 
  have continuous_on S  $(\lambda x. \text{norm } (f\ x))$ 
    using assms by (intro continuous_intros)
  from continuous_attains_sup[OF  $\langle \text{compact } S \rangle \langle S \neq \{\} \rangle$  this]
  obtain  $M$  where  $M: \bigwedge x. x \in S \implies \text{norm } (f\ x) \leq M$ 
    by auto
  show ?thesis
  proof (rule integrable_bound)
    show  $\text{integrable lborel } (\lambda x. \text{indicator } S\ x * M)$ 
      using assms by (auto intro!: emeasure_compact_finite borel_compact integrable_mult_left)
    show  $(\lambda x. \text{indicator } S\ x *_{\mathbb{R}} f\ x) \in \text{borel\_measurable lborel}$ 
      using assms by (auto intro!: borel_measurable_continuous_on_indicator borel_compact)
    show  $\text{AE } x \text{ in lborel. } \text{norm } (\text{indicator } S\ x *_{\mathbb{R}} f\ x) \leq \text{norm } (\text{indicator } S\ x * M)$ 
      by (auto split: split_indicator simp: abs_real_def dest!: M)
  qed
qed simp

```

```

lemma borel_integrable_atLeastAtMost:
  fixes  $f :: \text{real} \Rightarrow \text{real}$ 
  assumes  $f: \bigwedge x. a \leq x \implies x \leq b \implies \text{isCont } f\ x$ 
  shows  $\text{integrable lborel } (\lambda x. f\ x * \text{indicator } \{a .. b\}\ x)$  (is integrable _ ?f)
proof –
  have  $\text{integrable lborel } (\lambda x. \text{indicator } \{a .. b\}\ x *_{\mathbb{R}} f\ x)$ 
  proof (rule borel_integrable_compact)
    from  $f$  show continuous_on {a..b} f
      by (auto intro: continuous_at_imp_continuous_on)
  qed simp
  then show ?thesis
    by (auto simp: mult.commute)

```

qed

6.13.6 Lebesgue measurable sets

abbreviation *lmeasurable* :: 'a::euclidean_space set set

where

lmeasurable $\equiv$ *fmeasurable lebesgue*

lemma *not_measurable_UNIV* [simp]: *UNIV* $\notin$ *lmeasurable*
by (*simp add: fmeasurable_def*)

lemma *lmeasurable_iff_integrable*:

S $\in$ *lmeasurable* $\longleftrightarrow$ *integrable lebesgue* (*indicator S* :: 'a::euclidean_space $\Rightarrow$ real)

by (*auto simp: fmeasurable_def integrable_iff_bounded borel_measurable_indicator_iff ennreal_indicator*)

lemma *lmeasurable_cbox* [iff]: *cbox a b* $\in$ *lmeasurable*

and *lmeasurable_box* [iff]: *box a b* $\in$ *lmeasurable*

by (*auto simp: fmeasurable_def emeasure_lborel_box_eq emeasure_lborel_cbox_eq*)

lemma

fixes *a*::real

shows *lmeasurable_interval* [iff]: $\{a..b\} \in$ *lmeasurable* $\{a <..<b\} \in$ *lmeasurable*

apply (*metis box_real(2) lmeasurable_cbox*)

by (*metis box_real(1) lmeasurable_box*)

lemma *fmeasurable_compact*: *compact S* $\implies S \in$ *fmeasurable lborel*

using *emeasure_compact_finite*[of *S*] **by** (*intro fmeasurableI*) (*auto simp: borel_compact*)

lemma *lmeasurable_compact*: *compact S* $\implies S \in$ *lmeasurable*

using *fmeasurable_compact* **by** (*force simp: fmeasurable_def*)

lemma *measure_frontier*:

bounded S $\implies$ *measure lebesgue* (*frontier S*) = *measure lebesgue* (*closure S*) – *measure lebesgue* (*interior S*)

using *closure_subset interior_subset*

by (*auto simp: frontier_def fmeasurable_compact intro!: measurable_measure_Diff*)

lemma *lmeasurable_closure*:

bounded S $\implies$ *closure S* $\in$ *lmeasurable*

by (*simp add: lmeasurable_compact*)

lemma *lmeasurable_frontier*:

bounded S $\implies$ *frontier S* $\in$ *lmeasurable*

by (*simp add: compact_frontier_bounded lmeasurable_compact*)

lemma *lmeasurable_open*: *bounded S* $\implies$ *open S* $\implies S \in$ *lmeasurable*

using *emeasure_bounded_finite*[of *S*] **by** (*intro fmeasurableI*) (*auto simp: borel_open*)

lemma *lmeasurable_ball* [*simp*]: *ball a r ∈ lmeasurable*
by (*simp add: lmeasurable_open*)

lemma *lmeasurable_cball* [*simp*]: *cball a r ∈ lmeasurable*
by (*simp add: lmeasurable_compact*)

lemma *lmeasurable_interior*: *bounded S ⇒ interior S ∈ lmeasurable*
by (*simp add: bounded_interior lmeasurable_open*)

lemma *null_sets_cbox_Diff_box*: *cbox a b - box a b ∈ null_sets lborel*
proof -
have *emeasure lborel (cbox a b - box a b) = 0*
by (*subst emeasure_Diff*) (*auto simp: emeasure_lborel_cbox_eq emeasure_lborel_box_eq box_subset_cbox*)
then have *cbox a b - box a b ∈ null_sets lborel*
by (*auto simp: null_sets_def*)
then show *?thesis*
by (*auto dest!: AE_not_in*)
qed

lemma *bounded_set_imp_lmeasurable*:
assumes *bounded S S ∈ sets lebesgue* **shows** *S ∈ lmeasurable*
by (*metis assms bounded_Un emeasure_bounded_finite emeasure_completion fmeasurableI main_part_null_part_Un*)

lemma *finite_measure_lebesgue_on*: *S ∈ lmeasurable ⇒ finite_measure (lebesgue_on S)*
by (*auto simp: finite_measureI fmeasurable_def emeasure_restrict_space*)

lemma *integrable_const_ivl* [*iff*]:
fixes *a::'a::ordered_euclidean_space*
shows *integrable (lebesgue_on {a..b}) (λx. c)*
by (*metis cbox_interval finite_measure.integrable_const finite_measure_lebesgue_on lmeasurable_cbox*)

6.13.7 Translation preserves Lebesgue measure

lemma *sigma_sets_image*:
assumes *S: S ∈ sigma_sets Ω M* **and** *M ⊆ Pow Ω f ' Ω = Ω inj_on f Ω*
and *M: ∧y. y ∈ M ⇒ f ' y ∈ M*
shows *(f ' S) ∈ sigma_sets Ω M*
using *S*
proof (*induct S rule: sigma_sets.induct*)
case (*Basic a*) **then show** *?case*
by (*simp add: M*)
next
case *Empty* **then show** *?case*
by (*simp add: sigma_sets.Empty*)

```

next
  case (Compl a)
  then have  $\Omega - a \subseteq \Omega$   $a \subseteq \Omega$ 
    by (auto simp: sigma_sets_into_sp [OF ‹ $M \subseteq \text{Pow } \Omega$ ›])
  then show ?case
    by (auto simp: inj_on_image_set_diff [OF ‹inj_on f  $\Omega$ ›] assms intro: Compl
sigma_sets.Compl)
next
  case (Union a) then show ?case
    by (metis image_UN sigma_sets.simps)
qed

```

lemma *null_sets_translation:*

```

  assumes  $N \in \text{null\_sets } \text{lborel}$  shows  $\{x. x - a \in N\} \in \text{null\_sets } \text{lborel}$ 
proof -
  have [simp]:  $(\lambda x. x + a) \text{ ` } N = \{x. x - a \in N\}$ 
    by force
  show ?thesis
    using assms emeasure_lebesgue_affine [of 1 a N] by (auto simp: null_sets_def)
qed

```

lemma *lebesgue_sets_translation:*

```

  fixes  $f :: 'a \Rightarrow 'a::\text{euclidean\_space}$ 
  assumes  $S: S \in \text{sets } \text{lebesgue}$ 
  shows  $((\lambda x. a + x) \text{ ` } S) \in \text{sets } \text{lebesgue}$ 
proof -
  have im_eq:  $(+) a \text{ ` } A = \{x. x - a \in A\}$  for  $A$ 
    by force
  have  $((\lambda x. a + x) \text{ ` } S) = ((\lambda x. -a + x) \text{ ` } S) \cap (\text{space } \text{lebesgue})$ 
    using image_iff by fastforce
  also have  $\dots \in \text{sets } \text{lebesgue}$ 
proof (rule measurable_sets [OF measurableI assms])
  fix  $A :: 'b \text{ set}$ 
  assume  $A: A \in \text{sets } \text{lebesgue}$ 
  have vim_eq:  $(\lambda x. x - a) \text{ ` } A = (+) a \text{ ` } A$  for  $A$ 
    by force
  have  $\exists s \ n \ N'. (+) a \text{ ` } (S \cup N) = s \cup n \wedge s \in \text{sets } \text{borel} \wedge N' \in \text{null\_sets}$ 
 $\text{lborel} \wedge n \subseteq N'$ 
    if  $S \in \text{sets } \text{borel}$  and  $N' \in \text{null\_sets } \text{lborel}$  and  $N \subseteq N'$  for  $S \ N \ N'$ 
proof (intro exI conjI)
  show  $(+) a \text{ ` } (S \cup N) = (\lambda x. a + x) \text{ ` } S \cup (\lambda x. a + x) \text{ ` } N$ 
    by auto
  show  $(\lambda x. a + x) \text{ ` } N' \in \text{null\_sets } \text{lborel}$ 
    using that by (auto simp: null_sets_translation im_eq)
qed (use that im_eq in auto)
with  $A$  have  $(\lambda x. x - a) \text{ ` } A \in \text{sets } \text{lebesgue}$ 
  by (force simp: vim_eq completion_def intro!: sigma_sets_image)
then show  $(+) (- a) \text{ ` } A \cap \text{space } \text{lebesgue} \in \text{sets } \text{lebesgue}$ 
  by (auto simp: vimage_def im_eq)

```

1716

```
qed auto
finally show ?thesis .
qed
```

```
lemma measurable_translation:
   $S \in \text{lmeasurable} \implies ((+) a \, ' S) \in \text{lmeasurable}$ 
  using emeasure_lebesgue_affine [of 1 a S]
  apply (auto intro: lebesgue_sets_translation simp add: fmeasurable_def cong:
image_cong_simp)
  apply (simp add: ac_simps)
done
```

```
lemma measurable_translation_subtract:
   $S \in \text{lmeasurable} \implies ((\lambda x. x - a) \, ' S) \in \text{lmeasurable}$ 
  using measurable_translation [of S - a] by (simp cong: image_cong_simp)
```

```
lemma measure_translation:
   $\text{measure lebesgue } ((+) a \, ' S) = \text{measure lebesgue } S$ 
  using measure_lebesgue_affine [of 1 a S] by (simp add: ac_simps cong: im-
age_cong_simp)
```

```
lemma measure_translation_subtract:
   $\text{measure lebesgue } ((\lambda x. x - a) \, ' S) = \text{measure lebesgue } S$ 
  using measure_translation [of - a] by (simp cong: image_cong_simp)
```

6.13.8 A nice lemma for negligibility proofs

```
lemma summable_iff_suminf_neq_top:  $(\bigwedge n. f \, n \geq 0) \implies \neg \text{summable } f \implies$ 
 $(\sum i. \text{ennreal } (f \, i)) = \text{top}$ 
  by (metis summable_suminf_not_top)
```

```
proposition starlike_negligible_bounded_gmeasurable:
  fixes S :: 'a :: euclidean_space set
  assumes S:  $S \in \text{sets lebesgue}$  and bounded S
  and eq1:  $\bigwedge c \, x. \llbracket (c *_{\mathbb{R}} x) \in S; 0 \leq c; x \in S \rrbracket \implies c = 1$ 
  shows  $S \in \text{null\_sets lebesgue}$ 
```

```
proof -
  obtain M where  $0 < M$   $S \subseteq \text{ball } 0 \, M$ 
  using ⟨bounded S⟩ by (auto dest: bounded_subset_ballD)
```

```
let ?f =  $\lambda n. \text{root } \text{DIM}('a) \, (\text{Suc } n)$ 
```

```
have vimage_eq_image:  $(*_R) \, (?f \, n) - ' S = (*_R) \, (1 / ?f \, n) \, ' S$  for n
  apply safe
  subgoal for x by (rule image_eqI [of _ _ ?f n *_R x]) auto
  subgoal by auto
done
```

```
have eq:  $(1 / ?f \, n) \wedge \text{DIM}('a) = 1 / \text{Suc } n$  for n
```



```

by (simp add: field_simps)

{ fix n x assume x: root DIM('a) (1 + real n) *R x ∈ S
  have 1 * norm x ≤ root DIM('a) (1 + real n) * norm x
    by (rule mult_mono) auto
  also have ... < M
    using x ⟨S ⊆ ball 0 M⟩ by auto
  finally have norm x < M by simp }
note less_M = this

have (∑ n. ennreal (1 / Suc n)) = top
  using not_summable_harmonic[where 'a=real] summable_Suc_iff[where
f=λn. 1 / (real n)]
  by (intro summable_iff_suminf_neq_top) (auto simp add: inverse_eq_divide)
  then have top * emeasure lebesgue S = (∑ n. (1 / ?f n)DIM('a) * emeasure
lebesgue S)
  unfolding ennreal_suminf_multc_eq by simp
  also have ... = (∑ n. emeasure lebesgue ((*_R) (?f n) - ' S))
  unfolding vimage_eq_image using emeasure_lebesgue_affine[of 1 / ?f n 0 S
for n] by simp
  also have ... = emeasure lebesgue (⋃ n. (*_R) (?f n) - ' S)
  proof (intro suminf_emeasure)
    show disjoint_family (λn. (*_R) (?f n) - ' S)
      unfolding disjoint_family_on_def
    proof safe
      fix m n :: nat and x assume m ≠ n ?f m *R x ∈ S ?f n *R x ∈ S
      with eq1[of ?f m / ?f n ?f n *R x] show x ∈ {}
      by auto
    qed
    have (*_R) (?f i) - ' S ∈ sets lebesgue for i
      using measurable_sets[OF lebesgue_measurable_scaling[of ?f i] S] by auto
    then show range (λi. (*_R) (?f i) - ' S) ⊆ sets lebesgue
      by auto
    qed
  also have ... ≤ emeasure lebesgue (ball 0 M :: 'a set)
    using less_M by (intro emeasure_mono) auto
  also have ... < top
    using lmeasurable_ball by (auto simp: fmeasurable_def)
  finally have emeasure lebesgue S = 0
    by (simp add: ennreal_top_mult_split: if_split_asm)
  then show S ∈ null_sets lebesgue
    unfolding null_sets_def using ⟨S ∈ sets lebesgue⟩ by auto
qed

corollary starlike_negligible_compact:
  compact S ⟹ (⋀ c x. ⟦(c *R x) ∈ S; 0 ≤ c; x ∈ S⟧ ⟹ c = 1) ⟹ S ∈ null_sets
lebesgue
  using starlike_negligible_bounded_gmeasurable[of S] by (auto simp: compact_eq_bounded_closed)

```

proposition *outer_regular_lborel_le*:

assumes $B[\text{measurable}]$: $B \in \text{sets borel}$ **and** $0 < (e::\text{real})$

obtains U **where** $\text{open } U \ B \subseteq U$ **and** $\text{emeasure lborel } (U - B) \leq e$

proof –

let $? \mu = \text{emeasure lborel}$

let $?B = \lambda n::\text{nat. ball } 0 \ n :: 'a \text{ set}$

let $?e = \lambda n. e * ((1/2)^\wedge \text{Suc } n)$

have $\forall n. \exists U. \text{open } U \wedge ?B \ n \cap B \subseteq U \wedge ? \mu (U - B) < ?e \ n$

proof

fix $n :: \text{nat}$

let $?A = \text{density lborel } (\text{indicator } (?B \ n))$

have $\text{emeasure\_A}: X \in \text{sets borel} \implies \text{emeasure } ?A \ X = ? \mu (?B \ n \cap X)$ **for** X

by (*auto simp: emeasure_density borel_measurable_indicator indicator_inter_arith[symmetric]*)

have $\text{finite\_A}: \text{emeasure } ?A \ (\text{space } ?A) \neq \infty$

using $\text{emeasure\_bounded\_finite[of } ?B \ n]$ **by** (*auto simp: emeasure_A*)

interpret $A: \text{finite\_measure } ?A$

by *rule fact*

have $\text{emeasure } ?A \ B + ?e \ n > (\text{INF } U \in \{U. B \subseteq U \wedge \text{open } U\}. \text{emeasure } ?A \ U)$

using $\langle 0 < e \rangle$ **by** (*auto simp: outer_regular[OF finite_A B, symmetric]*)

then obtain U **where** $U: B \subseteq U \text{ open } U$ **and** $\mu U: ? \mu (?B \ n \cap B) + ?e \ n > ? \mu (?B \ n \cap U)$

unfolding INF_less_iff **by** (*auto simp: emeasure_A*)

moreover

{ have $? \mu ((?B \ n \cap U) - B) = ? \mu ((?B \ n \cap U) - (?B \ n \cap B))$

using U **by** (*intro arg_cong[where f=?μ] auto*)

also have $\dots = ? \mu (?B \ n \cap U) - ? \mu (?B \ n \cap B)$

using $U.A.\text{emeasure\_finite[of } B]$

by (*intro emeasure_Diff*) (*auto simp del: A.emeasure_finite simp: emeasure_A*)

also have $\dots < ?e \ n$

using $U \ \mu U \ A.\text{emeasure\_finite[of } B]$

by (*subst minus_less_iff ennreal*)

(*auto simp del: A.emeasure_finite simp: emeasure_A less_top ac_simps intro!: emeasure_mono*)

finally have $? \mu ((?B \ n \cap U) - B) < ?e \ n . \}$

ultimately show $\exists U. \text{open } U \wedge ?B \ n \cap B \subseteq U \wedge ? \mu (U - B) < ?e \ n$

by (*intro exI[of _ ?B \ n \cap U] auto*)

qed

then obtain U

where $U: \bigwedge n. \text{open } (U \ n) \ \bigwedge n. ?B \ n \cap B \subseteq U \ n \ \bigwedge n. ? \mu (U \ n - B) < ?e \ n$

by *metis*

show *?thesis*

proof

{ fix x **assume** $x \in B$

moreover

obtain n **where** $\text{norm } x < \text{real } n$

using $\text{reals\_Archimedean2}$ **by** *blast*

```

      ultimately have  $x \in (\bigcup n. U\ n)$ 
      using  $U(2)[of\ n]$  by auto }
    note * = this
    then show  $open\ (\bigcup n. U\ n)\ B \subseteq (\bigcup n. U\ n)$ 
      using  $U$  by auto
    have  $?μ\ (\bigcup n. U\ n - B) \leq (\sum n. ?μ\ (U\ n - B))$ 
      using  $U(1)$  by (intro emeasure_subadditive_countably) auto
    also have  $\dots \leq (\sum n. ennreal\ (?e\ n))$ 
      using  $U(3)$  by (intro suminf_le) (auto intro: less_imp_le)
    also have  $\dots = ennreal\ (e * 1)$ 
      using  $\langle 0 < e \rangle$  by (intro suminf_ennreal_eq_sums_mult_power_half_series)
  auto
  finally show emeasure lborel  $((\bigcup n. U\ n) - B) \leq ennreal\ e$ 
    by simp
qed
qed

```

```

lemma outer_regular_lborel:
  assumes  $B: B \in sets\ lborel$  and  $0 < (e::real)$ 
  obtains  $U$  where  $open\ U\ B \subseteq U$  emeasure lborel  $(U - B) < e$ 
proof -
  obtain  $U$  where  $U: open\ U\ B \subseteq U$  and emeasure lborel  $(U - B) \leq e/2$ 
    using outer_regular_lborel_le  $[OF\ B, of\ e/2]$   $\langle e > 0 \rangle$ 
    by force
  moreover have  $ennreal\ (e/2) < ennreal\ e$ 
    using  $\langle e > 0 \rangle$  by (simp add: ennreal_lessI)
  ultimately have emeasure lborel  $(U - B) < e$ 
    by auto
  with  $U$  show ?thesis
    using that by auto
qed

```

```

lemma completion_upper:
  assumes  $A: A \in sets\ (completion\ M)$ 
  obtains  $A'$  where  $A \subseteq A'$   $A' \in sets\ M$   $A' - A \in null\_sets\ (completion\ M)$ 
    emeasure  $(completion\ M)\ A = \textit{emeasure}\ M\ A'$ 
proof -
  from AE_notin_null_part  $[OF\ A]$  obtain  $N$  where  $N: N \in null\_sets\ M$  null_part
     $M\ A \subseteq N$ 
  unfolding eventually_ae_filter using null_part_null_sets  $[OF\ A, THEN\ null\_setsD2,$ 
    THEN sets_into_space] by auto
  let  $?A' = \textit{main\_part}\ M\ A \cup N$ 
  show ?thesis
proof
  show  $A \subseteq ?A'$ 
    using  $\langle \textit{null\_part}\ M\ A \subseteq N \rangle$  by (subst main_part_null_part_Un  $[symmetric,$ 
    OF\ A]) auto
  have main_part  $M\ A \subseteq A$ 
    using assms main_part_null_part_Un by auto

```

```

    then have ?A' - A  $\subseteq$  N
    by blast
    with N show ?A' - A  $\in$  null_sets (completion M)
    by (blast intro: null_sets_completionI completion.complete_measure_axioms
complete_measure.complete2)
    show emeasure (completion M) A = emeasure M (main_part M A  $\cup$  N)
    using A  $\langle$  N  $\in$  null_sets M  $\rangle$  by (simp add: emeasure_Un_null_set)
  qed (use A N in auto)
qed

```

```

lemma sets_lebesgue_outer_open:
  fixes e::real
  assumes S: S  $\in$  sets lebesgue and e > 0
  obtains T where open T S  $\subseteq$  T (T - S)  $\in$  lmeasurable emeasure lebesgue (T
- S) < ennreal e
proof -
  obtain S' where S': S  $\subseteq$  S' S'  $\in$  sets borel
    and null: S' - S  $\in$  null_sets lebesgue
    and em: emeasure lebesgue S = emeasure lborel S'
  using completion_upper[of S lborel] S by auto
  then have f_S': S'  $\in$  sets borel
  using S by (auto simp: fmeasurable_def)
  with outer_regular_lborel[OF _  $\langle$  0 < e  $\rangle$ ]
  obtain U where U: open U S'  $\subseteq$  U emeasure lborel (U - S') < e
    by blast
  show thesis
proof
  show open U S  $\subseteq$  U
    using f_S' U S' by auto
  have (U - S) = (U - S')  $\cup$  (S' - S)
    using S' U by auto
  then have eq: emeasure lebesgue (U - S) = emeasure lborel (U - S')
    using null by (simp add: U(1) emeasure_Un_null_set f_S' sets.Diff)
  have (U - S)  $\in$  sets lebesgue
    by (simp add: S U(1) sets.Diff)
  then show (U - S)  $\in$  lmeasurable
    unfolding fmeasurable_def using U(3) eq less_le_trans by fastforce
  with eq U show emeasure lebesgue (U - S) < e
    by (simp add: eq)
  qed
qed

```

```

lemma sets_lebesgue_inner_closed:
  fixes e::real
  assumes S  $\in$  sets lebesgue e > 0
  obtains T where closed T T  $\subseteq$  S S-T  $\in$  lmeasurable emeasure lebesgue (S -
T) < ennreal e
proof -
  have -S  $\in$  sets lebesgue

```

```

    using assms by (simp add: Compl_in_sets_lebesgue)
  then obtain T where open T  $-S \subseteq T$ 
    and T:  $(T - -S) \in \text{lmeasurable } \text{emeasure } \text{lebesgue } (T - -S) < e$ 
    using sets_lebesgue_outer_open assms by blast
  show thesis
  proof
    show closed  $(-T)$ 
      using  $\langle \text{open } T \rangle$  by blast
    show  $-T \subseteq S$ 
      using  $\langle -S \subseteq T \rangle$  by auto
    show  $S - (-T) \in \text{lmeasurable } \text{emeasure } \text{lebesgue } (S - (-T)) < e$ 
      using T by (auto simp: Int_commute)
  qed
qed

```

lemma *lebesgue_openin*:

```

 $\llbracket \text{openin } (\text{top\_of\_set } S) \ T; S \in \text{sets } \text{lebesgue} \rrbracket \implies T \in \text{sets } \text{lebesgue}$ 
  by (metis borel_open openin_open sets.Int sets_completionI_sets sets_lborel)

```

lemma *lebesgue_closedin*:

```

 $\llbracket \text{closedin } (\text{top\_of\_set } S) \ T; S \in \text{sets } \text{lebesgue} \rrbracket \implies T \in \text{sets } \text{lebesgue}$ 
  by (metis borel_closed closedin_closed sets.Int sets_completionI_sets sets_lborel)

```

6.13.9 *F_sigma* and *G_delta* sets.

inductive *fsigma* :: $'a::\text{topological\_space}$ *set* $\Rightarrow$ *bool* **where**
 $(\bigwedge n::\text{nat}. \text{closed } (F \ n)) \implies \text{fsigma } (\bigcup (F \text{ ' } \text{UNIV}))$

inductive *gdelta* :: $'a::\text{topological\_space}$ *set* $\Rightarrow$ *bool* **where**
 $(\bigwedge n::\text{nat}. \text{open } (F \ n)) \implies \text{gdelta } (\bigcap (F \text{ ' } \text{UNIV}))$

lemma *fsigma_Union_compact*:

```

  fixes S ::  $'a::\{\text{real\_normed\_vector}, \text{heine\_borel}\}$  set
  shows  $\text{fsigma } S \longleftrightarrow (\exists F::\text{nat} \Rightarrow 'a \text{ set}. \text{range } F \subseteq \text{Collect compact} \wedge S = \bigcup (F \text{ ' } \text{UNIV}))$ 

```

proof *safe*

```

  assume fsigma S
  then obtain F ::  $\text{nat} \Rightarrow 'a \text{ set}$  where  $F: \text{range } F \subseteq \text{Collect closed } S = \bigcup (F \text{ ' } \text{UNIV})$ 

```

```

    by (meson fsigma.cases image_subsetI mem_Collect_eq)
    then have  $\exists D::\text{nat} \Rightarrow 'a \text{ set}. \text{range } D \subseteq \text{Collect compact} \wedge \bigcup (D \text{ ' } \text{UNIV}) = F$ 
  i for i

```

```

    using closed_Union_compact_subsets [of F i]
    by (metis image_subsetI mem_Collect_eq range_subsetD)
  then obtain D ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ set}$ 
    where  $D: \bigwedge i. \text{range } (D \ i) \subseteq \text{Collect compact} \wedge \bigcup ((D \ i) \text{ ' } \text{UNIV}) = F \ i$ 
    by metis
  let ?DD =  $\lambda n. (\lambda(i,j). D \ i \ j) \ (\text{prod\_decode } n)$ 
  show  $\exists F::\text{nat} \Rightarrow 'a \text{ set}. \text{range } F \subseteq \text{Collect compact} \wedge S = \bigcup (F \text{ ' } \text{UNIV})$ 

```

```

proof (intro exI conjI)
  show range ?DD  $\subseteq$  Collect compact
    using D by clarsimp (metis mem_Collect_eq rangeI split_conv subsetCE
surj_pair)
  show  $S = \bigcup (\text{range } ?DD)$ 
  proof
    show  $S \subseteq \bigcup (\text{range } ?DD)$ 
    using D F
    by clarsimp (metis UN_iff old.prod.case prod_decode_inverse prod_encode_eq)
    show  $\bigcup (\text{range } ?DD) \subseteq S$ 
    using D F by fastforce
  qed
qed
next
  fix F :: nat  $\Rightarrow$  'a set
  assume range F  $\subseteq$  Collect compact and  $S = \bigcup (F \text{ ' UNIV})$ 
  then show fsigma ( $\bigcup (F \text{ ' UNIV})$ )
    by (simp add: compact_imp_closed fsigma.intros image_subset_iff)
qed

lemma gdelta_imp_fsigma: gdelta S  $\implies$  fsigma ( $- S$ )
proof (induction rule: gdelta.induct)
  case (1 F)
  have  $-\bigcap (F \text{ ' UNIV}) = (\bigcup i. -(F i))$ 
    by auto
  then show ?case
    by (simp add: fsigma.intros closed_Compl 1)
qed

lemma fsigma_imp_gdelta: fsigma S  $\implies$  gdelta ( $- S$ )
proof (induction rule: fsigma.induct)
  case (1 F)
  have  $-\bigcup (F \text{ ' UNIV}) = (\bigcap i. -(F i))$ 
    by auto
  then show ?case
    by (simp add: 1 gdelta.intros open_closed)
qed

lemma gdelta_complement: gdelta( $- S$ )  $\longleftrightarrow$  fsigma S
  using fsigma_imp_gdelta gdelta_imp_fsigma by force

lemma lebesgue_set_almost_fsigma:
  assumes S  $\in$  sets lebesgue
  obtains C T where fsigma C T  $\in$  null_sets lebesgue C  $\cup$  T = S disjnt C T
proof -
  { fix n::nat
    obtain T where closed T T  $\subseteq$  S S-T  $\in$  lmeasurable emeasure lebesgue (S -
T) < ennreal (1 / Suc n)
    using sets_lebesgue_inner_closed [OF assms]

```

```

    by (metis of_nat_0_less_iff zero_less_Suc zero_less_divide_1_iff)
  then have  $\exists T. \text{closed } T \wedge T \subseteq S \wedge S - T \in \text{lmeasurable} \wedge \text{measure lebesgue } (S - T) < 1 / \text{Suc } n$ 
    by (metis emeasure_eq_measure2 ennreal_leI not_le)
}
then obtain F where  $F: \bigwedge n::\text{nat}. \text{closed } (F\ n) \wedge F\ n \subseteq S \wedge S - F\ n \in \text{lmeasurable} \wedge \text{measure lebesgue } (S - F\ n) < 1 / \text{Suc } n$ 
  by metis
let ?C =  $\bigcup (F \text{ ` } \text{UNIV})$ 
show thesis
proof
  show fsigma ?C
    using F by (simp add: fsigma.intros)
  show  $(S - ?C) \in \text{null\_sets lebesgue}$ 
  proof (clarsimp simp add: completion.null_sets_outer_le)
    fix e :: real
    assume  $0 < e$ 
    then obtain n where  $n: 1 / \text{Suc } n < e$ 
      using nat_approx_posE by metis
    show  $\exists T \in \text{lmeasurable}. S - (\bigcup x. F\ x) \subseteq T \wedge \text{measure lebesgue } T \leq e$ 
    proof (intro bexI conjI)
      show  $\text{measure lebesgue } (S - F\ n) \leq e$ 
        by (meson F n less_trans not_le order.asym)
      qed (use F in auto)
    qed
  qed
  show  $?C \cup (S - ?C) = S$ 
    using F by blast
  show  $\text{disjnt } ?C (S - ?C)$ 
    by (auto simp: disjnt_def)
  qed
qed

lemma lebesgue_set_almost_gdelta:
  assumes  $S \in \text{sets lebesgue}$ 
  obtains C T where  $\text{gdelta } C\ T \in \text{null\_sets lebesgue } S \cup T = C \text{ disjnt } S\ T$ 
proof -
  have  $-S \in \text{sets lebesgue}$ 
    using assms Compl_in_sets_lebesgue by blast
  then obtain C T where  $C: \text{fsigma } C\ T \in \text{null\_sets lebesgue } C \cup T = -S \text{ disjnt } C\ T$ 
    using lebesgue_set_almost_fsigma by metis
  show thesis
  proof
    show  $\text{gdelta } (-C)$ 
      by (simp add:  $\langle \text{fsigma } C \rangle \text{fsigma\_imp\_gdelta}$ )
    show  $S \cup T = -C \text{ disjnt } S\ T$ 
      using C by (auto simp: disjnt_def)
    qed (use C in auto)
  qed
qed

```

end

6.14 Tagged Divisions for Henstock-Kurzweil Integration

theory *Tagged_Division*

imports *Topology_Euclidean_Space*

begin

lemma *sum_Sigma_product*:

assumes *finite S*

and $\bigwedge i. i \in S \implies \text{finite } (T\ i)$

shows $(\sum_{i \in S} \text{sum } (x\ i) (T\ i)) = (\sum_{(i, j) \in \text{Sigma } S\ T. x\ i\ j})$

using *assms*

proof *induct*

case *empty*

then show *?case*

by *simp*

next

case *(insert a S)*

show *?case*

by *(simp add: insert.hyps(1) insert.premis sum.Sigma)*

qed

lemmas *scaleR_simps* = *scaleR_zero_left scaleR_minus_left scaleR_left_diff_distrib*

scaleR_zero_right scaleR_minus_right scaleR_right_diff_distrib scaleR_eq_0_iff

scaleR_cancel_left scaleR_cancel_right scaleR_add_right scaleR_add_left real_vector_class.scaleR_c

6.14.1 Sundries

A transitive relation is well-founded if all initial segments are finite.

lemma *wf_finite_segments*:

assumes *irrefl r* **and** *trans r* **and** $\bigwedge x. \text{finite } \{y. (y, x) \in r\}$

shows *wf (r)*

apply *(simp add: trans_wf_iff wf_iff_acyclic_if_finite converse_def assms)*

using *acyclic_def assms irrefl_def trans_Restr* **by** *fastforce*

For creating values between *u* and *v*.

lemma *scaling_mono*:

fixes *u::'a::linordered_field*

assumes $u \leq v$ $0 \leq r$ $r \leq s$

shows $u + r * (v - u) / s \leq v$

proof $-$

have $r/s \leq 1$ **using** *assms*

using *divide_le_eq_1* **by** *fastforce*

then have $(r/s) * (v - u) \leq 1 * (v - u)$


```

    by (meson diff_ge_0_iff_ge mult_right_mono ⟨u ≤ v⟩)
  then show ?thesis
    by (simp add: field_simps)
qed

```

6.14.2 Some useful lemmas about intervals

```

lemma interior_subset_union_intervals:
  assumes i = cbox a b
    and j = cbox c d
    and interior j ≠ {}
    and i ⊆ j ∪ S
    and interior i ∩ interior j = {}
  shows interior i ⊆ interior S
proof -
  have box a b ∩ cbox c d = {}
    using Int_interval_mixed_eq_empty[of c d a b] assms
    unfolding interior_cbox by auto
  moreover
  have box a b ⊆ cbox c d ∪ S
    apply (rule order_trans, rule box_subset_cbox)
    using assms by auto
  ultimately
  show ?thesis
    unfolding assms interior_cbox
    by auto (metis IntI UnE empty_iff interior_maximal open_box subsetCE
subsetI)
qed

```

```

lemma interior_Union_subset_cbox:
  assumes finite f
  assumes f: ∧s. s ∈ f ⟹ ∃ a b. s = cbox a b ∧ s. s ∈ f ⟹ interior s ⊆ t
    and t: closed t
  shows interior (⋃f) ⊆ t
proof -
  have [simp]: s ∈ f ⟹ closed s for s
    using f by auto
  define E where E = {s ∈ f. interior s = {}}
  then have finite E E ⊆ {s ∈ f. interior s = {}}
    using ⟨finite f⟩ by auto
  then have interior (⋃f) = interior (⋃(f - E))
  proof (induction E rule: finite_subset_induct)
    case (insert s f')
    have interior (⋃(f - insert s f') ∪ s) = interior (⋃(f - insert s f'))
      using insert.hyps ⟨finite f⟩ by (intro interior_closed_Un_empty_interior)
    auto
    also have ⋃(f - insert s f') ∪ s = ⋃(f - f')
      using insert.hyps by auto
    finally show ?case

```

```

      by (simp add: insert.IH)
    qed simp
  also have ...  $\subseteq \bigcup (f - E)$ 
    by (rule interior_subset)
  also have ...  $\subseteq t$ 
  proof (rule Union_least)
    fix s assume  $s \in f - E$ 
    with f[of s] obtain a b where  $s \in f \ s = \text{cbox } a \ b \ \text{box } a \ b \neq \{\}$ 
      by (fastforce simp: E_def)
    have closure (interior s)  $\subseteq$  closure t
      by (intro closure_mono f  $\langle s \in f \rangle$ )
    with s  $\langle \text{closed } t \rangle$  show  $s \subseteq t$ 
      by simp
  qed
  finally show ?thesis .
qed

```

lemma *Int_interior_Union_intervals*:

$$\llbracket \text{finite } \mathcal{F}; \text{ open } S; \bigwedge T. T \in \mathcal{F} \implies \exists a \ b. T = \text{cbox } a \ b; \bigwedge T. T \in \mathcal{F} \implies S \cap (\text{interior } T) = \{\} \rrbracket$$

$$\implies S \cap \text{interior } (\bigcup \mathcal{F}) = \{\}$$

using *interior_Union_subset_cbox*[of $\mathcal{F}$ UNIV - S] by auto

lemma *interval_split*:

fixes $a :: 'a::\text{euclidean\_space}$

assumes $k \in \text{Basis}$

shows

$$\text{cbox } a \ b \cap \{x. x \cdot k \leq c\} = \text{cbox } a \ (\sum_{i \in \text{Basis}. (\text{if } i = k \text{ then } \min (b \cdot k) \ c \text{ else } b \cdot i) *_{\mathbb{R}} i)$$

$$\text{cbox } a \ b \cap \{x. x \cdot k \geq c\} = \text{cbox } (\sum_{i \in \text{Basis}. (\text{if } i = k \text{ then } \max (a \cdot k) \ c \text{ else } a \cdot i) *_{\mathbb{R}} i) \ b$$

using *assms* by (rule_tac set_eqI; auto simp: mem_box)+

lemma *interval_not_empty*: $\forall i \in \text{Basis}. a \cdot i \leq b \cdot i \implies \text{cbox } a \ b \neq \{\}$

by (simp add: box_ne_empty)

6.14.3 Bounds on intervals where they exist

definition *interval_upperbound* :: $('a::\text{euclidean\_space}) \text{ set} \Rightarrow 'a$

where *interval_upperbound* $s = (\sum_{i \in \text{Basis}. (\text{SUP } x \in s. x \cdot i) *_{\mathbb{R}} i)$

definition *interval_lowerbound* :: $('a::\text{euclidean\_space}) \text{ set} \Rightarrow 'a$

where *interval_lowerbound* $s = (\sum_{i \in \text{Basis}. (\text{INF } x \in s. x \cdot i) *_{\mathbb{R}} i)$

lemma *interval_upperbound[simp]*:

$$\forall i \in \text{Basis}. a \cdot i \leq b \cdot i \implies$$

$$\text{interval\_upperbound } (\text{cbox } a \ b) = (b :: 'a::\text{euclidean\_space})$$

unfolding *interval_upperbound_def* *euclidean_representation_sum* *cbox_def*

by (safe intro!: cSup_eq) auto

lemma *interval_lowerbound[simp]*:

$\forall i \in \text{Basis}. a \cdot i \leq b \cdot i \implies$

$\text{interval\_lowerbound} (\text{cbox } a \ b) = (a :: 'a :: \text{euclidean\_space})$

unfolding *interval_lowerbound_def euclidean_representation_sum cbox_def*

by (*safe intro!*; *cInf_eq*) *auto*

lemmas *interval_bounds = interval_upperbound interval_lowerbound*

lemma

fixes *X::real set*

shows *interval_upperbound_real[simp]*: $\text{interval\_upperbound } X = \text{Sup } X$

and *interval_lowerbound_real[simp]*: $\text{interval\_lowerbound } X = \text{Inf } X$

by (*auto simp: interval_upperbound_def interval_lowerbound_def*)

lemma *interval_bounds'[simp]*:

assumes $\text{cbox } a \ b \neq \{\}$

shows $\text{interval\_upperbound} (\text{cbox } a \ b) = b$

and $\text{interval\_lowerbound} (\text{cbox } a \ b) = a$

using *assms unfolding box_ne_empty* **by** *auto*

lemma *interval_upperbound_Times*:

assumes $A \neq \{\}$ **and** $B \neq \{\}$

shows $\text{interval\_upperbound} (A \times B) = (\text{interval\_upperbound } A, \text{interval\_upperbound } B)$

proof–

from *assms have fst_image_times'*: $A = \text{fst } (A \times B)$ **by** *simp*

have $(\sum i \in \text{Basis}. (\text{SUP } x \in A \times B. x \cdot (i, 0)) *_{\mathbb{R}} i) = (\sum i \in \text{Basis}. (\text{SUP } x \in A. x \cdot i) *_{\mathbb{R}} i)$

by (*subst (2) fst_image_times'*) (*simp del: fst_image_times add: image_comp inner_Pair_0*)

moreover from *assms have snd_image_times'*: $B = \text{snd } (A \times B)$ **by** *simp*

have $(\sum i \in \text{Basis}. (\text{SUP } x \in A \times B. x \cdot (0, i)) *_{\mathbb{R}} i) = (\sum i \in \text{Basis}. (\text{SUP } x \in B. x \cdot i) *_{\mathbb{R}} i)$

by (*subst (2) snd_image_times'*) (*simp del: snd_image_times add: image_comp inner_Pair_0*)

ultimately show *?thesis unfolding interval_upperbound_def*

by (*subst sum_Basis_prod_eq*) (*auto simp add: sum_prod*)

qed

lemma *interval_lowerbound_Times*:

assumes $A \neq \{\}$ **and** $B \neq \{\}$

shows $\text{interval\_lowerbound} (A \times B) = (\text{interval\_lowerbound } A, \text{interval\_lowerbound } B)$

proof–

from *assms have fst_image_times'*: $A = \text{fst } (A \times B)$ **by** *simp*

have $(\sum i \in \text{Basis}. (\text{INF } x \in A \times B. x \cdot (i, 0)) *_{\mathbb{R}} i) = (\sum i \in \text{Basis}. (\text{INF } x \in A. x \cdot i) *_{\mathbb{R}} i)$

by (*subst (2) fst_image_times'*) (*simp del: fst_image_times add: image_comp*)

```

inner_Pair_0)
  moreover from assms have snd_image_times':  $B = \text{snd } (A \times B)$  by simp
  have  $(\sum_{i \in \text{Basis}} (\text{INF } x \in A \times B. x \cdot (0, i)) *_R i) = (\sum_{i \in \text{Basis}} (\text{INF } x \in B. x \cdot i) *_R i)$ 
  by (subst (2) snd_image_times') (simp del: snd_image_times add: image_comp inner_Pair_0)
  ultimately show ?thesis unfolding interval_lowerbound_def
  by (subst sum_Basis_prod_eq) (auto simp add: sum_prod)
qed

```

6.14.4 The notion of a gauge — simply an open set containing the point

definition $\text{gauge } \gamma \longleftrightarrow (\forall x. x \in \gamma \wedge \text{open } (\gamma \ x))$

```

lemma gaugeI:
  assumes  $\bigwedge x. x \in \gamma$ 
  and  $\bigwedge x. \text{open } (\gamma \ x)$ 
  shows gauge  $\gamma$ 
  using assms unfolding gauge_def by auto

```

```

lemma gaugeD[dest]:
  assumes gauge  $\gamma$ 
  shows  $x \in \gamma$ 
  and  $\text{open } (\gamma \ x)$ 
  using assms unfolding gauge_def by auto

```

```

lemma gauge_ball_dependent:  $\forall x. 0 < e \implies \text{gauge } (\lambda x. \text{ball } x \ e \ x)$ 
  unfolding gauge_def by auto

```

```

lemma gauge_ball[intro]:  $0 < e \implies \text{gauge } (\lambda x. \text{ball } x \ e)$ 
  unfolding gauge_def by auto

```

```

lemma gauge_trivial[intro!]: gauge  $(\lambda x. \text{ball } x \ 1)$ 
  by (rule gauge_ball) auto

```

```

lemma gauge_Int[intro]: gauge  $\gamma_1 \implies \text{gauge } \gamma_2 \implies \text{gauge } (\lambda x. \gamma_1 \ x \cap \gamma_2 \ x)$ 
  unfolding gauge_def by auto

```

```

lemma gauge_reflect:
  fixes  $\gamma :: 'a::\text{euclidean\_space} \Rightarrow 'a \text{ set}$ 
  shows gauge  $\gamma \implies \text{gauge } (\lambda x. \text{uminus } (x))$ 
  using equation_minus_iff
  by (auto simp: gauge_def surj_def intro!: open_surjective_linear_image linear_uminus)

```

```

lemma gauge_Inter:
  assumes finite  $S$ 
  and  $\bigwedge u. u \in S \implies \text{gauge } (f \ u)$ 

```

```

  shows gauge  $(\lambda x. \bigcap \{f\ u\ x \mid u. u \in S\})$ 
proof -
  have *:  $\bigwedge x. \{f\ u\ x \mid u. u \in S\} = (\lambda u. f\ u\ x) \text{ ` } S$ 
    by auto
  show ?thesis
    unfolding gauge_def unfolding *
    using assms unfolding Ball_def Inter_iff mem_Collect_eq gauge_def by
auto
qed

```

lemma gauge_existence_lemma:

```

 $(\forall x. \exists d :: \text{real}. p\ x \longrightarrow 0 < d \wedge q\ d\ x) \longleftrightarrow (\forall x. \exists d > 0. p\ x \longrightarrow q\ d\ x)$ 
by (metis zero_less_one)

```

6.14.5 Attempt a systematic general set of "offset" results for components

lemma gauge_modify:

```

assumes  $(\forall S. \text{open } S \longrightarrow \text{open } \{x. f(x) \in S\})$  gauge d
shows gauge  $(\lambda x. \{y. f\ y \in d\ (f\ x)\})$ 
using assms unfolding gauge_def by force

```

6.14.6 Divisions

definition division_of (infixl division'_of 40)

where

```

s division_of i  $\longleftrightarrow$ 
  finite s  $\wedge$ 
   $(\forall K \in s. K \subseteq i \wedge K \neq \{\}) \wedge (\exists a\ b. K = \text{cbox } a\ b) \wedge$ 
   $(\forall K1 \in s. \forall K2 \in s. K1 \neq K2 \longrightarrow \text{interior}(K1) \cap \text{interior}(K2) = \{\}) \wedge$ 
   $(\bigcup s = i)$ 

```

lemma division_ofD[dest]:

```

assumes s division_of i
shows finite s
  and  $\bigwedge K. K \in s \implies K \subseteq i$ 
  and  $\bigwedge K. K \in s \implies K \neq \{\}$ 
  and  $\bigwedge K. K \in s \implies \exists a\ b. K = \text{cbox } a\ b$ 
  and  $\bigwedge K1\ K2. K1 \in s \implies K2 \in s \implies K1 \neq K2 \implies \text{interior}(K1) \cap \text{interior}(K2) = \{\}$ 
  and  $\bigcup s = i$ 
using assms unfolding division_of_def by auto

```

lemma division_ofI:

```

assumes finite s
  and  $\bigwedge K. K \in s \implies K \subseteq i$ 
  and  $\bigwedge K. K \in s \implies K \neq \{\}$ 
  and  $\bigwedge K. K \in s \implies \exists a\ b. K = \text{cbox } a\ b$ 
  and  $\bigwedge K1\ K2. K1 \in s \implies K2 \in s \implies K1 \neq K2 \implies \text{interior } K1 \cap \text{interior}$ 

```

1730

```

K2 = {}
  and  $\bigcup s = i$ 
  shows  $s$  division_of  $i$ 
  using assms unfolding division_of_def by auto

```

```

lemma division_of_finite:  $s$  division_of  $i \implies \text{finite } s$ 
  by auto

```

```

lemma division_of_self[intro]:  $\text{cbox } a \ b \neq \{\} \implies \{\text{cbox } a \ b\}$  division_of  $(\text{cbox } a \ b)$ 
  unfolding division_of_def by auto

```

```

lemma division_of_trivial[simp]:  $s$  division_of  $\{\} \longleftrightarrow s = \{\}$ 
  unfolding division_of_def by auto

```

```

lemma division_of_sing[simp]:
   $s$  division_of  $\text{cbox } a \ (a::'a::\text{euclidean\_space}) \longleftrightarrow s = \{\text{cbox } a \ a\}$ 
  (is ?l = ?r)

```

```

proof
  assume ?r
  moreover
  { fix  $K$ 
    assume  $s = \{\{a\}\}$   $K \in s$ 
    then have  $\exists x \ y. K = \text{cbox } x \ y$ 
      apply (rule_tac  $x=a$  in exI) +
      apply force
      done
  }
  ultimately show ?l
    unfolding division_of_def cbox_sing by auto
next
  assume ?l
  have  $x = \{a\}$  if  $x \in s$  for  $x$ 
    by (metis  $\langle s$  division_of  $\text{cbox } a \ a \rangle$  cbox_sing division_ofD(2) division_ofD(3)
    subset_singletonD that)
  moreover have  $s \neq \{\}$ 
    using  $\langle s$  division_of  $\text{cbox } a \ a \rangle$  by auto
  ultimately show ?r
    unfolding cbox_sing by auto
qed

```

```

lemma elementary_empty: obtains  $p$  where  $p$  division_of  $\{\}$ 
  unfolding division_of_trivial by auto

```

```

lemma elementary_interval: obtains  $p$  where  $p$  division_of  $(\text{cbox } a \ b)$ 
  by (metis division_of_trivial division_of_self)

```

```

lemma division_contains:  $s$  division_of  $i \implies \forall x \in i. \exists k \in s. x \in k$ 
  unfolding division_of_def by auto

```

lemma *forall_in_division*:

$d \text{ division\_of } i \implies (\forall x \in d. P \ x) \longleftrightarrow (\forall a \ b. \text{cbox } a \ b \in d \longrightarrow P \ (\text{cbox } a \ b))$
unfolding *division_of_def* **by** *fastforce*

lemma *cbox_division_memE*:

assumes $\mathcal{D}: K \in \mathcal{D} \ \mathcal{D} \text{ division\_of } S$
obtains $a \ b$ **where** $K = \text{cbox } a \ b \ K \neq \{\}$ $K \subseteq S$
using *assms* **unfolding** *division_of_def* **by** *metis*

lemma *division_of_subset*:

assumes $p \text{ division\_of } (\bigcup p)$
and $q \subseteq p$
shows $q \text{ division\_of } (\bigcup q)$
proof (*rule division_ofI*)
note $* = \text{division\_ofD}[OF \text{ assms}(1)]$
show *finite* q
using $*(1) \text{ assms}(2) \text{ infinite\_super}$ **by** *auto*
{
fix k
assume $k \in q$
then have $kp: k \in p$
using *assms*(2) **by** *auto*
show $k \subseteq \bigcup q$
using $\langle k \in q \rangle$ **by** *auto*
show $\exists a \ b. k = \text{cbox } a \ b$
using $*(4)[OF \ kp]$ **by** *auto*
show $k \neq \{\}$
using $*(3)[OF \ kp]$ **by** *auto*
}
fix $k1 \ k2$
assume $k1 \in q \ k2 \in q \ k1 \neq k2$
then have $** : k1 \in p \ k2 \in p \ k1 \neq k2$
using *assms*(2) **by** *auto*
show $\text{interior } k1 \cap \text{interior } k2 = \{\}$
using $*(5)[OF \ **]$ **by** *auto*
qed *auto*

lemma *division_of_union_self[intro]*: $p \text{ division\_of } s \implies p \text{ division\_of } (\bigcup p)$

unfolding *division_of_def* **by** *auto*

lemma *division_inter*:

fixes $s1 \ s2 :: 'a::\text{euclidean\_space set}$
assumes $p1 \text{ division\_of } s1$
and $p2 \text{ division\_of } s2$
shows $\{k1 \cap k2 \mid k1 \ k2. k1 \in p1 \wedge k2 \in p2 \wedge k1 \cap k2 \neq \{\}\} \text{ division\_of } (s1 \cap s2)$
(is ?A' division_of _)
proof –

```

let ?A = {s. s ∈ (λ(k1,k2). k1 ∩ k2) ‘ (p1 × p2) ∧ s ≠ {}}
have *: ?A' = ?A by auto
show ?thesis
  unfolding *
proof (rule division_ofI)
  have ?A ⊆ (λ(x, y). x ∩ y) ‘ (p1 × p2)
  by auto
moreover have finite (p1 × p2)
  using assms unfolding division_of_def by auto
ultimately show finite ?A by auto
have *: ⋂ s. ⋃ {x ∈ s. x ≠ {}} = ⋃ s
  by auto
show ⋃ ?A = s1 ∩ s2
  unfolding *
  using division_ofD(6)[OF assms(1)] and division_ofD(6)[OF assms(2)] by
auto
{
  fix k
  assume k ∈ ?A
  then obtain k1 k2 where k: k = k1 ∩ k2 k1 ∈ p1 k2 ∈ p2 k ≠ {}
  by auto
  then show k ≠ {}
  by auto
  show k ⊆ s1 ∩ s2
  using division_ofD(2)[OF assms(1) k(2)] and division_ofD(2)[OF assms(2)
k(3)]
  unfolding k by auto
  obtain a1 b1 where k1: k1 = cbox a1 b1
  using division_ofD(4)[OF assms(1) k(2)] by blast
  obtain a2 b2 where k2: k2 = cbox a2 b2
  using division_ofD(4)[OF assms(2) k(3)] by blast
  show ∃ a b. k = cbox a b
  unfolding k k1 k2 unfolding Int_interval by auto
}
fix k1 k2
assume k1 ∈ ?A
then obtain x1 y1 where k1: k1 = x1 ∩ y1 x1 ∈ p1 y1 ∈ p2 k1 ≠ {}
  by auto
assume k2 ∈ ?A
then obtain x2 y2 where k2: k2 = x2 ∩ y2 x2 ∈ p1 y2 ∈ p2 k2 ≠ {}
  by auto
assume k1 ≠ k2
then have th: x1 ≠ x2 ∨ y1 ≠ y2
  unfolding k1 k2 by auto
have *: interior x1 ∩ interior x2 = {} ∨ interior y1 ∩ interior y2 = {} ⇒
  interior (x1 ∩ y1) ⊆ interior x1 ⇒ interior (x1 ∩ y1) ⊆ interior y1 ⇒
  interior (x2 ∩ y2) ⊆ interior x2 ⇒ interior (x2 ∩ y2) ⊆ interior y2 ⇒
  interior (x1 ∩ y1) ∩ interior (x2 ∩ y2) = {} by auto
show interior k1 ∩ interior k2 = {}

```



```

    unfolding k1 k2
    apply (rule *)
    using assms division_ofD(5) k1 k2(2) k2(3) th apply auto
    done
  qed
qed

lemma division_inter_1:
  assumes d division_of i
  and cbox a (b::'a::euclidean_space)  $\subseteq$  i
  shows {cbox a b  $\cap$  k | k. k  $\in$  d  $\wedge$  cbox a b  $\cap$  k  $\neq$  {}} division_of (cbox a b)
proof (cases cbox a b = {})
  case True
  show ?thesis
    unfolding True and division_of_trivial by auto
next
  case False
  have *: cbox a b  $\cap$  i = cbox a b using assms(2) by auto
  show ?thesis
    using division_inter[OF division_of_self[OF False] assms(1)]
    unfolding * by auto
qed

lemma elementary_Int:
  fixes s t :: 'a::euclidean_space set
  assumes p1 division_of s
  and p2 division_of t
  shows  $\exists p. p$  division_of (s  $\cap$  t)
using assms division_inter by blast

lemma elementary_Inter:
  assumes finite f
  and f  $\neq$  {}
  and  $\forall s \in f. \exists p. p$  division_of (s::('a::euclidean_space) set)
  shows  $\exists p. p$  division_of ( $\bigcap$  f)
using assms
proof (induct f rule: finite_induct)
  case (insert x f)
  show ?case
  proof (cases f = {})
    case True
    then show ?thesis
      unfolding True using insert by auto
  next
    case False
    obtain p where p division_of  $\bigcap$  f
    using insert(3)[OF False insert(5)[unfolded ball_simps, THEN conjunct2]] ..
    moreover obtain px where px division_of x
    using insert(5)[rule_format, OF insertI1] ..

```

```

    ultimately show ?thesis
      by (simp add: elementary_Int Inter_insert)
  qed
qed auto

lemma division_disjoint_union:
  assumes p1 division_of s1
    and p2 division_of s2
    and interior s1  $\cap$  interior s2 = {}
  shows (p1  $\cup$  p2) division_of (s1  $\cup$  s2)
proof (rule division_ofI)
  note d1 = division_ofD[OF assms(1)]
  note d2 = division_ofD[OF assms(2)]
  show finite (p1  $\cup$  p2)
    using d1(1) d2(1) by auto
  show  $\bigcup$  (p1  $\cup$  p2) = s1  $\cup$  s2
    using d1(6) d2(6) by auto
  {
    fix k1 k2
    assume as: k1  $\in$  p1  $\cup$  p2 k2  $\in$  p1  $\cup$  p2 k1  $\neq$  k2
    moreover
    let ?g = interior k1  $\cap$  interior k2 = {}
    {
      assume as: k1  $\in$  p1 k2  $\in$  p2
      have ?g
        using interior_mono[OF d1(2)[OF as(1)]] interior_mono[OF d2(2)[OF
as(2)]]
        using assms(3) by blast
    }
    moreover
    {
      assume as: k1  $\in$  p2 k2  $\in$  p1
      have ?g
        using interior_mono[OF d1(2)[OF as(2)]] interior_mono[OF d2(2)[OF
as(1)]]
        using assms(3) by blast
    }
    ultimately show ?g
      using d1(5)[OF __ as(3)] and d2(5)[OF __ as(3)] by auto
  }
  fix k
  assume k: k  $\in$  p1  $\cup$  p2
  show k  $\subseteq$  s1  $\cup$  s2
    using k d1(2) d2(2) by auto
  show k  $\neq$  {}
    using k d1(3) d2(3) by auto
  show  $\exists a b. k = \text{cbox } a b$ 
    using k d1(4) d2(4) by auto
qed

```

```

lemma partial_division_extend_1:
  fixes a b c d :: 'a::euclidean_space
  assumes incl: cbox c d  $\subseteq$  cbox a b
  and nonempty: cbox c d  $\neq$  {}
  obtains p where p division_of (cbox a b) cbox c d  $\in$  p
proof
  let ?B =  $\lambda f::'a \Rightarrow 'a \times 'a.$ 
    cbox ( $\sum i \in \text{Basis}. (\text{fst } (f i) \cdot i) *_R i$ ) ( $\sum i \in \text{Basis}. (\text{snd } (f i) \cdot i) *_R i$ )
  define p where p = ?B ' (Basis  $\rightarrow_E$  {(a, c), (c, d), (d, b)})

  show cbox c d  $\in$  p
  unfolding p_def
  by (auto simp add: box_eq_empty cbox_def intro!: image_eqI [where x= $\lambda(i::'a) \in \text{Basis}.$ 
    (c, d)])
  have ord: a  $\cdot$  i  $\leq$  c  $\cdot$  i c  $\cdot$  i  $\leq$  d  $\cdot$  i d  $\cdot$  i  $\leq$  b  $\cdot$  i if i  $\in$  Basis for i
  using incl nonempty that
  unfolding box_eq_empty subset_box by (auto simp: not_le)

  show p division_of (cbox a b)
proof (rule division_ofI)
  show finite p
  unfolding p_def by (auto intro!: finite_PiE)
  {
    fix k
    assume k  $\in$  p
    then obtain f where f: f  $\in$  Basis  $\rightarrow_E$  {(a, c), (c, d), (d, b)} and k: k =
    ?B f
    by (auto simp: p_def)
    then show  $\exists a b. k = \text{cbox } a b$ 
    by auto
    have k  $\subseteq$  cbox a b  $\wedge$  k  $\neq$  {}
    proof (simp add: k box_eq_empty subset_box not_less, safe)
      fix i :: 'a
      assume i: i  $\in$  Basis
      with f have f i = (a, c)  $\vee$  f i = (c, d)  $\vee$  f i = (d, b)
      by (auto simp: PiE_iff)
      with i ord[of i]
      show a  $\cdot$  i  $\leq$  fst (f i)  $\cdot$  i snd (f i)  $\cdot$  i  $\leq$  b  $\cdot$  i fst (f i)  $\cdot$  i  $\leq$  snd (f i)  $\cdot$  i
      by auto
    qed
    then show k  $\neq$  {} k  $\subseteq$  cbox a b
    by auto
  }
  {
    fix l
    assume l  $\in$  p
    then obtain g where g: g  $\in$  Basis  $\rightarrow_E$  {(a, c), (c, d), (d, b)} and l: l =
    ?B g
    by (auto simp: p_def)
  }

```

```

    assume  $l \neq k$ 
    have  $\exists i \in \text{Basis}. f\ i \neq g\ i$ 
    proof (rule ccontr)
      assume  $\neg ?thesis$ 
      with  $f\ g$  have  $f = g$ 
      by (auto simp:  $\text{PiE\_iff}\ \text{extensional\_def}\ \text{fun\_eq\_iff}$ )
      with  $\langle l \neq k \rangle$  show False
      by (simp add:  $l\ k$ )
    qed
    then obtain  $i$  where  $*: i \in \text{Basis}\ f\ i \neq g\ i ..$ 
    then have  $f\ i = (a, c) \vee f\ i = (c, d) \vee f\ i = (d, b)$ 
       $g\ i = (a, c) \vee g\ i = (c, d) \vee g\ i = (d, b)$ 
      using  $f\ g$  by (auto simp:  $\text{PiE\_iff}$ )
    with  $*\ \text{ord}[of\ i]$  show  $\text{interior}\ l \cap \text{interior}\ k = \{\}$ 
      by (auto simp add:  $l\ k\ \text{disjoint\_interval}\ \text{intro!}\ \text{bexI}[of\ _\ i]$ )
  }
  note  $\langle k \subseteq \text{cbox}\ a\ b \rangle$ 
}
moreover
{
  fix  $x$  assume  $x: x \in \text{cbox}\ a\ b$ 
  have  $\forall i \in \text{Basis}. \exists l. x \cdot i \in \{\text{fst}\ l \cdot i .. \text{snd}\ l \cdot i\} \wedge l \in \{(a, c), (c, d), (d, b)\}$ 
  proof
    fix  $i :: 'a$ 
    assume  $i \in \text{Basis}$ 
    with  $x\ \text{ord}[of\ i]$ 
    have  $(a \cdot i \leq x \cdot i \wedge x \cdot i \leq c \cdot i) \vee (c \cdot i \leq x \cdot i \wedge x \cdot i \leq d \cdot i) \vee$ 
       $(d \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i)$ 
      by (auto simp:  $\text{cbox\_def}$ )
    then show  $\exists l. x \cdot i \in \{\text{fst}\ l \cdot i .. \text{snd}\ l \cdot i\} \wedge l \in \{(a, c), (c, d), (d, b)\}$ 
      by auto
  qed
  then obtain  $f$  where
     $f: \forall i \in \text{Basis}. x \cdot i \in \{\text{fst}\ (f\ i) \cdot i .. \text{snd}\ (f\ i) \cdot i\} \wedge f\ i \in \{(a, c), (c, d), (d, b)\}$ 
    unfolding  $\text{bchoice\_iff} ..$ 
  moreover from  $f$  have  $\text{restrict}\ f\ \text{Basis} \in \text{Basis} \rightarrow_E \{(a, c), (c, d), (d, b)\}$ 
    by auto
  moreover from  $f$  have  $x \in ?B\ (\text{restrict}\ f\ \text{Basis})$ 
    by (auto simp:  $\text{mem\_box}$ )
  ultimately have  $\exists k \in p. x \in k$ 
    unfolding  $p\_def$  by blast
}
ultimately show  $\bigcup p = \text{cbox}\ a\ b$ 
  by auto
qed
qed

```

proposition *partial_division_extend_interval:*
 assumes $p\ \text{division\_of}\ (\bigcup p)\ (\bigcup p) \subseteq \text{cbox}\ a\ b$

```

obtains  $q$  where  $p \subseteq q$   $q$  division_of  $\text{cbox } a \text{ } (b::'a::\text{euclidean\_space})$ 
proof (cases  $p = \{\}$ )
  case True
    obtain  $q$  where  $q$  division_of ( $\text{cbox } a \text{ } b$ )
    by (rule elementary_interval)
    then show ?thesis
    using True that by blast
  next
    case False
    note  $p = \text{division\_ofD}[OF \text{ assms}(1)]$ 
    have  $\text{div\_cbox}: \forall k \in p. \exists q. q \text{ division\_of } \text{cbox } a \text{ } b \wedge k \in q$ 
    proof
      fix  $k$ 
      assume  $kp: k \in p$ 
      obtain  $c \text{ } d$  where  $k: k = \text{cbox } c \text{ } d$ 
      using  $p(4)[OF kp]$  by blast
      have  $*$ :  $\text{cbox } c \text{ } d \subseteq \text{cbox } a \text{ } b \text{ } \text{cbox } c \text{ } d \neq \{\}$ 
      using  $p(2,3)[OF kp, \text{unfolded } k]$  using  $\text{assms}(2)$ 
      by (blast intro: order.trans)+
      obtain  $q$  where  $q$  division_of  $\text{cbox } a \text{ } b \text{ } \text{cbox } c \text{ } d \in q$ 
      by (rule partial_division_extend_1[OF *])
      then show  $\exists q. q \text{ division\_of } \text{cbox } a \text{ } b \wedge k \in q$ 
      unfolding  $k$  by auto
    qed
    obtain  $q$  where  $q: \bigwedge x. x \in p \implies q \text{ } x \text{ } \text{division\_of } \text{cbox } a \text{ } b \bigwedge x. x \in p \implies x \in$ 
 $q \text{ } x$ 
    using  $\text{bchoice}[OF \text{ div\_cbox}]$  by blast
    have  $q \text{ } x \text{ } \text{division\_of } \bigcup (q \text{ } x)$  if  $x: x \in p$  for  $x$ 
    apply (rule division_ofI)
    using  $\text{division\_ofD}[OF q(1)[OF x]]$  by auto
    then have  $di: \bigwedge x. x \in p \implies \exists d. d \text{ division\_of } \bigcup (q \text{ } x - \{x\})$ 
    by (meson Diff_subset division_of_subset)
    have  $\exists d. d \text{ division\_of } \bigcap ((\lambda i. \bigcup (q \text{ } i - \{i\})) \text{ } 'p)$ 
    apply (rule elementary_Inter [OF finite_imageI[OF p(1)]])
    apply (auto dest: di simp: False elementary_Inter [OF finite_imageI[OF p(1)]])
    done
    then obtain  $d$  where  $d: d \text{ division\_of } \bigcap ((\lambda i. \bigcup (q \text{ } i - \{i\})) \text{ } 'p) \dots$ 
    have  $d \cup p \text{ division\_of } \text{cbox } a \text{ } b$ 
    proof –
      have  $te: \bigwedge S f T. S \neq \{\} \implies \forall i \in S. f \text{ } i \cup i = T \implies T = \bigcap (f \text{ } 'S) \cup \bigcup S$  by
auto
      have  $\text{cbox\_eq}: \text{cbox } a \text{ } b = \bigcap ((\lambda i. \bigcup (q \text{ } i - \{i\})) \text{ } 'p) \cup \bigcup p$ 
      proof (rule te[OF False], clarify)
        fix  $i$ 
        assume  $i: i \in p$ 
        show  $\bigcup (q \text{ } i - \{i\}) \cup i = \text{cbox } a \text{ } b$ 
        using  $\text{division\_ofD}(6)[OF q(1)[OF i]]$  using  $q(2)[OF i]$  by auto
      qed
    { fix }  $K$ 

```

```

assume  $K$ :  $K \in p$ 
note  $qk = \text{division\_of } D[OF\ q(1)[OF\ K]]$ 
have *:  $\bigwedge u\ T\ S. T \cap S = \{\} \implies u \subseteq S \implies u \cap T = \{\}$ 
by auto
have  $\text{interior } (\bigcap_{i \in p}. \bigcup (q\ i - \{i\})) \cap \text{interior } K = \{\}$ 
proof (rule * [OF Int_interior_Union_intervals])
  show  $\bigwedge T. T \in q\ K - \{K\} \implies \text{interior } K \cap \text{interior } T = \{\}$ 
    using  $qk(5)$  using  $q(2)[OF\ K]$  by auto
  show  $\text{interior } (\bigcap_{i \in p}. \bigcup (q\ i - \{i\})) \subseteq \text{interior } (\bigcup (q\ K - \{K\}))$ 
    apply (rule interior_mono)+
    using  $K$  by auto
  qed (use  $qk$  in auto) } note [simp] = this
show  $d \cup p\ \text{division\_of } (\text{cbox } a\ b)$ 
unfolding cbox_eq
apply (rule division_disjoint_union [OF  $d\ \text{assms}(1)$ ])
apply (rule Int_interior_Union_intervals)
apply (rule p_open_interior ballI)+
apply simp_all
done
qed
then show ?thesis
by (meson Un_upper2 that)
qed

lemma elementary_bounded[dest]:
  fixes  $S :: 'a::\text{euclidean\_space}\ \text{set}$ 
  shows  $p\ \text{division\_of } S \implies \text{bounded } S$ 
  unfolding division_of_def by (metis bounded_Union bounded_cbox)

lemma elementary_subset_cbox:
   $p\ \text{division\_of } S \implies \exists a\ b. S \subseteq \text{cbox } a\ b$  ( $b :: 'a::\text{euclidean\_space}$ )
  by (meson bounded_subset_cbox_symmetric elementary_bounded)

proposition division_union_intervals_exists:
  fixes  $a\ b :: 'a::\text{euclidean\_space}$ 
  assumes  $\text{cbox } a\ b \neq \{\}$ 
  obtains  $p$  where (insert (cbox  $a\ b$ )  $p$ )  $\text{division\_of } (\text{cbox } a\ b \cup \text{cbox } c\ d)$ 
proof (cases  $\text{cbox } c\ d = \{\}$ )
  case True
    with assms that show ?thesis by force
  next
    case False
    show ?thesis
    proof (cases  $\text{cbox } a\ b \cap \text{cbox } c\ d = \{\}$ )
      case True
        then show ?thesis
        by (metis that False assms division_disjoint_union division_of_self insert_is_Un interior_Int interior_empty)
      next

```

```

case False
obtain u v where uv: cbox a b  $\cap$  cbox c d = cbox u v
  unfolding Int_interval by auto
have uv_sub: cbox u v  $\subseteq$  cbox c d using uv by auto
obtain p where pd: p division_of cbox c d and cbox u v  $\in$  p
  by (rule partial_division_extend_1[OF uv_sub False[unfolded uv]])
note p = this division_ofD[OF pd]
have interior (cbox a b  $\cap$   $\bigcup$  (p - {cbox u v})) = interior(cbox u v  $\cap$   $\bigcup$  (p - {cbox u v}))
  apply (rule arg_cong[of _ _ interior])
  using p(8) uv by auto
also have ... = {}
  unfolding interior_Int
  apply (rule Int_interior_Union_intervals)
  using p(6) p(7)[OF p(2)]  $\langle$ finite p $\rangle$ 
  apply auto
  done
finally have [simp]: interior (cbox a b)  $\cap$  interior ( $\bigcup$  (p - {cbox u v})) = {}
by simp
have cbe: cbox a b  $\cup$  cbox c d = cbox a b  $\cup$   $\bigcup$  (p - {cbox u v})
  using p(8) unfolding uv[symmetric] by auto
have insert (cbox a b) (p - {cbox u v}) division_of cbox a b  $\cup$   $\bigcup$  (p - {cbox u v})
proof -
  have {cbox a b} division_of cbox a b
    by (simp add: assms division_of_self)
  then show insert (cbox a b) (p - {cbox u v}) division_of cbox a b  $\cup$   $\bigcup$  (p - {cbox u v})
    by (metis (no_types) Diff_subset  $\langle$ interior (cbox a b)  $\cap$  interior ( $\bigcup$  (p - {cbox u v})) = {} $\rangle$  division_disjoint_union division_of_subset insert_is_Un p(1) p(8))
  qed
with that[of p - {cbox u v}] show ?thesis by (simp add: cbe)
qed
qed

```

lemma division_of_Union:

```

assumes finite f
and  $\bigwedge p. p \in f \implies p$  division_of ( $\bigcup p$ )
and  $\bigwedge k1 k2. k1 \in \bigcup f \implies k2 \in \bigcup f \implies k1 \neq k2 \implies$  interior k1  $\cap$  interior k2 = {}
shows  $\bigcup f$  division_of  $\bigcup$  ( $\bigcup f$ )
using assms by (auto intro!: division_ofI)

```

lemma elementary_union_interval:

```

fixes a b :: 'a::euclidean_space
assumes p division_of  $\bigcup p$ 
obtains q where q division_of (cbox a b  $\cup$   $\bigcup p$ )
proof (cases p = {}  $\vee$  cbox a b = {})

```

```

case True
obtain p where p division_of (cbox a b)
  by (rule elementary_interval)
then show thesis
  using True assms that by auto
next
case False
then have p ≠ {} cbox a b ≠ {}
  by auto
note pdiv = division_ofD[OF assms]
show ?thesis
proof (cases interior (cbox a b) = {})
case True
show ?thesis
  apply (rule that [of insert (cbox a b) p, OF division_ofI])
  using pdiv(1-4) True False apply auto
  apply (metis IntI equals0D pdiv(5))
  by (metis disjoint_iff_not_equal pdiv(5))
next
case False
have ∀ K ∈ p. ∃ q. (insert (cbox a b) q) division_of (cbox a b ∪ K)
proof
fix K
assume kp: K ∈ p
from pdiv(4)[OF kp] obtain c d where K = cbox c d by blast
then show ∃ q. (insert (cbox a b) q) division_of (cbox a b ∪ K)
  by (meson ‹cbox a b ≠ {}› division_union_intervals_exists)
qed
from bchoice[OF this] obtain q where ∀ x ∈ p. insert (cbox a b) (q x) division_of
(cbox a b) ∪ x ..
note q = division_ofD[OF this[rule_format]]
let ?D = ⋃ {insert (cbox a b) (q K) | K. K ∈ p}
show thesis
proof (rule that[OF division_ofI])
have *: {insert (cbox a b) (q K) | K. K ∈ p} = (λK. insert (cbox a b) (q K))
‘ p
  by auto
show finite ?D
  using * pdiv(1) q(1) by auto
have ⋃ ?D = (⋃ x ∈ p. ⋃ (insert (cbox a b) (q x)))
  by auto
also have ... = ⋃ {cbox a b ∪ t | t. t ∈ p}
  using q(6) by auto
also have ... = cbox a b ∪ ⋃ p
  using ‹p ≠ {}› by auto
finally show ⋃ ?D = cbox a b ∪ ⋃ p .
show K ⊆ cbox a b ∪ ⋃ p K ≠ {} if K ∈ ?D for K
  using q that by blast+
show ∃ a b. K = cbox a b if K ∈ ?D for K

```



```

    using q(4) that by auto
  next
    fix K' K
    assume K: K ∈ ?D and K': K' ∈ ?D K ≠ K'
    obtain x where x: K ∈ insert (cbox a b) (q x) x ∈ p
      using K by auto
    obtain x' where x': K' ∈ insert (cbox a b) (q x') x' ∈ p
      using K' by auto
    show interior K ∩ interior K' = {}
    proof (cases x = x' ∨ K = cbox a b ∨ K' = cbox a b)
      case True
      then show ?thesis
        using True K' q(5) x' x by auto
    next
      case False
      then have as': K ≠ cbox a b K' ≠ cbox a b
        by auto
      obtain c d where K: K = cbox c d
        using q(4) x by blast
      have interior K ∩ interior (cbox a b) = {}
        using as' K'(2) q(5) x by blast
      then have interior K ⊆ interior x
        by (metis ⟨interior (cbox a b) ≠ {}⟩ K q(2) x interior_subset_union_intervals)
      moreover
      obtain c d where c_d: K' = cbox c d
        using q(4)[OF x'(2,1)] by blast
      have interior K' ∩ interior (cbox a b) = {}
        using as'(2) q(5) x' by blast
      then have interior K' ⊆ interior x'
        by (metis ⟨interior (cbox a b) ≠ {}⟩ c_d interior_subset_union_intervals
          q(2) x'(1) x'(2))
      moreover have interior x ∩ interior x' = {}
        by (meson False assms division_ofD(5) x'(2) x(2))
      ultimately show ?thesis
        using ⟨interior K ⊆ interior x⟩ ⟨interior K' ⊆ interior x'⟩ by auto
    qed
  qed
qed
qed
qed

```

```

lemma elementary_unions_intervals:
  assumes fin: finite f
    and ∧s. s ∈ f ⟹ ∃ a b. s = cbox a (b::'a::euclidean_space)
  obtains p where p division_of (⋃ f)
proof -
  have ∃ p. p division_of (⋃ f)
  proof (induct_tac f rule:finite_subset_induct)

```

```

    show  $\exists p. p \text{ division\_of } \bigcup \{\}$  using elementary_empty by auto
  next
  fix  $x F$ 
  assume as: finite  $F$   $x \notin F$   $\exists p. p \text{ division\_of } \bigcup F$   $x \in f$ 
  from this(3) obtain  $p$  where  $p: p \text{ division\_of } \bigcup F$  ..
  from assms(2)[OF as(4)] obtain  $a b$  where  $x: x = \text{cbox } a b$  by blast
  have *:  $\bigcup F = \bigcup p$ 
    using division_ofD[OF p] by auto
  show  $\exists p. p \text{ division\_of } \bigcup (\text{insert } x F)$ 
    using elementary_union_interval[OF p[unfolded *], of a b]
    unfolding Union_insert  $x$  * by metis
  qed (insert assms, auto)
  then show ?thesis
    using that by auto
  qed

```

```

lemma elementary_union:
  fixes  $S T :: 'a::\text{euclidean\_space}$  set
  assumes  $ps \text{ division\_of } S$   $pt \text{ division\_of } T$ 
  obtains  $p$  where  $p \text{ division\_of } (S \cup T)$ 
proof -
  have *:  $S \cup T = \bigcup ps \cup \bigcup pt$ 
    using assms unfolding division_of_def by auto
  show ?thesis
    apply (rule elementary_unions_intervals[of ps  $\cup$  pt])
    using assms apply auto
    by (simp add: * that)
  qed

```

```

lemma partial_division_extend:
  fixes  $T :: 'a::\text{euclidean\_space}$  set
  assumes  $p \text{ division\_of } S$ 
    and  $q \text{ division\_of } T$ 
    and  $S \subseteq T$ 
  obtains  $r$  where  $p \subseteq r$  and  $r \text{ division\_of } T$ 
proof -
  note  $\text{divp} = \text{division\_ofD}[OF \text{assms}(1)]$  and  $\text{divq} = \text{division\_ofD}[OF \text{assms}(2)]$ 
  obtain  $a b$  where  $ab: T \subseteq \text{cbox } a b$ 
    using elementary_subset_cbox[OF assms(2)] by auto
  obtain  $r1$  where  $p \subseteq r1$   $r1 \text{ division\_of } (\text{cbox } a b)$ 
    using assms
    by (metis ab dual_order.trans partial_division_extend_interval  $\text{divp}$ (6))
  note  $r1 = \text{this division\_ofD}[OF \text{this}(2)]$ 
  obtain  $p'$  where  $p' \text{ division\_of } \bigcup (r1 - p)$ 
    apply (rule elementary_unions_intervals[of r1  $- p$ ])
    using  $r1$ (3,6)
    apply auto
  done
  then obtain  $r2$  where  $r2: r2 \text{ division\_of } (\bigcup (r1 - p)) \cap (\bigcup q)$ 

```

```

  by (metis assms(2) divq(6) elementary_Int)
{
  fix x
  assume x:  $x \in T \wedge x \notin S$ 
  then obtain R where r:  $R \in r1 \wedge x \in R$ 
    unfolding r1 using ab
    by (meson division_contains r1(2) subsetCE)
  moreover have  $R \notin p$ 
  proof
    assume  $R \in p$ 
    then have  $x \in S$  using divp(2) r by auto
    then show False using x by auto
  qed
  ultimately have  $x \in \bigcup (r1 - p)$  by auto
}
then have Teq:  $T = \bigcup p \cup (\bigcup (r1 - p) \cap \bigcup q)$ 
  unfolding divp divq using assms(3) by auto
have interior  $S \cap \text{interior } (\bigcup (r1 - p)) = \{\}$ 
proof (rule Int_interior_Union_intervals)
  have *:  $\bigwedge S. (\bigwedge x. x \in S \implies \text{False}) \implies S = \{\}$ 
  by auto
  show interior  $S \cap \text{interior } m = \{\}$  if  $m \in r1 - p$  for m
  proof -
    have interior  $m \cap \text{interior } (\bigcup p) = \{\}$ 
    proof (rule Int_interior_Union_intervals)
      show  $\bigwedge T. T \in p \implies \text{interior } m \cap \text{interior } T = \{\}$ 
      by (metis DiffD1 DiffD2 that r1(1) r1(7) rev_subsetD)
    qed (use divp in auto)
    then show interior  $S \cap \text{interior } m = \{\}$ 
    unfolding divp by auto
  qed
qed (use r1 in auto)
then have interior  $S \cap \text{interior } (\bigcup (r1 - p) \cap (\bigcup q)) = \{\}$ 
  using interior_subset by auto
then have div:  $p \cup r2 \text{ division\_of } \bigcup p \cup \bigcup (r1 - p) \cap \bigcup q$ 
  by (simp add: assms(1) division_disjoint_union divp(6) r2)
show ?thesis
  apply (rule that[of  $p \cup r2$ ])
  apply (auto simp: div Teq)
done
qed

```

lemma division_split:

```

  fixes a :: 'a::euclidean_space
  assumes p division_of (cbox a b)
  and k:  $k \in \text{Basis}$ 
  shows  $\{l \cap \{x. x \cdot k \leq c\} \mid l \in p \wedge l \cap \{x. x \cdot k \leq c\} \neq \{\}\} \text{ division\_of } (\text{cbox } a$ 
 $b \cap \{x. x \cdot k \leq c\})$ 

```

```

      (is ?p1 division_of ?I1)
    and {l ∩ {x. x·k ≥ c} | l. l ∈ p ∧ l ∩ {x. x·k ≥ c} ≠ {}} division_of (cbox a
b ∩ {x. x·k ≥ c})
      (is ?p2 division_of ?I2)
  proof (rule_tac[!] division_ofI)
    note p = division_ofD[OF assms(1)]
    show finite ?p1 finite ?p2
      using p(1) by auto
    show ∪ ?p1 = ?I1 ∪ ?p2 = ?I2
      unfolding p(6)[symmetric] by auto
  {
    fix K
    assume K ∈ ?p1
    then obtain l where l: K = l ∩ {x. x · k ≤ c} l ∈ p l ∩ {x. x · k ≤ c} ≠ {}
      by blast
    obtain u v where uv: l = cbox u v
      using assms(1) l(2) by blast
    show K ⊆ ?I1
      using l p(2) uv by force
    show K ≠ {}
      by (simp add: l)
    show ∃ a b. K = cbox a b
      apply (simp add: l uv p(2-3)[OF l(2)])
      apply (subst interval_split[OF k])
      apply (auto intro: order.trans)
      done
    fix K'
    assume K' ∈ ?p1
    then obtain l' where l': K' = l' ∩ {x. x · k ≤ c} l' ∈ p l' ∩ {x. x · k ≤ c}
      ≠ {}
      by blast
    assume K ≠ K'
    then show interior K ∩ interior K' = {}
      unfolding l l' using p(5)[OF l(2) l'(2)] by auto
  }
{
  fix K
  assume K ∈ ?p2
  then obtain l where l: K = l ∩ {x. c ≤ x · k} l ∈ p l ∩ {x. c ≤ x · k} ≠ {}
    by blast
  obtain u v where uv: l = cbox u v
    using l(2) p(4) by blast
  show K ⊆ ?I2
    using l p(2) uv by force
  show K ≠ {}
    by (simp add: l)
  show ∃ a b. K = cbox a b
    apply (simp add: l uv p(2-3)[OF l(2)])
    apply (subst interval_split[OF k])

```

```

    apply (auto intro: order.trans)
  done
  fix K'
  assume K' ∈ ?p2
  then obtain l' where l': K' = l' ∩ {x. c ≤ x · k} l' ∈ p l' ∩ {x. c ≤ x · k}
≠ {}
    by blast
  assume K ≠ K'
  then show interior K ∩ interior K' = {}
    unfolding l l' using p(5)[OF l(2) l'(2)] by auto
}
qed

```

6.14.7 Tagged (partial) divisions

definition *tagged_partial_division_of* (**infixr** *tagged'_partial'_division'_of* 40)

where $s \text{ tagged\_partial\_division\_of } i \longleftrightarrow$

$$\text{finite } s \wedge$$

$$(\forall x K. (x, K) \in s \longrightarrow x \in K \wedge K \subseteq i \wedge (\exists a b. K = \text{cbox } a \ b)) \wedge$$

$$(\forall x1 K1 x2 K2. (x1, K1) \in s \wedge (x2, K2) \in s \wedge (x1, K1) \neq (x2, K2) \longrightarrow$$

$$\text{interior } K1 \cap \text{interior } K2 = \{\})$$

lemma *tagged_partial_division_ofD*:

assumes $s \text{ tagged\_partial\_division\_of } i$

shows *finite s*

and $\bigwedge x K. (x, K) \in s \implies x \in K$

and $\bigwedge x K. (x, K) \in s \implies K \subseteq i$

and $\bigwedge x K. (x, K) \in s \implies \exists a b. K = \text{cbox } a \ b$

and $\bigwedge x1 K1 x2 K2. (x1, K1) \in s \implies$

$$(x2, K2) \in s \implies (x1, K1) \neq (x2, K2) \implies \text{interior } K1 \cap \text{interior } K2 = \{\}$$

using *assms* **unfolding** *tagged_partial_division_of_def* **by** *blast+*

definition *tagged_division_of* (**infixr** *tagged'_division'_of* 40)

where $s \text{ tagged\_division\_of } i \longleftrightarrow s \text{ tagged\_partial\_division\_of } i \wedge (\bigcup \{K. \exists x. (x, K) \in s\} = i)$

lemma *tagged_division_of_finite*: $s \text{ tagged\_division\_of } i \implies \text{finite } s$

unfolding *tagged_division_of_def* *tagged_partial_division_of_def* **by** *auto*

lemma *tagged_division_of*:

$s \text{ tagged\_division\_of } i \longleftrightarrow$

$$\text{finite } s \wedge$$

$$(\forall x K. (x, K) \in s \longrightarrow x \in K \wedge K \subseteq i \wedge (\exists a b. K = \text{cbox } a \ b)) \wedge$$

$$(\forall x1 K1 x2 K2. (x1, K1) \in s \wedge (x2, K2) \in s \wedge (x1, K1) \neq (x2, K2) \longrightarrow$$

$$\text{interior } K1 \cap \text{interior } K2 = \{\}) \wedge$$

$$(\bigcup \{K. \exists x. (x, K) \in s\} = i)$$

unfolding *tagged_division_of_def* *tagged_partial_division_of_def* **by** *auto*

lemma *tagged_division_ofI*:

assumes *finite s*
and $\bigwedge x K. (x, K) \in s \implies x \in K$
and $\bigwedge x K. (x, K) \in s \implies K \subseteq i$
and $\bigwedge x K. (x, K) \in s \implies \exists a b. K = \text{cbox } a \ b$
and $\bigwedge x1 K1 x2 K2. (x1, K1) \in s \implies (x2, K2) \in s \implies (x1, K1) \neq (x2, K2)$
 $\implies$
 $\text{interior } K1 \cap \text{interior } K2 = \{\}$
and $(\bigcup \{K. \exists x. (x, K) \in s\} = i)$
shows *s tagged_division_of i*
unfolding *tagged_division_of*
using *assms* **by** *fastforce*

lemma *tagged_division_ofD[dest]:*
assumes *s tagged_division_of i*
shows *finite s*
and $\bigwedge x K. (x, K) \in s \implies x \in K$
and $\bigwedge x K. (x, K) \in s \implies K \subseteq i$
and $\bigwedge x K. (x, K) \in s \implies \exists a b. K = \text{cbox } a \ b$
and $\bigwedge x1 K1 x2 K2. (x1, K1) \in s \implies (x2, K2) \in s \implies (x1, K1) \neq (x2, K2)$
 $\implies$
 $\text{interior } K1 \cap \text{interior } K2 = \{\}$
and $(\bigcup \{K. \exists x. (x, K) \in s\} = i)$
using *assms* **unfolding** *tagged_division_of* **by** *blast+*

lemma *division_of_tagged_division:*
assumes *s tagged_division_of i*
shows $(\text{snd } 's) \text{ division\_of } i$
proof *(rule division_ofI)*
note *assm = tagged_division_ofD[OF assms]*
show $\bigcup (\text{snd } 's) = i$ *finite (snd 's)*
using *assm* **by** *auto*
fix *k*
assume *k: k ∈ snd 's*
then obtain *xk where xk: (xk, k) ∈ s*
by *auto*
then show $k \subseteq i$ $k \neq \{\}$ $\exists a b. k = \text{cbox } a \ b$
using *assm* **by** *fastforce+*
fix *k'*
assume *k': k' ∈ snd 's* $k \neq k'$
from *this(1)* **obtain** *xk' where xk': (xk', k') ∈ s*
by *auto*
then show $\text{interior } k \cap \text{interior } k' = \{\}$
using *assm(5) k'(2) xk* **by** *blast*
qed

lemma *partial_division_of_tagged_division:*
assumes *s tagged_partial_division_of i*
shows $(\text{snd } 's) \text{ division\_of } \bigcup (\text{snd } 's)$
proof *(rule division_ofI)*

```

note assm = tagged_partial_division_ofD[OF assms]
show finite (snd ' s)  $\bigcup$  (snd ' s) =  $\bigcup$  (snd ' s)
  using assm by auto
fix k
assume k:  $k \in \textit{snd} ' s
then obtain xk where xk:  $(xk, k) \in s$ 
  by auto
then show  $k \neq \{\}$   $\exists a b. k = \textit{cbox} a b  $k \subseteq \bigcup (\textit{snd} ' s)$$$ 
```

using *assm* **by** *auto*

```

fix k'
assume k':  $k' \in \textit{snd} ' s  $k \neq k'$ 
from this(1) obtain xk' where xk':  $(xk', k') \in s$ 
  by auto
then show  $\textit{interior}$  k  $\cap$   $\textit{interior}$  k' =  $\{\}$ 
  using assm(5) k'(2) xk by auto
qed

lemma tagged_partial_division_subset:
  assumes s tagged_partial_division_of i
    and  $t \subseteq s$ 
  shows t tagged_partial_division_of i
  using assms finite_subset[OF assms(2)]
  unfolding tagged_partial_division_of_def
  by blast

lemma tag_in_interval:  $p$  tagged_division_of i  $\implies (x, k) \in p \implies x \in i$ 
  by auto

lemma tagged_division_of_empty:  $\{\}$  tagged_division_of  $\{\}$ 
  unfolding tagged_division_of by auto

lemma tagged_partial_division_of_trivial[simp]:  $p$  tagged_partial_division_of  $\{\}$ 
 $\longleftrightarrow p = \{\}$ 
  unfolding tagged_partial_division_of_def by auto

lemma tagged_division_of_trivial[simp]:  $p$  tagged_division_of  $\{\}$   $\longleftrightarrow p = \{\}$ 
  unfolding tagged_division_of by auto

lemma tagged_division_of_self:  $x \in \textit{cbox}$  a b  $\implies \{(x, \textit{cbox}$  a b)\} tagged_division_of
(cbox a b)
  by (rule tagged_division_ofI) auto

lemma tagged_division_of_self_real:  $x \in \{a \dots b::\textit{real}\} \implies \{(x, \{a \dots b\})\}$  tagged_division_of
 $\{a \dots b\}$ 
  unfolding box_real[symmetric]
  by (rule tagged_division_of_self)

lemma tagged_division_Un:
  assumes p1 tagged_division_of s1$ 
```

```

    and p2 tagged_division_of s2
    and interior s1 ∩ interior s2 = {}
  shows (p1 ∪ p2) tagged_division_of (s1 ∪ s2)
proof (rule tagged_division_ofI)
  note p1 = tagged_division_ofD[OF assms(1)]
  note p2 = tagged_division_ofD[OF assms(2)]
  show finite (p1 ∪ p2)
    using p1(1) p2(1) by auto
  show  $\bigcup \{k. \exists x. (x, k) \in p1 \cup p2\} = s1 \cup s2$ 
    using p1(6) p2(6) by blast
  fix x k
  assume xk:  $(x, k) \in p1 \cup p2$ 
  show  $x \in k \exists a b. k = \text{cbox } a b$ 
    using xk p1(2,4) p2(2,4) by auto
  show  $k \subseteq s1 \cup s2$ 
    using xk p1(3) p2(3) by blast
  fix x' k'
  assume xk':  $(x', k') \in p1 \cup p2$   $(x, k) \neq (x', k')$ 
  have *:  $\bigwedge a b. a \subseteq s1 \implies b \subseteq s2 \implies \text{interior } a \cap \text{interior } b = \{\}$ 
    using assms(3) interior_mono by blast
  show  $\text{interior } k \cap \text{interior } k' = \{\}$ 
    apply (cases  $(x, k) \in p1$ )
    apply (meson * UnE assms(1) assms(2) p1(5) tagged_division_ofD(3) xk'(1)
      xk'(2))
    by (metis * UnE assms(1) assms(2) inf_sup_aci(1) p2(5) tagged_division_ofD(3)
      xk xk'(1) xk'(2))
qed

```

lemma *tagged_division_Union*:

```

  assumes finite I
    and tag:  $\bigwedge i. i \in I \implies \text{pfn } i \text{ tagged\_division\_of } i$ 
    and disj:  $\bigwedge i1 i2. \llbracket i1 \in I; i2 \in I; i1 \neq i2 \rrbracket \implies \text{interior}(i1) \cap \text{interior}(i2) = \{\}$ 
  shows  $\bigcup (\text{pfn } ` I) \text{ tagged\_division\_of } (\bigcup I)$ 
proof (rule tagged_division_ofI)
  note assm = tagged_division_ofD[OF tag]
  show finite  $(\bigcup (\text{pfn } ` I))$ 
    using assms by auto
  have  $\bigcup \{k. \exists x. (x, k) \in \bigcup (\text{pfn } ` I)\} = \bigcup ((\lambda i. \bigcup \{k. \exists x. (x, k) \in \text{pfn } i\}) ` I)$ 
    by blast
  also have  $\dots = \bigcup I$ 
    using assm(6) by auto
  finally show  $\bigcup \{k. \exists x. (x, k) \in \bigcup (\text{pfn } ` I)\} = \bigcup I$  .
  fix x k
  assume xk:  $(x, k) \in \bigcup (\text{pfn } ` I)$ 
  then obtain i where  $i \in I$   $(x, k) \in \text{pfn } i$ 
    by auto
  show  $x \in k \exists a b. k = \text{cbox } a b \subseteq \bigcup I$ 
    using assm(2-4)[OF i] using i(1) by auto

```



```

fix x' k'
assume xk': (x', k') ∈ ⋃ (pfn ' I) (x, k) ≠ (x', k')
then obtain i' where i': i' ∈ I (x', k') ∈ pfn i'
  by auto
have *: ⋀ a b. i ≠ i' ⇒ a ⊆ i ⇒ b ⊆ i' ⇒ interior a ∩ interior b = {}
  using i(1) i'(1) disj interior_mono by blast
show interior k ∩ interior k' = {}
proof (cases i = i')
  case True then show ?thesis
    using assm(5) i' i xk'(2) by blast
  next
    case False then show ?thesis
      using * assm(3) i' i by auto
qed
qed

```

```

lemma tagged_partial_division_of_Union_self:
  assumes p tagged_partial_division_of s
  shows p tagged_division_of (⋃ (snd ' p))
  apply (rule tagged_division_ofI)
  using tagged_partial_division_ofD[OF assms]
  apply auto
  done

```

```

lemma tagged_division_of_union_self:
  assumes p tagged_division_of s
  shows p tagged_division_of (⋃ (snd ' p))
  apply (rule tagged_division_ofI)
  using tagged_division_ofD[OF assms]
  apply auto
  done

```

```

lemma tagged_division_Un_interval:
  fixes a :: 'a::euclidean_space
  assumes p1 tagged_division_of (cbox a b ∩ {x. x•k ≤ (c::real)})
    and p2 tagged_division_of (cbox a b ∩ {x. x•k ≥ c})
    and k: k ∈ Basis
  shows (p1 ∪ p2) tagged_division_of (cbox a b)
proof -
  have *: cbox a b = (cbox a b ∩ {x. x•k ≤ c}) ∪ (cbox a b ∩ {x. x•k ≥ c})
    by auto
  show ?thesis
    apply (subst *)
    apply (rule tagged_division_Un[OF assms(1-2)])
    unfolding interval_split[OF k] interior_cbox
    using k
    apply (auto simp add: box_def elim!: ballE[where x=k])
    done
qed

```

lemma *tagged_division_Un_interval_real*:

fixes $a :: \text{real}$
assumes $p1 \text{ tagged\_division\_of } (\{a .. b\} \cap \{x. x \cdot k \leq (c :: \text{real})\})$
and $p2 \text{ tagged\_division\_of } (\{a .. b\} \cap \{x. x \cdot k \geq c\})$
and $k: k \in \text{Basis}$
shows $(p1 \cup p2) \text{ tagged\_division\_of } \{a .. b\}$
using *assms*
unfolding *box_real[symmetric]*
by (*rule tagged_division_Un_interval*)

lemma *tagged_division_split_left_inj*:

assumes $d: d \text{ tagged\_division\_of } i$
and $\text{tags}: (x1, K1) \in d \ (x2, K2) \in d$
and $K1 \neq K2$
and $\text{eq}: K1 \cap \{x. x \cdot k \leq c\} = K2 \cap \{x. x \cdot k \leq c\}$
shows $\text{interior } (K1 \cap \{x. x \cdot k \leq c\}) = \{\}$
proof –
have $\text{interior } (K1 \cap K2) = \{\} \vee (x2, K2) = (x1, K1)$
using *tags d by (metis (no_types) interior_Int tagged_division_ofD(5))*
then show *?thesis*
using $\text{eq } \langle K1 \neq K2 \rangle$ **by** (*metis (no_types) inf_assoc inf_bot_left inf_left_idem interior_Int old.prod.inject*)
qed

lemma *tagged_division_split_right_inj*:

assumes $d: d \text{ tagged\_division\_of } i$
and $\text{tags}: (x1, K1) \in d \ (x2, K2) \in d$
and $K1 \neq K2$
and $\text{eq}: K1 \cap \{x. x \cdot k \geq c\} = K2 \cap \{x. x \cdot k \geq c\}$
shows $\text{interior } (K1 \cap \{x. x \cdot k \geq c\}) = \{\}$
proof –
have $\text{interior } (K1 \cap K2) = \{\} \vee (x2, K2) = (x1, K1)$
using *tags d by (metis (no_types) interior_Int tagged_division_ofD(5))*
then show *?thesis*
using $\text{eq } \langle K1 \neq K2 \rangle$ **by** (*metis (no_types) inf_assoc inf_bot_left inf_left_idem interior_Int old.prod.inject*)
qed

lemma (*in comm_monoid_set*) *over_tagged_division_lemma*:

assumes $p \text{ tagged\_division\_of } i$
and $\bigwedge u v. \text{box } u v = \{\} \implies d (\text{cbox } u v) = \mathbf{1}$
shows $F (\lambda(\_, k). d k) p = F d (\text{snd } p)$

proof –

have $*$: $(\lambda(\_, k). d k) = d \circ \text{snd}$
by (*simp add: fun_eq_iff*)
note $\text{assm} = \text{tagged\_division\_ofD}[OF \text{ assms}(1)]$
show *?thesis*
unfolding $*$

```

proof (rule reindex_nontrivial[symmetric])
  show finite p
    using assm by auto
  fix x y
  assume x ∈ p y ∈ p x ≠ y snd x = snd y
  obtain a b where ab: snd x = cbox a b
    using assm(4)[of fst x snd x] ⟨x ∈ p⟩ by auto
  have (fst x, snd y) ∈ p (fst x, snd y) ≠ y
    using ⟨x ∈ p⟩ ⟨x ≠ y⟩ ⟨snd x = snd y⟩ [symmetric] by auto
  with ⟨x ∈ p⟩ ⟨y ∈ p⟩ have interior (snd x) ∩ interior (snd y) = {}
    by (intro assm(5)[of fst x _ fst y]) auto
  then have box a b = {}
    unfolding ⟨snd x = snd y⟩ [symmetric] ab by auto
  then have d (cbox a b) = 1
    using assm(2)[of fst x snd x] ⟨x ∈ p⟩ ab[symmetric] by (intro assms(2)) auto
  then show d (snd x) = 1
    unfolding ab by auto
qed
qed

```

6.14.8 Functions closed on boxes: morphisms from boxes to monoids

This auxiliary structure is used to sum up over the elements of a division. Main theorem is *operative_division*. Instances for the monoid are *'a option*, *real*, and *bool*.

Using additivity of lifted function to encode definedness. **definition**

lift_option :: ('a ⇒ 'b ⇒ 'c) ⇒ 'a option ⇒ 'b option ⇒ 'c option

where

lift_option f a' b' = Option.bind a' (λa. Option.bind b' (λb. Some (f a b)))

lemma *lift_option_simps*[simp]:

lift_option f (Some a) (Some b) = Some (f a b)

lift_option f None b' = None

lift_option f a' None = None

by (auto simp: *lift_option_def*)

lemma *comm_monoid_lift_option*:

assumes *comm_monoid* f z

shows *comm_monoid* (*lift_option* f) (Some z)

proof –

from *assms* **interpret** *comm_monoid* f z .

show ?thesis

by standard (auto simp: *lift_option_def* *ac_simps* *split*: *bind_split*)

qed

lemma *comm_monoid_and*: *comm_monoid* HOL.conj True

by *standard auto*

lemma *comm_monoid_set_and*: *comm_monoid_set HOL.conj True*
 by (rule *comm_monoid_set.intro*) (fact *comm_monoid_and*)

Misc lemma *interval_real_split*:
 $\{a \dots b :: \text{real}\} \cap \{x. x \leq c\} = \{a \dots \min b c\}$
 $\{a \dots b\} \cap \{x. c \leq x\} = \{\max a c \dots b\}$
 apply (metis *Int_atLeastAtMostL1 atMost_def*)
 apply (metis *Int_atLeastAtMostL2 atLeast_def*)
 done

lemma *bgaugue_existence_lemma*: $(\forall x \in s. \exists d :: \text{real}. 0 < d \wedge q d x) \longleftrightarrow (\forall x. \exists d > 0. x \in s \longrightarrow q d x)$
 by (meson *zero_less_one*)

Division points definition *division_points* (*k*::('a::euclidean_space) set) *d* =
 $\{(j, x). j \in \text{Basis} \wedge (\text{interval\_lowerbound } k) \cdot j < x \wedge x < (\text{interval\_upperbound } k) \cdot j \wedge$
 $(\exists i \in d. (\text{interval\_lowerbound } i) \cdot j = x \vee (\text{interval\_upperbound } i) \cdot j = x)\}$

lemma *division_points_finite*:
 fixes *i* :: 'a::euclidean_space set
 assumes *d division_of i*
 shows *finite (division_points i d)*
proof –
 note *assm = division_ofD[OF assms]*
 let *?M* = $\lambda j. \{(j, x) | x. (\text{interval\_lowerbound } i) \cdot j < x \wedge x < (\text{interval\_upperbound } i) \cdot j \wedge$
 $(\exists i \in d. (\text{interval\_lowerbound } i) \cdot j = x \vee (\text{interval\_upperbound } i) \cdot j = x)\}$
 have *: *division_points i d* = $\bigcup (?M \text{ `Basis})$
 unfolding *division_points_def* by auto
 show *?thesis*
 unfolding * using *assm* by auto
 qed

lemma *division_points_subset*:
 fixes *a* :: 'a::euclidean_space
 assumes *d division_of (cbox a b)*
 and $\forall i \in \text{Basis}. a \cdot i < b \cdot i \quad a \cdot k < c < b \cdot k$
 and *k*: *k* ∈ *Basis*
 shows *division_points (cbox a b ∩ {x. x·k ≤ c}) {l ∩ {x. x·k ≤ c} | l. l ∈ d ∧ l ∩ {x. x·k ≤ c} ≠ {}}* ⊆
division_points (cbox a b) d (is *?t1*)
 and *division_points (cbox a b ∩ {x. x·k ≥ c}) {l ∩ {x. x·k ≥ c} | l. l ∈ d ∧ ¬(l ∩ {x. x·k ≥ c} = {})}* ⊆
division_points (cbox a b) d (is *?t2*)
proof –
 note *assm = division_ofD[OF assms(1)]*

```

have *:  $\forall i \in \text{Basis}. a \cdot i \leq b \cdot i$ 
 $\forall i \in \text{Basis}. a \cdot i \leq (\sum i \in \text{Basis}. (\text{if } i = k \text{ then } \min(b \cdot k) \ c \text{ else } b \cdot i) *_R i) \cdot i$ 
 $\forall i \in \text{Basis}. (\sum i \in \text{Basis}. (\text{if } i = k \text{ then } \max(a \cdot k) \ c \text{ else } a \cdot i) *_R i) \cdot i \leq b \cdot i$ 
 $\min(b \cdot k) \ c = c \ \max(a \cdot k) \ c = c$ 
using assms using less_imp_le by auto
have  $\exists i \in d. \text{interval\_lowerbound } i \cdot x = y \vee \text{interval\_upperbound } i \cdot x = y$ 
if  $a \cdot x < y \vee y < (\text{if } x = k \text{ then } c \text{ else } b \cdot x)$ 
 $\text{interval\_lowerbound } i \cdot x = y \vee \text{interval\_upperbound } i \cdot x = y$ 
 $i = l \cap \{x. x \cdot k \leq c\} \ l \in d \ l \cap \{x. x \cdot k \leq c\} \neq \{\}$ 
 $x \in \text{Basis}$  for  $i \ l \ x \ y$ 
proof -
obtain  $u \ v$  where  $l: l = \text{cbox } u \ v$ 
using  $\langle l \in d \rangle$  assms(1) by blast
have *:  $\forall i \in \text{Basis}. u \cdot i \leq (\sum i \in \text{Basis}. (\text{if } i = k \text{ then } \min(v \cdot k) \ c \text{ else } v \cdot i) *_R i) \cdot i$ 
using that(6) unfolding l interval_split[OF k] box_ne_empty that .
have **:  $\forall i \in \text{Basis}. u \cdot i \leq v \cdot i$ 
using l using that(6) unfolding box_ne_empty[symmetric] by auto
show ?thesis
apply (rule bexI[OF _  $\langle l \in d \rangle$ ])
using that(1-3,5)  $\langle x \in \text{Basis} \rangle$ 
unfolding l interval_bounds[OF **] interval_bounds[OF *] interval_split[OF
k] that
apply (auto split: if_split_asm)
done
qed
moreover have  $\bigwedge x \ y. \llbracket y < (\text{if } x = k \text{ then } c \text{ else } b \cdot x) \rrbracket \implies y < b \cdot x$ 
using  $\langle c < b \cdot k \rangle$  by (auto split: if_split_asm)
ultimately show ?t1
unfolding division_points_def interval_split[OF k, of a b]
unfolding interval_bounds[OF *(1)] interval_bounds[OF *(2)] interval_bounds[OF
*(3)]
unfolding * by force
have  $\bigwedge x \ y \ i \ l. (\text{if } x = k \text{ then } c \text{ else } a \cdot x) < y \implies a \cdot x < y$ 
using  $\langle a \cdot k < c \rangle$  by (auto split: if_split_asm)
moreover have  $\exists i \in d. \text{interval\_lowerbound } i \cdot x = y \vee$ 
 $\text{interval\_upperbound } i \cdot x = y$ 
if  $(\text{if } x = k \text{ then } c \text{ else } a \cdot x) < y \vee y < b \cdot x$ 
 $\text{interval\_lowerbound } i \cdot x = y \vee \text{interval\_upperbound } i \cdot x = y$ 
 $i = l \cap \{x. c \leq x \cdot k\} \ l \in d \ l \cap \{x. c \leq x \cdot k\} \neq \{\}$ 
 $x \in \text{Basis}$  for  $x \ y \ i \ l$ 
proof -
obtain  $u \ v$  where  $l: l = \text{cbox } u \ v$ 
using  $\langle l \in d \rangle$  assm(4) by blast
have *:  $\forall i \in \text{Basis}. (\sum i \in \text{Basis}. (\text{if } i = k \text{ then } \max(u \cdot k) \ c \text{ else } u \cdot i) *_R i) \cdot$ 
 $i \leq v \cdot i$ 
using that(6) unfolding l interval_split[OF k] box_ne_empty that .
have **:  $\forall i \in \text{Basis}. u \cdot i \leq v \cdot i$ 
using l using that(6) unfolding box_ne_empty[symmetric] by auto

```

```

show  $\exists i \in d. \text{interval\_lowerbound } i \cdot x = y \vee \text{interval\_upperbound } i \cdot x = y$ 
apply (rule beXI[OF  $\langle l \in d \rangle$ ])
using that(1-3,5)  $\langle x \in \text{Basis} \rangle$ 
unfolding l interval_bounds[OF **] interval_bounds[OF *] interval_split[OF
k] that
apply (auto split: if_split_asm)
done
qed
ultimately show ?t2
unfolding division_points_def interval_split[OF k, of a b]
unfolding interval_bounds[OF *(1)] interval_bounds[OF *(2)] interval_bounds[OF
*(3)]
unfolding *
by force
qed

lemma division_points_psubset:
fixes a :: 'a::euclidean_space
assumes d: d division_of (cbox a b)
and alth:  $\forall i \in \text{Basis}. a \cdot i < b \cdot i \quad a \cdot k < c \quad c < b \cdot k$ 
and l  $\in d$ 
and disj:  $\text{interval\_lowerbound } l \cdot k = c \vee \text{interval\_upperbound } l \cdot k = c$ 
and k: k  $\in \text{Basis}$ 
shows division_points (cbox a b  $\cap \{x. x \cdot k \leq c\}$ )  $\{l \cap \{x. x \cdot k \leq c\} \mid l. l \in d \wedge l$ 
 $\cap \{x. x \cdot k \leq c\} \neq \{\}\} \subset$ 
division_points (cbox a b) d (is ?D1  $\subset$  ?D)
and division_points (cbox a b  $\cap \{x. x \cdot k \geq c\}$ )  $\{l \cap \{x. x \cdot k \geq c\} \mid l. l \in d \wedge l \cap$ 
 $\{x. x \cdot k \geq c\} \neq \{\}\} \subset$ 
division_points (cbox a b) d (is ?D2  $\subset$  ?D)
proof -
have ab:  $\forall i \in \text{Basis}. a \cdot i \leq b \cdot i$ 
using alth by (auto intro!: less_imp_le)
obtain u v where l: l = cbox u v
using d  $\langle l \in d \rangle$  by blast
have uv:  $\forall i \in \text{Basis}. u \cdot i \leq v \cdot i \quad \forall i \in \text{Basis}. a \cdot i \leq u \cdot i \wedge v \cdot i \leq b \cdot i$ 
apply (metis assms(5) box_ne_empty(1) cbox_division_memE d l)
by (metis assms(5) box_ne_empty(1) cbox_division_memE d l subset_box(1))
have *:  $\text{interval\_upperbound } (cbox a b \cap \{x. x \cdot k \leq \text{interval\_upperbound } l \cdot k\})$ 
 $\cdot k = \text{interval\_upperbound } l \cdot k$ 
 $\text{interval\_upperbound } (cbox a b \cap \{x. x \cdot k \leq \text{interval\_lowerbound } l \cdot k\}) \cdot$ 
 $k = \text{interval\_lowerbound } l \cdot k$ 
unfolding l interval_split[OF k] interval_bounds[OF uv(1)]
using uv[rule_format, of k] ab k
by auto
have  $\exists x. x \in ?D - ?D1$ 
using assms(3-)
unfolding division_points_def interval_bounds[OF ab]
by (force simp add: *)
moreover have ?D1  $\subset$  ?D

```

```

    by (auto simp add: assms division_points_subset)
  ultimately show ?D1  $\subset$  ?D
    by blast
  have *: interval_lowerbound (cbox a b  $\cap$  {x. x  $\cdot$  k  $\geq$  interval_lowerbound l  $\cdot$  k})
 $\cdot$  k = interval_lowerbound l  $\cdot$  k
    interval_lowerbound (cbox a b  $\cap$  {x. x  $\cdot$  k  $\geq$  interval_upperbound l  $\cdot$  k})  $\cdot$  k =
interval_upperbound l  $\cdot$  k
    unfolding l interval_split[OF k] interval_bounds[OF uv(1)]
    using uv[rule_format, of k] ab k
    by auto
  have  $\exists x. x \in ?D - ?D2$ 
    using assms( $\exists$ -)
    unfolding division_points_def interval_bounds[OF ab]
    by (force simp add: *)
  moreover have ?D2  $\subseteq$  ?D
    by (auto simp add: assms division_points_subset)
  ultimately show ?D2  $\subset$  ?D
    by blast
qed

```

```

lemma division_split_left_inj:
  fixes S :: 'a::euclidean_space set
  assumes div:  $\mathcal{D}$  division_of S
    and eq:  $K1 \cap \{x::'a. x \cdot k \leq c\} = K2 \cap \{x. x \cdot k \leq c\}$ 
    and  $K1 \in \mathcal{D} \ K2 \in \mathcal{D} \ K1 \neq K2$ 
  shows interior ( $K1 \cap \{x. x \cdot k \leq c\}$ ) = {}
proof -
  have interior  $K2 \cap$  interior {a. a  $\cdot$  k  $\leq$  c} = interior  $K1 \cap$  interior {a. a  $\cdot$  k
 $\leq$  c}
    by (metis (no_types) eq interior_Int)
  moreover have  $\bigwedge A. \text{interior } A \cap \text{interior } K2 = \{\} \vee A = K2 \vee A \notin \mathcal{D}$ 
    by (meson div  $\langle K2 \in \mathcal{D} \rangle$  division_of_def)
  ultimately show ?thesis
    using  $\langle K1 \in \mathcal{D} \rangle \ \langle K1 \neq K2 \rangle$  by auto
qed

```

```

lemma division_split_right_inj:
  fixes S :: 'a::euclidean_space set
  assumes div:  $\mathcal{D}$  division_of S
    and eq:  $K1 \cap \{x::'a. x \cdot k \geq c\} = K2 \cap \{x. x \cdot k \geq c\}$ 
    and  $K1 \in \mathcal{D} \ K2 \in \mathcal{D} \ K1 \neq K2$ 
  shows interior ( $K1 \cap \{x. x \cdot k \geq c\}$ ) = {}
proof -
  have interior  $K2 \cap$  interior {a. a  $\cdot$  k  $\geq$  c} = interior  $K1 \cap$  interior {a. a  $\cdot$  k
 $\geq$  c}
    by (metis (no_types) eq interior_Int)
  moreover have  $\bigwedge A. \text{interior } A \cap \text{interior } K2 = \{\} \vee A = K2 \vee A \notin \mathcal{D}$ 
    by (meson div  $\langle K2 \in \mathcal{D} \rangle$  division_of_def)
  ultimately show ?thesis

```

using $\langle K1 \in \mathcal{D} \rangle \langle K1 \neq K2 \rangle$ by auto
qed

lemma *interval_doublesplit*:
fixes $a :: 'a::euclidean\_space$
assumes $k \in \text{Basis}$
shows $\text{cbox } a \ b \cap \{x. |x \cdot k - c| \leq (e::real)\} =$
 $\text{cbox } (\sum_{i \in \text{Basis}. (if } i = k \text{ then } \max(a \cdot k) (c - e) \text{ else } a \cdot i) *_{\mathbb{R}} i)$
 $(\sum_{i \in \text{Basis}. (if } i = k \text{ then } \min(b \cdot k) (c + e) \text{ else } b \cdot i) *_{\mathbb{R}} i)$
proof –
have *: $\bigwedge x \ c \ e::real. |x - c| \leq e \longleftrightarrow x \geq c - e \wedge x \leq c + e$
by auto
have **: $\bigwedge s \ P \ Q. s \cap \{x. P \ x \wedge Q \ x\} = (s \cap \{x. Q \ x\}) \cap \{x. P \ x\}$
by blast
show ?thesis
unfolding * ** *interval_split*[OF assms] **by** (rule refl)
qed

lemma *division_doublesplit*:
fixes $a :: 'a::euclidean\_space$
assumes $p \ \text{division\_of} \ (\text{cbox } a \ b)$
and $k: k \in \text{Basis}$
shows $(\lambda l. l \cap \{x. |x \cdot k - c| \leq e\}) \cdot \{\{l \in p. l \cap \{x. |x \cdot k - c| \leq e\} \neq \{\}\}$
 $\text{division\_of} \ (\text{cbox } a \ b \cap \{x. |x \cdot k - c| \leq e\})$
proof –
have *: $\bigwedge x \ c. |x - c| \leq e \longleftrightarrow x \geq c - e \wedge x \leq c + e$
by auto
have **: $\bigwedge p \ q \ p' \ q'. p \ \text{division\_of} \ q \implies p = p' \implies q = q' \implies p' \ \text{division\_of} \ q'$
by auto
note *division_split*(1)[OF assms, **where** $c=c+e$, *unfolded interval_split*[OF k]]
note *division_split*(2)[OF this, **where** $c=c-e$ **and** $k=k$, OF k]
then show ?thesis
apply (rule **)
subgoal
apply (*simp add: abs_diff_le_iff field_simps Collect_conj_eq setcompr_eq_image*
[symmetric] cong: image_cong_simp)
apply (rule equalityI)
apply blast
apply clarsimp
apply (rule_tac $x=xa \cap \{x. c + e \geq x \cdot k\}$ in exI)
apply auto
done
by (*simp add: interval_split k interval_doublesplit*)
qed

Operative **locale** *operative* = *comm_monoid_set* +
fixes $g :: 'b::euclidean\_space \text{ set} \Rightarrow 'a$
assumes *box_empty_imp*: $\bigwedge a \ b. \text{box } a \ b = \{\} \implies g (\text{cbox } a \ b) = 1$
and *Basis_imp*: $\bigwedge a \ b \ c \ k. k \in \text{Basis} \implies g (\text{cbox } a \ b) = g (\text{cbox } a \ b \cap \{x. x \cdot k$

$\leq c\}$) * $g (cbox\ a\ b \cap \{x. x \cdot k \geq c\})$
begin

lemma *empty [simp]*:

$g\ \{\} = 1$
proof –
have *: $cbox\ One\ (-One) = (\{\}::'b\ set)$
by (*auto simp: box_eq_empty inner_sum_left inner_Basis sum.If_cases ex_in_conv*)
moreover **have** $box\ One\ (-One) = (\{\}::'b\ set)$
using $box\_subset\_cbox[of\ One\ -One]$ **by** (*auto simp: **)
ultimately **show** *?thesis*
using $box\_empty\_imp[of\ One\ -One]$ **by** *simp*
qed

lemma *division*:

$F\ g\ d = g (cbox\ a\ b)$ **if** $d\ division\_of\ (cbox\ a\ b)$

proof –
define C **where** $[abs\_def]: C = card\ (division\_points\ (cbox\ a\ b)\ d)$
then **show** *?thesis*
using *that* **proof** (*induction C arbitrary: a b d rule: less_induct*)
case ($less\ a\ b\ d$)
show *?case*
proof *cases*
assume $box\ a\ b = \{\}$
{ fix k **assume** $k \in d$
then **obtain** $a'\ b'$ **where** $k: k = cbox\ a'\ b'$
using $division\_ofD(4)[OF\ less.prems]$ **by** *blast*
with $\langle k \in d \rangle\ division\_ofD(2)[OF\ less.prems]$ **have** $cbox\ a'\ b' \subseteq cbox\ a\ b$
by *auto*
then **have** $box\ a'\ b' \subseteq box\ a\ b$
unfolding $subset\_box$ **by** *auto*
then **have** $g\ k = 1$
using $box\_empty\_imp[of\ a'\ b']\ k$ **by** (*simp add: $\langle box\ a\ b = \{\} \rangle$*) **}**
then **show** $box\ a\ b = \{\} \implies F\ g\ d = g (cbox\ a\ b)$
by (*auto intro!: neutral simp: box_empty_imp*)
next
assume $box\ a\ b \neq \{\}$
then **have** $ab: \forall i \in Basis. a \cdot i < b \cdot i$ **and** $ab': \forall i \in Basis. a \cdot i \leq b \cdot i$
by (*auto simp: box_ne_empty*)
show $F\ g\ d = g (cbox\ a\ b)$
proof (*cases division_points (cbox a b) d = {}*)
case *True*
{ fix $u\ v$ **and** $j :: 'b$
assume $j: j \in Basis$ **and** $as: cbox\ u\ v \in d$
then **have** $cbox\ u\ v \neq \{\}$
using *less.prems* **by** *blast*
then **have** $uv: \forall i \in Basis. u \cdot i \leq v \cdot i$ $u \cdot j \leq v \cdot j$
using j **unfolding** box_ne_empty **by** *auto*
have *: $\bigwedge p\ r\ Q. \neg j \in Basis \vee p \vee r \vee (\forall x \in d. Q\ x) \implies p \vee r \vee Q (cbox$

```

u v)
  using as j by auto
  have (j, u·j) ∉ division_points (cbox a b) d
    (j, v·j) ∉ division_points (cbox a b) d using True by auto
  note this[unfolded de_Morgan_conj division_points_def mem_Collect_eq
split_conv interval_bounds[OF ab'] bex_simps]
  note *[OF this(1)] *[OF this(2)] note this[unfolded interval_bounds[OF
uv(1)]]]
  moreover
  have a·j ≤ u·j v·j ≤ b·j
    using division_ofD(2,2,3)[OF ⟨d division_of cbox a b⟩ as]
  apply (metis j subset_box(1) uv(1))
  by (metis ⟨cbox u v ⊆ cbox a b⟩ j subset_box(1) uv(1))
  ultimately have u·j = a·j ∧ v·j = a·j ∨ u·j = b·j ∧ v·j = b·j ∨ u·j =
a·j ∧ v·j = b·j
    unfolding not_less de_Morgan_disj using ab[rule_format, of j] uv(2)
j by force }
  then have d': ∀ i ∈ d. ∃ u v. i = cbox u v ∧
    (∀ j ∈ Basis. u·j = a·j ∧ v·j = a·j ∨ u·j = b·j ∧ v·j = b·j ∨ u·j = a·j ∧
v·j = b·j)
    unfolding forall_in_division[OF less.prem] by blast
  have (1/2) *R (a+b) ∈ cbox a b
    unfolding mem_box using ab by (auto simp: inner_simps)
  note this[unfolded division_ofD(6)[OF ⟨d division_of cbox a b⟩, symmetric]
Union_iff]
  then obtain i where i: i ∈ d (1 / 2) *R (a + b) ∈ i ..
  obtain u v where uv: i = cbox u v
    ∀ j ∈ Basis. u · j = a · j ∧ v · j = a · j ∨
      u · j = b · j ∧ v · j = b · j ∨
      u · j = a · j ∧ v · j = b · j

  using d' i(1) by auto
  have cbox a b ∈ d
  proof -
    have u = a v = b
      unfolding euclidean_eq_iff[where 'a='b]
    proof safe
      fix j :: 'b
      assume j: j ∈ Basis
      note i(2)[unfolded uv mem_box, rule_format, of j]
      then show u · j = a · j and v · j = b · j
        using uv(2)[rule_format, of j] j by (auto simp: inner_simps)
    qed
    then have i = cbox a b using uv by auto
    then show ?thesis using i by auto
  qed
  then have deq: d = insert (cbox a b) (d - {cbox a b})
    by auto
  have F g (d - {cbox a b}) = 1
  proof (intro neutral ballI)

```

```

fix x
assume x:  $x \in d - \{cbox\ a\ b\}$ 
then have  $x \in d$ 
  by auto note  $d'[rule\_format, OF\ this]$ 
then obtain u v where  $uv: x = cbox\ u\ v$ 
   $\forall j \in Basis. u \cdot j = a \cdot j \wedge v \cdot j = a \cdot j \vee$ 
   $u \cdot j = b \cdot j \wedge v \cdot j = b \cdot j \vee$ 
   $u \cdot j = a \cdot j \wedge v \cdot j = b \cdot j$ 

  by blast
have  $u \neq a \vee v \neq b$ 
  using  $x[unfolded\ uv]$  by auto
then obtain j where  $u \cdot j \neq a \cdot j \vee v \cdot j \neq b \cdot j$  and  $j: j \in Basis$ 
  unfolding euclidean_eq_iff[where 'a='b] by auto
then have  $u \cdot j = v \cdot j$ 
  using  $uv(2)[rule\_format, OF\ j]$  by auto
then have  $box\ u\ v = \{\}$ 
  using j unfolding box_eq_empty by (auto intro!: boxI[of _ j])
then show  $g\ x = 1$ 
  unfolding  $uv(1)$  by (rule box_empty_imp)
qed
then show  $F\ g\ d = g\ (cbox\ a\ b)$ 
  using division_ofD[OF less.prems]
  apply (subst deq)
  apply (subst insert)
  apply auto
done
next
case False
then have  $\exists x. x \in division\_points\ (cbox\ a\ b)\ d$ 
  by auto
then obtain k c
  where  $k \in Basis\ interval\_lowerbound\ (cbox\ a\ b) \cdot k < c$ 
   $c < interval\_upperbound\ (cbox\ a\ b) \cdot k$ 
   $\exists i \in d. interval\_lowerbound\ i \cdot k = c \vee interval\_upperbound\ i \cdot k = c$ 
  unfolding division_points_def by auto
then obtain j where  $a \cdot k < c < b \cdot k$ 
  and  $j \in d$  and  $j: interval\_lowerbound\ j \cdot k = c \vee interval\_upperbound\ j \cdot k = c$ 
  by (metis division_of_trivial empty_iff interval_bounds' less.prems)
let ?lec =  $\{x. x \cdot k \leq c\}$  let ?gec =  $\{x. x \cdot k \geq c\}$ 
define d1 where  $d1 = \{l \cap ?lec \mid l. l \in d \wedge l \cap ?lec \neq \{\}\}$ 
define d2 where  $d2 = \{l \cap ?gec \mid l. l \in d \wedge l \cap ?gec \neq \{\}\}$ 
define cb where  $cb = (\sum i \in Basis. (if\ i = k\ then\ c\ else\ b \cdot i) *_{\mathbb{R}} i)$ 
define ca where  $ca = (\sum i \in Basis. (if\ i = k\ then\ c\ else\ a \cdot i) *_{\mathbb{R}} i)$ 
have division_points (cbox a b  $\cap$  ?lec)  $\{l \cap ?lec \mid l. l \in d \wedge l \cap ?lec \neq \{\}\}$ 
   $\subset division\_points\ (cbox\ a\ b)\ d$ 
  by (rule division_points_psubset[OF  $\langle d\ division\_of\ cbox\ a\ b \rangle\ ab\ \langle a \cdot k < c \rangle\ \langle c < b \cdot k \rangle\ \langle j \in d \rangle\ j\ \langle k \in Basis \rangle$ ])
with division_points_finite[OF  $\langle d\ division\_of\ cbox\ a\ b \rangle$ ]

```

```

have card
  (division_points (cbox a b  $\cap$  ?lec) { $l \cap ?lec \mid l. l \in d \wedge l \cap ?lec \neq \{\}$ })
    < card (division_points (cbox a b) d)
    by (rule psubset_card_mono)
moreover have division_points (cbox a b  $\cap$  { $x. c \leq x \cdot k$ }) { $l \cap \{x. c \leq x \cdot k\} \mid l. l \in d \wedge l \cap \{x. c \leq x \cdot k\} \neq \{\}$ }
   $\subset$  division_points (cbox a b) d
    by (rule division_points_psubset[OF  $\langle d \text{ division\_of cbox a b} \rangle$  ab  $\langle a \cdot k < c \rangle \langle c < b \cdot k \rangle \langle j \in d \rangle j \langle k \in \text{Basis} \rangle$ ])
with division_points_finite[OF  $\langle d \text{ division\_of cbox a b} \rangle$ ]
have card (division_points (cbox a b  $\cap$  ?gec) { $l \cap ?gec \mid l. l \in d \wedge l \cap ?gec \neq \{\}$ })
  < card (division_points (cbox a b) d)
    by (rule psubset_card_mono)
ultimately have *:  $F \ g \ d1 = g \ (cbox \ a \ b \cap ?lec) \ F \ g \ d2 = g \ (cbox \ a \ b \cap ?gec)$ 
  unfolding interval_split[OF  $\langle k \in \text{Basis} \rangle$ ]
  apply (rule_tac[!] less.hyps)
  using division_split[OF  $\langle d \text{ division\_of cbox a b} \rangle$ , where  $k=k$  and  $c=c$ ]
 $\langle k \in \text{Basis} \rangle$ 
  by (simp_all add: interval_split d1_def d2_def division_points_finite[OF  $\langle d \text{ division\_of cbox a b} \rangle$ ])
have f $k\_le$ :  $g \ (l \cap ?lec) = 1$ 
  if  $l \in d \ y \in d \ l \cap ?lec = y \cap ?lec \ l \neq y$  for  $l \ y$ 
proof –
  obtain  $u \ v$  where  $leq: l = cbox \ u \ v$ 
    using  $\langle l \in d \rangle$  less.premis by auto
  have interior (cbox  $u \ v \cap ?lec$ ) = {}
    using that division_split_left_inj leq less.premis by blast
  then show ?thesis
    unfolding leq_interval_split [OF  $\langle k \in \text{Basis} \rangle$ ]
    by (auto intro: box_empty_imp)
qed
have f $k\_ge$ :  $g \ (l \cap \{x. x \cdot k \geq c\}) = 1$ 
  if  $l \in d \ y \in d \ l \cap ?gec = y \cap ?gec \ l \neq y$  for  $l \ y$ 
proof –
  obtain  $u \ v$  where  $leq: l = cbox \ u \ v$ 
    using  $\langle l \in d \rangle$  less.premis by auto
  have interior (cbox  $u \ v \cap ?gec$ ) = {}
    using that division_split_right_inj leq less.premis by blast
  then show ?thesis
    unfolding leq_interval_split[OF  $\langle k \in \text{Basis} \rangle$ ]
    by (auto intro: box_empty_imp)
qed
have d1_alt:  $d1 = (\lambda l. l \cap ?lec) \cdot \{l \in d. l \cap ?lec \neq \{\}$ 
  using d1_def by auto
have d2_alt:  $d2 = (\lambda l. l \cap ?gec) \cdot \{l \in d. l \cap ?gec \neq \{\}$ 
  using d2_def by auto
have  $g \ (cbox \ a \ b) = F \ g \ d1 * F \ g \ d2$  (is  $\_ = ?prev$ )

```

```

    unfolding * using ⟨k ∈ Basis⟩
    by (auto dest: Basis_imp)
  also have  $F\ g\ d1 = F\ (\lambda l. g\ (l \cap ?lec))\ d$ 
    unfolding d1_alt using division_of_finite[OF less.prem] fvk_le
    by (subst reindex_nontrivial) (auto intro!: mono_neutral_cong_left)
  also have  $F\ g\ d2 = F\ (\lambda l. g\ (l \cap ?gec))\ d$ 
    unfolding d2_alt using division_of_finite[OF less.prem] fvk_ge
    by (subst reindex_nontrivial) (auto intro!: mono_neutral_cong_left)
  also have *:  $\forall x \in d. g\ x = g\ (x \cap ?lec) * g\ (x \cap ?gec)$ 
    unfolding forall_in_division[OF ⟨d division_of cbox a b⟩]
    using ⟨k ∈ Basis⟩
    by (auto dest: Basis_imp)
  have  $F\ (\lambda l. g\ (l \cap ?lec))\ d * F\ (\lambda l. g\ (l \cap ?gec))\ d = F\ g\ d$ 
    using * by (simp add: distrib)
  finally show ?thesis by auto
qed
qed
qed
qed

proposition tagged_division:
  assumes  $d\ \text{tagged\_division\_of}\ (cbox\ a\ b)$ 
  shows  $F\ (\lambda(\_, l). g\ l)\ d = g\ (cbox\ a\ b)$ 
proof -
  have  $F\ (\lambda(\_, k). g\ k)\ d = F\ g\ (snd\ 'd)$ 
    using assms box_empty_imp by (rule over_tagged_division_lemma)
  then show ?thesis
    unfolding assms [THEN division_of_tagged_division, THEN division] .
qed

end

locale operative_real = comm_monoid_set +
  fixes  $g :: real\ set \Rightarrow 'a$ 
  assumes neutral:  $b \leq a \implies g\ \{a..b\} = 1$ 
  assumes coalesce_less:  $a < c \implies c < b \implies g\ \{a..c\} * g\ \{c..b\} = g\ \{a..b\}$ 
begin

sublocale operative where  $g = g$ 
  rewrites box = (greaterThanLessThan :: real  $\Rightarrow$  _)
  and cbox = (atLeastAtMost :: real  $\Rightarrow$  _)
  and  $\bigwedge x::real. x \in Basis \longleftrightarrow x = 1$ 
proof -
  show operative f z g
  proof
    show  $g\ (cbox\ a\ b) = 1$  if  $box\ a\ b = \{\}$  for  $a\ b$ 
      using that by (simp add: neutral)
    show  $g\ (cbox\ a\ b) = g\ (cbox\ a\ b \cap \{x. x \cdot k \leq c\}) * g\ (cbox\ a\ b \cap \{x. c \leq x \cdot k\})$ 

```

```

    if  $k \in \text{Basis}$  for  $a\ b\ c\ k$ 
  proof -
    from that have [simp]:  $k = 1$ 
    by simp
    from neutral [of 0 1] neutral [of a a for a] coalesce_less
  have [simp]:  $g\ \{\} = 1 \wedge a. g\ \{a\} = 1$ 
     $\wedge a\ b\ c. a < c \implies c < b \implies g\ \{a..c\} * g\ \{c..b\} = g\ \{a..b\}$ 
  by auto
    have  $g\ \{a..b\} = g\ \{a..min\ b\ c\} * g\ \{max\ a\ c..b\}$ 
  by (auto simp: min_def max_def le_less)
    then show  $g\ (cbox\ a\ b) = g\ (cbox\ a\ b \cap \{x. x \cdot k \leq c\}) * g\ (cbox\ a\ b \cap \{x.$ 
 $c \leq x \cdot k\})$ 
    by (simp add: atMost_def [symmetric] atLeast_def [symmetric])
qed
qed
show  $box = (greaterThanLessThan :: real \Rightarrow \_)$ 
and  $cbox = (atLeastAtMost :: real \Rightarrow \_)$ 
and  $\wedge x::real. x \in \text{Basis} \longleftrightarrow x = 1$ 
by (simp_all add: fun_eq_iff)
qed

lemma coalesce_less_eq:
   $g\ \{a..c\} * g\ \{c..b\} = g\ \{a..b\}$  if  $a \leq c \wedge c \leq b$ 
proof (cases  $c = a \vee c = b$ )
  case False
  with that have  $a < c < b$ 
  by auto
  then show ?thesis
  by (rule coalesce_less)
next
  case True
  with that box_empty_imp [of a a] box_empty_imp [of b b] show ?thesis
  by safe simp_all
qed

end

lemma operative_realI:
  operative_real  $f\ z\ g$  if operative  $f\ z\ g$ 
proof -
  interpret operative  $f\ z\ g$ 
  using that .
  show ?thesis
proof
  show  $g\ \{a..b\} = z$  if  $b \leq a$  for  $a\ b$ 
  using that box_empty_imp by simp
  show  $f\ (g\ \{a..c\})\ (g\ \{c..b\}) = g\ \{a..b\}$  if  $a < c < b$  for  $a\ b\ c$ 
  using that
  using Basis_imp [of 1 a b c]

```

```

    by (simp_all add: atMost_def [symmetric] atLeast_def [symmetric] max_def
min_def)
qed
qed

```

6.14.9 Special case of additivity we need for the FTC

```

lemma additive_tagged_division_1:
  fixes f :: real  $\Rightarrow$  'a::real_normed_vector
  assumes a  $\leq$  b
  and p tagged_division_of {a..b}
  shows sum ( $\lambda(x,k). f(\text{Sup } k) - f(\text{Inf } k)$ ) p = f b - f a
proof -
  let ?f = ( $\lambda k::(\text{real}) \text{ set}. \text{if } k = \{\} \text{ then } 0 \text{ else } f(\text{interval\_upperbound } k) -$ 
 $f(\text{interval\_lowerbound } k)$ )
  interpret operative_real_plus 0 ?f
  rewrites comm_monoid_set.F (+) 0 = sum
  by standard[1] (auto simp add: sum_def)
  have p_td: p tagged_division_of cbox a b
  using assms(2) box_real(2) by presburger
  have **: cbox a b  $\neq \{\}$ 
  using assms(1) by auto
  then have f b - f a = ( $\sum (x, l) \in p. \text{if } l = \{\} \text{ then } 0 \text{ else } f(\text{interval\_upperbound } l) -$ 
 $f(\text{interval\_lowerbound } l)$ )
  proof -
    have (if cbox a b =  $\{\}$  then 0 else f (interval_upperbound (cbox a b)) - f
(interval_lowerbound (cbox a b))) = f b - f a
    using assms by auto
    then show ?thesis
    using p_td assms by (simp add: tagged_division)
  qed
  then show ?thesis
  using assms by (auto intro!: sum.cong)
qed

```

6.14.10 Fine-ness of a partition w.r.t. a gauge

```

definition fine (infixr fine 46)
  where d fine s  $\longleftrightarrow (\forall (x,k) \in s. k \subseteq d x)$ 

```

```

lemma fineI:
  assumes  $\bigwedge x k. (x, k) \in s \implies k \subseteq d x$ 
  shows d fine s
  using assms unfolding fine_def by auto

```

```

lemma fineD[dest]:
  assumes d fine s
  shows  $\bigwedge x k. (x,k) \in s \implies k \subseteq d x$ 
  using assms unfolding fine_def by auto

```

1764

lemma *fine_Int*: $(\lambda x. d1\ x \cap d2\ x)\ fine\ p \longleftrightarrow d1\ fine\ p \wedge d2\ fine\ p$
unfolding *fine_def* **by** *auto*

lemma *fine_Inter*:
 $(\lambda x. \bigcap \{f\ d\ x \mid d. d \in s\})\ fine\ p \longleftrightarrow (\forall d \in s. (f\ d)\ fine\ p)$
unfolding *fine_def* **by** *blast*

lemma *fine_Un*: $d\ fine\ p1 \implies d\ fine\ p2 \implies d\ fine\ (p1 \cup p2)$
unfolding *fine_def* **by** *blast*

lemma *fine_Union*: $(\bigwedge p. p \in ps \implies d\ fine\ p) \implies d\ fine\ (\bigcup ps)$
unfolding *fine_def* **by** *auto*

lemma *fine_subset*: $p \subseteq q \implies d\ fine\ q \implies d\ fine\ p$
unfolding *fine_def* **by** *blast*

6.14.11 Some basic combining lemmas

lemma *tagged_division_Union_exists*:
assumes *finite I*
and $\forall i \in I. \exists p. p\ tagged\_division\_of\ i \wedge d\ fine\ p$
and $\forall i1 \in I. \forall i2 \in I. i1 \neq i2 \longrightarrow interior\ i1 \cap interior\ i2 = \{\}$
and $\bigcup I = i$
obtains *p* **where** $p\ tagged\_division\_of\ i$ **and** $d\ fine\ p$
proof –
obtain *pfn* **where** *pfn*:
 $\bigwedge x. x \in I \implies pfn\ x\ tagged\_division\_of\ x$
 $\bigwedge x. x \in I \implies d\ fine\ pfn\ x$
using *bchoice[OF assms(2)]* **by** *auto*
show *thesis*
apply $(rule\_tac\ p = \bigcup (pfn\ ` I)\ in\ that)$
using *assms(1) assms(3) assms(4) pfn(1) tagged_division_Union* **apply** *force*
by $(metis\ (mono\_tags,\ lifting)\ fine\_Union\ imageE\ pfn(2))$
qed

6.14.12 The set we're concerned with must be closed

lemma *division_of_closed*:
fixes *i* :: *'n::euclidean_space set*
shows $s\ division\_of\ i \implies closed\ i$
unfolding *division_of_def* **by** *fastforce*

6.14.13 General bisection principle for intervals; might be useful elsewhere

lemma *interval_bisection_step*:
fixes *type* :: *'a::euclidean_space*
assumes *emp: P {}*
and *Un*: $\bigwedge S\ T. \llbracket P\ S; P\ T; interior(S) \cap interior(T) = \{\} \rrbracket \implies P\ (S \cup T)$
and *non*: $\neg P\ (cbox\ a\ (b::'a))$


```

obtains  $c\ d$  where  $\neg P\ (cbox\ c\ d)$ 
  and  $\bigwedge i. i \in Basis \implies a \cdot i \leq c \cdot i \wedge c \cdot i \leq d \cdot i \wedge d \cdot i \leq b \cdot i \wedge 2 * (d \cdot i - c \cdot i) \leq$ 
 $b \cdot i - a \cdot i$ 
proof -
  have  $cbox\ a\ b \neq \{\}$ 
  using emp non by metis
  then have  $ab: \bigwedge i. i \in Basis \implies a \cdot i \leq b \cdot i$ 
  by (force simp: mem_box)
  have  $UN\_cases: \llbracket finite\ \mathcal{F};$ 
 $\bigwedge S. S \in \mathcal{F} \implies P\ S;$ 
 $\bigwedge S. S \in \mathcal{F} \implies \exists a\ b. S = cbox\ a\ b;$ 
 $\bigwedge S\ T. S \in \mathcal{F} \implies T \in \mathcal{F} \implies S \neq T \implies interior\ S \cap interior\ T = \{\} \rrbracket \implies$ 
 $P\ (\bigcup \mathcal{F})$  for  $\mathcal{F}$ 
  proof (induct  $\mathcal{F}$  rule: finite_induct)
    case empty show ?case
    using emp by auto
  next
    case (insert  $x\ f$ )
    then show ?case
    unfolding Union_insert by (metis Int_interior_Union_intervals Un insert_iff open_interior)
  qed
  let  $?ab = \lambda i. (a \cdot i + b \cdot i) / 2$ 
  let  $?A = \{cbox\ c\ d \mid c\ d :: 'a. \forall i \in Basis. (c \cdot i = a \cdot i) \wedge (d \cdot i = ?ab\ i) \vee$ 
 $(c \cdot i = ?ab\ i) \wedge (d \cdot i = b \cdot i)\}$ 
  have  $P\ (\bigcup ?A)$ 
  if  $\bigwedge c\ d. \forall i \in Basis. a \cdot i \leq c \cdot i \wedge c \cdot i \leq d \cdot i \wedge d \cdot i \leq b \cdot i \wedge 2 * (d \cdot i - c \cdot i) \leq b \cdot i$ 
 $- a \cdot i \implies P\ (cbox\ c\ d)$ 
  proof (rule UN_cases)
    let  $?B = (\lambda S. cbox\ (\sum i \in Basis. (if\ i \in S\ then\ a \cdot i\ else\ ?ab\ i) *_{\mathbb{R}} i :: 'a)$ 
 $(\sum i \in Basis. (if\ i \in S\ then\ ?ab\ i\ else\ b \cdot i) *_{\mathbb{R}} i))\ '\{s. s \subseteq Basis\}$ 
    have  $?A \subseteq ?B$ 
  proof
    fix  $x$ 
    assume  $x \in ?A$ 
    then obtain  $c\ d$ 
    where  $x: x = cbox\ c\ d$ 
 $\bigwedge i. i \in Basis \implies$ 
 $c \cdot i = a \cdot i \wedge d \cdot i = ?ab\ i \vee c \cdot i = ?ab\ i \wedge d \cdot i = b \cdot i$ 
    by blast
    have  $c = (\sum i \in Basis. (if\ c \cdot i = a \cdot i\ then\ a \cdot i\ else\ ?ab\ i) *_{\mathbb{R}} i)$ 
 $d = (\sum i \in Basis. (if\ c \cdot i = a \cdot i\ then\ ?ab\ i\ else\ b \cdot i) *_{\mathbb{R}} i)$ 
    using  $x(2)\ ab$  by (fastforce simp add: euclidean_eq_iff[where 'a='a]) +
    then show  $x \in ?B$ 
    unfolding  $x$  by (rule_tac  $x = \{i. i \in Basis \wedge c \cdot i = a \cdot i\}$  in image_eqI) auto
  qed
  then show finite ?A
  by (rule finite_subset) auto
next

```

```

fix S
assume S ∈ ?A
then obtain c d
  where s: S = cbox c d
     $\bigwedge i. i \in \text{Basis} \implies c \cdot i = a \cdot i \wedge d \cdot i = ?ab\ i \vee c \cdot i = ?ab\ i \wedge d \cdot i$ 
= b · i
  by blast
show P S
  unfolding s using ab s(2) by (fastforce intro!: that)
show  $\exists a\ b. S = \text{cbox } a\ b$ 
  unfolding s by auto
fix T
assume T ∈ ?A
then obtain e f where t:
  T = cbox e f
   $\bigwedge i. i \in \text{Basis} \implies e \cdot i = a \cdot i \wedge f \cdot i = ?ab\ i \vee e \cdot i = ?ab\ i \wedge f \cdot i = b \cdot i$ 
  by blast
assume S ≠ T
then have  $\neg (c = e \wedge d = f)$ 
  unfolding s t by auto
then obtain i where  $c \cdot i \neq e \cdot i \vee d \cdot i \neq f \cdot i$  and i': i ∈ Basis
  unfolding euclidean_eq_iff[where 'a='a'] by auto
then have i:  $c \cdot i \neq e \cdot i \vee d \cdot i \neq f \cdot i$ 
  using s(2) t(2) apply fastforce
  using t(2)[OF i']  $\langle c \cdot i \neq e \cdot i \vee d \cdot i \neq f \cdot i \rangle i'$  s(2) t(2) by fastforce
have *:  $\bigwedge s\ t. (\bigwedge a. a \in s \implies a \in t \implies \text{False}) \implies s \cap t = \{\}$ 
  by auto
show interior S ∩ interior T = {}
  unfolding s t interior_cbox
proof (rule *)
  fix x
  assume x ∈ box c d x ∈ box e f
  then have x:  $c \cdot i < d \cdot i \wedge e \cdot i < f \cdot i \wedge c \cdot i < f \cdot i \wedge e \cdot i < d \cdot i$ 
  unfolding mem_box using i' by force+
  show False using s(2)[OF i'] t(2)[OF i'] and i x
  by auto
qed
qed
also have  $\bigcup ?A = \text{cbox } a\ b$ 
proof (rule set_eqI, rule)
  fix x
  assume x ∈  $\bigcup ?A$ 
  then obtain c d where x:
    x ∈ cbox c d
     $\bigwedge i. i \in \text{Basis} \implies c \cdot i = a \cdot i \wedge d \cdot i = ?ab\ i \vee c \cdot i = ?ab\ i \wedge d \cdot i = b \cdot i$ 
    by blast
  then show x ∈ cbox a b
    unfolding mem_box by force
next

```

```

fix x
assume x:  $x \in \text{cbox } a \ b$ 
then have  $\forall i \in \text{Basis}. \exists c \ d. (c = a \cdot i \wedge d = ?ab \ i \vee c = ?ab \ i \wedge d = b \cdot i) \wedge$ 
 $c \leq x \cdot i \wedge x \cdot i \leq d$ 
  (is  $\forall i \in \text{Basis}. \exists c \ d. ?P \ i \ c \ d$ )
  unfolding mem_box by (metis linear)
then obtain  $\alpha \ \beta$  where  $\forall i \in \text{Basis}. (\alpha \cdot i = a \cdot i \wedge \beta \cdot i = ?ab \ i \vee$ 
 $\alpha \cdot i = ?ab \ i \wedge \beta \cdot i = b \cdot i) \wedge \alpha \cdot i \leq x \cdot i \wedge x \cdot i \leq \beta \cdot i$ 
  by (auto simp: choice_Basis_iff)
then show  $x \in \bigcup ?A$ 
  by (force simp add: mem_box)
qed
finally show thesis
  by (metis (no_types, lifting) assms(3) that)
qed

```

lemma *interval_bisection*:

```

fixes type :: 'a::euclidean_space
assumes  $P \ \{\}$ 
  and  $Un: \bigwedge S \ T. \llbracket P \ S; P \ T; \text{interior}(S) \cap \text{interior}(T) = \{\} \rrbracket \implies P \ (S \cup T)$ 
  and  $\neg P \ (\text{cbox } a \ (b::'a))$ 
obtains  $x$  where  $x \in \text{cbox } a \ b$ 
  and  $\forall e > 0. \exists c \ d. x \in \text{cbox } c \ d \wedge \text{cbox } c \ d \subseteq \text{ball } x \ e \wedge \text{cbox } c \ d \subseteq \text{cbox } a \ b \wedge$ 
 $\neg P \ (\text{cbox } c \ d)$ 
proof -
  have  $\forall x. \exists y. \neg P \ (\text{cbox } (\text{fst } x) \ (\text{snd } x)) \longrightarrow (\neg P \ (\text{cbox } (\text{fst } y) \ (\text{snd } y)) \wedge$ 
 $(\forall i \in \text{Basis}. \text{fst } x \cdot i \leq \text{fst } y \cdot i \wedge \text{fst } y \cdot i \leq \text{snd } y \cdot i \wedge \text{snd } y \cdot i \leq \text{snd } x \cdot i \wedge$ 
 $2 * (\text{snd } y \cdot i - \text{fst } y \cdot i) \leq \text{snd } x \cdot i - \text{fst } x \cdot i))$  (is  $\forall x. ?P \ x$ )
  proof
    show  $?P \ x$  for  $x$ 
    proof (cases  $P \ (\text{cbox } (\text{fst } x) \ (\text{snd } x))$ )
      case True
        then show  $?thesis$  by auto
    next
      case False
        obtain  $c \ d$  where  $\neg P \ (\text{cbox } c \ d)$ 
         $\bigwedge i. i \in \text{Basis} \implies$ 
           $\text{fst } x \cdot i \leq c \cdot i \wedge$ 
           $c \cdot i \leq d \cdot i \wedge$ 
           $d \cdot i \leq \text{snd } x \cdot i \wedge$ 
           $2 * (d \cdot i - c \cdot i) \leq \text{snd } x \cdot i - \text{fst } x \cdot i$ 
        by (blast intro: interval_bisection_step[of P, OF assms(1-2) False])
        then show  $?thesis$ 
        by (rule_tac x=(c,d) in exI) auto
    qed
  qed
then obtain  $f$  where  $f:$ 
 $\forall x.$ 
 $\neg P \ (\text{cbox } (\text{fst } x) \ (\text{snd } x)) \longrightarrow$ 

```

```

    ¬ P (cbox (fst (f x)) (snd (f x))) ∧
    (∀ i ∈ Basis.
      fst x · i ≤ fst (f x) · i ∧
      fst (f x) · i ≤ snd (f x) · i ∧
      snd (f x) · i ≤ snd x · i ∧
      2 * (snd (f x) · i - fst (f x) · i) ≤ snd x · i - fst x · i) by metis
  define AB A B where ab_def: AB n = (f  $\frown$  n) (a,b) A n = fst(AB n) B n =
  snd(AB n) for n
  have [simp]: A 0 = a B 0 = b and ABRAW:  $\bigwedge n. \neg P (cbox (A(Suc n)) (B(Suc n))) \wedge$ 
  ( $\forall i \in Basis. A(n) \cdot i \leq A(Suc n) \cdot i \wedge A(Suc n) \cdot i \leq B(Suc n) \cdot i \wedge B(Suc n) \cdot i \leq$ 
   $B(n) \cdot i \wedge$ 
   $2 * (B(Suc n) \cdot i - A(Suc n) \cdot i) \leq B(n) \cdot i - A(n) \cdot i$ ) (is  $\bigwedge n. ?P n$ )
  proof -
  show A 0 = a B 0 = b
  unfolding ab_def by auto
  note S = ab_def funpow.simps o_def id_apply
  show ?P n for n
  proof (induct n)
  case 0
  then show ?case
  unfolding S using  $\langle \neg P (cbox a b) \rangle f$  by auto
  next
  case (Suc n)
  show ?case
  unfolding S
  apply (rule f[rule_format])
  using Suc
  unfolding S
  apply auto
  done
  qed
  qed
  then have AB:  $A(n) \cdot i \leq A(Suc n) \cdot i \wedge A(Suc n) \cdot i \leq B(Suc n) \cdot i$ 
   $B(Suc n) \cdot i \leq B(n) \cdot i \wedge 2 * (B(Suc n) \cdot i - A(Suc n) \cdot i) \leq B(n) \cdot i -$ 
   $A(n) \cdot i$ 
  if i ∈ Basis for i n
  using that by blast+
  have notPAB:  $\neg P (cbox (A(Suc n)) (B(Suc n)))$  for n
  using ABRAW by blast
  have interv:  $\exists n. \forall x \in cbox (A n) (B n). \forall y \in cbox (A n) (B n). dist x y < e$ 
  if e: 0 < e for e
  proof -
  obtain n where n:  $(\sum i \in Basis. b \cdot i - a \cdot i) / e < 2 \frown n$ 
  using real_arch_pow[of 2 (sum ( $\lambda i. b \cdot i - a \cdot i$ ) Basis) / e] by auto
  show ?thesis
  proof (rule exI [where x=n], clarify)
  fix x y
  assume xy:  $x \in cbox (A n) (B n) \wedge y \in cbox (A n) (B n)$ 

```

```

have  $\text{dist } x \ y \leq \text{sum } (\lambda i. |(x - y) \cdot i|) \ \text{Basis}$ 
  unfolding  $\text{dist\_norm}$  by (rule  $\text{norm\_le\_l1}$ )
also have  $\dots \leq \text{sum } (\lambda i. B \ n \cdot i - A \ n \cdot i) \ \text{Basis}$ 
proof (rule  $\text{sum\_mono}$ )
  fix  $i :: 'a$ 
  assume  $i: i \in \text{Basis}$ 
  show  $|(x - y) \cdot i| \leq B \ n \cdot i - A \ n \cdot i$ 
    using  $xy[\text{unfolded mem\_box}, \text{THEN bspec}, \text{OF } i]$ 
    by (auto simp:  $\text{inner\_diff\_left}$ )
qed
also have  $\dots \leq \text{sum } (\lambda i. b \cdot i - a \cdot i) \ \text{Basis} / 2^n$ 
  unfolding  $\text{sum\_divide\_distrib}$ 
proof (rule  $\text{sum\_mono}$ )
  show  $B \ n \cdot i - A \ n \cdot i \leq (b \cdot i - a \cdot i) / 2^n$  if  $i: i \in \text{Basis}$  for  $i$ 
  proof (induct  $n$ )
    case 0
    then show ?case
      unfolding  $AB$  by auto
  next
    case (Suc  $n$ )
    have  $B \ (\text{Suc } n) \cdot i - A \ (\text{Suc } n) \cdot i \leq (B \ n \cdot i - A \ n \cdot i) / 2$ 
      using  $AB(3)$  that  $AB(4)[\text{of } i \ n]$  using  $i$  by auto
    also have  $\dots \leq (b \cdot i - a \cdot i) / 2^{\text{Suc } n}$ 
      using  $\text{Suc}$  by (auto simp add:  $\text{field\_simps}$ )
    finally show ?case .
  qed
qed
also have  $\dots < e$ 
  using  $n$  using  $e$  by (auto simp add:  $\text{field\_simps}$ )
finally show  $\text{dist } x \ y < e$  .
qed
qed
{
  fix  $n \ m :: \text{nat}$ 
  assume  $m \leq n$  then have  $\text{cbox } (A \ n) \ (B \ n) \subseteq \text{cbox } (A \ m) \ (B \ m)$ 
  proof (induction rule:  $\text{inc\_induct}$ )
    case (step  $i$ )
    show ?case
      using  $AB$  by (intro  $\text{order\_trans}[\text{OF step.IH}] \ \text{subset\_box\_imp}$ ) auto
  qed simp
} note  $AB\text{subset} = \text{this}$ 
have  $\bigwedge n. \text{cbox } (A \ n) \ (B \ n) \neq \{\}$ 
  by (meson  $AB \ \text{dual\_order.trans interval\_not\_empty}$ )
then obtain  $x0$  where  $x0: \bigwedge n. x0 \in \text{cbox } (A \ n) \ (B \ n)$ 
  using  $\text{decreasing\_closed\_nest} [\text{OF closed\_cbox}] \ AB\text{subset interv}$  by blast
show  $\text{thesis}$ 
proof (rule  $\text{that}[\text{rule\_format}, \text{of } x0]$ )
  show  $x0 \in \text{cbox } a \ b$ 
    using  $\langle A \ 0 = a \rangle \ \langle B \ 0 = b \rangle \ x0$  by blast

```

```

fix e :: real
assume e > 0
from interval[OF this] obtain n
  where n:  $\forall x \in \text{cbox } (A \ n) \ (B \ n). \forall y \in \text{cbox } (A \ n) \ (B \ n). \text{dist } x \ y < e ..$ 
have  $\neg P \ (\text{cbox } (A \ n) \ (B \ n))$ 
proof (cases 0 < n)
  case True then show ?thesis
    by (metis Suc_pred' notPAB)
next
  case False then show ?thesis
    using  $\langle A \ 0 = a \rangle \langle B \ 0 = b \rangle \langle \neg P \ (\text{cbox } a \ b) \rangle$  by blast
qed
moreover have  $\text{cbox } (A \ n) \ (B \ n) \subseteq \text{ball } x0 \ e$ 
  using n using x0[of n] by auto
moreover have  $\text{cbox } (A \ n) \ (B \ n) \subseteq \text{cbox } a \ b$ 
  using ABsubset  $\langle A \ 0 = a \rangle \langle B \ 0 = b \rangle$  by blast
ultimately show  $\exists c \ d. x0 \in \text{cbox } c \ d \wedge \text{cbox } c \ d \subseteq \text{ball } x0 \ e \wedge \text{cbox } c \ d \subseteq$ 
 $\text{cbox } a \ b \wedge \neg P \ (\text{cbox } c \ d)$ 
  apply (rule_tac x=A n in exI)
  apply (rule_tac x=B n in exI)
  apply (auto simp: x0)
done
qed
qed

```

6.14.14 Cousin's lemma

```

lemma fine_division_exists:
  fixes a b :: 'a::euclidean_space
  assumes gauge g
  obtains p where p tagged_division_of (cbox a b) g fine p
proof (cases  $\exists p. p \text{ tagged\_division\_of } (\text{cbox } a \ b) \wedge g \text{ fine } p$ )
  case True
  then show ?thesis
    using that by auto
next
  case False
  assume  $\neg (\exists p. p \text{ tagged\_division\_of } (\text{cbox } a \ b) \wedge g \text{ fine } p)$ 
  obtain x where x:
     $x \in (\text{cbox } a \ b)$ 
     $\bigwedge e. 0 < e \implies$ 
     $\exists c \ d.$ 
     $x \in \text{cbox } c \ d \wedge$ 
     $\text{cbox } c \ d \subseteq \text{ball } x \ e \wedge$ 
     $\text{cbox } c \ d \subseteq (\text{cbox } a \ b) \wedge$ 
     $\neg (\exists p. p \text{ tagged\_division\_of } \text{cbox } c \ d \wedge g \text{ fine } p)$ 
  apply (rule interval_bisection[of  $\lambda s. \exists p. p \text{ tagged\_division\_of } s \wedge g \text{ fine } p,$ 
    OF _ False])
  apply (simp add: fine_def)

```

```

  apply (metis tagged_division_Un fine_Un)
  apply auto
  done
obtain e where e:  $e > 0$  ball  $x \in g \ x$ 
  using gaugeD[OF assms, of x] unfolding open_contains_ball by auto
from x(2)[OF e(1)]
obtain c d where c_d:  $x \in \text{cbox } c \ d$ 
  cbox  $c \ d \subseteq \text{ball } x \ e$ 
  cbox  $c \ d \subseteq \text{cbox } a \ b$ 
   $\neg (\exists p. p \text{ tagged\_division\_of } \text{cbox } c \ d \wedge g \text{ fine } p)$ 
  by blast
have g fine {(x, cbox c d)}
  unfolding fine_def using e using c_d(2) by auto
then show ?thesis
  using tagged_division_of_self[OF c_d(1)] using c_d by auto
qed

```

```

lemma fine_division_exists_real:
  fixes a b :: real
  assumes gauge g
  obtains p where p tagged_division_of {a .. b} g fine p
  by (metis assms box_real(2) fine_division_exists)

```

6.14.15 A technical lemma about "refinement" of division

```

lemma tagged_division_finer:
  fixes p :: ('a::euclidean_space  $\times$  ('a::euclidean_space set)) set
  assumes ptag: p tagged_division_of (cbox a b)
  and gauge d
  obtains q where q tagged_division_of (cbox a b)
  and d fine q
  and  $\forall (x,k) \in p. k \subseteq d(x) \longrightarrow (x,k) \in q$ 
proof -
  have p: finite p p tagged_partial_division_of (cbox a b)
    using ptag tagged_division_of_def by blast+
  have  $(\exists q. q \text{ tagged\_division\_of } (\bigcup \{k. \exists x. (x,k) \in p\}) \wedge d \text{ fine } q \wedge (\forall (x,k) \in p. k \subseteq d(x) \longrightarrow (x,k) \in q))$ 
    if finite p p tagged_partial_division_of (cbox a b) gauge d for p
    using that
  proof (induct p)
    case empty
    show ?case
      by (force simp add: fine_def)
  next
    case (insert xk p)
    obtain x k where xk:  $xk = (x, k)$ 
      using surj_pair[of xk] by metis
    obtain q1 where q1: q1 tagged_division_of  $\bigcup \{k. \exists x. (x, k) \in p\}$ 
      and d fine q1

```

```

      and q1I:  $\bigwedge x k. \llbracket (x, k) \in p; k \subseteq d \ x \rrbracket \implies (x, k) \in q1$ 
    using case_prodD tagged_partial_division_subset[OF insert(4) subset_insertI]
      by (metis (mono_tags, lifting) insert.hyps(3) insert.prem(2))
    have *:  $\bigcup \{l. \exists y. (y, l) \in \text{insert } xk \ p\} = k \cup \bigcup \{l. \exists y. (y, l) \in p\}$ 
      unfolding xk by auto
    note p = tagged_partial_division_ofD[OF insert(4)]
    obtain u v where uv:  $k = \text{cbox } u \ v$ 
      using p(4) xk by blast
    have finite  $\{k. \exists x. (x, k) \in p\}$ 
      apply (rule finite_subset[of__ snd 'p])
      using image_iff apply fastforce
      using insert.hyps(1) by blast
    then have int:  $\text{interior } (\text{cbox } u \ v) \cap \text{interior } (\bigcup \{k. \exists x. (x, k) \in p\}) = \{\}$ 
    proof (rule Int_interior_Union_intervals)
      show open  $(\text{interior } (\text{cbox } u \ v))$ 
        by auto
      show  $\bigwedge T. T \in \{k. \exists x. (x, k) \in p\} \implies \exists a \ b. T = \text{cbox } a \ b$ 
        using p(4) by auto
      show  $\bigwedge T. T \in \{k. \exists x. (x, k) \in p\} \implies \text{interior } (\text{cbox } u \ v) \cap \text{interior } T = \{\}$ 
        by clarify (metis insert.hyps(2) insert_iff interior_cbox p(5) uv xk)
    qed
  show ?case
  proof (cases cbox u v  $\subseteq d \ x$ )
    case True
      have  $\{(x, \text{cbox } u \ v)\}$  tagged_division_of cbox u v
        by (simp add: p(2) uv xk tagged_division_of_self)
      then have  $\{(x, \text{cbox } u \ v)\} \cup q1$  tagged_division_of  $\bigcup \{k. \exists x. (x, k) \in \text{insert } xk \ p\}$ 
        unfolding * uv by (metis (no_types, lifting) int q1 tagged_division_Un)
      with True show ?thesis
        apply (rule_tac x= $\{(x, \text{cbox } u \ v)\} \cup q1$  in exI)
        using <d fine q1> fine_def q1I uv xk apply fastforce
        done
    next
      case False
      obtain q2 where q2: q2 tagged_division_of cbox u v d fine q2
        using fine_division_exists[OF assms(2)] by blast
      show ?thesis
        apply (rule_tac x=q2  $\cup q1$  in exI)
        apply (intro conjI)
        unfolding * uv
        apply (rule tagged_division_Un q2 q1 int fine_Un)+
        apply (auto intro: q1 q2 fine_Un <d fine q1> simp add: False q1I uv xk)
        done
    qed
  qed
  with p obtain q where q: q tagged_division_of  $\bigcup \{k. \exists x. (x, k) \in p\}$  d fine q
     $\forall (x, k) \in p. k \subseteq d \ x \longrightarrow (x, k) \in q$ 
    by (meson <gauge d>)

```


with ptag that show ?thesis by auto
qed

Covering lemma

Some technical lemmas used in the approximation results that follow. Proof of the covering lemma is an obvious multidimensional generalization of Lemma 3, p65 of Swartz's "Introduction to Gauge Integrals".

proposition *covering_lemma*:

assumes $S \subseteq \text{cbox } a \ b$ $\text{box } a \ b \neq \{\}$ *gauge* g

obtains $\mathcal{D}$ **where**

countable $\mathcal{D} \cup \mathcal{D} \subseteq \text{cbox } a \ b$

$\bigwedge K. K \in \mathcal{D} \implies \text{interior } K \neq \{\} \wedge (\exists c \ d. K = \text{cbox } c \ d)$

pairwise $(\lambda A \ B. \text{interior } A \cap \text{interior } B = \{\}) \ \mathcal{D}$

$\bigwedge K. K \in \mathcal{D} \implies \exists x \in S \cap K. K \subseteq g \ x$

$\bigwedge u \ v. \text{cbox } u \ v \in \mathcal{D} \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot i - a \cdot i) / 2^n$
 $S \subseteq \bigcup \mathcal{D}$

proof –

have *aibi*: $\bigwedge i. i \in \text{Basis} \implies a \cdot i \leq b \cdot i$ **and** *normab*: $0 < \text{norm}(b - a)$

using $\langle \text{box } a \ b \neq \{\} \rangle$ *box_eq_empty* *box_sing* **by** *fastforce*+

let $?K0 = \lambda(n, f::'a \Rightarrow \text{nat}).$

$\text{cbox } (\sum i \in \text{Basis}. (a \cdot i + (f \ i / 2^n) * (b \cdot i - a \cdot i)) *_{\mathbb{R}} i)$
 $(\sum i \in \text{Basis}. (a \cdot i + ((f \ i + 1) / 2^n) * (b \cdot i - a \cdot i)) *_{\mathbb{R}} i)$

let $?D0 = ?K0 \ '(\text{SIGMA } n:\text{UNIV}. \text{Pi}_E \ \text{Basis} \ (\lambda i::'a. \text{lessThan } (2^n)))$

obtain $\mathcal{D}0$ **where** *count*: *countable* $\mathcal{D}0$

and *sub*: $\bigcup \mathcal{D}0 \subseteq \text{cbox } a \ b$

and *int*: $\bigwedge K. K \in \mathcal{D}0 \implies (\text{interior } K \neq \{\}) \wedge (\exists c \ d. K = \text{cbox } c \ d)$

and *intdj*: $\bigwedge A \ B. \llbracket A \in \mathcal{D}0; B \in \mathcal{D}0 \rrbracket \implies A \subseteq B \vee B \subseteq A \vee \text{interior } A \cap \text{interior } B = \{\}$

and *SK*: $\bigwedge x. x \in S \implies \exists K \in \mathcal{D}0. x \in K \wedge K \subseteq g \ x$

and *cbox*: $\bigwedge u \ v. \text{cbox } u \ v \in \mathcal{D}0 \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot i - a \cdot i) / 2^n$

and *fin*: $\bigwedge K. K \in \mathcal{D}0 \implies \text{finite } \{L \in \mathcal{D}0. K \subseteq L\}$

proof

show *countable* $?D0$

by (*simp* *add*: *countable_PiE*)

next

show $\bigcup ?D0 \subseteq \text{cbox } a \ b$

apply (*simp* *add*: *UN_subset_iff*)

apply (*intro* *conjI* *allI* *ballI* *subset_box_imp*)

apply (*simp* *add*: *field_simps*)

apply (*auto* *intro*: *mult_right_mono* *aibi*)

apply (*force* *simp*: *aibi* *scaling_mono* *nat_less_real_le* *dest*: *PiE_mem* *intro*: *mult_right_mono*)

done

next

show $\bigwedge K. K \in ?D0 \implies \text{interior } K \neq \{\} \wedge (\exists c \ d. K = \text{cbox } c \ d)$

using $\langle \text{box } a \ b \neq \{\} \rangle$

```

    by (clarsimp simp: box_eq_empty) (fastforce simp add: field_split_simps dest:
PiE_mem)
  next
    have realff: (real w) * 2m < (real v) * 2n  $\longleftrightarrow$  w * 2m < v * 2n for m n
  v w
    using of_nat_less_iff less_imp_of_nat_less by fastforce
    have *:  $\forall v w. ?K0(m,v) \subseteq ?K0(n,w) \vee ?K0(n,w) \subseteq ?K0(m,v) \vee \text{interior}(?K0(m,v)) \cap \text{interior}(?K0(n,w)) = \{\}$ 
    for m n — The symmetry argument requires a single HOL formula
    proof (rule linorder_wlog [where a=m and b=n], intro allI impI)
      fix v w m and n::nat
      assume m  $\leq$  n — WLOG we can assume  $m \leq n$ , when the first disjunct
      becomes impossible
      have  $?K0(n,w) \subseteq ?K0(m,v) \vee \text{interior}(?K0(m,v)) \cap \text{interior}(?K0(n,w)) = \{\}$ 
    {}
      apply (simp add: subset_box disjoint_interval)
      apply (rule ccontr)
      apply (clarsimp simp add: aibi mult_le_cancel_right divide_le_cancel
not_less not_le)
      apply (drule_tac x=i in bspec, assumption)
      using  $\langle m \leq n \rangle$  realff [of _ 1 + _] realff [of 1 + _ 1 + _]
      apply (auto simp: divide_simps add.commute not_le nat_le_iff_add realff)
      apply (simp_all add: power_add)
      apply (metis (no_types, opaque_lifting) mult_Suc mult_less_cancel2
not_less_eq mult.assoc)
      apply (metis (no_types, opaque_lifting) mult_Suc mult_less_cancel2
not_less_eq mult.assoc)
      done
      then show  $?K0(m,v) \subseteq ?K0(n,w) \vee ?K0(n,w) \subseteq ?K0(m,v) \vee \text{interior}(?K0(m,v)) \cap \text{interior}(?K0(n,w)) = \{\}$ 
      by meson
    qed auto
    show  $\bigwedge A B. [A \in ?D0; B \in ?D0] \implies A \subseteq B \vee B \subseteq A \vee \text{interior } A \cap \text{interior } B = \{\}$ 
    apply (erule imageE SigmaE)+
    using * by simp
  next
    show  $\exists K \in ?D0. x \in K \wedge K \subseteq g \ x$  if  $x \in S$  for x
    proof (simp only: bex_simps split_paired_Bex_Sigma)
      show  $\exists n. \exists f \in \text{Basis} \rightarrow_E \{.. < 2^n\}. x \in ?K0(n,f) \wedge ?K0(n,f) \subseteq g \ x$ 
      proof —
        obtain e where 0 < e
          and e:  $\bigwedge y. (\bigwedge i. i \in \text{Basis} \implies |x \cdot i - y \cdot i| \leq e) \implies y \in g \ x$ 
        proof —
          have  $x \in g \ x$  open (g x)
          using  $\langle \text{gauge } g \rangle$  by (auto simp: gauge_def)
          then obtain  $\varepsilon$  where 0 <  $\varepsilon$  and  $\varepsilon$ :  $\text{ball } x \ \varepsilon \subseteq g \ x$ 
          using openE by blast
          have  $\text{norm } (x - y) < \varepsilon$ 

```

```

      if ( $\bigwedge i. i \in \text{Basis} \implies |x \cdot i - y \cdot i| \leq \varepsilon / (2 * \text{real DIM}('a))$ ) for  $y$ 
    proof -
      have  $\text{norm } (x - y) \leq (\sum i \in \text{Basis}. |x \cdot i - y \cdot i|)$ 
      by (metis (no_types, lifting) inner_diff_left norm_le_l1 sum.cong)
      also have  $\dots \leq \text{DIM}('a) * (\varepsilon / (2 * \text{real DIM}('a)))$ 
      by (meson sum_bounded_above that)
      also have  $\dots = \varepsilon / 2$ 
      by (simp add: field_split_simps)
      also have  $\dots < \varepsilon$ 
      by (simp add:  $\langle 0 < \varepsilon \rangle$ )
      finally show ?thesis .
    qed
  then show ?thesis
    by (rule_tac  $e = \varepsilon / 2 / \text{DIM}('a)$  in that) (simp_all add:  $\langle 0 < \varepsilon \rangle$ )
dist_norm subsetD [OF  $\varepsilon$ ]
qed
have  $xab: x \in \text{cbox } a \ b$ 
  using  $\langle x \in S \rangle \langle S \subseteq \text{cbox } a \ b \rangle$  by blast
obtain  $n$  where  $n: \text{norm } (b - a) / 2^n < e$ 
  using real_arch_pow_inv [of  $e / \text{norm}(b - a) \ 1/2$ ] normab  $\langle 0 < e \rangle$ 
  by (auto simp: field_split_simps)
then have  $\text{norm } (b - a) < e * 2^n$ 
  by (auto simp: field_split_simps)
then have  $bai: b \cdot i - a \cdot i < e * 2^n$  if  $i \in \text{Basis}$  for  $i$ 
proof -
  have  $b \cdot i - a \cdot i \leq \text{norm } (b - a)$ 
  by (metis abs_of_nonneg dual_order.trans inner_diff_left linear
norm_ge_zero Basis_le_norm that)
  also have  $\dots < e * 2^n$ 
  using  $\langle \text{norm } (b - a) < e * 2^n \rangle$  by blast
  finally show ?thesis .
qed
have  $D: (a + n \leq x \wedge x \leq a + m) \implies (a + n \leq y \wedge y \leq a + m) \implies$ 
 $\text{abs}(x - y) \leq m - n$ 
  for  $a \ m \ n \ x$  and  $y::\text{real}$ 
  by auto
have  $\forall i \in \text{Basis}. \exists k < 2^n. (a \cdot i + \text{real } k * (b \cdot i - a \cdot i) / 2^n \leq x \cdot i \wedge$ 
 $x \cdot i \leq a \cdot i + (\text{real } k + 1) * (b \cdot i - a \cdot i) / 2^n)$ 
proof
  fix  $i::'a$  assume  $i \in \text{Basis}$ 
  consider  $x \cdot i = b \cdot i \mid x \cdot i < b \cdot i$ 
  using  $\langle i \in \text{Basis} \rangle \text{mem\_box}(2) \ xab$  by force
  then show  $\exists k < 2^n. (a \cdot i + \text{real } k * (b \cdot i - a \cdot i) / 2^n \leq x \cdot i \wedge$ 
 $x \cdot i \leq a \cdot i + (\text{real } k + 1) * (b \cdot i - a \cdot i) / 2^n)$ 
  proof cases
    case 1 then show ?thesis
      by (rule_tac  $x = 2^n - 1$  in exI) (auto simp: algebra_simps
field_split_simps of_nat_diff  $\langle i \in \text{Basis} \rangle \ aibi$ )
  next

```

```

case 2
then have abi_less:  $a \cdot i < b \cdot i$ 
  using  $\langle i \in \text{Basis} \rangle$  xab by (auto simp: mem_box)
let ?k = nat  $\lfloor 2^n * (x \cdot i - a \cdot i) / (b \cdot i - a \cdot i) \rfloor$ 
show ?thesis
proof (intro exI conjI)
  show  $?k < 2^n$ 
    using aibi xab  $\langle i \in \text{Basis} \rangle$ 
    by (force simp: nat_less_iff floor_less_iff field_split_simps 2
mem_box)
  next
  have  $a \cdot i + \text{real } ?k * (b \cdot i - a \cdot i) / 2^n \leq$ 
 $a \cdot i + (2^n * (x \cdot i - a \cdot i) / (b \cdot i - a \cdot i)) * (b \cdot i - a \cdot i) /$ 
 $2^n$ 
    apply (intro add_left_mono mult_right_mono divide_right_mono
of_nat_floor)
    using aibi [OF  $\langle i \in \text{Basis} \rangle$ ] xab 2
    apply (simp_all add:  $\langle i \in \text{Basis} \rangle$  mem_box field_split_simps)
    done
  also have  $\dots = x \cdot i$ 
    using abi_less by (simp add: field_split_simps)
  finally show  $a \cdot i + \text{real } ?k * (b \cdot i - a \cdot i) / 2^n \leq x \cdot i$  .
  next
  have  $x \cdot i \leq a \cdot i + (2^n * (x \cdot i - a \cdot i) / (b \cdot i - a \cdot i)) * (b \cdot i$ 
 $- a \cdot i) / 2^n$ 
    using abi_less by (simp add: field_split_simps)
  also have  $\dots \leq a \cdot i + (\text{real } ?k + 1) * (b \cdot i - a \cdot i) / 2^n$ 
    apply (intro add_left_mono mult_right_mono divide_right_mono
of_nat_floor)
    using aibi [OF  $\langle i \in \text{Basis} \rangle$ ] xab
    apply (auto simp:  $\langle i \in \text{Basis} \rangle$  mem_box divide_simps)
    done
  finally show  $x \cdot i \leq a \cdot i + (\text{real } ?k + 1) * (b \cdot i - a \cdot i) / 2^n$  .
qed
qed
qed
then have  $\exists f \in \text{Basis} \rightarrow_E \{.. < 2^n\}. x \in ?K0(n, f)$ 
  apply (simp add: mem_box Bex_def)
  apply (clarify dest!: bchoice)
  apply (rule_tac x=restrict f Basis in exI, simp)
  done
moreover have  $\bigwedge f. x \in ?K0(n, f) \implies ?K0(n, f) \subseteq g \ x$ 
  apply (clarsimp simp add: mem_box)
  apply (rule e)
  apply (drule bspec D, assumption)+
  apply (erule order_trans)
  apply (simp add: divide_simps)
using bai apply (force simp add: algebra_simps)
done

```

```

      ultimately show ?thesis by auto
    qed
  qed
next
  show  $\bigwedge u v. \text{cbox } u v \in ?D0 \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot i - a \cdot i) / 2^n$ 
    by (force simp: eq_cbox box_eq_empty field_simps dest!: aibi)
  next
    obtain  $j::'a$  where  $j \in \text{Basis}$ 
      using nonempty_Basis by blast
    have finite  $\{L \in ?D0. ?K0(n,f) \subseteq L\}$  if  $f \in \text{Basis} \rightarrow_E \{..<2^n\}$  for  $n f$ 
      proof (rule finite_subset)
        let  $?B = (\lambda(n, f::'a \Rightarrow \text{nat}). \text{cbox } (\sum i \in \text{Basis}. (a \cdot i + (f i) / 2^n * (b \cdot i - a \cdot i))) *_R i)$ 

$$(\sum i \in \text{Basis}. (a \cdot i + ((f i) + 1) / 2^n * (b \cdot i - a \cdot i))) *_R i)$$

        ' (SIGMA  $m:\text{atMost } n. \text{Pi}_E \text{Basis } (\lambda i::'a. \text{lessThan } (2^m))$ )
        have  $?K0(m,g) \in ?B$  if  $g \in \text{Basis} \rightarrow_E \{..<2^m\}$   $?K0(n,f) \subseteq ?K0(m,g)$ 
      for  $m g$ 
      proof -
        have  $dd: w / m \leq v / n \wedge (v+1) / n \leq (w+1) / m$ 

$$\implies \text{inverse } n \leq \text{inverse } m$$

        for  $w m v n::\text{real}$ 
        by (auto simp: field_split_simps)
        have  $\text{bjaj}: b \cdot j - a \cdot j > 0$ 
          using  $\langle j \in \text{Basis} \rangle \langle \text{box } a b \neq \{\} \rangle \text{box\_eq\_empty}(1)$  by fastforce
        have  $((g j) / 2^m) * (b \cdot j - a \cdot j) \leq ((f j) / 2^n) * (b \cdot j - a \cdot j) \wedge$ 

$$(((f j) + 1) / 2^n) * (b \cdot j - a \cdot j) \leq (((g j) + 1) / 2^m) * (b \cdot j - a \cdot j)$$

          using that  $\langle j \in \text{Basis} \rangle$  by (simp add: subset_box field_split_simps aibi)
        then have  $((g j) / 2^m) \leq ((f j) / 2^n) \wedge$ 

$$((\text{real}(f j) + 1) / 2^n) \leq ((\text{real}(g j) + 1) / 2^m)$$

        by (metis bjaj mult.commute of_nat_1 of_nat_add mult_le_cancel_iff2)
        then have  $\text{inverse } (2^n) \leq (\text{inverse } (2^m) :: \text{real})$ 
        by (rule dd)
        then have  $m \leq n$ 
        by auto
        show ?thesis
          by (rule imageI) (simp add:  $\langle m \leq n \rangle$  that)
      qed
    then show  $\{L \in ?D0. ?K0(n,f) \subseteq L\} \subseteq ?B$ 
      by auto
    show finite ?B
      by (intro finite_imageI finite_SigmaI finite_atMost finite_lessThan finite_PiE finite_Basis)
    qed
    then show finite  $\{L \in ?D0. K \subseteq L\}$  if  $K \in ?D0$  for  $K$ 
      using that by auto
    qed
  let  $?D1 = \{K \in \mathcal{D}0. \exists x \in S \cap K. K \subseteq g x\}$ 

```

```

obtain  $\mathcal{D}$  where count: countable  $\mathcal{D}$ 
  and sub:  $\bigcup \mathcal{D} \subseteq \text{cbox } a \ b \ S \subseteq \bigcup \mathcal{D}$ 
  and int:  $\bigwedge K. K \in \mathcal{D} \implies (\text{interior } K \neq \{\}) \wedge (\exists c \ d. K = \text{cbox } c \ d)$ 
  and intdj:  $\bigwedge A \ B. \llbracket A \in \mathcal{D}; B \in \mathcal{D} \rrbracket \implies A \subseteq B \vee B \subseteq A \vee \text{interior } A$ 
 $\cap \text{interior } B = \{\}$ 
  and SK:  $\bigwedge K. K \in \mathcal{D} \implies \exists x. x \in S \cap K \wedge K \subseteq g \ x$ 
  and cbox:  $\bigwedge u \ v. \text{cbox } u \ v \in \mathcal{D} \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot$ 
 $i - a \cdot i) / 2^n$ 
  and fin:  $\bigwedge K. K \in \mathcal{D} \implies \text{finite } \{L. L \in \mathcal{D} \wedge K \subseteq L\}$ 
proof
  show countable ?D1 using count countable_subset
    by (simp add: count countable_subset)
  show  $\bigcup ?D1 \subseteq \text{cbox } a \ b$ 
    using sub by blast
  show  $S \subseteq \bigcup ?D1$ 
    using SK by (force simp:)
  show  $\bigwedge K. K \in ?D1 \implies (\text{interior } K \neq \{\}) \wedge (\exists c \ d. K = \text{cbox } c \ d)$ 
    using int by blast
  show  $\bigwedge A \ B. \llbracket A \in ?D1; B \in ?D1 \rrbracket \implies A \subseteq B \vee B \subseteq A \vee \text{interior } A \cap \text{interior}$ 
 $B = \{\}$ 
    using intdj by blast
  show  $\bigwedge K. K \in ?D1 \implies \exists x. x \in S \cap K \wedge K \subseteq g \ x$ 
    by auto
  show  $\bigwedge u \ v. \text{cbox } u \ v \in ?D1 \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot i - a \cdot$ 
 $i) / 2^n$ 
    using cbox by blast
  show  $\bigwedge K. K \in ?D1 \implies \text{finite } \{L. L \in ?D1 \wedge K \subseteq L\}$ 
    using fin by simp (metis (mono_tags, lifting) Collect_mono rev_finite_subset)
qed
let ?D =  $\{K \in \mathcal{D}. \forall K'. K' \in \mathcal{D} \wedge K \neq K' \longrightarrow \neg(K \subseteq K')\}$ 
show ?thesis
proof (rule that)
  show countable ?D
    by (blast intro: countable_subset [OF__ count])
  show  $\bigcup ?D \subseteq \text{cbox } a \ b$ 
    using sub by blast
  show  $S \subseteq \bigcup ?D$ 
proof clarsimp
  fix x
  assume  $x \in S$ 
  then obtain X where  $x \in X \ X \in \mathcal{D}$  using  $\langle S \subseteq \bigcup \mathcal{D} \rangle$  by blast
  let ?R =  $\{(K, L). K \in \mathcal{D} \wedge L \in \mathcal{D} \wedge L \subset K\}$ 
  have irrR: irrefl ?R by (force simp: irrefl_def)
  have traR: trans ?R by (force simp: trans_def)
  have finR:  $\bigwedge x. \text{finite } \{y. (y, x) \in ?R\}$ 
  by simp (metis (mono_tags, lifting) fin  $\langle X \in \mathcal{D} \rangle$  finite_subset mem_Collect_eq
psubset_imp_subset subsetI)
  have  $\{X \in \mathcal{D}. x \in X\} \neq \{\}$ 
    using  $\langle X \in \mathcal{D} \rangle \langle x \in X \rangle$  by blast

```

```

    then obtain Y where Y ∈ {X ∈ D. x ∈ X} ∧ Y'. (Y', Y) ∈ ?R ⇒ Y' ∉
    {X ∈ D. x ∈ X}
    by (rule wfE_min' [OF wf_finite_segments [OF irrR traR finR]]) blast
    then show ∃ Y. Y ∈ D ∧ (∀ K'. K' ∈ D ∧ Y ≠ K' ⟶ ¬ Y ⊆ K') ∧ x ∈ Y
    by blast
  qed
  show ∧K. K ∈ ?D ⟹ interior K ≠ {} ∧ (∃ c d. K = cbox c d)
  by (blast intro: dest: int)
  show pairwise (λA B. interior A ∩ interior B = {}) ?D
  using intdj by (simp add: pairwise_def) metis
  show ∧K. K ∈ ?D ⟹ ∃ x ∈ S ∩ K. K ⊆ g x
  using SK by force
  show ∧u v. cbox u v ∈ ?D ⟹ ∃ n. ∀ i ∈ Basis. v · i - u · i = (b · i - a · i) /
  2n
  using cbox by force
  qed
qed

```

6.14.16 Division filter

Divisions over all gauges towards finer divisions.

definition *division_filter* :: 'a::euclidean_space set ⇒ ('a × 'a set) set filter
 where *division_filter* s = (INF g ∈ {g. gauge g}. principal {p. p tagged_division_of
 s ∧ g fine p})

proposition *eventually_division_filter*:

(∀_F p in *division_filter* s. P p) ⟷
 (∃ g. gauge g ∧ (∀ p. p tagged_division_of s ∧ g fine p ⟶ P p))

unfolding *division_filter_def*

proof (subst *eventually_INF_base*; clarsimp)

fix g1 g2 :: 'a ⇒ 'a set show gauge g1 ⟹ gauge g2 ⟹ ∃ x. gauge x ∧
 {p. p tagged_division_of s ∧ x fine p} ⊆ {p. p tagged_division_of s ∧ g1 fine
 p} ∧
 {p. p tagged_division_of s ∧ x fine p} ⊆ {p. p tagged_division_of s ∧ g2 fine
 p}

by (intro exI[of _ λx. g1 x ∩ g2 x]) (auto simp: fine_Int)

qed (auto simp: *eventually_principal*)

lemma *division_filter_not_empty*: *division_filter* (cbox a b) ≠ bot

unfolding *trivial_limit_def* *eventually_division_filter*

by (auto elim: *fine_division_exists*)

lemma *eventually_division_filter_tagged_division*:

eventually (λp. p tagged_division_of s) (*division_filter* s)

unfolding *eventually_division_filter* by (intro exI[of _ λx. ball x 1]) auto

end

6.15 Henstock-Kurzweil Gauge Integration in Many Dimensions

```

theory Henstock_Kurzweil_Integration
imports
  Lebesgue_Measure Tagged_Division
begin

lemma norm_diff2:  $\llbracket y = y1 + y2; x = x1 + x2; e = e1 + e2; \text{norm}(y1 - x1) \leq e1; \text{norm}(y2 - x2) \leq e2 \rrbracket$ 
   $\implies \text{norm}(y - x) \leq e$ 
  using norm_triangle_mono [of  $y1 - x1$   $e1$   $y2 - x2$   $e2$ ]
  by (simp add: add_diff_add)

lemma setcomp_dot1:  $\{z. P (z \cdot (i, 0))\} = \{(x, y). P(x \cdot i)\}$ 
  by auto

lemma setcomp_dot2:  $\{z. P (z \cdot (0, i))\} = \{(x, y). P(y \cdot i)\}$ 
  by auto

lemma Sigma_Int_Paircomp1:  $(\text{Sigma } A \ B) \cap \{(x, y). P \ x\} = \text{Sigma } (A \cap \{x. P \ x\}) \ B$ 
  by blast

lemma Sigma_Int_Paircomp2:  $(\text{Sigma } A \ B) \cap \{(x, y). P \ y\} = \text{Sigma } A \ (\lambda z. B \ z \cap \{y. P \ y\})$ 
  by blast

### 6.15.1 Content (length, area, volume...) of an interval

abbreviation content :: 'a::euclidean_space set  $\Rightarrow$  real'
  where content s  $\equiv$  measure lborel s

lemma content_cbox_cases:
  content (cbox a b) = (if  $\forall i \in \text{Basis}. a \cdot i \leq b \cdot i$  then prod ( $\lambda i. b \cdot i - a \cdot i$ ) Basis else 0)
  by (simp add: measure_lborel_cbox_eq inner_diff)

lemma content_cbox:  $\forall i \in \text{Basis}. a \cdot i \leq b \cdot i \implies \text{content (cbox a b)} = (\prod i \in \text{Basis}. b \cdot i - a \cdot i)$ 
  unfolding content_cbox_cases by simp

lemma content_cbox':  $\text{cbox a b} \neq \{\}$   $\implies \text{content (cbox a b)} = (\prod i \in \text{Basis}. b \cdot i - a \cdot i)$ 
  by (simp add: box_ne_empty inner_diff)

lemma content_cbox_if: content (cbox a b) = (if cbox a b =  $\{\}$  then 0 else  $\prod i \in \text{Basis}. b \cdot i - a \cdot i$ )
  by (simp add: content_cbox')

```



```

lemma content_cbox_cart:
  cbox a b  $\neq \{\}$   $\implies$  content(cbox a b) = prod ( $\lambda i. b\$i - a\$i$ ) UNIV
  by (simp add: content_cbox_if Basis_vec_def cart_eq_inner_axis axis_eq_axis
    prod.UNION_disjoint)

lemma content_cbox_if_cart:
  content(cbox a b) = (if cbox a b =  $\{\}$  then 0 else prod ( $\lambda i. b\$i - a\$i$ ) UNIV)
  by (simp add: content_cbox_cart)

lemma content_division_of:
  assumes  $K \in \mathcal{D} \ \mathcal{D} \text{ division\_of } S$ 
  shows content  $K = (\prod i \in \text{Basis. interval\_upperbound } K \cdot i - \text{interval\_lowerbound } K \cdot i)$ 
proof -
  obtain a b where  $K = \text{cbox } a \ b$ 
  using cbox_division_memE assms by metis
  then show ?thesis
  using assms by (force simp: division_of_def content_cbox')
qed

lemma content_real:  $a \leq b \implies \text{content } \{a..b\} = b - a$ 
  by simp

lemma abs_eq_content:  $|y - x| = (\text{if } x \leq y \text{ then content } \{x..y\} \text{ else content } \{y..x\})$ 
  by (auto simp: content_real)

lemma content_singleton: content  $\{a\} = 0$ 
  by simp

lemma content_unit[iff]: content (cbox 0 (One::'a::euclidean_space)) = 1
  by simp

lemma content_pos_le [iff]:  $0 \leq \text{content } X$ 
  by simp

corollary content_nonneg [simp]:  $\neg \text{content } (\text{cbox } a \ b) < 0$ 
  using not_le by blast

lemma content_pos_lt:  $\forall i \in \text{Basis. } a \cdot i < b \cdot i \implies 0 < \text{content } (\text{cbox } a \ b)$ 
  by (auto simp: less_imp_le inner_diff box_eq_empty intro!: prod_pos)

lemma content_eq_0: content (cbox a b) = 0  $\longleftrightarrow (\exists i \in \text{Basis. } b \cdot i \leq a \cdot i)$ 
  by (auto simp: content_cbox_cases not_le intro: less_imp_le antisym eq_refl)

lemma content_eq_0_interior: content (cbox a b) = 0  $\longleftrightarrow \text{interior}(\text{cbox } a \ b) = \{\}$ 
  unfolding content_eq_0 interior_cbox box_eq_empty by auto

```

lemma *content_pos_lt_eq*: $0 < \text{content } (\text{cbox } a \text{ } (b::'a::\text{euclidean\_space})) \longleftrightarrow (\forall i \in \text{Basis}. a \cdot i < b \cdot i)$

by (*auto simp add: content_cbox_cases less_le prod_nonneg*)

lemma *content_empty* [*simp*]: $\text{content } \{\} = 0$

by *simp*

lemma *content_real_if* [*simp*]: $\text{content } \{a..b\} = (\text{if } a \leq b \text{ then } b - a \text{ else } 0)$

by (*simp add: content_real*)

lemma *content_subset*: $\text{cbox } a \text{ } b \subseteq \text{cbox } c \text{ } d \implies \text{content } (\text{cbox } a \text{ } b) \leq \text{content } (\text{cbox } c \text{ } d)$

unfolding *measure_def*

by (*intro enn2real_mono emeasure_mono*) (*auto simp: emeasure_lborel_cbox_eq*)

lemma *content_lt_nz*: $0 < \text{content } (\text{cbox } a \text{ } b) \longleftrightarrow \text{content } (\text{cbox } a \text{ } b) \neq 0$

unfolding *content_pos_lt_eq content_eq_0* **unfolding** *not_ex not_le* **by** *fast-force*

lemma *content_Pair*: $\text{content } (\text{cbox } (a,c) \text{ } (b,d)) = \text{content } (\text{cbox } a \text{ } b) * \text{content } (\text{cbox } c \text{ } d)$

unfolding *measure_lborel_cbox_eq Basis_prod_def*

apply (*subst prod.union_disjoint*)

apply (*auto simp: bex_Un ball_Un*)

apply (*subst (1 2) prod.reindex_nontrivial*)

apply *auto*

done

lemma *content_cbox_pair_eq0_D*:

$\text{content } (\text{cbox } (a,c) \text{ } (b,d)) = 0 \implies \text{content } (\text{cbox } a \text{ } b) = 0 \vee \text{content } (\text{cbox } c \text{ } d) = 0$

by (*simp add: content_Pair*)

lemma *content_cbox_plus*:

fixes $x :: 'a::\text{euclidean\_space}$

shows $\text{content}(\text{cbox } x \text{ } (x + h *_{\mathbb{R}} \text{One})) = (\text{if } h \geq 0 \text{ then } h \wedge \text{DIM}('a) \text{ else } 0)$

by (*simp add: algebra_simps content_cbox_if box_eq_empty*)

lemma *content_0_subset*: $\text{content}(\text{cbox } a \text{ } b) = 0 \implies s \subseteq \text{cbox } a \text{ } b \implies \text{content } s = 0$

using *emeasure_mono*[*of s cbox a b lborel*]

by (*auto simp: measure_def enn2real_eq_0_iff emeasure_lborel_cbox_eq*)

lemma *content_ball_pos*:

assumes $r > 0$

shows $\text{content } (\text{ball } c \text{ } r) > 0$

proof —

from *rational_boxes*[*OF assms, of c*] **obtain** $a \text{ } b$ **where** $ab: c \in \text{box } a \text{ } b \text{ box } a \text{ } b \subseteq \text{ball } c \text{ } r$

```

    by auto
  from ab have 0 < content (box a b)
    by (subst measure_lborel_box_eq) (auto intro!: prod_pos simp: algebra_simps
box_def)
  have emeasure_lborel (box a b) ≤ emeasure_lborel (ball c r)
    using ab by (intro emeasure_mono) auto
  also have emeasure_lborel (box a b) = ennreal (content (box a b))
    using emeasure_lborel_box_finite[of a b] by (intro emeasure_eq_ennreal_measure)
auto
  also have emeasure_lborel (ball c r) = ennreal (content (ball c r))
    using emeasure_lborel_ball_finite[of c r] by (intro emeasure_eq_ennreal_measure)
auto
  finally show ?thesis
    using ⟨content (box a b) > 0⟩ by simp
qed

lemma content_cball_pos:
  assumes r > 0
  shows content (cball c r) > 0
proof -
  from rational_boxes[OF assms, of c] obtain a b where ab: c ∈ box a b box a b
  ⊆ ball c r
    by auto
  from ab have 0 < content (box a b)
    by (subst measure_lborel_box_eq) (auto intro!: prod_pos simp: algebra_simps
box_def)
  have emeasure_lborel (box a b) ≤ emeasure_lborel (ball c r)
    using ab by (intro emeasure_mono) auto
  also have ... ≤ emeasure_lborel (cball c r)
    by (intro emeasure_mono) auto
  also have emeasure_lborel (box a b) = ennreal (content (box a b))
    using emeasure_lborel_box_finite[of a b] by (intro emeasure_eq_ennreal_measure)
auto
  also have emeasure_lborel (cball c r) = ennreal (content (cball c r))
    using emeasure_lborel_cball_finite[of c r] by (intro emeasure_eq_ennreal_measure)
auto
  finally show ?thesis
    using ⟨content (box a b) > 0⟩ by simp
qed

lemma content_split:
  fixes a :: 'a::euclidean_space
  assumes k ∈ Basis
  shows content (cbox a b) = content(cbox a b ∩ {x. x·k ≤ c}) + content(cbox a
b ∩ {x. x·k ≥ c})
  — Prove using measure theory
proof (cases ∀ i ∈ Basis. a · i ≤ b · i)
case True
  have 1: ∧ X Y Z. (∏ i ∈ Basis. Z i (if i = k then X else Y i)) = Z k X *

```

```

( $\prod i \in \text{Basis} - \{k\}. Z\ i\ (Y\ i)$ )
  by (simp add: if_distrib prod.delta_remove assms)
  note_simps = interval_split[OF assms] content_cbox_cases
  have 2: ( $\prod i \in \text{Basis}. b \cdot i - a \cdot i = (\prod i \in \text{Basis} - \{k\}. b \cdot i - a \cdot i) * (b \cdot k - a \cdot k)$ )
    by (metis (no_types, lifting) assms finite_Basis mult.commute prod.remove)
  have  $\bigwedge x. \min(b \cdot k)\ c = \max(a \cdot k)\ c \implies$ 
     $x * (b \cdot k - a \cdot k) = x * (\max(a \cdot k)\ c - a \cdot k) + x * (b \cdot k - \max(a \cdot k)\ c)$ 
    by (auto simp add: field_simps)
  moreover
  have **: ( $\prod i \in \text{Basis}. ((\sum i \in \text{Basis}. (\text{if } i = k \text{ then } \min(b \cdot k)\ c \text{ else } b \cdot i) *_R i)$ 
 $\cdot i - a \cdot i)) =$ 
    ( $\prod i \in \text{Basis}. (\text{if } i = k \text{ then } \min(b \cdot k)\ c \text{ else } b \cdot i) - a \cdot i$ )
    ( $\prod i \in \text{Basis}. b \cdot i - ((\sum i \in \text{Basis}. (\text{if } i = k \text{ then } \max(a \cdot k)\ c \text{ else } a \cdot i) *_R i)$ 
 $\cdot i)) =$ 
    ( $\prod i \in \text{Basis}. b \cdot i - (\text{if } i = k \text{ then } \max(a \cdot k)\ c \text{ else } a \cdot i)$ )
    by (auto intro!: prod.cong)
  have  $\neg a \cdot k \leq c \implies \neg c \leq b \cdot k \implies \text{False}$ 
    unfolding not_le using True assms by auto
  ultimately show ?thesis
    using assms unfolding_simps ** 1[of  $\lambda i\ x. b \cdot i - x$ ] 1[of  $\lambda i\ x. x - a \cdot i$ ] 2
    by auto
next
case False
then have cbox_a_b = {}
  unfolding box_eq_empty by (auto simp: not_le)
then show ?thesis
  by (auto simp: not_le)
qed

```

```

lemma division_of_content_0:
  assumes content (cbox a b) = 0 d division_of (cbox a b)  $K \in d$ 
  shows content  $K = 0$ 
  unfolding forall_in_division[OF assms(2)]
  by (meson assms content_0_subset division_of_def)

```

```

lemma sum_content_null:
  assumes content (cbox a b) = 0
  and p tagged_division_of (cbox a b)
  shows ( $\sum (x, K) \in p. \text{content } K *_R f\ x$ ) = ( $0 :: 'a :: \text{real\_normed\_vector}$ )
proof (rule sum.neutral, rule)
  fix y
  assume y:  $y \in p$ 
  obtain x K where xk:  $y = (x, K)$ 
  using surj_pair[of y] by blast
  then obtain c d where k:  $K = \text{cbox } c\ d\ K \subseteq \text{cbox } a\ b$ 
  by (metis assms(2) tagged_division_ofD(3) tagged_division_ofD(4) y)
  have ( $\lambda (x', K'). \text{content } K' *_R f\ x'$ ) y = content  $K *_R f\ x$ 
  unfolding xk by auto
  also have ... = 0

```

```

    using assms(1) content_0_subset k(2) by auto
    finally show  $(\lambda(x, k). \text{content } k *_{\mathbb{R}} f x) y = 0$  .
qed

```

```

global_interpretation sum_content: operative plus 0 content
  rewrites comm_monoid_set.F plus 0 = sum
proof -
  interpret operative plus 0 content
  by standard (auto simp add: content_split [symmetric] content_eq_0_interior)
  show operative plus 0 content
  by standard
  show comm_monoid_set.F plus 0 = sum
  by (simp add: sum_def)
qed

```

```

lemma additive_content_division:  $d \text{ division\_of } (\text{cbox } a \ b) \implies \text{sum content } d = \text{content } (\text{cbox } a \ b)$ 
  by (fact sum_content.division)

```

```

lemma additive_content_tagged_division:
 $d \text{ tagged\_division\_of } (\text{cbox } a \ b) \implies \text{sum } (\lambda(x,l). \text{content } l) d = \text{content } (\text{cbox } a \ b)$ 
  by (fact sum_content.tagged_division)

```

```

lemma subadditive_content_division:
  assumes  $\mathcal{D} \text{ division\_of } S \ S \subseteq \text{cbox } a \ b$ 
  shows  $\text{sum content } \mathcal{D} \leq \text{content}(\text{cbox } a \ b)$ 
proof -
  have  $\mathcal{D} \text{ division\_of } \bigcup \mathcal{D} \ \bigcup \mathcal{D} \subseteq \text{cbox } a \ b$ 
  using assms by auto
  then obtain  $\mathcal{D}'$  where  $\mathcal{D} \subseteq \mathcal{D}' \ \mathcal{D}' \text{ division\_of } \text{cbox } a \ b$ 
  using partial_division_extend_interval by metis
  then have  $\text{sum content } \mathcal{D} \leq \text{sum content } \mathcal{D}'$ 
  using sum_mono2 by blast
  also have  $\dots \leq \text{content}(\text{cbox } a \ b)$ 
  by (simp add:  $\langle \mathcal{D}' \text{ division\_of } \text{cbox } a \ b \rangle \text{ additive\_content\_division less\_eq\_real\_def}$ )
  finally show ?thesis .
qed

```

```

lemma content_real_eq_0:  $\text{content } \{a..b::\text{real}\} = 0 \iff a \geq b$ 
  by (metis atLeastatMost_empty_iff2 content_empty content_real diff_self eq_iff
  le_cases le_iff_diff_le_0)

```

```

lemma property_empty_interval:  $\forall a \ b. \text{content } (\text{cbox } a \ b) = 0 \longrightarrow P (\text{cbox } a \ b)$ 
 $\implies P \{\}$ 
  using content_empty unfolding empty_as_interval by auto

```

```

lemma interval_bounds_nz_content [simp]:
  assumes  $\text{content } (\text{cbox } a \ b) \neq 0$ 

```

shows *interval_upperbound* (*cbox* *a* *b*) = *b*
and *interval_lowerbound* (*cbox* *a* *b*) = *a*
by (*metis assms content_empty interval_bounds'*)⁺

6.15.2 Gauge integral

Case distinction to define it first on compact intervals first, then use a limit. This is only much later unified. In Fremlin: Measure Theory, Volume 4I this is generalized using residual sets.

definition *has_integral* :: ('*n*::euclidean_space $\Rightarrow$ '*b*::real_normed_vector) $\Rightarrow$ '*b* $\Rightarrow$ '*n* set $\Rightarrow$ bool

(**infix** *has'_integral* 46)
where (*f has_integral I*) *s* $\longleftrightarrow$
 (if $\exists a\ b. s = \text{cbox } a\ b$
 then $((\lambda p. \sum_{(x,k) \in p. \text{content } k *_R f\ x}) \longrightarrow I) (\text{division\_filter } s)$
 else $(\forall e > 0. \exists B > 0. \forall a\ b. \text{ball } 0\ B \subseteq \text{cbox } a\ b \longrightarrow$
 $(\exists z. ((\lambda p. \sum_{(x,k) \in p. \text{content } k *_R (\text{if } x \in s \text{ then } f\ x \text{ else } 0)) \longrightarrow z)$
 $(\text{division\_filter } (\text{cbox } a\ b)) \wedge$
 $\text{norm } (z - I) < e)))$

lemma *has_integral_cbox*:

(*f has_integral I*) (*cbox* *a* *b*) $\longleftrightarrow ((\lambda p. \sum_{(x,k) \in p. \text{content } k *_R f\ x}) \longrightarrow I)$
 $(\text{division\_filter } (\text{cbox } a\ b))$
by (*auto simp add: has_integral_def*)

lemma *has_integral*:

(*f has_integral y*) (*cbox* *a* *b*) $\longleftrightarrow$
 $(\forall e > 0. \exists \gamma. \text{gauge } \gamma \wedge$
 $(\forall \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } (\text{cbox } a\ b) \wedge \gamma \text{ fine } \mathcal{D} \longrightarrow$
 $\text{norm } (\sum (\lambda(x,k). \text{content}(k) *_R f\ x) \mathcal{D} - y) < e))$
by (*auto simp: dist_norm eventually_division_filter has_integral_def tendsto_iff*)

lemma *has_integral_real*:

(*f has_integral y*) {*a..b*::real} $\longleftrightarrow$
 $(\forall e > 0. \exists \gamma. \text{gauge } \gamma \wedge$
 $(\forall \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } \{a..b\} \wedge \gamma \text{ fine } \mathcal{D} \longrightarrow$
 $\text{norm } (\sum (\lambda(x,k). \text{content}(k) *_R f\ x) \mathcal{D} - y) < e))$
unfolding *box_real[symmetric]* **by** (*rule has_integral*)

lemma *has_integralD[dest]*:

assumes (*f has_integral y*) (*cbox* *a* *b*)
and *e* > 0
obtains γ
where *gauge* γ
and $\bigwedge \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } (\text{cbox } a\ b) \implies \gamma \text{ fine } \mathcal{D} \implies$
 $\text{norm } ((\sum_{(x,k) \in \mathcal{D}. \text{content } k *_R f\ x}) - y) < e$
using *assms unfolding has_integral by auto*

lemma *has_integral_alt*:

```

(f has_integral y) i  $\longleftrightarrow$ 
  (if  $\exists a b. i = \text{cbox } a b$ 
    then (f has_integral y) i
    else ( $\forall e > 0. \exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow$ 
      ( $\exists z. ((\lambda x. \text{if } x \in i \text{ then } f x \text{ else } 0) \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm } (z - y) < e$ )))
  by (subst has_integral_def) (auto simp add: has_integral_cbox)

```

```

lemma has_integral_altD:
  assumes (f has_integral y) i
  and  $\neg (\exists a b. i = \text{cbox } a b)$ 
  and  $e > 0$ 
  obtains B where  $B > 0$ 
  and  $\forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow$ 
    ( $\exists z. ((\lambda x. \text{if } x \in i \text{ then } f(x) \text{ else } 0) \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm}(z - y) < e$ )
  using assms has_integral_alt[of f y i] by auto

```

```

definition integrable_on (infixr integrable'_on 46)
  where f integrable_on i  $\longleftrightarrow (\exists y. (f \text{ has\_integral } y) i)$ 

```

```

definition integral i f = (SOME y. (f has_integral y) i  $\vee \neg f \text{ integrable\_on } i \wedge y = 0$ )

```

```

lemma integrable_integral[intro]: f integrable_on i  $\implies (f \text{ has\_integral } (\text{integral } i f)) i$ 
  unfolding integrable_on_def integral_def by (metis (mono_tags, lifting) someI_ex)

```

```

lemma not_integrable_integral:  $\neg f \text{ integrable\_on } i \implies \text{integral } i f = 0$ 
  unfolding integrable_on_def integral_def by blast

```

```

lemma has_integral_integrable[dest]: (f has_integral i) s  $\implies f \text{ integrable\_on } s$ 
  unfolding integrable_on_def by auto

```

```

lemma has_integral_integral: f integrable_on s  $\longleftrightarrow (f \text{ has\_integral } (\text{integral } s f)) s$ 
  by auto

```

6.15.3 Basic theorems about integrals

```

lemma has_integral_eq_rhs: (f has_integral j) S  $\implies i = j \implies (f \text{ has\_integral } i) S$ 
  by (rule forw_subst)

```

```

lemma has_integral_unique_cbox:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::real_normed_vector
  shows (f has_integral k1) (cbox a b)  $\implies (f \text{ has\_integral } k2) (\text{cbox } a b) \implies k1 = k2$ 
  by (auto simp: has_integral_cbox intro: tendsto_unique[OF division_filter_not_empty])

```

```

lemma has_integral_unique:
  fixes  $f :: 'n::euclidean\_space \Rightarrow 'a::real\_normed\_vector$ 
  assumes  $(f \text{ has\_integral } k1) \ i \ (f \text{ has\_integral } k2) \ i$ 
  shows  $k1 = k2$ 
proof (rule ccontr)
  let  $?e = norm \ (k1 - k2)/2$ 
  let  $?F = (\lambda x. \text{if } x \in i \text{ then } f \ x \text{ else } 0)$ 
  assume  $k1 \neq k2$ 
  then have  $e: ?e > 0$ 
    by auto
  have nonbox:  $\neg (\exists a \ b. i = \text{cbox } a \ b)$ 
    using  $\langle k1 \neq k2 \rangle \text{ assms } has\_integral\_unique\_cbox$  by blast
  obtain  $B1$  where  $B1$ :
     $0 < B1$ 
     $\bigwedge a \ b. \text{ball } 0 \ B1 \subseteq \text{cbox } a \ b \implies$ 
     $\exists z. (?F \text{ has\_integral } z) (\text{cbox } a \ b) \wedge norm \ (z - k1) < norm \ (k1 - k2)/2$ 
    by (rule has_integral_altD[OF assms(1) nonbox, OF e]) blast
  obtain  $B2$  where  $B2$ :
     $0 < B2$ 
     $\bigwedge a \ b. \text{ball } 0 \ B2 \subseteq \text{cbox } a \ b \implies$ 
     $\exists z. (?F \text{ has\_integral } z) (\text{cbox } a \ b) \wedge norm \ (z - k2) < norm \ (k1 - k2)/2$ 
    by (rule has_integral_altD[OF assms(2) nonbox, OF e]) blast
  obtain  $a \ b :: 'n$  where  $ab: \text{ball } 0 \ B1 \subseteq \text{cbox } a \ b \ \text{ball } 0 \ B2 \subseteq \text{cbox } a \ b$ 
    by (metis Un_subset_iff bounded_Un bounded_ball bounded_subset_cbox_symmetric)
  obtain  $w$  where  $w: (?F \text{ has\_integral } w) (\text{cbox } a \ b) \ norm \ (w - k1) < norm \ (k1$ 
     $- k2)/2$ 
    using  $B1(2)[\text{OF } ab(1)]$  by blast
  obtain  $z$  where  $z: (?F \text{ has\_integral } z) (\text{cbox } a \ b) \ norm \ (z - k2) < norm \ (k1$ 
     $- k2)/2$ 
    using  $B2(2)[\text{OF } ab(2)]$  by blast
  have  $z = w$ 
    using has_integral_unique_cbox[OF  $w(1) \ z(1)$ ] by auto
  then have  $norm \ (k1 - k2) \leq norm \ (z - k2) + norm \ (w - k1)$ 
    using norm_triangle_ineq4 [of  $k1 - w \ k2 - z$ ]
    by (auto simp add: norm_minus_commute)
  also have  $\dots < norm \ (k1 - k2)/2 + norm \ (k1 - k2)/2$ 
    by (metis add_strict_mono  $z(2) \ w(2)$ )
  finally show False by auto
qed

```

```

lemma integral_unique [intro]:  $(f \text{ has\_integral } y) \ k \implies \text{integral } k \ f = y$ 
  unfolding integral_def
  by (rule some_equality) (auto intro: has_integral_unique)

```

```

lemma has_integral_iff:  $(f \text{ has\_integral } i) \ S \longleftrightarrow (f \text{ integrable\_on } S \wedge \text{integral } S$ 
 $f = i)$ 
  by blast

```


lemma *eq_integralD*: $\text{integral } k \ f = y \implies (f \text{ has\_integral } y) \ k \vee \neg f \text{ integrable\_on } k \wedge y=0$

unfolding *integral_def integrable_on_def*
apply (*erule subst*)
apply (*rule someI_ex*)
by *blast*

lemma *has_integral_const* [*intro*]:

fixes $a \ b :: 'a::\text{euclidean\_space}$
shows $((\lambda x. \ c) \text{ has\_integral } (\text{content } (\text{cbox } a \ b) *_{\mathbb{R}} c)) \ (\text{cbox } a \ b)$
using *eventually_division_filter_tagged_division*[*of cbox a b*]
additive_content_tagged_division[*of _ a b*]
by (*auto simp: has_integral_cbox split_beta' scaleR_sum_left[symmetric]*
elim!: eventually_mono intro!: tendsto_cong[THEN iffD1, OF _ tendsto_const])

lemma *has_integral_const_real* [*intro*]:

fixes $a \ b :: \text{real}$
shows $((\lambda x. \ c) \text{ has\_integral } (\text{content } \{a..b\} *_{\mathbb{R}} c)) \ \{a..b\}$
by (*metis box_real(2) has_integral_const*)

lemma *has_integral_integrable_integral*: $(f \text{ has\_integral } i) \ s \longleftrightarrow f \text{ integrable\_on } s \wedge \text{integral } s \ f = i$

by *blast*

lemma *integral_const* [*simp*]:

fixes $a \ b :: 'a::\text{euclidean\_space}$
shows $\text{integral } (\text{cbox } a \ b) \ (\lambda x. \ c) = \text{content } (\text{cbox } a \ b) *_{\mathbb{R}} c$
by (*rule integral_unique*) (*rule has_integral_const*)

lemma *integral_const_real* [*simp*]:

fixes $a \ b :: \text{real}$
shows $\text{integral } \{a..b\} \ (\lambda x. \ c) = \text{content } \{a..b\} *_{\mathbb{R}} c$
by (*metis box_real(2) integral_const*)

lemma *has_integral_is_0_cbox*:

fixes $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{real\_normed\_vector}$
assumes $\bigwedge x. x \in \text{cbox } a \ b \implies f \ x = 0$
shows $(f \text{ has\_integral } 0) \ (\text{cbox } a \ b)$
unfolding *has_integral_cbox*
using *eventually_division_filter_tagged_division*[*of cbox a b*] *assms*
by (*subst tendsto_cong[where g= $\lambda \_.$ 0]*)
(auto elim!: eventually_mono intro!: sum.neutral simp: tag_in_interval)

lemma *has_integral_is_0*:

fixes $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{real\_normed\_vector}$
assumes $\bigwedge x. x \in S \implies f \ x = 0$
shows $(f \text{ has\_integral } 0) \ S$

```

proof (cases (∃ a b. S = cbox a b))
  case True with assms has_integral_is_0_cbox show ?thesis
    by blast
next
  case False
  have *: (λx. if x ∈ S then f x else 0) = (λx. 0)
    by (auto simp add: assms)
  show ?thesis
    using has_integral_is_0_cbox False
    by (subst has_integral_alt) (force simp add: *)
qed

```

```

lemma has_integral_0[simp]: ((λx::'n::euclidean_space. 0) has_integral 0) S
  by (rule has_integral_is_0) auto

```

```

lemma has_integral_0_eq[simp]: ((λx. 0) has_integral i) S ⟷ i = 0
  using has_integral_unique[OF has_integral_0] by auto

```

```

lemma has_integral_linear_cbox:
  fixes f :: 'n::euclidean_space ⇒ 'a::real_normed_vector
  assumes f: (f has_integral y) (cbox a b)
    and h: bounded_linear h
  shows ((h ∘ f) has_integral (h y)) (cbox a b)
proof -
  interpret bounded_linear h using h .
  show ?thesis
    unfolding has_integral_cbox using tendsto [OF f [unfolded has_integral_cbox]]
    by (simp add: sum scaleR split_beta')
qed

```

```

lemma has_integral_linear:
  fixes f :: 'n::euclidean_space ⇒ 'a::real_normed_vector
  assumes f: (f has_integral y) S
    and h: bounded_linear h
  shows ((h ∘ f) has_integral (h y)) S
proof (cases (∃ a b. S = cbox a b))
  case True with f h has_integral_linear_cbox show ?thesis
    by blast
next
  case False
  interpret bounded_linear h using h .
  from pos_bounded obtain B where B: 0 < B ∧ x. norm (h x) ≤ norm x * B
    by blast
  let ?S = λf x. if x ∈ S then f x else 0
  show ?thesis
proof (subst has_integral_alt, clarsimp simp: False)
  fix e :: real
  assume e: e > 0
  have *: 0 < e/B using e B(1) by simp

```

```

obtain  $M$  where  $M$ :
   $M > 0$ 
   $\bigwedge a\ b. \text{ball } 0\ M \subseteq \text{cbox } a\ b \implies$ 
     $\exists z. (?S\ f\ \text{has\_integral } z) (\text{cbox } a\ b) \wedge \text{norm } (z - y) < e/B$ 
  using  $\text{has\_integral\_altD}[OF\ f\ \text{False } *]$  by  $\text{blast}$ 
show  $\exists B > 0. \forall a\ b. \text{ball } 0\ B \subseteq \text{cbox } a\ b \longrightarrow$ 
   $(\exists z. (?S(h \circ f)\ \text{has\_integral } z) (\text{cbox } a\ b) \wedge \text{norm } (z - h\ y) < e)$ 
proof ( $\text{rule } \text{exI}, \text{intro } \text{allI } \text{conjI } \text{impI}$ )
  show  $M > 0$  using  $M$  by  $\text{metis}$ 
next
  fix  $a\ b :: 'n$ 
  assume  $sb: \text{ball } 0\ M \subseteq \text{cbox } a\ b$ 
  obtain  $z$  where  $z: (?S\ f\ \text{has\_integral } z) (\text{cbox } a\ b) \wedge \text{norm } (z - y) < e/B$ 
  using  $M(2)[OF\ sb]$  by  $\text{blast}$ 
  have  $*$ :  $?S(h \circ f) = h \circ ?S\ f$ 
  using  $\text{zero}$  by  $\text{auto}$ 
  show  $\exists z. (?S(h \circ f)\ \text{has\_integral } z) (\text{cbox } a\ b) \wedge \text{norm } (z - h\ y) < e$ 
  proof ( $\text{intro } \text{exI } \text{conjI}$ )
    show  $(?S(h \circ f)\ \text{has\_integral } h\ z) (\text{cbox } a\ b)$ 
    by ( $\text{simp add: } * \text{ has\_integral\_linear\_cbox}[OF\ z(1)\ h]$ )
    show  $\text{norm } (h\ z - h\ y) < e$ 
    by ( $\text{metis } B\ \text{diff } \text{le\_less\_trans } \text{pos\_less\_divide\_eq } z(2)$ )
  qed
qed
qed
qed

```

```

lemma  $\text{has\_integral\_scaleR\_left}$ :
   $(f\ \text{has\_integral } y)\ S \implies ((\lambda x. f\ x *_{\mathbb{R}} c)\ \text{has\_integral } (y *_{\mathbb{R}} c))\ S$ 
  using  $\text{has\_integral\_linear}[OF\ \_ \text{ bounded\_linear\_scaleR\_left}]$  by ( $\text{simp add: comp\_def}$ )

```

```

lemma  $\text{integrable\_on\_scaleR\_left}$ :
  assumes  $f\ \text{integrable\_on } A$ 
  shows  $(\lambda x. f\ x *_{\mathbb{R}} y)\ \text{integrable\_on } A$ 
  using  $\text{assms } \text{has\_integral\_scaleR\_left}$  unfolding  $\text{integrable\_on\_def}$  by  $\text{blast}$ 

```

```

lemma  $\text{has\_integral\_mult\_left}$ :
  fixes  $c :: \_ :: \text{real\_normed\_algebra}$ 
  shows  $(f\ \text{has\_integral } y)\ S \implies ((\lambda x. f\ x * c)\ \text{has\_integral } (y * c))\ S$ 
  using  $\text{has\_integral\_linear}[OF\ \_ \text{ bounded\_linear\_mult\_left}]$  by ( $\text{simp add: comp\_def}$ )

```

```

lemma  $\text{has\_integral\_divide}$ :
  fixes  $c :: \_ :: \text{real\_normed\_div\_algebra}$ 
  shows  $(f\ \text{has\_integral } y)\ S \implies ((\lambda x. f\ x / c)\ \text{has\_integral } (y / c))\ S$ 
  unfolding  $\text{divide\_inverse}$  by ( $\text{simp add: has\_integral\_mult\_left}$ )

```

The case analysis eliminates the condition $f\ \text{integrable\_on } S$ at the cost of the type class constraint division_ring

```

corollary integral_mult_left [simp]:
  fixes  $c :: 'a :: \{\text{real\_normed\_algebra}, \text{division\_ring}\}$ 
  shows  $\text{integral } S (\lambda x. f\ x * c) = \text{integral } S f * c$ 
proof (cases  $f$  integrable_on  $S \vee c = 0$ )
  case True then show ?thesis
    by (force intro: has_integral_mult_left)
next
  case False then have  $\neg (\lambda x. f\ x * c)$  integrable_on  $S$ 
    using has_integral_mult_left [of  $(\lambda x. f\ x * c)$   $S$  inverse  $c$ ]
    by (auto simp add: mult.assoc)
  with False show ?thesis by (simp add: not_integrable_integral)
qed

```

```

corollary integral_mult_right [simp]:
  fixes  $c :: 'a :: \{\text{real\_normed\_field}\}$ 
  shows  $\text{integral } S (\lambda x. c * f\ x) = c * \text{integral } S f$ 
by (simp add: mult.commute [of  $c$ ])

```

```

corollary integral_divide [simp]:
  fixes  $z :: 'a :: \text{real\_normed\_field}$ 
  shows  $\text{integral } S (\lambda x. f\ x / z) = \text{integral } S (\lambda x. f\ x) / z$ 
using integral_mult_left [of  $S\ f$  inverse  $z$ ]
  by (simp add: divide_inverse_commute)

```

```

lemma has_integral_mult_right:
  fixes  $c :: 'a :: \text{real\_normed\_algebra}$ 
  shows  $(f$  has_integral  $y) \ i \implies ((\lambda x. c * f\ x)$  has_integral  $(c * y)) \ i$ 
  using has_integral_linear[OF  $\_ \text{bounded\_linear\_mult\_right}$ ] by (simp add:
comp_def)

```

```

lemma has_integral_cmul:  $(f$  has_integral  $k) \ S \implies ((\lambda x. c *_R f\ x)$  has_integral
 $(c *_R k)) \ S$ 
  unfolding o_def[symmetric]
  by (metis has_integral_linear bounded_linear_scaleR_right)

```

```

lemma has_integral_cmult_real:
  fixes  $c :: \text{real}$ 
  assumes  $c \neq 0 \implies (f$  has_integral  $x) \ A$ 
  shows  $((\lambda x. c * f\ x)$  has_integral  $c * x) \ A$ 
proof (cases  $c = 0$ )
  case True
  then show ?thesis by simp
next
  case False
  from has_integral_cmul[OF assms[OF this], of  $c$ ] show ?thesis
  unfolding real_scaleR_def .
qed

```

```

lemma has_integral_neg:  $(f$  has_integral  $k) \ S \implies ((\lambda x. -(f\ x))$  has_integral  $-k$ )

```

```

S
  by (drule_tac c=-1 in has_integral_cmul) auto

lemma has_integral_neg_iff:  $((\lambda x. - f x) \text{ has\_integral } k) S \longleftrightarrow (f \text{ has\_integral } - k) S$ 
  using has_integral_neg[of f - k] has_integral_neg[of  $\lambda x. - f x$  k] by auto

lemma has_integral_add_cbox:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::real_normed_vector
  assumes (f has_integral k) (cbox a b) (g has_integral l) (cbox a b)
  shows  $((\lambda x. f x + g x) \text{ has\_integral } (k + l)) (cbox a b)$ 
  using assms
    unfolding has_integral_cbox
    by (simp add: split_beta' scaleR_add_right sum.distrib[abs_def] tendsto_add)

lemma has_integral_add:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::real_normed_vector
  assumes f: (f has_integral k) S and g: (g has_integral l) S
  shows  $((\lambda x. f x + g x) \text{ has\_integral } (k + l)) S$ 
proof (cases  $\exists a b. S = \text{cbox } a b$ )
  case True with has_integral_add_cbox assms show ?thesis
    by blast
next
  let ?S =  $\lambda f x. \text{if } x \in S \text{ then } f x \text{ else } 0$ 
  case False
  then show ?thesis
  proof (subst has_integral_alt, clarsimp, goal_cases)
    case (1 e)
    then have e2:  $e/2 > 0$ 
    by auto
    obtain Bf where  $0 < Bf$ 
    and Bf:  $\bigwedge a b. \text{ball } 0 Bf \subseteq \text{cbox } a b \implies$ 
       $\exists z. (?S f \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm } (z - k) < e/2$ 
    using has_integral_altD[OF f False e2] by blast
    obtain Bg where  $0 < Bg$ 
    and Bg:  $\bigwedge a b. \text{ball } 0 Bg \subseteq (\text{cbox } a b) \implies$ 
       $\exists z. (?S g \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm } (z - l) < e/2$ 
    using has_integral_altD[OF g False e2] by blast
    show ?case
  proof (rule_tac x=max Bf Bg in exI, clarsimp simp add: max.strict_coboundedI1
     $\langle 0 < Bf \rangle$ )
    fix a b
    assume ball 0 (max Bf Bg)  $\subseteq \text{cbox } a (b::'n)$ 
    then have fs:  $\text{ball } 0 Bf \subseteq \text{cbox } a (b::'n)$  and gs:  $\text{ball } 0 Bg \subseteq \text{cbox } a (b::'n)$ 
    by auto
    obtain w where  $w: (?S f \text{ has\_integral } w) (\text{cbox } a b) \wedge \text{norm } (w - k) < e/2$ 
    using Bf[OF fs] by blast
    obtain z where  $z: (?S g \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm } (z - l) < e/2$ 
    using Bg[OF gs] by blast

```

```

have *:  $\bigwedge x. (if\ x \in S\ then\ f\ x + g\ x\ else\ 0) = (?S\ f\ x) + (?S\ g\ x)$ 
  by auto
show  $\exists z. (?S(\lambda x. f\ x + g\ x)\ has\_integral\ z)\ (cbox\ a\ b) \wedge norm\ (z - (k + l)) < e$ 
  proof (intro exI conjI)
    show  $(?S(\lambda x. f\ x + g\ x)\ has\_integral\ (w + z))\ (cbox\ a\ b)$ 
    by (simp add: has\_integral\_add\_cbox[OF w(1) z(1), unfolded *[symmetric]])
    show  $norm\ (w + z - (k + l)) < e$ 
    by (metis dist\_norm dist\_triangle\_add\_half w(2) z(2))
  qed
qed
qed
qed
qed

```

```

lemma has\_integral\_diff:
   $(f\ has\_integral\ k)\ S \implies (g\ has\_integral\ l)\ S \implies$ 
   $((\lambda x. f\ x - g\ x)\ has\_integral\ (k - l))\ S$ 
  using has\_integral\_add[OF _ has\_integral\_neg, of f k S g l]
  by (auto simp: algebra_simps)

```

```

lemma integral\_0 [simp]:
   $integral\ S\ (\lambda x::'n::euclidean\_space.\ 0::'m::real\_normed\_vector) = 0$ 
  by (rule integral\_unique has\_integral\_0)+

```

```

lemma integral\_add:  $f\ integrable\_on\ S \implies g\ integrable\_on\ S \implies$ 
   $integral\ S\ (\lambda x. f\ x + g\ x) = integral\ S\ f + integral\ S\ g$ 
  by (rule integral\_unique) (metis integrable\_integral has\_integral\_add)

```

```

lemma integral\_cmul [simp]:  $integral\ S\ (\lambda x. c *_{\mathbb{R}} f\ x) = c *_{\mathbb{R}} integral\ S\ f$ 
proof (cases  $f\ integrable\_on\ S \vee c = 0$ )
  case True with has\_integral\_cmul integrable\_integral show ?thesis
    by fastforce
next
  case False then have  $\neg (\lambda x. c *_{\mathbb{R}} f\ x)\ integrable\_on\ S$ 
    using has\_integral\_cmul [of  $(\lambda x. c *_{\mathbb{R}} f\ x)\ S\ inverse\ c$ ] by auto
  with False show ?thesis by (simp add: not\_integrable\_integral)
qed

```

```

lemma integral\_mult:
  fixes  $K::real$ 
  shows  $f\ integrable\_on\ X \implies K * integral\ X\ f = integral\ X\ (\lambda x. K * f\ x)$ 
  unfolding real\_scaleR\_def[symmetric] integral\_cmul ..

```

```

lemma integral\_neg [simp]:  $integral\ S\ (\lambda x. - f\ x) = - integral\ S\ f$ 
proof (cases  $f\ integrable\_on\ S$ )
  case True then show ?thesis
    by (simp add: has\_integral\_neg integrable\_integral integral\_unique)
next
  case False then have  $\neg (\lambda x. - f\ x)\ integrable\_on\ S$ 

```

```

    using has_integral_neg [of  $(\lambda x. -f x)$   $S$ ] by auto
  with False show ?thesis by (simp add: not_integrable_integral)
qed

```

```

lemma integral_diff:  $f$  integrable_on  $S \implies g$  integrable_on  $S \implies$ 
   $\text{integral } S (\lambda x. f x - g x) = \text{integral } S f - \text{integral } S g$ 
  by (rule integral_unique) (metis integrable_integral has_integral_diff)

```

```

lemma integrable_0:  $(\lambda x. 0)$  integrable_on  $S$ 
  unfolding integrable_on_def using has_integral_0 by auto

```

```

lemma integrable_add:  $f$  integrable_on  $S \implies g$  integrable_on  $S \implies (\lambda x. f x + g x)$ 
  integrable_on  $S$ 
  unfolding integrable_on_def by (auto intro: has_integral_add)

```

```

lemma integrable_cmul:  $f$  integrable_on  $S \implies (\lambda x. c *_{\mathbb{R}} f(x))$  integrable_on  $S$ 
  unfolding integrable_on_def by (auto intro: has_integral_cmul)

```

```

lemma integrable_on_scaleR_iff [simp]:
  fixes  $c :: \text{real}$ 
  assumes  $c \neq 0$ 
  shows  $(\lambda x. c *_{\mathbb{R}} f x)$  integrable_on  $S \longleftrightarrow f$  integrable_on  $S$ 
  using integrable_cmul [of  $\lambda x. c *_{\mathbb{R}} f x$   $S$   $1 / c$ ] integrable_cmul [of  $f$   $S$   $c$ ]  $\langle c \neq 0 \rangle$ 
  by auto

```

```

lemma integrable_on_cmult_iff [simp]:
  fixes  $c :: \text{real}$ 
  assumes  $c \neq 0$ 
  shows  $(\lambda x. c * f x)$  integrable_on  $S \longleftrightarrow f$  integrable_on  $S$ 
  using integrable_on_scaleR_iff [of  $c$   $f$ ] assms by simp

```

```

lemma integrable_on_cmult_left:
  assumes  $f$  integrable_on  $S$ 
  shows  $(\lambda x. \text{of\_real } c * f x)$  integrable_on  $S$ 
  using integrable_cmul [of  $f$   $S$   $\text{of\_real } c$ ] assms
  by (simp add: scaleR_conv_of_real)

```

```

lemma integrable_neg:  $f$  integrable_on  $S \implies (\lambda x. -f(x))$  integrable_on  $S$ 
  unfolding integrable_on_def by (auto intro: has_integral_neg)

```

```

lemma integrable_neg_iff:  $(\lambda x. -f(x))$  integrable_on  $S \longleftrightarrow f$  integrable_on  $S$ 
  using integrable_neg by fastforce

```

```

lemma integrable_diff:
   $f$  integrable_on  $S \implies g$  integrable_on  $S \implies (\lambda x. f x - g x)$  integrable_on  $S$ 
  unfolding integrable_on_def by (auto intro: has_integral_diff)

```

```

lemma integrable_linear:
   $f$  integrable_on  $S \implies \text{bounded\_linear } h \implies (h \circ f)$  integrable_on  $S$ 

```

unfolding *integrable_on_def* **by** (*auto intro: has_integral_linear*)

lemma *integral_linear*:

f integrable_on S $\implies$ *bounded_linear h* $\implies$ *integral S (h $\circ$ f) = h (integral S f)*
by (*meson has_integral_iff has_integral_linear*)

lemma *integrable_on_cnj_iff*:

($\lambda x. \text{cnj } (f x)$) *integrable_on A* $\longleftrightarrow$ *f integrable_on A*
using *integrable_linear[OF _ bounded_linear_cnj, of f A]*
integrable_linear[OF _ bounded_linear_cnj, of cnj $\circ$ f A]
by (*auto simp: o_def*)

lemma *integral_cnj*: *cnj (integral A f) = integral A ($\lambda x. \text{cnj } (f x)$)*

by (*cases f integrable_on A*)
(simp_all add: integral_linear[OF _ bounded_linear_cnj, symmetric]
o_def integrable_on_cnj_iff not_integrable_integral)

lemma *integral_component_eq[simp]*:

fixes *f* :: '*n::euclidean_space* $\Rightarrow$ '*m::euclidean_space*
assumes *f integrable_on S*
shows *integral S ($\lambda x. f x \cdot k$) = integral S f $\cdot$ k*
unfolding *integral_linear[OF assms(1) bounded_linear_inner_left, unfolded o_def]*

..

lemma *has_integral_sum*:

assumes *finite T*
and $\bigwedge a. a \in T \implies ((f a) \text{ has\_integral } (i a)) S$
shows $((\lambda x. \text{sum } (\lambda a. f a x) T) \text{ has\_integral } (\text{sum } i T)) S$
using *assms(1) subset_refl[of T]*
proof (*induct rule: finite_subset_induct*)
case empty
then show ?*case* **by** *auto*
next
case (*insert x F*)
with *assms* **show** ?*case*
by (*simp add: has_integral_add*)

qed

lemma *integral_sum*:

$\llbracket \text{finite } I; \bigwedge a. a \in I \implies f a \text{ integrable\_on } S \rrbracket \implies$
integral S ($\lambda x. \sum a \in I. f a x$) = ($\sum a \in I. \text{integral S } (f a)$)
by (*simp add: has_integral_sum integrable_integral integral_unique*)

lemma *integrable_sum*:

$\llbracket \text{finite } I; \bigwedge a. a \in I \implies f a \text{ integrable\_on } S \rrbracket \implies (\lambda x. \sum a \in I. f a x) \text{ integrable\_on } S$
unfolding *integrable_on_def* **using** *has_integral_sum[of I]* **by** *metis*

lemma *has_integral_eq*:


```

assumes  $\bigwedge x. x \in s \implies f\ x = g\ x$ 
and (f has_integral k) s
shows (g has_integral k) s
using has_integral_diff[OF assms(2), of  $\lambda x. f\ x - g\ x\ 0$ ]
using has_integral_is_0[of s  $\lambda x. f\ x - g\ x$ ]
using assms(1)
by auto

```

```

lemma integrable_eq:  $\llbracket f \text{ integrable\_on } s; \bigwedge x. x \in s \implies f\ x = g\ x \rrbracket \implies g \text{ integrable\_on } s$ 
unfolding integrable_on_def
using has_integral_eq[of s f g] has_integral_eq by blast

```

```

lemma has_integral_cong:
assumes  $\bigwedge x. x \in s \implies f\ x = g\ x$ 
shows (f has_integral i) s = (g has_integral i) s
using has_integral_eq[of s f g] has_integral_eq[of s g f] assms
by auto

```

```

lemma integral_cong:
assumes  $\bigwedge x. x \in s \implies f\ x = g\ x$ 
shows  $\text{integral } s\ f = \text{integral } s\ g$ 
unfolding integral_def
by (metis (full_types, opaque_lifting) assms has_integral_cong integrable_eq)

```

```

lemma integrable_on_cmult_left_iff [simp]:
assumes  $c \neq 0$ 
shows  $(\lambda x. \text{of\_real } c * f\ x) \text{ integrable\_on } s \longleftrightarrow f \text{ integrable\_on } s$ 
(is ?lhs = ?rhs)
proof
assume ?lhs
then have  $(\lambda x. \text{of\_real } (1 / c) * (\text{of\_real } c * f\ x)) \text{ integrable\_on } s$ 
using integrable_cmul[of  $\lambda x. \text{of\_real } c * f\ x\ s\ 1 / \text{of\_real } c$ ]
by (simp add: scaleR_conv_of_real)
then have  $(\lambda x. (\text{of\_real } (1 / c) * \text{of\_real } c * f\ x)) \text{ integrable\_on } s$ 
by (simp add: algebra_simps)
with  $\langle c \neq 0 \rangle$  show ?rhs
by (metis (no_types, lifting) integrable_eq mult.left_neutral nonzero_divide_eq_eq
of_real_1 of_real_mult)
qed (blast intro: integrable_on_cmult_left)

```

```

lemma integrable_on_cmult_right:
fixes f ::  $\_ \Rightarrow 'b :: \{\text{comm\_ring}, \text{real\_algebra\_1}, \text{real\_normed\_vector}\}$ 
assumes f integrable_on s
shows  $(\lambda x. f\ x * \text{of\_real } c) \text{ integrable\_on } s$ 
using integrable_on_cmult_left [OF assms] by (simp add: mult.commute)

```

```

lemma integrable_on_cmult_right_iff [simp]:
fixes f ::  $\_ \Rightarrow 'b :: \{\text{comm\_ring}, \text{real\_algebra\_1}, \text{real\_normed\_vector}\}$ 

```

assumes $c \neq 0$
shows $(\lambda x. f\ x * \text{of\_real } c) \text{ integrable\_on } s \longleftrightarrow f \text{ integrable\_on } s$
using *integrable_on_cmult_left_iff* [*OF assms*] **by** (*simp add: mult.commute*)

lemma *integrable_on_cdivide*:
fixes $f :: \_ \Rightarrow 'b :: \text{real\_normed\_field}$
assumes $f \text{ integrable\_on } s$
shows $(\lambda x. f\ x / \text{of\_real } c) \text{ integrable\_on } s$
by (*simp add: integrable_on_cmult_right divide_inverse assms flip: of_real_inverse*)

lemma *integrable_on_cdivide_iff* [*simp*]:
fixes $f :: \_ \Rightarrow 'b :: \text{real\_normed\_field}$
assumes $c \neq 0$
shows $(\lambda x. f\ x / \text{of\_real } c) \text{ integrable\_on } s \longleftrightarrow f \text{ integrable\_on } s$
by (*simp add: divide_inverse assms flip: of_real_inverse*)

lemma *has_integral_null* [*intro*]: $\text{content}(\text{cbox } a\ b) = 0 \implies (f \text{ has\_integral } 0)$
(cbox a b)
unfolding *has_integral_cbox*
using *eventually_division_filter_tagged_division*[*of cbox a b*]
by (*subst tendsto_cong[where g= $\lambda \_.$ 0]*) (*auto elim: eventually_mono intro: sum_content_null*)

lemma *has_integral_null_real* [*intro*]: $\text{content } \{a..b::\text{real}\} = 0 \implies (f \text{ has\_integral } 0)$
 $\{a..b\}$
by (*metis box_real(2) has_integral_null*)

lemma *has_integral_null_eq*[*simp*]: $\text{content}(\text{cbox } a\ b) = 0 \implies (f \text{ has\_integral } i)$
(cbox a b) $\longleftrightarrow i = 0$
by (*auto simp add: has_integral_null dest!: integral_unique*)

lemma *integral_null* [*simp*]: $\text{content}(\text{cbox } a\ b) = 0 \implies \text{integral}(\text{cbox } a\ b)\ f = 0$
by (*metis has_integral_null integral_unique*)

lemma *integrable_on_null* [*intro*]: $\text{content}(\text{cbox } a\ b) = 0 \implies f \text{ integrable\_on } (\text{cbox } a\ b)$
by (*simp add: has_integral_integrable*)

lemma *has_integral_empty*[*intro*]: $(f \text{ has\_integral } 0) \ \{\}$
by (*meson ex_in_conv has_integral_is_0*)

lemma *has_integral_empty_eq*[*simp*]: $(f \text{ has\_integral } i) \ \{\} \longleftrightarrow i = 0$
by (*auto simp add: has_integral_empty has_integral_unique*)

lemma *integrable_on_empty*[*intro*]: $f \text{ integrable\_on } \{\}$
unfolding *integrable_on_def* **by** *auto*

lemma *integral_empty*[*simp*]: $\text{integral } \{\} \ f = 0$
by (*rule integral_unique*) (*rule has_integral_empty*)

```

lemma has_integral_refl[intro]:
  fixes a :: 'a::euclidean_space
  shows (f has_integral 0) (cbox a a)
    and (f has_integral 0) {a}
proof -
  show (f has_integral 0) (cbox a a)
    by (rule has_integral_null) simp
  then show (f has_integral 0) {a}
    by simp
qed

```

```

lemma integrable_on_refl[intro]: f integrable_on cbox a a
  unfolding integrable_on_def by auto

```

```

lemma integral_refl [simp]: integral (cbox a a) f = 0
  by (rule integral_unique) auto

```

```

lemma integral_singleton [simp]: integral {a} f = 0
  by auto

```

```

lemma integral_blinfun_apply:
  assumes f integrable_on s
  shows integral s ( $\lambda x. \text{blinfun\_apply } h (f x) = \text{blinfun\_apply } h (\text{integral } s f)$ )
    by (subst integral_linear[symmetric, OF assms blinfun.bounded_linear_right])
  (simp add: o_def)

```

```

lemma blinfun_apply_integral:
  assumes f integrable_on s
  shows blinfun_apply (integral s f) x = integral s ( $\lambda y. \text{blinfun\_apply } (f y) x$ )
    by (metis (no_types, lifting) assms blinfun.prod_left_rep_eq integral_blinfun_apply
  integral_cong)

```

```

lemma has_integral_componentwise_iff:
  fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: euclidean_space
  shows (f has_integral y) A  $\longleftrightarrow$  ( $\forall b \in \text{Basis}. ((\lambda x. f x \cdot b) \text{ has\_integral } (y \cdot b))$ 
  A)

```

```

proof safe
  fix b :: 'b assume (f has_integral y) A
  from has_integral_linear[OF this(1) bounded_linear_inner_left, of b]
  show (( $\lambda x. f x \cdot b$ ) has_integral (y  $\cdot$  b)) A by (simp add: o_def)
next
  assume ( $\forall b \in \text{Basis}. ((\lambda x. f x \cdot b) \text{ has\_integral } (y \cdot b)) A$ )
  hence  $\forall b \in \text{Basis}. (((\lambda x. x *_{\mathbb{R}} b) \circ (\lambda x. f x \cdot b)) \text{ has\_integral } ((y \cdot b) *_{\mathbb{R}} b)) A$ 
    by (intro ballI has_integral_linear) (simp_all add: bounded_linear_scaleR_left)
  hence ( $\lambda x. \sum b \in \text{Basis}. (f x \cdot b) *_{\mathbb{R}} b$ ) has_integral ( $\sum b \in \text{Basis}. (y \cdot b) *_{\mathbb{R}} b$ )
  A
  by (intro has_integral_sum) (simp_all add: o_def)
  thus (f has_integral y) A by (simp add: euclidean_representation)

```

1800

qed

lemma *has_integral_componentwise*:

fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$

shows $(\bigwedge b. b \in \text{Basis} \implies ((\lambda x. f\ x \cdot b) \text{ has\_integral } (y \cdot b))\ A) \implies (f \text{ has\_integral } y)\ A$

by (subst has_integral_componentwise_iff) blast

lemma *integrable_componentwise_iff*:

fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$

shows $f \text{ integrable\_on } A \longleftrightarrow (\forall b \in \text{Basis}. (\lambda x. f\ x \cdot b) \text{ integrable\_on } A)$

proof

assume $f \text{ integrable\_on } A$

then obtain y **where** $(f \text{ has\_integral } y)\ A$ **by** (auto simp: integrable_on_def)

hence $(\forall b \in \text{Basis}. ((\lambda x. f\ x \cdot b) \text{ has\_integral } (y \cdot b))\ A)$

by (subst (asm) has_integral_componentwise_iff)

thus $(\forall b \in \text{Basis}. (\lambda x. f\ x \cdot b) \text{ integrable\_on } A)$ **by** (auto simp: integrable_on_def)

next

assume $(\forall b \in \text{Basis}. (\lambda x. f\ x \cdot b) \text{ integrable\_on } A)$

then obtain y **where** $\forall b \in \text{Basis}. ((\lambda x. f\ x \cdot b) \text{ has\_integral } y\ b)\ A$

unfolding integrable_on_def **by** (subst (asm) bchoice_iff) blast

hence $\forall b \in \text{Basis}. (((\lambda x. x *_R b) \circ (\lambda x. f\ x \cdot b)) \text{ has\_integral } (y\ b *_R b))\ A$

by (intro ballI has_integral_linear) (simp_all add: bounded_linear_scaleR_left)

hence $((\lambda x. \sum b \in \text{Basis}. (f\ x \cdot b) *_R b) \text{ has\_integral } (\sum b \in \text{Basis}. y\ b *_R b))\ A$

by (intro has_integral_sum) (simp_all add: o_def)

thus $f \text{ integrable\_on } A$ **by** (auto simp: integrable_on_def o_def euclidean_representation)

qed

lemma *integrable_componentwise*:

fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$

shows $(\bigwedge b. b \in \text{Basis} \implies (\lambda x. f\ x \cdot b) \text{ integrable\_on } A) \implies f \text{ integrable\_on } A$

by (subst integrable_componentwise_iff) blast

lemma *integral_componentwise*:

fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$

assumes $f \text{ integrable\_on } A$

shows $\text{integral } A\ f = (\sum b \in \text{Basis}. \text{integral } A\ (\lambda x. (f\ x \cdot b) *_R b))$

proof –

from assms **have** integrable: $\forall b \in \text{Basis}. (\lambda x. x *_R b) \circ (\lambda x. (f\ x \cdot b)) \text{ integrable\_on } A$

by (subst (asm) integrable_componentwise_iff, intro integrable_linear ballI)

(simp_all add: bounded_linear_scaleR_left)

have $\text{integral } A\ f = \text{integral } A\ (\lambda x. \sum b \in \text{Basis}. (f\ x \cdot b) *_R b)$

by (simp add: euclidean_representation)

also from integrable **have** $\dots = (\sum a \in \text{Basis}. \text{integral } A\ (\lambda x. (f\ x \cdot a) *_R a))$

by (subst integral_sum) (simp_all add: o_def)

finally show ?thesis .

qed

lemma *integrable_component*:

f integrable_on $A \implies (\lambda x. f\ x \cdot (y :: 'b :: euclidean\_space))$ integrable_on A
by (drule integrable_linear[OF _ bounded_linear_inner_left[of y]]) (simp add: o_def)

6.15.4 Cauchy-type criterion for integrability

proposition *integrable_Cauchy*:

fixes $f :: 'n :: euclidean\_space \Rightarrow 'a :: \{real\_normed\_vector, complete\_space\}$
shows f integrable_on $cbox\ a\ b \longleftrightarrow$
 $(\forall e > 0. \exists \gamma. gauge\ \gamma \wedge$
 $(\forall \mathcal{D}1\ \mathcal{D}2. \mathcal{D}1\ tagged\_division\_of\ (cbox\ a\ b) \wedge \gamma\ fine\ \mathcal{D}1 \wedge$
 $\mathcal{D}2\ tagged\_division\_of\ (cbox\ a\ b) \wedge \gamma\ fine\ \mathcal{D}2 \longrightarrow$
 $norm\ ((\sum (x,K) \in \mathcal{D}1. content\ K *_R f\ x) - (\sum (x,K) \in \mathcal{D}2. content\ K *_R$
 $f\ x)) < e))$

(**is** ? $l = (\forall e > 0. \exists \gamma. ?P\ e\ \gamma)$)

proof (intro iffI allI impI)

assume ? l

then obtain y

where $y: \bigwedge e. e > 0 \implies$

$\exists \gamma. gauge\ \gamma \wedge$

$(\forall \mathcal{D}. \mathcal{D}\ tagged\_division\_of\ cbox\ a\ b \wedge \gamma\ fine\ \mathcal{D} \longrightarrow$

$norm\ ((\sum (x,K) \in \mathcal{D}. content\ K *_R f\ x) - y) < e)$

by (auto simp: integrable_on_def has_integral)

show $\exists \gamma. ?P\ e\ \gamma$ **if** $e > 0$ **for** e

proof –

have $e/2 > 0$ **using** that **by** auto

with y **obtain** γ **where** $gauge\ \gamma$

and $\gamma: \bigwedge \mathcal{D}. \mathcal{D}\ tagged\_division\_of\ cbox\ a\ b \wedge \gamma\ fine\ \mathcal{D} \implies$

$norm\ ((\sum (x,K) \in \mathcal{D}. content\ K *_R f\ x) - y) < e/2$

by meson

show ?thesis

apply (rule_tac $x=\gamma$ in exI, clarsimp simp: ‹ $gauge\ \gamma$ ›)

by (blast intro!: $\gamma\ dist\_triangle\_half\_l$ [where $y=y, unfolded\ dist\_norm$])

qed

next

assume $\forall e > 0. \exists \gamma. ?P\ e\ \gamma$

then have $\forall n :: nat. \exists \gamma. ?P\ (1 / (n + 1))\ \gamma$

by auto

then obtain $\gamma :: nat \Rightarrow 'n \Rightarrow 'n$ **set** **where** $\gamma:$

$\bigwedge m. gauge\ (\gamma\ m)$

$\bigwedge m\ \mathcal{D}1\ \mathcal{D}2. \llbracket \mathcal{D}1\ tagged\_division\_of\ cbox\ a\ b;$

$\gamma\ m\ fine\ \mathcal{D}1; \mathcal{D}2\ tagged\_division\_of\ cbox\ a\ b; \gamma\ m\ fine\ \mathcal{D}2 \rrbracket$

$\implies norm\ ((\sum (x,K) \in \mathcal{D}1. content\ K *_R f\ x) - (\sum (x,K) \in \mathcal{D}2.$

$content\ K *_R f\ x))$

$< 1 / (m + 1)$

by metis

have $gauge\ (\lambda x. \bigcap \{\gamma\ i\ x \mid i. i \in \{0..n\}\})$ **for** n

using γ **by** (intro gauge_Inter) auto

```

then have  $\forall n. \exists p. p \text{ tagged\_division\_of } (cbox\ a\ b) \wedge (\lambda x. \bigcap \{\gamma\ i\ x \mid i. i \in \{0..n\}\}) \text{ fine } p$ 
by (meson fine_division_exists)
then obtain  $p$  where  $p: \bigwedge z. p\ z \text{ tagged\_division\_of } cbox\ a\ b$ 
 $\bigwedge z. (\lambda x. \bigcap \{\gamma\ i\ x \mid i. i \in \{0..z\}\}) \text{ fine } p\ z$ 
by meson
have  $dp: \bigwedge i\ n. i \leq n \implies \gamma\ i \text{ fine } p\ n$ 
using p_unfolding fine_Inter
using atLeastAtMost_iff by blast
have Cauchy  $(\lambda n. \text{sum } (\lambda(x,K). \text{content } K *_R (f\ x))\ (p\ n))$ 
proof (rule CauchyI)
  fix  $e::\text{real}$ 
  assume  $0 < e$ 
  then obtain  $N$  where  $N \neq 0$  and  $N: \text{inverse } (\text{real } N) < e$ 
    using real_arch_inverse[of e] by blast
    show  $\exists M. \forall m \geq M. \forall n \geq M. \text{norm } ((\sum (x,K) \in p\ m. \text{content } K *_R f\ x) - (\sum (x,K) \in p\ n. \text{content } K *_R f\ x)) < e$ 
    proof (intro exI allI impI)
      fix  $m\ n$ 
      assume  $mn: N \leq m\ N \leq n$ 
      have  $\text{norm } ((\sum (x,K) \in p\ m. \text{content } K *_R f\ x) - (\sum (x,K) \in p\ n. \text{content } K *_R f\ x)) < 1 / (\text{real } N + 1)$ 
      by (simp add: p(1) dp mn  $\gamma$ )
      also have  $\dots < e$ 
      using  $N \langle N \neq 0 \rangle \langle 0 < e \rangle$  by (auto simp: field_simps)
      finally show  $\text{norm } ((\sum (x,K) \in p\ m. \text{content } K *_R f\ x) - (\sum (x,K) \in p\ n. \text{content } K *_R f\ x)) < e$  .
    qed
  qed
then obtain  $y$  where  $y: \exists no. \forall n \geq no. \text{norm } ((\sum (x,K) \in p\ n. \text{content } K *_R f\ x) - y) < r$  if  $r > 0$  for  $r$ 
by (auto simp: convergent_eq_Cauchy[symmetric] dest: LIMSEQ_D)
show ?l
  unfolding integrable_on_def has_integral
proof (rule_tac x=y in exI, clarify)
    fix  $e::\text{real}$ 
    assume  $e > 0$ 
    then have  $e2: e/2 > 0$  by auto
    then obtain  $N1::\text{nat}$  where  $N1: N1 \neq 0 \text{ inverse } (\text{real } N1) < e/2$ 
    using real_arch_inverse by blast
    obtain  $N2::\text{nat}$  where  $N2: \bigwedge n. n \geq N2 \implies \text{norm } ((\sum (x,K) \in p\ n. \text{content } K *_R f\ x) - y) < e/2$ 
    using  $y[OF\ e2]$  by metis
    show  $\exists \gamma. \text{gauge } \gamma \wedge$ 
       $(\forall \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } (cbox\ a\ b) \wedge \gamma \text{ fine } \mathcal{D} \longrightarrow$ 
 $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f\ x) - y) < e)$ 
    proof (intro exI conjI allI impI)
      show gauge  $(\gamma\ (N1+N2))$ 
      using  $\gamma$  by auto

```

```

  show norm (( $\sum (x,K) \in q.$  content  $K *_R f x$ ) -  $y$ ) <  $e$ 
    if  $q$  tagged_division_of cbox  $a b \wedge \gamma (N1+N2)$  fine  $q$  for  $q$ 
  proof (rule norm_triangle_half_r)
    have norm (( $\sum (x,K) \in p (N1+N2).$  content  $K *_R f x$ ) - ( $\sum (x,K) \in q.$ 
    content  $K *_R f x$ ))
      < 1 / (real (N1+N2) + 1)
    by (rule  $\gamma$ ; simp add: dp  $p$  that)
    also have ... <  $e/2$ 
      using N1 <0 <  $e$ > by (auto simp: field_simps intro: less_le_trans)
    finally show norm (( $\sum (x,K) \in p (N1+N2).$  content  $K *_R f x$ ) - ( $\sum (x,K) \in q.$ 
    content  $K *_R f x$ )) <  $e/2$  .
    show norm (( $\sum (x,K) \in p (N1+N2).$  content  $K *_R f x$ ) -  $y$ ) <  $e/2$ 
      using N2 le_add_same_cancel2 by blast
  qed
qed
qed
qed

```

6.15.5 Additivity of integral on abutting intervals

lemma tagged_division_split_left_inj_content:

assumes $\mathcal{D}$: $\mathcal{D}$ tagged_division_of S

and $(x1, K1) \in \mathcal{D}$ $(x2, K2) \in \mathcal{D}$ $K1 \neq K2$ $K1 \cap \{x. x \cdot k \leq c\} = K2 \cap \{x.$
 $x \cdot k \leq c\}$ $k \in \text{Basis}$

shows content $(K1 \cap \{x. x \cdot k \leq c\}) = 0$

proof -

from tagged_division_ofD(4)[OF $\mathcal{D}$ $\langle (x1, K1) \in \mathcal{D} \rangle$] obtain $a b$ where $K1$:
 $K1 = \text{cbox } a b$

by auto

then have interior $(K1 \cap \{x. x \cdot k \leq c\}) = \{\}$

by (metis tagged_division_split_left_inj assms)

then show ?thesis

unfolding $K1$ interval_split[OF $\langle k \in \text{Basis} \rangle$] by (auto simp: content_eq_0_interior)

qed

lemma tagged_division_split_right_inj_content:

assumes $\mathcal{D}$: $\mathcal{D}$ tagged_division_of S

and $(x1, K1) \in \mathcal{D}$ $(x2, K2) \in \mathcal{D}$ $K1 \neq K2$ $K1 \cap \{x. x \cdot k \geq c\} = K2 \cap \{x.$
 $x \cdot k \geq c\}$ $k \in \text{Basis}$

shows content $(K1 \cap \{x. x \cdot k \geq c\}) = 0$

proof -

from tagged_division_ofD(4)[OF $\mathcal{D}$ $\langle (x1, K1) \in \mathcal{D} \rangle$] obtain $a b$ where $K1$:
 $K1 = \text{cbox } a b$

by auto

then have interior $(K1 \cap \{x. c \leq x \cdot k\}) = \{\}$

by (metis tagged_division_split_right_inj assms)

then show ?thesis

unfolding $K1$ interval_split[OF $\langle k \in \text{Basis} \rangle$]

by (auto simp: content_eq_0_interior)

qed

proposition *has_integral_split*:

fixes $f :: 'a::euclidean\_space \Rightarrow 'b::real\_normed\_vector$

assumes $fi: (f \text{ has\_integral } i) (cbox\ a\ b \cap \{x. x \cdot k \leq c\})$

and $fj: (f \text{ has\_integral } j) (cbox\ a\ b \cap \{x. x \cdot k \geq c\})$

and $k: k \in Basis$

shows $(f \text{ has\_integral } (i + j)) (cbox\ a\ b)$

unfolding *has_integral*

proof *clarify*

fix $e::real$

assume $0 < e$

then have $e: e/2 > 0$

by *auto*

obtain $\gamma 1$ **where** $\gamma 1: gauge\ \gamma 1$

and $\gamma 1norm$:

$\bigwedge \mathcal{D}. [\mathcal{D}\ \text{tagged\_division\_of}\ cbox\ a\ b \cap \{x. x \cdot k \leq c\};\ \gamma 1\ \text{fine}\ \mathcal{D}]$

$\implies norm\ ((\sum (x, K) \in \mathcal{D}. content\ K *_{\mathbb{R}} f\ x) - i) < e/2$

apply $(rule\ has\_integralD[OF\ fi[unboxed\_interval\_split[OF\ k]]\ e])$

apply $(simp\ add: interval\_split[symmetric]\ k)$

done

obtain $\gamma 2$ **where** $\gamma 2: gauge\ \gamma 2$

and $\gamma 2norm$:

$\bigwedge \mathcal{D}. [\mathcal{D}\ \text{tagged\_division\_of}\ cbox\ a\ b \cap \{x. c \leq x \cdot k\};\ \gamma 2\ \text{fine}\ \mathcal{D}]$

$\implies norm\ ((\sum (x, k) \in \mathcal{D}. content\ k *_{\mathbb{R}} f\ x) - j) < e/2$

apply $(rule\ has\_integralD[OF\ fj[unboxed\_interval\_split[OF\ k]]\ e])$

apply $(simp\ add: interval\_split[symmetric]\ k)$

done

let $? \gamma = \lambda x. if\ x \cdot k = c\ then\ (\gamma 1\ x \cap \gamma 2\ x)\ else\ ball\ x\ |x \cdot k - c| \cap \gamma 1\ x \cap \gamma 2\ x$

have *gauge* $? \gamma$

using $\gamma 1\ \gamma 2$ **unfolding** *gauge_def* **by** *auto*

then show $\exists \gamma. gauge\ \gamma \wedge$

$(\forall \mathcal{D}. \mathcal{D}\ \text{tagged\_division\_of}\ cbox\ a\ b \wedge \gamma\ \text{fine}\ \mathcal{D} \longrightarrow$

$norm\ ((\sum (x, k) \in \mathcal{D}. content\ k *_{\mathbb{R}} f\ x) - (i + j)) < e)$

proof $(rule\_tac\ x=? \gamma\ in\ exI, safe)$

fix p

assume $p: p\ \text{tagged\_division\_of}\ (cbox\ a\ b)\ \text{and}\ ? \gamma\ \text{fine}\ p$

have $ab\_eqp: cbox\ a\ b = \bigcup \{K. \exists x. (x, K) \in p\}$

using p **by** *blast*

have $xk\_le\_c: x \cdot k \leq c\ \text{if}\ as: (x, K) \in p\ \text{and}\ K: K \cap \{x. x \cdot k \leq c\} \neq \{\}$ **for** x

K

proof $(rule\ ccontr)$

assume $**:$ $\neg x \cdot k \leq c$

then have $K \subseteq ball\ x\ |x \cdot k - c|$

using $\langle ? \gamma\ \text{fine}\ p \rangle$ **as** **by** $(fastforce\ simp: not\_le\ algebra\_simps)$

with K **obtain** y **where** $y: y \in ball\ x\ |x \cdot k - c|\ y \cdot k \leq c$

by *blast*

then have $|x \cdot k - y \cdot k| < |x \cdot k - c|$


```

    using Basis_le_norm[OF k, of x - y]
  by (auto simp add: dist_norm inner_diff_left intro: le_less_trans)
with y show False
  using ** by (auto simp add: field_simps)
qed
have xk_ge_c:  $x \cdot k \geq c$  if as:  $(x, K) \in p$  and  $K: K \cap \{x. x \cdot k \geq c\} \neq \{\}$  for  $x$ 
K
proof (rule ccontr)
  assume **:  $\neg x \cdot k \geq c$ 
  then have  $K \subseteq \text{ball } x \mid x \cdot k - c \mid$ 
    using  $\langle ?\gamma \text{ fine } p \rangle$  as by (fastforce simp: not_le algebra_simps)
  with K obtain y where  $y: y \in \text{ball } x \mid x \cdot k - c \mid y \cdot k \geq c$ 
    by blast
  then have  $|x \cdot k - y \cdot k| < |x \cdot k - c|$ 
    using Basis_le_norm[OF k, of x - y]
    by (auto simp add: dist_norm inner_diff_left intro: le_less_trans)
  with y show False
    using ** by (auto simp add: field_simps)
qed
have fin_finite: finite  $\{(x, f K) \mid x K. (x, K) \in s \wedge P x K\}$ 
  if finite s for s and  $f :: 'a \text{ set} \Rightarrow 'a \text{ set}$  and  $P :: 'a \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ 
proof -
  from that have finite  $((\lambda(x, K). (x, f K)) \text{ ` } s)$ 
    by auto
  then show ?thesis
    by (rule rev_finite_subset) auto
qed
{ fix  $\mathcal{G} :: 'a \text{ set} \Rightarrow 'a \text{ set}$ 
  fix  $i :: 'a \times 'a \text{ set}$ 
  assume  $i \in (\lambda(x, k). (x, \mathcal{G} k)) \text{ ` } p - \{(x, \mathcal{G} k) \mid x k. (x, k) \in p \wedge \mathcal{G} k \neq \{\}\}$ 
  then obtain  $x K$  where  $xk: i = (x, \mathcal{G} K) \quad (x, K) \in p$ 
     $(x, \mathcal{G} K) \notin \{(x, \mathcal{G} K) \mid x K. (x, K) \in p \wedge \mathcal{G} K \neq \{\}\}$ 
    by auto
  have content  $(\mathcal{G} K) = 0$ 
    using xk using content_empty by auto
  then have  $(\lambda(x, K). \text{content } K *_R f x) i = 0$ 
    unfolding xk split_conv by auto
} note [simp] = this
have finite p
  using p by blast
let ?M1 =  $\{(x, K \cap \{x. x \cdot k \leq c\}) \mid x K. (x, K) \in p \wedge K \cap \{x. x \cdot k \leq c\} \neq \{\}\}$ 
have  $\gamma 1\_fine: \gamma 1 \text{ fine } ?M1$ 
  using  $\langle ?\gamma \text{ fine } p \rangle$  by (fastforce simp: fine_def split: if_split_asm)
have norm  $((\sum (x, k) \in ?M1. \text{content } k *_R f x) - i) < e/2$ 
proof (rule  $\gamma 1 \text{ norm } [OF \text{ tagged\_division\_ofI } \gamma 1\_fine]$ )
  show finite ?M1
    by (rule fin_finite) (use p in blast)
  show  $\bigcup \{k. \exists x. (x, k) \in ?M1\} = \text{cbox } a \ b \cap \{x. x \cdot k \leq c\}$ 
    by (auto simp: ab_eqp)

```

```

fix  $x\ L$ 
assume  $xL: (x, L) \in ?M1$ 
then obtain  $x'\ L'$  where  $xL': x = x'\ L = L' \cap \{x. x \cdot k \leq c\}$ 
 $(x', L') \in p\ L' \cap \{x. x \cdot k \leq c\} \neq \{\}$ 

  by blast
then obtain  $a'\ b'$  where  $ab': L' = cbox\ a'\ b'$ 
  using  $p$  by blast
show  $x \in L\ L \subseteq cbox\ a\ b \cap \{x. x \cdot k \leq c\}$ 
  using  $p\ xk\_le\_c\ xL'$  by auto
show  $\exists a\ b. L = cbox\ a\ b$ 
  using  $p\ xL'\ ab'$  by (auto simp add: interval_split[OF k,where c=c])

fix  $y\ R$ 
assume  $yR: (y, R) \in ?M1$ 
then obtain  $y'\ R'$  where  $yR': y = y'\ R = R' \cap \{x. x \cdot k \leq c\}$ 
 $(y', R') \in p\ R' \cap \{x. x \cdot k \leq c\} \neq \{\}$ 

  by blast
assume  $as: (x, L) \neq (y, R)$ 
show  $interior\ L \cap interior\ R = \{\}$ 
proof (cases  $L' = R' \longrightarrow x' = y'$ )
  case False
    have  $interior\ R' = \{\}$ 
    by (metis (no_types) False Pair_inject inf.idem tagged_division_ofD(5))
  [OF  $p$ ]  $xL'(\mathfrak{I})\ yR'(\mathfrak{I})$ 
    then show ?thesis
    using  $yR'$  by simp
  next
  case True
    then have  $L' \neq R'$ 
    using  $as$  unfolding  $xL'\ yR'$  by auto
    have  $interior\ L' \cap interior\ R' = \{\}$ 
    by (metis (no_types) Pair_inject  $\langle L' \neq R' \rangle\ p\ tagged\_division\_ofD$ (5))
   $xL'(\mathfrak{I})\ yR'(\mathfrak{I})$ 
    then show ?thesis
    using  $xL'(\mathfrak{I})\ yR'(\mathfrak{I})$  by auto
  qed
qed
moreover
let  $?M2 = \{(x, K \cap \{x. x \cdot k \geq c\}) \mid x\ K. (x, K) \in p \wedge K \cap \{x. x \cdot k \geq c\} \neq \{\}\}$ 
have  $\gamma2\_fine: \gamma2\ fine\ ?M2$ 
  using  $\langle ?\gamma\ fine\ p \rangle$  by (fastforce simp: fine_def split: if_split_asm)
have  $norm\ ((\sum (x, k) \in ?M2. content\ k *_{\mathbb{R}} f\ x) - j) < e/2$ 
proof (rule  $\gamma2norm\ [OF\ tagged\_division\_ofI\ \gamma2\_fine]$ )
  show finite  $?M2$ 
    by (rule fin_finite) (use  $p$  in blast)
show  $\bigcup \{k. \exists x. (x, k) \in ?M2\} = cbox\ a\ b \cap \{x. x \cdot k \geq c\}$ 
  by (auto simp: ab_eqp)

```

```

fix x L
assume xL: (x, L) ∈ ?M2
then obtain x' L' where xL': x = x' L = L' ∩ {x. x • k ≥ c}
      (x', L') ∈ p L' ∩ {x. x • k ≥ c} ≠ {}

  by blast
then obtain a' b' where ab': L' = cbox a' b'
  using p by blast
show x ∈ L L ⊆ cbox a b ∩ {x. x • k ≥ c}
  using p xk_ge_c xL' by auto
show ∃ a b. L = cbox a b
  using p xL' ab' by (auto simp add: interval_split[OF k, where c=c])

fix y R
assume yR: (y, R) ∈ ?M2
then obtain y' R' where yR': y = y' R = R' ∩ {x. x • k ≥ c}
      (y', R') ∈ p R' ∩ {x. x • k ≥ c} ≠ {}

  by blast
assume as: (x, L) ≠ (y, R)
show interior L ∩ interior R = {}
proof (cases L' = R' ⟶ x' = y')
  case False
  have interior R' = {}
    by (metis (no_types) False Pair_inject inf.idem tagged_division_ofD(5)
      [OF p] xL'(3) yR'(3))
  then show ?thesis
    using yR' by simp
  next
  case True
  then have L' ≠ R'
    using as unfolding xL' yR' by auto
  have interior L' ∩ interior R' = {}
    by (metis (no_types) Pair_inject ⟨L' ≠ R'⟩ p tagged_division_ofD(5)
      xL'(3) yR'(3))
  then show ?thesis
    using xL'(2) yR'(2) by auto
qed
qed
ultimately
have norm (((∑ (x,K) ∈ ?M1. content K *R f x) - i) + ((∑ (x,K) ∈ ?M2.
content K *R f x) - j)) < e/2 + e/2
  using norm_add_less by blast
moreover have ((∑ (x,K) ∈ ?M1. content K *R f x) - i) +
  ((∑ (x,K) ∈ ?M2. content K *R f x) - j) =
  (∑ (x, ka) ∈ p. content ka *R f x) - (i + j)
proof -
  have eq0: ∧ x y. x = (0::real) ⟹ x *R (y::'b) = 0
    by auto
  have cont_eq: ∧ g. (λ(x,l). content l *R f x) ∘ (λ(x,l). (x,g l)) = (λ(x,l).
content (g l) *R f x)

```

```

    by auto
    have *:  $\bigwedge \mathcal{G} :: 'a \text{ set} \Rightarrow 'a \text{ set}.$ 
       $(\sum (x,K) \in \{(x, \mathcal{G} K) \mid x K. (x,K) \in p \wedge \mathcal{G} K \neq \{\}\}. \text{content } K *_R f$ 
 $x) =$ 
       $(\sum (x,K) \in (\lambda(x,K). (x, \mathcal{G} K)) \text{ ' } p. \text{content } K *_R f x)$ 
    by (rule sum.mono_neutral_left) (auto simp: <finite p>)
    have  $((\sum (x, k) \in ?M1. \text{content } k *_R f x) - i) + ((\sum (x, k) \in ?M2. \text{content } k$ 
 $*_R f x) - j) =$ 
 $(\sum (x, k) \in ?M1. \text{content } k *_R f x) + (\sum (x, k) \in ?M2. \text{content } k *_R f x) -$ 
 $(i + j)$ 
    by auto
    moreover have  $\dots = (\sum (x,K) \in p. \text{content } (K \cap \{x. x \cdot k \leq c\}) *_R f x) +$ 
 $(\sum (x,K) \in p. \text{content } (K \cap \{x. c \leq x \cdot k\}) *_R f x) - (i + j)$ 
    unfolding *
    apply (subst (1 2) sum.reindex_nontrivial)
    apply (auto intro!: k p eq0 tagged_division_split_left_inj_content
      tagged_division_split_right_inj_content
      simp: cont_eq <finite p>)
    done
    moreover have  $\bigwedge x. x \in p \Longrightarrow (\lambda(a,B). \text{content } (B \cap \{a. a \cdot k \leq c\}) *_R f$ 
 $a) x +$ 
 $(\lambda(a,B). \text{content } (B \cap \{a. c \leq a \cdot k\}) *_R f a) x =$ 
 $(\lambda(a,B). \text{content } B *_R f a) x$ 
  proof clarify
    fix a B
    assume  $(a, B) \in p$ 
    with p obtain u v where  $uv: B = \text{cbox } u \text{ } v$  by blast
    then show  $\text{content } (B \cap \{x. x \cdot k \leq c\}) *_R f a + \text{content } (B \cap \{x. c \leq x$ 
 $\cdot k\}) *_R f a = \text{content } B *_R f a$ 
      by (auto simp: scaleR_left_distrib uv content_split[OF k, of u v c])
    qed
    ultimately show ?thesis
      by (auto simp: sum.distrib[symmetric])
    qed
    ultimately show  $\text{norm } ((\sum (x, k) \in p. \text{content } k *_R f x) - (i + j)) < e$ 
      by auto
    qed
  qed

```

6.15.6 A sort of converse, integrability on subintervals

lemma *has_integral_separate_sides:*

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{real\_normed\_vector}$

assumes $f: (f \text{ has\_integral } i) (\text{cbox } a \text{ } b)$

and $e > 0$

and $k: k \in \text{Basis}$

obtains d **where** *gauge* d

$\forall p1 \ p2. p1 \text{ tagged\_division\_of } (\text{cbox } a \text{ } b \cap \{x. x \cdot k \leq c\}) \wedge d \text{ fine } p1 \wedge$
 $p2 \text{ tagged\_division\_of } (\text{cbox } a \text{ } b \cap \{x. x \cdot k \geq c\}) \wedge d \text{ fine } p2 \longrightarrow$

```

      norm ((sum (λ(x,k). content k *R f x) p1 + sum (λ(x,k). content k *R f x)
p2) - i) < e
proof -
  obtain γ where d: gauge γ
  ∧p. [[p tagged_division_of cbox a b; γ fine p]]
    ⇒ norm ((sum (x, k) ∈ p. content k *R f x) - i) < e
  using has_integralD[OF f <e> 0] by metis
  { fix p1 p2
    assume tdiv1: p1 tagged_division_of (cbox a b) ∩ {x. x • k ≤ c} and γ fine
  p1
    note p1=tagged_division_ofD[OF this(1)]
    assume tdiv2: p2 tagged_division_of (cbox a b) ∩ {x. c ≤ x • k} and γ fine
  p2
    note p2=tagged_division_ofD[OF this(1)]
    note tagged_division_Un_interval[OF tdiv1 tdiv2]
    note p12 = tagged_division_ofD[OF this] this
    { fix a b
      assume ab: (a, b) ∈ p1 ∩ p2
      have (a, b) ∈ p1
      using ab by auto
      obtain u v where uv: b = cbox u v
      using ⟨(a, b) ∈ p1⟩ p1(4) by mouna
      have b ⊆ {x. x•k = c}
      using ab p1(3)[of a b] p2(3)[of a b] by fastforce
      moreover
      have interior {x::'a. x • k = c} = {}
      proof (rule ccontr)
        assume ¬ ?thesis
        then obtain x where x: x ∈ interior {x::'a. x•k = c}
        by auto
        then obtain ε where 0 < ε and ε: ball x ε ⊆ {x. x • k = c}
        using mem_interior by metis
        have x: x•k = c
        using x interior_subset by fastforce
        have *: ∧i. i ∈ Basis ⇒ |(x - (x + (ε/2) *R k)) • i| = (if i = k then ε/2
else 0)
        using ⟨0 < ε⟩ k by (auto simp: inner_simps inner_not_same_Basis)
        have (sum i ∈ Basis. |(x - (x + (ε/2) *R k)) • i|) =
          (sum i ∈ Basis. (if i = k then ε/2 else 0))
        using * by (blast intro: sum.cong)
        also have ... < ε
        by (subst sum.delta) (use ⟨0 < ε⟩ in auto)
        finally have x + (ε/2) *R k ∈ ball x ε
        unfolding mem_ball dist_norm by (rule le_less_trans[OF norm_le_l1])
        then have x + (ε/2) *R k ∈ {x. x•k = c}
        using ε by auto
        then show False
        using ⟨0 < ε⟩ x k by (auto simp: inner_simps)
      qed
    }
  }

```

```

ultimately have content b = 0
  unfolding uv content_eq_0_interior
  using interior_mono by blast
then have content b *R f a = 0
  by auto
}
then have norm (( $\sum (x, k) \in p1. \text{content } k *_{\mathcal{R}} f x$ ) + ( $\sum (x, k) \in p2. \text{content } k *_{\mathcal{R}} f x$ ) - i) =
  norm (( $\sum (x, k) \in p1 \cup p2. \text{content } k *_{\mathcal{R}} f x$ ) - i)
  by (subst sum.union_inter_neutral) (auto simp: p1 p2)
also have ... < e
  using d(2) p12 by (simp add: fine_Un k ⟨ $\gamma$  fine p1⟩ ⟨ $\gamma$  fine p2⟩)
finally have norm (( $\sum (x, k) \in p1. \text{content } k *_{\mathcal{R}} f x$ ) + ( $\sum (x, k) \in p2. \text{content } k *_{\mathcal{R}} f x$ ) - i) < e .
}
then show ?thesis
  using d(1) that by auto
qed

```

lemma integrable_split [intro]:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::{real_normed_vector, complete_space}
assumes f: f integrable_on cbox a b
  and k: k  $\in$  Basis
shows f integrable_on (cbox a b  $\cap$  {x. x•k  $\leq$  c}) (is ?thesis1)
  and f integrable_on (cbox a b  $\cap$  {x. x•k  $\geq$  c}) (is ?thesis2)
proof -
  obtain y where y: (f has_integral y) (cbox a b)
  using f by blast
  define a' where a' = ( $\sum i \in \text{Basis}. (\text{if } i = k \text{ then } \max (a \cdot k) \ c \text{ else } a \cdot i) *_{\mathcal{R}} i$ )
  define b' where b' = ( $\sum i \in \text{Basis}. (\text{if } i = k \text{ then } \min (b \cdot k) \ c \text{ else } b \cdot i) *_{\mathcal{R}} i$ )
  have  $\exists d. \text{gauge } d \wedge$ 
    ( $\forall p1 \ p2. p1 \text{ tagged\_division\_of } \text{cbox } a \ b \cap \{x. x \cdot k \leq c\} \wedge d \text{ fine } p1 \wedge$ 
       $p2 \text{ tagged\_division\_of } \text{cbox } a \ b \cap \{x. x \cdot k \leq c\} \wedge d \text{ fine } p2 \longrightarrow$ 
       $\text{norm } ((\sum (x, K) \in p1. \text{content } K *_{\mathcal{R}} f x) - (\sum (x, K) \in p2. \text{content } K *_{\mathcal{R}} f x)) < e$ )
  if e > 0 for e
  proof -
    have e/2 > 0 using that by auto
    with has_integral_separate_sides[OF y this k, of c]
    obtain d
    where gauge d
      and d:  $\bigwedge p1 \ p2. \llbracket p1 \text{ tagged\_division\_of } \text{cbox } a \ b \cap \{x. x \cdot k \leq c\}; d \text{ fine } p1;$ 
       $p2 \text{ tagged\_division\_of } \text{cbox } a \ b \cap \{x. c \leq x \cdot k\}; d \text{ fine } p2 \rrbracket$ 
       $\implies \text{norm } ((\sum (x, K) \in p1. \text{content } K *_{\mathcal{R}} f x) + (\sum (x, K) \in p2. \text{content } K *_{\mathcal{R}} f x) - y) < e/2$ 
    by metis
    show ?thesis
      proof (rule_tac x=d in exI, clarsimp simp add: ⟨gauge d⟩)

```

```

fix p1 p2
assume as: p1 tagged_division_of (cbox a b)  $\cap \{x. x \cdot k \leq c\}$  d fine p1
           p2 tagged_division_of (cbox a b)  $\cap \{x. x \cdot k \leq c\}$  d fine p2
show norm (( $\sum (x, k) \in p1. \text{content } k *_R f x$ ) - ( $\sum (x, k) \in p2. \text{content } k *_R f x$ )) < e
proof (rule fine_division_exists[OF  $\langle \text{gauge } d \rangle$ , of a' b])
  fix p
  assume p tagged_division_of cbox a' b d fine p
  then show ?thesis
    using as norm_triangle_half_l[OF d[of p1 p] d[of p2 p]]
    unfolding interval_split[OF k] b'_def[symmetric] a'_def[symmetric]
    by (auto simp add: algebra_simps)
qed
qed
qed
with f show ?thesis1
  by (simp add: interval_split[OF k] integrable_Cauchy)
have  $\exists d. \text{gauge } d \wedge$ 
  ( $\forall p1 p2. p1 \text{ tagged\_division\_of } cbox a b \cap \{x. x \cdot k \geq c\} \wedge d \text{ fine } p1 \wedge$ 
    $p2 \text{ tagged\_division\_of } cbox a b \cap \{x. x \cdot k \geq c\} \wedge d \text{ fine } p2 \implies$ 
    $\text{norm } ((\sum (x, K) \in p1. \text{content } K *_R f x) - (\sum (x, K) \in p2. \text{content } K *_R f x)) < e$ )
  if e > 0 for e
proof -
  have e/2 > 0 using that by auto
  with has_integral_separate_sides[OF y this k, of c]
  obtain d
    where gauge d
      and d:  $\bigwedge p1 p2. \llbracket p1 \text{ tagged\_division\_of } cbox a b \cap \{x. x \cdot k \leq c\}; d \text{ fine } p1;$ 
            $p2 \text{ tagged\_division\_of } cbox a b \cap \{x. c \leq x \cdot k\}; d \text{ fine } p2 \rrbracket$ 
            $\implies \text{norm } ((\sum (x, K) \in p1. \text{content } K *_R f x) + (\sum (x, K) \in p2. \text{content } K *_R f x) - y) < e/2$ 
    by metis
  show ?thesis
    proof (rule_tac x=d in exI, clarsimp simp add:  $\langle \text{gauge } d \rangle$ )
      fix p1 p2
      assume as: p1 tagged_division_of (cbox a b)  $\cap \{x. x \cdot k \geq c\}$  d fine p1
           p2 tagged_division_of (cbox a b)  $\cap \{x. x \cdot k \geq c\}$  d fine p2
      show norm (( $\sum (x, k) \in p1. \text{content } k *_R f x$ ) - ( $\sum (x, k) \in p2. \text{content } k *_R f x$ )) < e
      proof (rule fine_division_exists[OF  $\langle \text{gauge } d \rangle$ , of a b])
        fix p
        assume p tagged_division_of cbox a b' d fine p
        then show ?thesis
          using as norm_triangle_half_l[OF d[of p p1] d[of p p2]]
          unfolding interval_split[OF k] b'_def[symmetric] a'_def[symmetric]
          by (auto simp add: algebra_simps)
        qed
      qed
    qed
  qed

```

1812

```

    qed
  qed
  with f show ?thesis2
    by (simp add: interval_split[OF k] integrable_Cauchy)
  qed

lemma operative_integralI:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::banach
  shows operative (lift_option (+)) (Some 0)
    ( $\lambda i.$  if f integrable_on i then Some (integral i f) else None)
  proof -
    interpret comm_monoid lift_option plus Some (0::'b)
    by (rule comm_monoid_lift_option)
      (rule add.comm_monoid_axioms)
    show ?thesis
    proof
      fix a b c
      fix k :: 'a
      assume k: k  $\in$  Basis
      show (if f integrable_on cbox a b then Some (integral (cbox a b) f) else None)
      =
        lift_option (+) (if f integrable_on cbox a b  $\cap$  {x. x  $\cdot$  k  $\leq$  c} then Some
        (integral (cbox a b  $\cap$  {x. x  $\cdot$  k  $\leq$  c}) f) else None)
        (if f integrable_on cbox a b  $\cap$  {x. c  $\leq$  x  $\cdot$  k} then Some (integral (cbox a b
         $\cap$  {x. c  $\leq$  x  $\cdot$  k}) f) else None)
      proof (cases f integrable_on cbox a b)
        case True
        with k show ?thesis
        by (auto simp: integrable_split intro: integral_unique [OF has_integral_split[OF
        _ _ k]])
      next
        case False
        have  $\neg$  (f integrable_on cbox a b  $\cap$  {x. x  $\cdot$  k  $\leq$  c})  $\vee$   $\neg$  (f integrable_on cbox
        a b  $\cap$  {x. c  $\leq$  x  $\cdot$  k})
        proof (rule ccontr)
          assume  $\neg$  ?thesis
          then have f integrable_on cbox a b
            unfolding integrable_on_def
            apply (rule_tac x=integral (cbox a b  $\cap$  {x. x  $\cdot$  k  $\leq$  c}) f + integral (cbox
            a b  $\cap$  {x. x  $\cdot$  k  $\geq$  c}) f in exI)
            apply (auto intro: has_integral_split[OF _ _ k])
            done
          then show False
            using False by auto
        qed
        then show ?thesis
          using False by auto
      qed
    next

```



```

    fix a b :: 'a
    assume box a b = {}
    then show (if f integrable_on cbox a b then Some (integral (cbox a b) f) else
None) = Some 0
      using has_integral_null_eq
      by (auto simp: integrable_on_null content_eq_0_interior)
  qed
qed

```

6.15.7 Bounds on the norm of Riemann sums and the integral itself

```

lemma dsum_bound:
  assumes p: p division_of (cbox a b)
  and norm c ≤ e
  shows norm (sum (λl. content l *R c) p) ≤ e * content (cbox a b)
proof -
  have sumeq: (∑ i∈p. |content i|) = sum content p
  by simp
  have e: 0 ≤ e
  using assms(2) norm_ge_zero order_trans by blast
  have norm (sum (λl. content l *R c) p) ≤ (∑ i∈p. norm (content i *R c))
  using norm_sum by blast
  also have ... ≤ e * (∑ i∈p. |content i|)
  by (simp add: sum_distrib_left[symmetric] mult.commute assms(2) mult_right_mono
sum_nonneg)
  also have ... ≤ e * content (cbox a b)
  by (metis additive_content_division p eq_iff sumeq)
  finally show ?thesis .
qed

```

```

lemma rsum_bound:
  assumes p: p tagged_division_of (cbox a b)
  and ∀ x∈cbox a b. norm (f x) ≤ e
  shows norm (sum (λ(x,k). content k *R f x) p) ≤ e * content (cbox a b)
proof (cases cbox a b = {})
  case True show ?thesis
    using p unfolding True tagged_division_of_trivial by auto
  next
  case False
  then have e: e ≥ 0
  by (meson ex_in_conv assms(2) norm_ge_zero order_trans)
  have sum_le: sum (content ∘ snd) p ≤ content (cbox a b)
  unfolding additive_content_tagged_division[OF p, symmetric] split_def
  by (auto intro: eq_refl)
  have con: ∧ xk. xk ∈ p ⇒ 0 ≤ content (snd xk)
  using tagged_division_ofD(4) [OF p] content_pos_le
  by force
  have norm (sum (λ(x,k). content k *R f x) p) ≤ (∑ i∈p. norm (case i of (x, k)

```

```

⇒ content k *R f x))
  by (rule norm_sum)
  also have ... ≤ e * content (cbox a b)
  proof -
    have  $\bigwedge xk. xk \in p \implies \text{norm } (f \text{ (fst } xk)) \leq e$ 
      using assms(2) p tag_in_interval by force
    moreover have  $(\sum i \in p. |\text{content } (\text{snd } i)| * e) \leq e * \text{content } (\text{cbox } a \text{ } b)$ 
      unfolding sum_distrib_right[symmetric]
      using con sum_le by (auto simp: mult.commute intro: mult_left_mono [OF
    _ e])
    ultimately show ?thesis
      unfolding split_def norm_scaleR
      by (metis (no_types, lifting) mult_left_mono[OF _ abs_ge_zero] or-
der_trans[OF sum_mono])
    qed
  finally show ?thesis .
qed

```

```

lemma rsum_diff_bound:
  assumes p tagged_division_of (cbox a b)
  and  $\forall x \in \text{cbox } a \text{ } b. \text{norm } (f \text{ } x - g \text{ } x) \leq e$ 
  shows  $\text{norm } (\text{sum } (\lambda(x,k). \text{content } k *_{\text{R}} f \text{ } x) \text{ } p - \text{sum } (\lambda(x,k). \text{content } k *_{\text{R}} g \text{ } x) \text{ } p) \leq$ 
     $e * \text{content } (\text{cbox } a \text{ } b)$ 
  using order_trans[OF _ rsum_bound[OF assms]]
  by (simp add: split_def scaleR_diff_right sum_subtractf eq_refl)

```

```

lemma has_integral_bound:
  fixes f :: 'a::euclidean_space ⇒ 'b::real_normed_vector
  assumes 0 ≤ B
  and f: (f has_integral i) (cbox a b)
  and  $\bigwedge x. x \in \text{cbox } a \text{ } b \implies \text{norm } (f \text{ } x) \leq B$ 
  shows  $\text{norm } i \leq B * \text{content } (\text{cbox } a \text{ } b)$ 
proof (rule ccontr)
  assume ¬ ?thesis
  then have  $\text{norm } i - B * \text{content } (\text{cbox } a \text{ } b) > 0$ 
    by auto
  with f[unfolded has_integral]
  obtain γ where gauge γ and γ:
     $\bigwedge p. \llbracket p \text{ tagged\_division\_of } \text{cbox } a \text{ } b; \gamma \text{ fine } p \rrbracket$ 
     $\implies \text{norm } ((\sum (x, K) \in p. \text{content } K *_{\text{R}} f \text{ } x) - i) < \text{norm } i - B * \text{content } (\text{cbox } a \text{ } b)$ 
  by metis
  then obtain p where p: p tagged_division_of cbox a b and γ fine p
    using fine_division_exists by blast
  have  $\bigwedge s \in B. \text{norm } s \leq B \implies \neg \text{norm } (s - i) < \text{norm } i - B$ 
    unfolding not_less
    by (metis diff_left_mono dist_commute dist_norm norm_triangle_ineq2 or-
der_trans)

```

```

then show False
  using  $\gamma$  [OF p  $\langle \gamma$  fine p  $\rangle$ ] rsum_bound[OF p] assms by metis
qed

```

```

corollary integrable_bound:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::real_normed_vector
  assumes  $0 \leq B$ 
    and f integrable_on (cbox a b)
    and  $\bigwedge x. x \in \text{cbox } a \ b \implies \text{norm } (f \ x) \leq B$ 
  shows  $\text{norm } (\text{integral } (\text{cbox } a \ b) \ f) \leq B * \text{content } (\text{cbox } a \ b)$ 
by (metis integrable_integral has_integral_bound assms)

```

6.15.8 Similar theorems about relationship among components

```

lemma rsum_component_le:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes p: p tagged_division_of (cbox a b)
    and  $\bigwedge x. x \in \text{cbox } a \ b \implies (f \ x) \cdot i \leq (g \ x) \cdot i$ 
  shows  $(\sum (x, K) \in p. \text{content } K *_R f \ x) \cdot i \leq (\sum (x, K) \in p. \text{content } K *_R g \ x)$ 
   $\cdot i$ 
unfolding inner_sum_left
proof (rule sum_mono, clarify)
  fix x K
  assume ab:  $(x, K) \in p$ 
  with p obtain u v where K: K = cbox u v
  by blast
  then show  $(\text{content } K *_R f \ x) \cdot i \leq (\text{content } K *_R g \ x) \cdot i$ 
  by (metis ab assms inner_scaleR_left measure_nonneg mult_left_mono tag_in_interval)
qed

```

```

lemma has_integral_component_le:
  fixes f g :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes k: k  $\in$  Basis
  assumes (f has_integral i) S (g has_integral j) S
    and f_le_g:  $\bigwedge x. x \in S \implies (f \ x) \cdot k \leq (g \ x) \cdot k$ 
  shows  $i \cdot k \leq j \cdot k$ 
proof –
  have ik_le_jk:  $i \cdot k \leq j \cdot k$ 
    if f_i: (f has_integral i) (cbox a b)
    and g_j: (g has_integral j) (cbox a b)
    and le:  $\forall x \in \text{cbox } a \ b. (f \ x) \cdot k \leq (g \ x) \cdot k$ 
    for a b i and j :: 'b and f g :: 'a  $\Rightarrow$  'b
  proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then have  $*$ :  $0 < (i \cdot k - j \cdot k) / 3$ 
    by auto
  obtain  $\gamma 1$  where gauge  $\gamma 1$ 
    and  $\gamma 1$ :  $\bigwedge p. \llbracket p \text{ tagged\_division\_of } \text{cbox } a \ b; \gamma 1 \text{ fine } p \rrbracket$ 

```

```

       $\implies \text{norm } ((\sum (x, k) \in p. \text{content } k *_R f x) - i) < (i \cdot k - j \cdot k) / 3$ 
    using f_i[unfolded has_integral, rule_format, OF *] by fastforce
  obtain  $\gamma 2$  where gauge  $\gamma 2$ 
  and  $\gamma 2$ :  $\bigwedge p. \llbracket p \text{ tagged\_division\_of } \text{cbox } a \text{ } b; \gamma 2 \text{ fine } p \rrbracket$ 
       $\implies \text{norm } ((\sum (x, k) \in p. \text{content } k *_R g x) - j) < (i \cdot k - j \cdot k) / 3$ 
    using g_j[unfolded has_integral, rule_format, OF *] by fastforce
  obtain  $p$  where  $p$ :  $p \text{ tagged\_division\_of } \text{cbox } a \text{ } b$  and  $\gamma 1 \text{ fine } p$   $\gamma 2 \text{ fine } p$ 
    using fine_division_exists[OF gauge_Int[OF  $\langle \text{gauge } \gamma 1 \rangle \langle \text{gauge } \gamma 2 \rangle$ ], of a
b] unfolding fine_Int
  by metis
  then have  $|((\sum (x, k) \in p. \text{content } k *_R f x) - i) \cdot k| < (i \cdot k - j \cdot k) / 3$ 
     $|((\sum (x, k) \in p. \text{content } k *_R g x) - j) \cdot k| < (i \cdot k - j \cdot k) / 3$ 
  using le_less_trans[OF Basis_le_norm[OF k]] k  $\gamma 1$   $\gamma 2$  by metis+
  then show False
  unfolding inner_simps
  using rsum_component_le[OF p] le
  by (fastforce simp add: abs_real_def split: if_split_asm)
qed
show ?thesis
proof (cases  $\exists a \text{ } b. S = \text{cbox } a \text{ } b$ )
  case True
  with ik_le_jk assms show ?thesis
  by auto
next
  case False
  show ?thesis
  proof (rule ccontr)
    assume  $\neg i \cdot k \leq j \cdot k$ 
    then have  $ij: (i \cdot k - j \cdot k) / 3 > 0$ 
    by auto
    obtain  $B1$  where  $0 < B1$ 
    and  $B1: \bigwedge a \text{ } b. \text{ball } 0 \text{ } B1 \subseteq \text{cbox } a \text{ } b \implies$ 
       $\exists z. ((\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ has\_integral } z) (\text{cbox } a \text{ } b) \wedge$ 
       $\text{norm } (z - i) < (i \cdot k - j \cdot k) / 3$ 
    using has_integral_altD[OF _ False ij] assms by blast
    obtain  $B2$  where  $0 < B2$ 
    and  $B2: \bigwedge a \text{ } b. \text{ball } 0 \text{ } B2 \subseteq \text{cbox } a \text{ } b \implies$ 
       $\exists z. ((\lambda x. \text{if } x \in S \text{ then } g x \text{ else } 0) \text{ has\_integral } z) (\text{cbox } a \text{ } b) \wedge$ 
       $\text{norm } (z - j) < (i \cdot k - j \cdot k) / 3$ 
    using has_integral_altD[OF _ False ij] assms by blast
    have bounded ( $\text{ball } 0 \text{ } B1 \cup \text{ball } (0::'a) \text{ } B2$ )
    unfolding bounded_Un by (rule conjI bounded_ball)+
    from bounded_subset_cbox_symmetric[OF this]
    obtain  $a \text{ } b::'a$  where  $ab: \text{ball } 0 \text{ } B1 \subseteq \text{cbox } a \text{ } b$   $\text{ball } 0 \text{ } B2 \subseteq \text{cbox } a \text{ } b$ 
    by (meson Un_subset_iff)
    then obtain  $w1 \text{ } w2$  where  $\text{int\_}w1: ((\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ has\_integral } w1)$ 
    ( $\text{cbox } a \text{ } b$ )
    and  $\text{norm\_}w1: \text{norm } (w1 - i) < (i \cdot k - j \cdot k) / 3$ 
    and  $\text{int\_}w2: ((\lambda x. \text{if } x \in S \text{ then } g x \text{ else } 0) \text{ has\_integral } w2)$ 

```

```

(cbox a b)
      and norm_w2: norm (w2 - j) < (i • k - j • k) / 3
    using B1 B2 by blast
    have *:  $\bigwedge w1\ w2\ j\ i::\text{real}.\ |w1 - i| < (i - j) / 3 \implies |w2 - j| < (i - j) / 3$ 
 $\implies w1 \leq w2 \implies \text{False}$ 
    by (simp add: abs_real_def split: if_split_asm)
    have  $| (w1 - i) \cdot k | < (i \cdot k - j \cdot k) / 3$ 
       $| (w2 - j) \cdot k | < (i \cdot k - j \cdot k) / 3$ 
    using Basis_le_norm k le_less_trans norm_w1 norm_w2 by blast+
    moreover
    have  $w1 \cdot k \leq w2 \cdot k$ 
    using ik_le_jk int_w1 int_w2 f_le_g by auto
    ultimately show False
    unfolding inner_simps by(rule *)
  qed
qed
qed

```

```

lemma integral_component_le:
  fixes g f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes k  $\in$  Basis
    and f integrable_on S g integrable_on S
    and  $\bigwedge x. x \in S \implies (f\ x) \cdot k \leq (g\ x) \cdot k$ 
  shows  $(\text{integral } S\ f) \cdot k \leq (\text{integral } S\ g) \cdot k$ 
  using has_integral_component_le assms by blast

```

```

lemma has_integral_component_nonneg:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes k  $\in$  Basis
    and (f has_integral i) S
    and  $\bigwedge x. x \in S \implies 0 \leq (f\ x) \cdot k$ 
  shows  $0 \leq i \cdot k$ 
  using has_integral_component_le[OF assms(1) has_integral_0 assms(2)]
  using assms(3-)
  by auto

```

```

lemma integral_component_nonneg:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes k  $\in$  Basis
    and  $\bigwedge x. x \in S \implies 0 \leq (f\ x) \cdot k$ 
  shows  $0 \leq (\text{integral } S\ f) \cdot k$ 
proof (cases f integrable_on S)
case True show ?thesis
  using True assms has_integral_component_nonneg by blast
next
case False then show ?thesis by (simp add: not_integrable_integral)
qed

```

```

lemma has_integral_component_neg:

```

```

fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
assumes  $k \in Basis$ 
  and  $(f \text{ has\_integral } i) \ S$ 
  and  $\bigwedge x. x \in S \implies (f\ x) \cdot k \leq 0$ 
shows  $i \cdot k \leq 0$ 
using  $\text{has\_integral\_component\_le}[OF \text{ assms}(1,2) \text{ has\_integral\_0}] \text{ assms}(2-)$ 
by auto

```

```

lemma  $\text{has\_integral\_component\_lbound}$ :
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $(f \text{ has\_integral } i) \ (cbox\ a\ b)$ 
    and  $\forall x \in cbox\ a\ b. B \leq f(x) \cdot k$ 
    and  $k \in Basis$ 
  shows  $B * \text{content } (cbox\ a\ b) \leq i \cdot k$ 
  using  $\text{has\_integral\_component\_le}[OF \text{ assms}(3) \text{ has\_integral\_const } \text{assms}(1), of$ 
 $(\sum i \in Basis. B * i)::'b] \text{ assms}(2-)$ 
  by  $(auto \text{ simp add: field\_simps})$ 

```

```

lemma  $\text{has\_integral\_component\_ubound}$ :
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $(f \text{ has\_integral } i) \ (cbox\ a\ b)$ 
    and  $\forall x \in cbox\ a\ b. f\ x \cdot k \leq B$ 
    and  $k \in Basis$ 
  shows  $i \cdot k \leq B * \text{content } (cbox\ a\ b)$ 
  using  $\text{has\_integral\_component\_le}[OF \text{ assms}(3,1) \text{ has\_integral\_const, of } \sum i \in Basis.$ 
 $B * i] \text{ assms}(2-)$ 
  by  $(auto \text{ simp add: field\_simps})$ 

```

```

lemma  $\text{integral\_component\_lbound}$ :
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $f \text{ integrable\_on } cbox\ a\ b$ 
    and  $\forall x \in cbox\ a\ b. B \leq f(x) \cdot k$ 
    and  $k \in Basis$ 
  shows  $B * \text{content } (cbox\ a\ b) \leq (\text{integral}(cbox\ a\ b) f) \cdot k$ 
  using  $\text{assms has\_integral\_component\_lbound by blast}$ 

```

```

lemma  $\text{integral\_component\_lbound\_real}$ :
  assumes  $f \text{ integrable\_on } \{a..b\}$ 
    and  $\forall x \in \{a..b\}. B \leq f(x) \cdot k$ 
    and  $k \in Basis$ 
  shows  $B * \text{content } \{a..b\} \leq (\text{integral } \{a..b\} f) \cdot k$ 
  using  $\text{assms}$ 
  by  $(metis \text{box\_real}(2) \text{integral\_component\_lbound})$ 

```

```

lemma  $\text{integral\_component\_ubound}$ :
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $f \text{ integrable\_on } cbox\ a\ b$ 
    and  $\forall x \in cbox\ a\ b. f\ x \cdot k \leq B$ 
    and  $k \in Basis$ 

```

shows $(\text{integral } (\text{cbox } a \ b) \ f) \cdot k \leq B * \text{content } (\text{cbox } a \ b)$
using *assms has_integral_component_ubound* **by** *blast*

lemma *integral_component_ubound_real*:
fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$
assumes $f \text{ integrable\_on } \{a..b\}$
and $\forall x \in \{a..b\}. f \ x \cdot k \leq B$
and $k \in \text{Basis}$
shows $(\text{integral } \{a..b\} \ f) \cdot k \leq B * \text{content } \{a..b\}$
using *assms*
by (*metis* *box_real(2)* *integral_component_ubound*)

6.15.9 Uniform limit of integrable functions is integrable

lemma *real_arch_invD*:
 $0 < (e::\text{real}) \implies (\exists n::\text{nat}. n \neq 0 \wedge 0 < \text{inverse } (\text{real } n) \wedge \text{inverse } (\text{real } n) < e)$
by (*subst(asm)* *real_arch_inverse*)

lemma *integrable_uniform_limit*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{banach}$
assumes $\bigwedge e. e > 0 \implies \exists g. (\forall x \in \text{cbox } a \ b. \text{norm } (f \ x - g \ x) \leq e) \wedge g \text{ integrable\_on } \text{cbox } a \ b$
shows $f \text{ integrable\_on } \text{cbox } a \ b$
proof (*cases* $\text{content } (\text{cbox } a \ b) > 0$)
case *False* **then show** *?thesis*
using *has_integral_null* **by** (*simp* *add: content_lt_nz integrable_on_def*)
next
case *True*
have $1 / (\text{real } n + 1) > 0$ **for** n
by *auto*
then have $\exists g. (\forall x \in \text{cbox } a \ b. \text{norm } (f \ x - g \ x) \leq 1 / (\text{real } n + 1)) \wedge g \text{ integrable\_on } \text{cbox } a \ b$ **for** n
using *assms* **by** *blast*
then obtain g **where** $g \text{ near\_} f: \bigwedge n \ x. x \in \text{cbox } a \ b \implies \text{norm } (f \ x - g \ n \ x) \leq 1 / (\text{real } n + 1)$
and $\text{int\_}g: \bigwedge n. g \ n \text{ integrable\_on } \text{cbox } a \ b$
by *metis*
then obtain h **where** $h: \bigwedge n. (g \ n \text{ has\_integral } h \ n) (\text{cbox } a \ b)$
unfolding *integrable_on_def* **by** *metis*
have *Cauchy* h
unfolding *Cauchy_def*
proof *clarify*
fix $e :: \text{real}$
assume $e > 0$
then have $e/4 / \text{content } (\text{cbox } a \ b) > 0$
using *True* **by** (*auto* *simp: field_simps*)
then obtain M **where** $M \neq 0$ **and** $M: 1 / (\text{real } M) < e/4 / \text{content } (\text{cbox } a \ b)$

```

    by (metis inverse_eq_divide real_arch_inverse)
  show  $\exists M. \forall m \geq M. \forall n \geq M. \text{dist } (h \ m) \ (h \ n) < e$ 
  proof (rule exI [where  $x=M$ ], clarify)
    fix  $m \ n$ 
    assume  $m: M \leq m$  and  $n: M \leq n$ 
    have  $e/4 > 0$  using  $\langle e > 0 \rangle$  by auto
    then obtain  $gm \ gn$  where gauge  $gm$  gauge  $gn$ 
      and  $gm: \bigwedge \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } cbox \ a \ b \wedge gm \text{ fine } \mathcal{D}$ 
       $\implies \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \ m \ x) - h \ m) <$ 
 $e/4$ 
      and  $gn: \bigwedge \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } cbox \ a \ b \wedge gn \text{ fine } \mathcal{D} \implies$ 
       $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \ n \ x) - h \ n) < e/4$ 
    using  $h[\text{unfolded has\_integral}]$  by meson
    then obtain  $\mathcal{D}$  where  $\mathcal{D}: \mathcal{D} \text{ tagged\_division\_of } cbox \ a \ b \ (\lambda x. gm \ x \cap gn \ x)$ 
  fine  $\mathcal{D}$ 
    by (metis (full_types) fine_division_exists gauge_Int)
    have triangle3:  $\text{norm } (i1 - i2) < e$ 
      if no:  $\text{norm}(s2 - s1) \leq e/2 \ \text{norm}(s1 - i1) < e/4 \ \text{norm}(s2 - i2) < e/4$ 
  for  $s1 \ s2 \ i1$  and  $i2::'b$ 
    proof -
      have  $\text{norm } (i1 - i2) \leq \text{norm } (i1 - s1) + \text{norm } (s1 - s2) + \text{norm } (s2 -$ 
 $i2)$ 
      using norm_triangle_ineq[of  $i1 - s1 \ s1 - i2$ ]
      using norm_triangle_ineq[of  $s1 - s2 \ s2 - i2$ ]
      by (auto simp: algebra_simps)
      also have  $\dots < e$ 
      using no by (auto simp: algebra_simps norm_minus_commute)
      finally show ?thesis .
    qed
  have finep:  $gm \text{ fine } \mathcal{D} \ gn \text{ fine } \mathcal{D}$ 
  using fine_Int  $\mathcal{D}$  by auto
  have norm_le:  $\text{norm } (g \ n \ x - g \ m \ x) \leq 2 / \text{real } M$  if  $x: x \in cbox \ a \ b$  for  $x$ 
  proof -
    have  $\text{norm } (f \ x - g \ n \ x) + \text{norm } (f \ x - g \ m \ x) \leq 1 / (\text{real } n + 1) + 1 /$ 
 $(\text{real } m + 1)$ 
    using  $g\_near\_f[OF \ x, \text{of } n] \ g\_near\_f[OF \ x, \text{of } m]$  by simp
    also have  $\dots \leq 1 / (\text{real } M) + 1 / (\text{real } M)$ 
    using  $\langle M \neq 0 \rangle \ m \ n$  by (intro add_mono; force simp: field_split_simps)
    also have  $\dots = 2 / \text{real } M$ 
    by auto
    finally show  $\text{norm } (g \ n \ x - g \ m \ x) \leq 2 / \text{real } M$ 
    using norm_triangle_le[of  $g \ n \ x - f \ x \ f \ x - g \ m \ x \ 2 / \text{real } M$ ]
    by (auto simp: algebra_simps simp add: norm_minus_commute)
  qed
  have  $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \ n \ x) - (\sum (x,K) \in \mathcal{D}. \text{content } K$ 
 $*_R g \ m \ x)) \leq 2 / \text{real } M * \text{content } (cbox \ a \ b)$ 
  by (blast intro: norm_le rsum_diff_bound[OF  $\mathcal{D}(1)$ ], where  $e=2 / \text{real } M$ )
  also have  $\dots \leq e/2$ 
  using  $M \text{ True}$ 

```



```

    by (auto simp: field_simps)
    finally have le_e2: norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \ n \ x$ ) - ( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \ m \ x$ ))  $\leq e/2$  .
    then show dist (h m) (h n)  $< e$ 
      unfolding dist_norm using gm gn  $\mathcal{D}$  finep by (auto intro!: triangle3)
  qed
qed
then obtain s where s:  $h \longrightarrow s$ 
  using convergent_eq_Cauchy[symmetric] by blast
show ?thesis
  unfolding integrable_on_def has_integral
proof (rule_tac x=s in exI, clarify)
  fix e::real
  assume e:  $0 < e$ 
  then have  $e/3 > 0$  by auto
  then obtain N1 where N1:  $\forall n \geq N1. \text{norm } (h \ n - s) < e/3$ 
    using LIMSEQ_D [OF s] by metis
  from e True have  $e/3 / \text{content } (\text{cbox } a \ b) > 0$ 
    by (auto simp: field_simps)
  then obtain N2 :: nat
    where N2  $\neq 0$  and N2:  $1 / (\text{real } N2) < e/3 / \text{content } (\text{cbox } a \ b)$ 
    by (metis inverse_eq_divide real_arch_inverse)
  obtain g' where gauge g'
    and g':  $\bigwedge \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b \wedge g' \text{ fine } \mathcal{D} \implies$ 
      norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \ (N1 + N2) \ x$ ) -  $h \ (N1 + N2)$ )  $< e/3$ 
    by (metis h has_integral  $\langle e/3 > 0 \rangle$ )
  have *: norm (sf - s)  $< e$ 
    if no: norm (sf - sg)  $\leq e/3$  norm(h - s)  $< e/3$  norm (sg - h)  $< e/3$  for
  sf sg h
  proof -
    have norm (sf - s)  $\leq$  norm (sf - sg) + norm (sg - h) + norm (h - s)
      using norm_triangle_ineq[of sf - sg sg - s]
      using norm_triangle_ineq[of sg - h h - s]
      by (auto simp: algebra_simps)
    also have ...  $< e$ 
      using no by (auto simp: algebra_simps norm_minus_commute)
    finally show ?thesis .
  qed
{ fix  $\mathcal{D}$ 
  assume ptag:  $\mathcal{D} \text{ tagged\_division\_of } (\text{cbox } a \ b)$  and g' fine  $\mathcal{D}$ 
  then have norm_less: norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \ (N1 + N2) \ x$ ) -  $h \ (N1 + N2)$ )  $< e/3$ 
    using g' by blast
  have content (cbox a b)  $< e/3 * (\text{of\_nat } N2)$ 
    using  $\langle N2 \neq 0 \rangle$  N2 using True by (auto simp: field_split_simps)
  moreover have  $e/3 * \text{of\_nat } N2 \leq e/3 * (\text{of\_nat } (N1 + N2) + 1)$ 
    using  $\langle e > 0 \rangle$  by auto
  ultimately have content (cbox a b)  $< e/3 * (\text{of\_nat } (N1 + N2) + 1)$ 
```

```

    by linarith
  then have le_e3: 1 / (real (N1 + N2) + 1) * content (cbox a b) ≤ e/3
    unfolding inverse_eq_divide
    by (auto simp: field_simps)
  have ne3: norm (h (N1 + N2) - s) < e/3
    using N1 by auto
  have norm ((∑ (x,K) ∈ D. content K *R f x) - (∑ (x,K) ∈ D. content K
*_R g (N1 + N2) x))
    ≤ 1 / (real (N1 + N2) + 1) * content (cbox a b)
    by (blast intro: g_near_f rsum_diff_bound[OF ptag])
  then have norm ((∑ (x,K) ∈ D. content K *R f x) - s) < e
    by (rule *[OF order_trans [OF le_e3] ne3 norm_less])
}
then show ∃ d. gauge d ∧
  (∀ D. D tagged_division_of cbox a b ∧ d fine D → norm ((∑ (x,K) ∈
D. content K *R f x) - s) < e)
  by (blast intro: g' <gauge g')
qed
qed

```

lemmas *integrable_uniform_limit_real* = *integrable_uniform_limit* [where 'a=real, simplified]

6.15.10 Negligible sets

definition *negligible* (*s* :: 'a::euclidean_space set) $\longleftrightarrow$
 (∀ a b. ((indicator s :: 'a⇒real) has_integral 0) (cbox a b))

Negligibility of hyperplane

```

lemma content_doublesplit:
  fixes a :: 'a::euclidean_space
  assumes 0 < e
  and k: k ∈ Basis
  obtains d where 0 < d and content (cbox a b ∩ {x. |x·k - c| ≤ d}) < e
proof cases
  assume *: a · k ≤ c ∧ c ≤ b · k ∧ (∀ j ∈ Basis. a · j ≤ b · j)
  define a' where a' d = (∑ j ∈ Basis. (if j = k then max (a·j) (c - d) else a·j)
*_R j) for d
  define b' where b' d = (∑ j ∈ Basis. (if j = k then min (b·j) (c + d) else b·j)
*_R j) for d

  have ((λ d. ∏ j ∈ Basis. (b' d - a' d) · j) → (∏ j ∈ Basis. (b' 0 - a' 0) · j))
    (at_right 0)
  by (auto simp: b'_def a'_def intro!: tendsto_min tendsto_max tendsto_eq_intros)
  also have (∏ j ∈ Basis. (b' 0 - a' 0) · j) = 0
    using k *
  by (intro prod_zero bexI[OF k])
  (auto simp: b'_def a'_def inner_diff inner_sum_left inner_not_same_Basis
intro!: sum.cong)

```

```

also have (( $\lambda d. \prod_{j \in \text{Basis}. (b' d - a' d) \cdot j} \longrightarrow 0$ ) (at_right 0) =
  (( $\lambda d. \text{content } (\text{cbox } a \ b \cap \{x. |x \cdot k - c| \leq d\})$ )  $\longrightarrow 0$ ) (at_right 0))
proof (intro tendsto_cong eventually_at_rightI)
  fix  $d :: \text{real}$  assume  $d: d \in \{0 <..<1\}$ 
  have  $\text{cbox } a \ b \cap \{x. |x \cdot k - c| \leq d\} = \text{cbox } (a' d) (b' d)$  for  $d$ 
  using *  $d \ k$  by (auto simp add: cbox_def set_eq_iff Int_def ball_conj_distrib
abs_diff_le_iff a'_def b'_def)
  moreover have  $j \in \text{Basis} \implies a' d \cdot j \leq b' d \cdot j$  for  $j$ 
  using *  $d \ k$  by (auto simp: a'_def b'_def)
  ultimately show ( $\prod_{j \in \text{Basis}. (b' d - a' d) \cdot j} = \text{content } (\text{cbox } a \ b \cap \{x. |x \cdot k - c| \leq d\})$ )
    by simp
qed simp
finally have (( $\lambda d. \text{content } (\text{cbox } a \ b \cap \{x. |x \cdot k - c| \leq d\})$ )  $\longrightarrow 0$ ) (at_right
0) .
  from order_tendstoD(2)[OF this  $\langle 0 < e \rangle$ ]
  obtain  $d'$  where  $0 < d'$  and  $d': \bigwedge y. y > 0 \implies y < d' \implies \text{content } (\text{cbox } a \ b$ 
 $\cap \{x. |x \cdot k - c| \leq y\}) < e$ 
  by (subst (asm) eventually_at_right[of _ 1]) auto
  show ?thesis
  by (rule that[of  $d'/2$ ], insert  $\langle 0 < d' \rangle$   $d'$ [of  $d'/2$ ], auto)
next
assume *:  $\neg (a \cdot k \leq c \wedge c \leq b \cdot k \wedge (\forall j \in \text{Basis}. a \cdot j \leq b \cdot j))$ 
then have ( $\exists j \in \text{Basis}. b \cdot j < a \cdot j$ )  $\vee (c < a \cdot k \vee b \cdot k < c)$ 
  by (auto simp: not_le)
show thesis
proof cases
  assume  $\exists j \in \text{Basis}. b \cdot j < a \cdot j$ 
  then have [simp]:  $\text{cbox } a \ b = \{\}$ 
  using box_ne_empty(1)[of  $a \ b$ ] by auto
  show ?thesis
  by (rule that[of 1]) (simp_all add:  $\langle 0 < e \rangle$ )
next
assume  $\neg (\exists j \in \text{Basis}. b \cdot j < a \cdot j)$ 
with * have  $c < a \cdot k \vee b \cdot k < c$ 
  by auto
then show thesis
proof
  assume  $c: c < a \cdot k$ 
  moreover have  $x \in \text{cbox } a \ b \implies c \leq x \cdot k$  for  $x$ 
  using  $k \ c$  by (auto simp: cbox_def)
  ultimately have  $\text{cbox } a \ b \cap \{x. |x \cdot k - c| \leq (a \cdot k - c)/2\} = \{\}$ 
  using  $k$  by (auto simp: cbox_def)
  with  $\langle 0 < e \rangle \ c$  that[of  $(a \cdot k - c)/2$ ] show ?thesis
  by auto
next
assume  $c: b \cdot k < c$ 
moreover have  $x \in \text{cbox } a \ b \implies x \cdot k \leq c$  for  $x$ 
  using  $k \ c$  by (auto simp: cbox_def)

```

```

ultimately have  $cbox\ a\ b \cap \{x. |x \cdot k - c| \leq (c - b \cdot k)/2\} = \{\}$ 
  using  $k$  by (auto simp: cbox_def)
with  $\langle 0 < e \rangle$   $c$  that[ $of\ (c - b \cdot k)/2$ ] show ?thesis
  by auto
qed
qed
qed

proposition negligible_standard_hyperplane[intro]:
  fixes  $k :: 'a::euclidean\_space$ 
  assumes  $k: k \in Basis$ 
  shows negligible  $\{x. x \cdot k = c\}$ 
  unfolding negligible_def has_integral
proof clarsimp
  fix  $a\ b$  and  $e::real$  assume  $e > 0$ 
  with  $k$  obtain  $d$  where  $0 < d$  and  $d: content\ (cbox\ a\ b \cap \{x. |x \cdot k - c| \leq d\})$ 
  <  $e$ 
  by (metis content_doublesplit)
  let  $?i = indicator\ \{x::'a. x \cdot k = c\} :: 'a \Rightarrow real$ 
  show  $\exists \gamma. gauge\ \gamma \wedge$ 
     $(\forall \mathcal{D}. \mathcal{D}\ tagged\_division\_of\ cbox\ a\ b \wedge \gamma\ fine\ \mathcal{D} \longrightarrow$ 
       $|\sum (x,K) \in \mathcal{D}. content\ K * ?i\ x| < e)$ 
proof (intro exI, safe)
  show gauge  $(\lambda x. ball\ x\ d)$ 
    using  $\langle 0 < d \rangle$  by blast
next
  fix  $\mathcal{D}$ 
  assume  $p: \mathcal{D}\ tagged\_division\_of\ (cbox\ a\ b)\ (\lambda x. ball\ x\ d)\ fine\ \mathcal{D}$ 
  have  $content\ L = content\ (L \cap \{x. |x \cdot k - c| \leq d\})$ 
  if  $(x, L) \in \mathcal{D}$   $?i\ x \neq 0$  for  $x\ L$ 
proof -
  have  $xk: x \cdot k = c$ 
    using that by (simp add: indicator_def split: if_split_asm)
  have  $L \subseteq \{x. |x \cdot k - c| \leq d\}$ 
proof
  fix  $y$ 
  assume  $y: y \in L$ 
  have  $L \subseteq ball\ x\ d$ 
    using  $p(2)$  that(1) by auto
  then have  $norm\ (x - y) < d$ 
    by (simp add: dist_norm subset_iff y)
  then have  $|(x - y) \cdot k| < d$ 
    using  $k\ norm\_bound\_Basis\_lt$  by blast
  then show  $y \in \{x. |x \cdot k - c| \leq d\}$ 
    unfolding inner_simps  $xk$  by auto
qed
then show  $content\ L = content\ (L \cap \{x. |x \cdot k - c| \leq d\})$ 
  by (metis inf.orderE)

```

```

qed
then have *: ( $\sum (x,K) \in \mathcal{D}. \text{content } K * ?i x$ ) = ( $\sum (x,K) \in \mathcal{D}. \text{content } (K \cap \{x. |x \cdot k - c| \leq d\}) * ?i x$ )
by (force simp add: split_paired_all intro!: sum.cong [OF refl])
note p' = tagged_division_ofD[OF p(1)] and p'' = division_of_tagged_division[OF p(1)]
have ( $\sum (x,K) \in \mathcal{D}. \text{content } (K \cap \{x. |x \cdot k - c| \leq d\}) * \text{indicator } \{x. x \cdot k = c\} x$ ) < e
proof -
  have ( $\sum (x,K) \in \mathcal{D}. \text{content } (K \cap \{x. |x \cdot k - c| \leq d\}) * ?i x$ )  $\leq$  ( $\sum (x,K) \in \mathcal{D}. \text{content } (K \cap \{x. |x \cdot k - c| \leq d\})$ )
  by (force simp add: indicator_def intro!: sum_mono)
  also have ... < e
  proof (subst sum.over_tagged_division_lemma[OF p(1)])
    fix u v :: 'a
    assume box u v = {}
    then have *: content (cbox u v) = 0
    unfolding content_eq_0_interior by simp
    have cbox u v  $\cap$  {x. |x · k - c| ≤ d}  $\subseteq$  cbox u v
    by auto
    then have content (cbox u v  $\cap$  {x. |x · k - c| ≤ d})  $\leq$  content (cbox u v)
    unfolding interval_doublesplit[OF k] by (rule content_subset)
    then show content (cbox u v  $\cap$  {x. |x · k - c| ≤ d}) = 0
    unfolding * interval_doublesplit[OF k]
    by (blast intro: antisym)
  next
    have ( $\sum l \in \text{snd } ' \mathcal{D}. \text{content } (l \cap \{x. |x \cdot k - c| \leq d\})$ ) =
      sum content (( $\lambda l. l \cap \{x. |x \cdot k - c| \leq d\}$ ) {l ∈ snd '  $\mathcal{D}. l \cap \{x. |x \cdot k - c| \leq d\} \neq \{\}$ })
    proof (subst (2) sum.reindex_nontrivial)
      fix x y assume x ∈ {l ∈ snd '  $\mathcal{D}. l \cap \{x. |x \cdot k - c| \leq d\} \neq \{\}}$  y ∈ {l ∈ snd '  $\mathcal{D}. l \cap \{x. |x \cdot k - c| \leq d\} \neq \{\}}$ 
      x  $\neq$  y and eq: x  $\cap$  {x. |x · k - c| ≤ d} = y  $\cap$  {x. |x · k - c| ≤ d}
      then obtain x' y' where (x', x) ∈  $\mathcal{D}$  x  $\cap$  {x. |x · k - c| ≤ d}  $\neq \{\}$  (y', y) ∈  $\mathcal{D}$  y  $\cap$  {x. |x · k - c| ≤ d}  $\neq \{\}$ 
      by (auto)
      from p'(5)[OF  $\langle (x', x) \in \mathcal{D} \rangle \langle (y', y) \in \mathcal{D} \rangle \langle x \neq y \rangle$ ] have interior (x  $\cap$  y) = {}
      by auto
      moreover have interior ((x  $\cap$  {x. |x · k - c| ≤ d})  $\cap$  (y  $\cap$  {x. |x · k - c| ≤ d}))  $\subseteq$  interior (x  $\cap$  y)
      by (auto intro: interior_mono)
      ultimately have interior (x  $\cap$  {x. |x · k - c| ≤ d}) = {}
      by (auto simp: eq)
      then show content (x  $\cap$  {x. |x · k - c| ≤ d}) = 0
      using p'(4)[OF  $\langle (x', x) \in \mathcal{D} \rangle$ ] by (auto simp: interval_doublesplit[OF k] content_eq_0_interior simp del: interior_Int)
    qed (insert p'(1), auto intro!: sum.mono_neutral_right)
    also have ...  $\leq$  norm ( $\sum l \in (\lambda l. l \cap \{x. |x \cdot k - c| \leq d\}) \{l \in \text{snd } ' \mathcal{D}. l \cap \{x. |x \cdot k - c| \leq d\} \neq \{\}$ )

```

```

{x. |x · k - c| ≤ d} ≠ {}}. content l *R 1::real)
  by simp
  also have ... ≤ 1 * content (cbox a b ∩ {x. |x · k - c| ≤ d})
    using division_doublesplit[OF p'' k, unfolded interval_doublesplit[OF k]]
    unfolding interval_doublesplit[OF k] by (intro dsum_bound) auto
  also have ... < e
    using d by simp
  finally show (∑ K ∈ snd 'D. content (K ∩ {x. |x · k - c| ≤ d})) < e .
qed
finally show (∑ (x, K) ∈ D. content (K ∩ {x. |x · k - c| ≤ d}) * ?i x) < e .
qed
then show |∑ (x, K) ∈ D. content K * ?i x| < e
  unfolding * by (simp add: sum_nonneg split: prod.split)
qed
qed

```

```

corollary negligible_standard_hyperplane_cart:
  fixes k :: 'a::finite
  shows negligible {x. x$k = (0::real)}
  by (simp add: cart_eq_inner_axis negligible_standard_hyperplane)

```

Hence the main theorem about negligible sets

```

lemma has_integral_negligible_cbox:
  fixes f :: 'b::euclidean_space ⇒ 'a::real_normed_vector
  assumes negs: negligible S
    and 0: ⋀ x. x ∉ S ⇒ f x = 0
  shows (f has_integral 0) (cbox a b)
  unfolding has_integral
proof clarify
  fix e::real
  assume e > 0
  then have nn_gt0: e/2 / ((real n+1) * (2 ^ n)) > 0 for n
    by simp
  then have ∃ γ. gauge γ ∧
    (∀ D. D tagged_division_of cbox a b ∧ γ fine D ⟹
      |∑ (x,K) ∈ D. content K *R indicator S x|
      < e/2 / ((real n + 1) * 2 ^ n)) for n
    using negs [unfolded negligible_def has_integral] by auto
  then obtain γ where
    gd: ⋀ n. gauge (γ n)
    and γ: ⋀ n D. [D tagged_division_of cbox a b; γ n fine D]
      ⟹ |∑ (x,K) ∈ D. content K *R indicator S x| < e/2 / ((real n +
1) * 2 ^ n)
    by metis
  show ∃ γ. gauge γ ∧
    (∀ D. D tagged_division_of cbox a b ∧ γ fine D ⟹
      norm ((∑ (x,K) ∈ D. content K *R f x) - 0) < e)
  proof (intro exI, safe)

```

```

show gauge ( $\lambda x. \gamma (\text{nat } \lfloor \text{norm } (f x) \rfloor) x$ )
  using gd by (auto simp: gauge_def)

show norm (( $\sum (x, K) \in \mathcal{D}. \text{content } K *_R f x - 0$ ) < e)
  if  $\mathcal{D}$  tagged_division_of (cbox a b) ( $\lambda x. \gamma (\text{nat } \lfloor \text{norm } (f x) \rfloor) x$ ) fine  $\mathcal{D}$  for  $\mathcal{D}$ 
proof (cases  $\mathcal{D} = \{\}$ )
  case True with  $\langle 0 < e \rangle$  show ?thesis by simp
next
  case False
  obtain N where Max (( $\lambda(x, K). \text{norm } (f x)$ ) '  $\mathcal{D}$ )  $\leq$  real N
    using real_arch_simple by blast
  then have N:  $\bigwedge x. x \in (\lambda(x, K). \text{norm } (f x)) ' \mathcal{D} \implies x \leq \text{real } N$ 
  by (meson Max_ge that(1) dual_order.trans finite_imageI tagged_division_of_finite)
  have  $\forall i. \exists q. q \text{ tagged\_division\_of } (\text{cbox } a \text{ } b) \wedge (\gamma i) \text{ fine } q \wedge (\forall (x, K) \in \mathcal{D}. K \subseteq (\gamma i) x \longrightarrow (x, K) \in q)$ 
    by (auto intro: tagged_division_finer[OF that(1) gd])
  from choice[OF this]
  obtain q where q:  $\bigwedge n. q \text{ n tagged\_division\_of cbox } a \text{ } b$ 
     $\bigwedge n. \gamma \text{ n fine } q \text{ n}$ 
     $\bigwedge n x K. \llbracket (x, K) \in \mathcal{D}; K \subseteq \gamma \text{ n } x \rrbracket \implies (x, K) \in q \text{ n}$ 
  by fastforce
  have finite  $\mathcal{D}$ 
    using that(1) by blast
  then have sum_le_inc:  $\llbracket \text{finite } T; \bigwedge x y. (x, y) \in T \implies (0::\text{real}) \leq g(x, y);$ 
     $\bigwedge y. y \in \mathcal{D} \implies \exists x. (x, y) \in T \wedge f(y) \leq g(x, y) \rrbracket \implies \text{sum } f \mathcal{D} \leq$ 
     $\text{sum } g T$  for  $f g T$ 
    by (rule sum_le_included[of  $\mathcal{D}$   $T$   $g$   $\text{snd } f$ ]; force)
  have norm ( $\sum (x, K) \in \mathcal{D}. \text{content } K *_R f x$ )  $\leq (\sum (x, K) \in \mathcal{D}. \text{norm } (\text{content } K *_R f x))$ 
    unfolding split_def by (rule norm_sum)
  also have ...  $\leq (\sum (i, j) \in \text{Sigma } \{..N + 1\} q. (\text{real } i + 1) * (\text{case } j \text{ of } (x, K) \Rightarrow \text{content } K *_R \text{indicator } S x))$ 
  proof (rule sum_le_inc, safe)
    show finite (Sigma  $\{..N + 1\} q$ )
      by (meson finite_SigmaI finite_atMost tagged_division_of_finite q(1))
  next
    fix x K
    assume xk:  $(x, K) \in \mathcal{D}$ 
    define n where  $n = \text{nat } \lfloor \text{norm } (f x) \rfloor$ 
    have *:  $\text{norm } (f x) \in (\lambda(x, K). \text{norm } (f x)) ' \mathcal{D}$ 
      using xk by auto
    have nfx:  $\text{real } n \leq \text{norm } (f x) \text{ norm } (f x) \leq \text{real } n + 1$ 
      unfolding n_def by auto
    then have  $n \in \{0..N + 1\}$ 
      using N[OF *] by auto
    moreover have  $K \subseteq \gamma (\text{nat } \lfloor \text{norm } (f x) \rfloor) x$ 
      using that(2) xk by auto
    moreover then have  $(x, K) \in q (\text{nat } \lfloor \text{norm } (f x) \rfloor)$ 

```

```

    by (simp add: q(3) xk)
  moreover then have  $(x, K) \in q \ n$ 
    using n_def by blast
  moreover
  have norm  $(\text{content } K *_R f x) \leq (\text{real } n + 1) * (\text{content } K * \text{indicator } S x)$ 
  proof (cases  $x \in S$ )
    case False
    then show ?thesis by (simp add: 0)
  next
    case True
    have *:  $\text{content } K \geq 0$ 
      using tagged_division_ofD(4)[OF that(1) xk] by auto
    moreover have  $\text{content } K * \text{norm } (f x) \leq \text{content } K * (\text{real } n + 1)$ 
      by (simp add: mult_left_mono nfx(2))
    ultimately show ?thesis
      using nfx True by (auto simp: field_simps)
  qed
  ultimately show  $\exists y. (y, x, K) \in (\text{Sigma } \{..N + 1\} q) \wedge \text{norm } (\text{content } K *_R f x) \leq$ 
     $(\text{real } y + 1) * (\text{content } K *_R \text{indicator } S x)$ 
    by force
  qed auto
  also have  $\dots = (\sum_{i \leq N + 1}. \sum_{j \in q \ i}. (\text{real } i + 1) * (\text{case } j \text{ of } (x, K) \Rightarrow \text{content } K *_R \text{indicator } S x))$ 
    using q(1) by (intro sum_Sigma_product [symmetric]) auto
  also have  $\dots \leq (\sum_{i \leq N + 1}. (\text{real } i + 1) * |\sum_{(x,K) \in q \ i}. \text{content } K *_R \text{indicator } S x|)$ 
    by (rule sum_mono) (simp add: sum_distrib_left [symmetric])
  also have  $\dots \leq (\sum_{i \leq N + 1}. e/2/2^i)$ 
  proof (rule sum_mono)
    show  $(\text{real } i + 1) * |\sum_{(x,K) \in q \ i}. \text{content } K *_R \text{indicator } S x| \leq e/2/2^i$ 
    ^ i
    if  $i \in \{..N + 1\}$  for i
      using  $\gamma[\text{of } q \ i \ i] \ q$  by (simp add: divide_simps mult.left_commute)
  qed
  also have  $\dots = e/2 * (\sum_{i \leq N + 1}. (1/2)^i)$ 
    unfolding sum_distrib_left by (metis divide_inverse inverse_eq_divide power_one_over)
  also have  $\dots < e/2 * 2$ 
  proof (rule mult_strict_left_mono)
    have  $\text{sum } (\text{power } (1/2)) \ \{..N + 1\} = \text{sum } (\text{power } (1/2::\text{real})) \ \{..<N + 2\}$ 
    using lessThan_Suc_atMost by auto
  also have  $\dots < 2$ 
    by (auto simp: geometric_sum)
  finally show  $\text{sum } (\text{power } (1/2::\text{real})) \ \{..N + 1\} < 2$  .
  qed (use  $0 < e$  in auto)
  finally show ?thesis by auto
  qed

```


qed
qed

proposition *has_integral_negligible*:

fixes $f :: 'b::\text{euclidean\_space} \Rightarrow 'a::\text{real\_normed\_vector}$
assumes *negs*: *negligible S*
and $\bigwedge x. x \in (T - S) \implies f\ x = 0$
shows $(f \text{ has\_integral } 0)\ T$
proof (*cases* $\exists a\ b. T = \text{cbox } a\ b$)
case *True*
then have $((\lambda x. \text{if } x \in T \text{ then } f\ x \text{ else } 0) \text{ has\_integral } 0)\ T$
using *assms* **by** (*auto intro!*: *has_integral_negligible_cbox*)
then show *?thesis*
by (*rule has_integral_eq [rotated]*) *auto*
next
case *False*
let $?f = (\lambda x. \text{if } x \in T \text{ then } f\ x \text{ else } 0)$
have $((\lambda x. \text{if } x \in T \text{ then } f\ x \text{ else } 0) \text{ has\_integral } 0)\ T$
apply (*auto simp: False has_integral_alt [of ?f]*)
apply (*rule_tac x=1 in exI, auto*)
apply (*rule_tac x=0 in exI, simp add: has_integral_negligible_cbox [OF negs]*
assms)
done
then show *?thesis*
by (*rule_tac f=?f in has_integral_eq*) *auto*
qed

lemma

assumes *negligible S*
shows *integrable_negligible*: $f \text{ integrable\_on } S$ **and** *integral_negligible*: $\text{integral } S\ f = 0$
using *has_integral_negligible [OF assms]*
by (*auto simp: has_integral_iff*)

lemma *has_integral_spike*:

fixes $f :: 'b::\text{euclidean\_space} \Rightarrow 'a::\text{real\_normed\_vector}$
assumes *negligible S*
and $gf: \bigwedge x. x \in T - S \implies g\ x = f\ x$
and *fint*: $(f \text{ has\_integral } y)\ T$
shows $(g \text{ has\_integral } y)\ T$
proof –
have $*(g \text{ has\_integral } y)\ (\text{cbox } a\ b)$
if $(f \text{ has\_integral } y)\ (\text{cbox } a\ b) \ \forall x \in \text{cbox } a\ b - S. g\ x = f\ x$ **for** $a\ b\ f$ **and**
 $g:: 'b \Rightarrow 'a$ **and** y
proof –
have $((\lambda x. f\ x + (g\ x - f\ x)) \text{ has\_integral } (y + 0))\ (\text{cbox } a\ b)$
using that **by** (*intro has_integral_add has_integral_negligible*) (*auto intro!*:
 $\langle \text{negligible } S \rangle$)

```

    then show ?thesis
    by auto
qed
have §:  $\bigwedge a b z. \llbracket \bigwedge x. x \in T \wedge x \notin S \implies g x = f x; \quad$ 
 $((\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0) \text{ has\_integral } z) (cbox a b) \rrbracket$ 
 $\implies ((\lambda x. \text{if } x \in T \text{ then } g x \text{ else } 0) \text{ has\_integral } z) (cbox a b)$ 
    by (auto dest!: *[where  $f = \lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0$  and  $g = \lambda x. \text{if } x \in T$ 
then  $g x \text{ else } 0$ ])
    show ?thesis
    using fint gf
    apply (subst has_integral_alt)
    apply (subst (asm) has_integral_alt)
    apply (auto split: if_split_asm)
    apply (blast dest: *)
    using § by meson
qed

```

```

lemma has_integral_spike_eq:
  assumes negligible S
  and gf:  $\bigwedge x. x \in T - S \implies g x = f x$ 
  shows  $(f \text{ has\_integral } y) T \longleftrightarrow (g \text{ has\_integral } y) T$ 
  using has_integral_spike [OF  $\langle \text{negligible } S \rangle$ ] gf
  by metis

```

```

lemma integrable_spike:
  assumes  $f \text{ integrable\_on } T$  negligible S  $\bigwedge x. x \in T - S \implies g x = f x$ 
  shows  $g \text{ integrable\_on } T$ 
  using assms unfolding integrable_on_def by (blast intro: has_integral_spike)

```

```

lemma integral_spike:
  assumes negligible S
  and  $\bigwedge x. x \in T - S \implies g x = f x$ 
  shows  $\text{integral } T f = \text{integral } T g$ 
  using has_integral_spike_eq [OF assms]
  by (auto simp: integral_def integrable_on_def)

```

6.15.11 Some other trivialities about negligible sets

```

lemma negligible_subset:
  assumes negligible s  $t \subseteq s$ 
  shows negligible t
  unfolding negligible_def
  by (metis (no_types) Diff_iff assms contra_subsetD has_integral_negligible
indicator_simps(2))

```

```

lemma negligible_diff[intro?]:
  assumes negligible s
  shows negligible (s - t)
  using assms by (meson Diff_subset negligible_subset)

```

```

lemma negligible_Int:
  assumes negligible s  $\vee$  negligible t
  shows negligible (s  $\cap$  t)
  using assms negligible_subset by force

lemma negligible_Un:
  assumes negligible S and T: negligible T
  shows negligible (S  $\cup$  T)
proof -
  have (indicat_real (S  $\cup$  T) has_integral 0) (cbox a b)
  if S0: (indicat_real S has_integral 0) (cbox a b)
    and (indicat_real T has_integral 0) (cbox a b) for a b
  proof (subst has_integral_spike_eq[OF T])
    show indicat_real S x = indicat_real (S  $\cup$  T) x if x  $\in$  cbox a b - T for x
    using that by (simp add: indicator_def)
    show (indicat_real S has_integral 0) (cbox a b)
    by (simp add: S0)
  qed
qed
with assms show ?thesis
  unfolding negligible_def by blast
qed

lemma negligible_Un_eq[simp]: negligible (s  $\cup$  t)  $\longleftrightarrow$  negligible s  $\wedge$  negligible t
  using negligible_Un negligible_subset by blast

lemma negligible_sing[intro]: negligible {a::'a::euclidean_space}
  using negligible_standard_hyperplane[OF SOME_Basis, of a  $\cdot$  (SOME i. i  $\in$  Basis)] negligible_subset by blast

lemma negligible_insert[simp]: negligible (insert a s)  $\longleftrightarrow$  negligible s
  by (metis insert_is_Un negligible_Un_eq negligible_sing)

lemma negligible_empty[iff]: negligible {}
  using negligible_insert by blast

Useful in this form for backchaining

lemma empty_imp_negligible: S = {}  $\implies$  negligible S
  by simp

lemma negligible_finite[intro]:
  assumes finite s
  shows negligible s
  using assms by (induct s) auto

lemma negligible_Union[intro]:
  assumes finite  $\mathcal{T}$ 
  and  $\bigwedge t. t \in \mathcal{T} \implies$  negligible t
  shows negligible( $\bigcup \mathcal{T}$ )

```

using *assms* by *induct auto*

lemma *negligible*: $\text{negligible } S \longleftrightarrow (\forall T. (\text{indicat\_real } S \text{ has\_integral } 0) \ T)$
proof (*intro iffI allI*)
 fix *T*
 assume *negligible S*
 then show $(\text{indicator } S \text{ has\_integral } 0) \ T$
 by (*meson Diff_iff has_integral_negligible indicator_simps(2)*)
qed (*simp add: negligible_def*)

6.15.12 Finite case of the spike theorem is quite commonly needed

lemma *has_integral_spike_finite*:
 assumes *finite S*
 and $\bigwedge x. x \in T - S \implies g \ x = f \ x$
 and $(f \text{ has\_integral } y) \ T$
 shows $(g \text{ has\_integral } y) \ T$
 using *assms has_integral_spike negligible_finite* by *blast*

lemma *has_integral_spike_finite_eq*:
 assumes *finite S*
 and $\bigwedge x. x \in T - S \implies g \ x = f \ x$
 shows $((f \text{ has\_integral } y) \ T \longleftrightarrow (g \text{ has\_integral } y) \ T)$
 by (*metis assms has_integral_spike_finite*)

lemma *integrable_spike_finite*:
 assumes *finite S*
 and $\bigwedge x. x \in T - S \implies g \ x = f \ x$
 and *f integrable_on T*
 shows *g integrable_on T*
 using *assms has_integral_spike_finite* by *blast*

lemma *has_integral_bound_spike_finite*:
 fixes *f* :: *'a::euclidean_space* $\Rightarrow$ *'b::real_normed_vector*
 assumes $0 \leq B$ *finite S*
 and *f*: $(f \text{ has\_integral } i) \ (\text{cbox } a \ b)$
 and *leB*: $\bigwedge x. x \in \text{cbox } a \ b - S \implies \text{norm } (f \ x) \leq B$
 shows $\text{norm } i \leq B * \text{content } (\text{cbox } a \ b)$
proof –
 define *g* where $g \equiv (\lambda x. \text{if } x \in S \text{ then } 0 \text{ else } f \ x)$
 then have $\bigwedge x. x \in \text{cbox } a \ b - S \implies \text{norm } (g \ x) \leq B$
 using *leB* by *simp*
 moreover have $(g \text{ has\_integral } i) \ (\text{cbox } a \ b)$
 using *has_integral_spike_finite* [*OF* $\langle \text{finite } S \rangle$ _ *f*]
 by (*simp add: g_def*)
 ultimately show *?thesis*
 by (*simp add: $\langle 0 \leq B \rangle$ g_def has_integral_bound*)
qed

corollary *has_integral_bound_real*:
fixes $f :: \text{real} \Rightarrow 'b::\text{real\_normed\_vector}$
assumes $0 \leq B$ *finite S*
and $(f \text{ has\_integral } i) \{a..b\}$
and $\bigwedge x. x \in \{a..b\} - S \implies \text{norm } (f x) \leq B$
shows $\text{norm } i \leq B * \text{content } \{a..b\}$
by $(\text{metis } \text{assms } \text{box\_real}(2) \text{ has\_integral\_bound\_spike\_finite})$

6.15.13 In particular, the boundary of an interval is negligible

lemma *negligible_frontier_interval*: $\text{negligible}(\text{cbox } (a::'a::\text{euclidean\_space}) \ b - \text{box } a \ b)$
proof –
let $?A = \bigcup ((\lambda k. \{x. x \cdot k = a \cdot k\} \cup \{x::'a. x \cdot k = b \cdot k\}) \text{ `Basis})$
have *negligible ?A*
by $(\text{force } \text{simp } \text{add: negligible\_Union}[OF \text{ finite\_imageI}])$
moreover have $\text{cbox } a \ b - \text{box } a \ b \subseteq ?A$
by $(\text{force } \text{simp } \text{add: mem\_box})$
ultimately show *?thesis*
by $(\text{rule negligible\_subset})$
qed

lemma *has_integral_spike_interior*:
assumes $f: (f \text{ has\_integral } y) (\text{cbox } a \ b)$ **and** $gf: \bigwedge x. x \in \text{box } a \ b \implies g x = f x$
shows $(g \text{ has\_integral } y) (\text{cbox } a \ b)$
by $(\text{meson } \text{Diff\_iff } gf \text{ has\_integral\_spike}[OF \text{ negligible\_frontier\_interval } \_ f])$

lemma *has_integral_spike_interior_eq*:
assumes $\bigwedge x. x \in \text{box } a \ b \implies g x = f x$
shows $(f \text{ has\_integral } y) (\text{cbox } a \ b) \longleftrightarrow (g \text{ has\_integral } y) (\text{cbox } a \ b)$
by $(\text{metis } \text{assms } \text{has\_integral\_spike\_interior})$

lemma *integrable_spike_interior*:
assumes $\bigwedge x. x \in \text{box } a \ b \implies g x = f x$
and $f \text{ integrable\_on } \text{cbox } a \ b$
shows $g \text{ integrable\_on } \text{cbox } a \ b$
using *assms has_integral_spike_interior_eq* **by** *blast*

6.15.14 Integrability of continuous functions

lemma *operative_approximableI*:
fixes $f :: 'b::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$
assumes $0 \leq e$
shows $\text{operative } \text{conj } \text{True } (\lambda i. \exists g. (\forall x \in i. \text{norm } (f x - g (x::'b)) \leq e) \wedge g \text{ integrable\_on } i)$
proof –
interpret *comm_monoid conj True*

```

    by standard auto
  show ?thesis
  proof (standard, safe)
    fix a b :: 'b
    show  $\exists g. (\forall x \in \text{cbox } a \ b. \text{norm } (f \ x - g \ x) \leq e) \wedge g \text{ integrable\_on cbox } a \ b$ 
      if  $\text{box } a \ b = \{\}$  for a b
      using assms that
    by (metis content_eq_0_interior integrable_on_null_interior_cbox norm_zero
right_minus_eq)
    {
      fix c g and k :: 'b
      assume fg:  $\forall x \in \text{cbox } a \ b. \text{norm } (f \ x - g \ x) \leq e$  and g:  $g \text{ integrable\_on cbox } a \ b$ 
      assume k:  $k \in \text{Basis}$ 
      show  $\exists g. (\forall x \in \text{cbox } a \ b \cap \{x. x \cdot k \leq c\}. \text{norm } (f \ x - g \ x) \leq e) \wedge g \text{ integrable\_on cbox } a \ b \cap \{x. x \cdot k \leq c\}$ 
         $\exists g. (\forall x \in \text{cbox } a \ b \cap \{x. c \leq x \cdot k\}. \text{norm } (f \ x - g \ x) \leq e) \wedge g \text{ integrable\_on cbox } a \ b \cap \{x. c \leq x \cdot k\}$ 
        using fg g k by auto
    }
    show  $\exists g. (\forall x \in \text{cbox } a \ b. \text{norm } (f \ x - g \ x) \leq e) \wedge g \text{ integrable\_on cbox } a \ b$ 
      if fg1:  $\forall x \in \text{cbox } a \ b \cap \{x. x \cdot k \leq c\}. \text{norm } (f \ x - g1 \ x) \leq e$ 
      and g1:  $g1 \text{ integrable\_on cbox } a \ b \cap \{x. x \cdot k \leq c\}$ 
      and fg2:  $\forall x \in \text{cbox } a \ b \cap \{x. c \leq x \cdot k\}. \text{norm } (f \ x - g2 \ x) \leq e$ 
      and g2:  $g2 \text{ integrable\_on cbox } a \ b \cap \{x. c \leq x \cdot k\}$ 
      and k:  $k \in \text{Basis}$ 
    for c k g1 g2
  proof -
    let ?g =  $\lambda x. \text{if } x \cdot k = c \text{ then } f \ x \text{ else if } x \cdot k \leq c \text{ then } g1 \ x \text{ else } g2 \ x$ 
    show  $\exists g. (\forall x \in \text{cbox } a \ b. \text{norm } (f \ x - g \ x) \leq e) \wedge g \text{ integrable\_on cbox } a \ b$ 
    proof (intro exI conjI ballI)
      show  $\text{norm } (f \ x - ?g \ x) \leq e$  if  $x \in \text{cbox } a \ b$  for x
        by (auto simp: that assms fg1 fg2)
      show ?g integrable_on cbox a b
      proof -
        have ?g integrable_on cbox a b  $\cap \{x. x \cdot k \leq c\}$  ?g integrable_on cbox a
b  $\cap \{x. x \cdot k \geq c\}$ 
        by (rule integrable_spike[OF _ negligible_standard_hyperplane[of k c]],
use k g1 g2 in auto)+
        with has_integral_split[OF _ _ k] show ?thesis
        unfolding integrable_on_def by blast
      qed
    qed
  qed
  qed
  qed

```

lemma *comm_monoid_set_F_and*: $\text{comm\_monoid\_set.F } (\wedge) \text{ True } f \ s \longleftrightarrow (\text{finite } s \longrightarrow (\forall x \in s. f \ x))$

```

proof –
  interpret bool: comm_monoid_set  $\langle (\wedge) \rangle$  True ..
  show ?thesis
    by (induction s rule: infinite_finite_induct) auto
qed

lemma approximable_on_division:
  fixes f :: 'b::euclidean_space  $\Rightarrow$  'a::banach
  assumes  $0 \leq e$ 
  and d: d division_of (cbox a b)
  and f:  $\forall i \in d. \exists g. (\forall x \in i. \text{norm } (f\ x - g\ x) \leq e) \wedge g \text{ integrable\_on } i$ 
  obtains g where  $\forall x \in \text{cbox } a\ b. \text{norm } (f\ x - g\ x) \leq e$  g integrable_on cbox a b
proof –
  interpret operative conj True  $\lambda i. \exists g. (\forall x \in i. \text{norm } (f\ x - g\ (x::'b)) \leq e) \wedge g$ 
integrable_on i
  using  $\langle 0 \leq e \rangle$  by (rule operative_approximableI)
  from f local.division [OF d] that show thesis
  by auto
qed

lemma integrable_continuous:
  fixes f :: 'b::euclidean_space  $\Rightarrow$  'a::banach
  assumes continuous_on (cbox a b) f
  shows f integrable_on cbox a b
proof (rule integrable_uniform_limit)
  fix e :: real
  assume e:  $e > 0$ 
  then obtain d where  $0 < d$  and d:  $\bigwedge x\ x'. \llbracket x \in \text{cbox } a\ b; x' \in \text{cbox } a\ b; \text{dist } x'$ 
 $x < d \rrbracket \implies \text{dist } (f\ x') (f\ x) < e$ 
  using compact_uniformly_continuous[OF assms compact_cbox] unfolding
uniformly_continuous_on_def by metis
  obtain p where ptag: p tagged_division_of cbox a b and finep:  $(\lambda x. \text{ball } x\ d)$ 
fine p
  using fine_division_exists[OF gauge_ball[OF  $\langle 0 < d \rangle$ ], of a b] .
  have *:  $\forall i \in \text{snd } 'p. \exists g. (\forall x \in i. \text{norm } (f\ x - g\ x) \leq e) \wedge g \text{ integrable\_on } i$ 
  proof (safe, unfold snd_conv)
  fix x l
  assume as:  $(x, l) \in p$ 
  obtain a b where l: l = cbox a b
  using as ptag by blast
  then have x:  $x \in \text{cbox } a\ b$ 
  using as ptag by auto
  show  $\exists g. (\forall x \in l. \text{norm } (f\ x - g\ x) \leq e) \wedge g \text{ integrable\_on } l$ 
  proof (intro exI conjI strip)
  show  $(\lambda y. f\ x) \text{ integrable\_on } l$ 
  unfolding integrable_on_def l by blast
next
  fix y
  assume y:  $y \in l$ 

```

```

    then have  $y \in \text{ball } x \ d$ 
    using as fine by fastforce
    then show  $\text{norm } (f \ y - f \ x) \leq e$ 
    using  $d \ x \ y$  as l
    by (metis dist_commute dist_norm less_imp_le mem_ball ptag_subsetCE
    tagged_division_ofD(3))
  qed
qed
from e have  $e \geq 0$ 
  by auto
from approximable_on_division[OF this division_of_tagged_division[OF ptag]]
*]
show  $\exists g. (\forall x \in \text{cbox } a \ b. \text{norm } (f \ x - g \ x) \leq e) \wedge g \text{ integrable\_on } \text{cbox } a \ b$ 
  by metis
qed

```

```

lemma integrable_continuous_interval:
  fixes  $f :: 'b::\text{ordered\_euclidean\_space} \Rightarrow 'a::\text{banach}$ 
  assumes continuous_on  $\{a..b\}$  f
  shows f integrable_on  $\{a..b\}$ 
  by (metis assms integrable_continuous_interval_cbox)

```

```

lemmas integrable_continuous_real = integrable_continuous_interval[where  $'b = \text{real}$ ]

```

```

lemma integrable_continuous_closed_segment:
  fixes  $f :: \text{real} \Rightarrow 'a::\text{banach}$ 
  assumes continuous_on (closed_segment a b) f
  shows f integrable_on (closed_segment a b)
  using assms
  by (auto intro!: integrable_continuous_interval simp: closed_segment_eq_real_ivl)

```

6.15.15 Specialization of additivity to one dimension

6.15.16 A useful lemma allowing us to factor out the content size

```

lemma has_integral_factor_content:
  (f has_integral i) (cbox a b)  $\longleftrightarrow$ 
  ( $\forall e > 0. \exists d. \text{gauge } d \wedge (\forall p. p \text{ tagged\_division\_of } (\text{cbox } a \ b) \wedge d \text{ fine } p \longrightarrow$ 
     $\text{norm } (\text{sum } (\lambda(x,k). \text{content } k *_R f \ x) \ p - i) \leq e * \text{content } (\text{cbox } a \ b)))$ 
proof (cases content (cbox a b) = 0)
  case True
  have  $\bigwedge e \ p. p \text{ tagged\_division\_of } \text{cbox } a \ b \implies \text{norm } ((\sum (x, k) \in p. \text{content } k *_R$ 
 $f \ x)) \leq e * \text{content } (\text{cbox } a \ b)$ 
    unfolding sum_content_null[OF True] True by force
  moreover have  $i = 0$ 
  if  $\bigwedge e. e > 0 \implies \exists d. \text{gauge } d \wedge$ 
    ( $\forall p. p \text{ tagged\_division\_of } \text{cbox } a \ b \wedge$ 
       $d \text{ fine } p \longrightarrow$ 
       $\text{norm } ((\sum (x, k) \in p. \text{content } k *_R f \ x) - i) \leq e * \text{content } (\text{cbox } a \ b))$ 

```



```

    using that [of 1]
  by (force simp add: True sum_content_null[OF True] intro: fine_division_exists[of
    _ a b])
  ultimately show ?thesis
    unfolding has_integral_null_eq[OF True]
    by (force simp add: )
next
case False
then have F:  $0 < \text{content } (\text{cbox } a \ b)$ 
  using zero_less_measure_iff by blast
let ?P =  $\lambda e \text{ opp. } \exists d. \text{ gauge } d \wedge$ 
  ( $\forall p. p \text{ tagged\_division\_of } (\text{cbox } a \ b) \wedge d \text{ fine } p \longrightarrow \text{opp } (\text{norm } ((\sum (x, k) \in p. \text{content } k *_R f x) - i)) \ e)$ )
show ?thesis
proof (subst has_integral, safe)
  fix e :: real
  assume e:  $e > 0$ 
  show ?P ( $e * \text{content } (\text{cbox } a \ b)$ ) ( $\leq$ ) if §[rule_format]:  $\forall \varepsilon > 0. ?P \ \varepsilon \ (<)$ 
    using § [of  $e * \text{content } (\text{cbox } a \ b)$ ]
    by (meson F e less_imp_le mult_pos_pos)
  show ?P e ( $<$ ) if §[rule_format]:  $\forall \varepsilon > 0. ?P \ (\varepsilon * \text{content } (\text{cbox } a \ b)) \ (\leq)$ 
    using § [of  $e/2 / \text{content } (\text{cbox } a \ b)$ ]
    using F e by (force simp add: algebra_simps)
qed
qed

```

lemma *has_integral_factor_content_real*:

```

(f has_integral i) {a..b::real}  $\longleftrightarrow$ 
  ( $\forall e > 0. \exists d. \text{ gauge } d \wedge (\forall p. p \text{ tagged\_division\_of } \{a..b\} \wedge d \text{ fine } p \longrightarrow$ 
     $\text{norm } (\text{sum } (\lambda(x,k). \text{content } k *_R f x) \ p - i) \leq e * \text{content } \{a..b\} \ ))$ )
unfolding box_real[symmetric]
by (rule has_integral_factor_content)

```

6.15.17 Fundamental theorem of calculus

lemma *interval_bounds_real*:

```

fixes q b :: real
assumes  $a \leq b$ 
shows  $\text{Sup } \{a..b\} = b$ 
  and  $\text{Inf } \{a..b\} = a$ 
using assms by auto

```

theorem *fundamental_theorem_of_calculus*:

```

fixes f :: real  $\Rightarrow$  'a::banach
assumes  $a \leq b$ 
  and vecd:  $\bigwedge x. x \in \{a..b\} \implies (f \text{ has\_vector\_derivative } f' \ x) \text{ (at } x \text{ within } \{a..b\})$ 
shows ( $f' \text{ has\_integral } (f \ b - f \ a) \ \{a..b\}$ )
unfolding has_integral_factor_content box_real[symmetric]

```

```

proof safe
  fix  $e :: \text{real}$ 
  assume  $e > 0$ 
  then have  $\forall x. \exists d > 0. x \in \{a..b\} \longrightarrow$ 
     $(\forall y \in \{a..b\}. \text{norm } (y-x) < d \longrightarrow \text{norm } (f y - f x - (y-x) *_R f' x) \leq e * \text{norm } (y-x))$ 
  using vecd[unfolded has_vector_derivative_def has_derivative_within_alt] by
    blast
  then obtain  $d$  where  $d: \bigwedge x. 0 < d x$ 
     $\bigwedge x y. [x \in \{a..b\}; y \in \{a..b\}; \text{norm } (y-x) < d x] \implies \text{norm } (f y - f x - (y-x) *_R f' x) \leq e * \text{norm } (y-x)$ 
  by metis
  show  $\exists d. \text{gauge } d \wedge (\forall p. p \text{ tagged\_division\_of } (\text{cbox } a \ b) \wedge d \text{ fine } p \longrightarrow$ 
     $\text{norm } ((\sum (x, k) \in p. \text{content } k *_R f' x) - (f b - f a)) \leq e * \text{content } (\text{cbox } a \ b))$ 
  proof (rule exI, safe)
    show gauge  $(\lambda x. \text{ball } x \ (d \ x))$ 
    using d(1) gauge_ball_dependent by blast
  next
    fix  $p$ 
    assume ptag:  $p \text{ tagged\_division\_of } \text{cbox } a \ b$  and finep:  $(\lambda x. \text{ball } x \ (d \ x)) \text{ fine } p$ 
    have ba:  $b - a = (\sum (x, K) \in p. \text{Sup } K - \text{Inf } K) f b - f a = (\sum (x, K) \in p. f(\text{Sup } K) - f(\text{Inf } K))$ 
    using additive_tagged_division_1 [where  $f = \lambda x. x$ ] additive_tagged_division_1 [where  $f = f$ ]
     $\langle a \leq b \rangle$  ptag by auto
    have  $\text{norm } (\sum (x, K) \in p. (\text{content } K *_R f' x) - (f(\text{Sup } K) - f(\text{Inf } K)))$ 
       $\leq (\sum n \in p. e * (\text{case } n \text{ of } (x, k) \Rightarrow \text{Sup } k - \text{Inf } k))$ 
    proof (rule sum_norm_le, safe)
      fix  $x \ K$ 
      assume  $(x, K) \in p$ 
      then have  $x \in K$  and kab:  $K \subseteq \text{cbox } a \ b$ 
      using ptag by blast+
      then obtain  $u \ v$  where  $k: K = \text{cbox } u \ v$  and  $x \in K$  and kab:  $K \subseteq \text{cbox } a \ b$ 
      using ptag  $\langle (x, K) \in p \rangle$  by auto
      have  $u \leq v$ 
      using  $\langle x \in K \rangle$  unfolding k by auto
      have ball:  $\forall y \in K. y \in \text{ball } x \ (d \ x)$ 
      using finep  $\langle (x, K) \in p \rangle$  by blast
      have  $u \in K \ v \in K$ 
      by (simp_all add:  $\langle u \leq v \rangle \ k$ )
      have  $\text{norm } ((v - u) *_R f' x - (f v - f u)) = \text{norm } (f u - f x - (u - x) *_R f' x - (f v - f x - (v - x) *_R f' x))$ 
      by (auto simp add: algebra_simps)
      also have  $\dots \leq \text{norm } (f u - f x - (u - x) *_R f' x) + \text{norm } (f v - f x - (v - x) *_R f' x)$ 
      by (rule norm_triangle_ineq4)
      finally have  $\text{norm } ((v - u) *_R f' x - (f v - f u)) \leq$ 
         $\text{norm } (f u - f x - (u - x) *_R f' x) + \text{norm } (f v - f x - (v - x) *_R f' x) .$ 
      also have  $\dots \leq e * \text{norm } (u - x) + e * \text{norm } (v - x)$ 

```

```

proof (rule add_mono)
  show norm (f u - f x - (u - x) *R f' x) ≤ e * norm (u - x)
proof (rule d)
  show norm (u - x) < d x
    using ⟨u ∈ K⟩ ball by (auto simp add: dist_real_def)
qed (use ⟨x ∈ K⟩ ⟨u ∈ K⟩ kab in auto)
show norm (f v - f x - (v - x) *R f' x) ≤ e * norm (v - x)
proof (rule d)
  show norm (v - x) < d x
    using ⟨v ∈ K⟩ ball by (auto simp add: dist_real_def)
qed (use ⟨x ∈ K⟩ ⟨v ∈ K⟩ kab in auto)
qed
also have ... ≤ e * (Sup K - Inf K)
  using ⟨x ∈ K⟩ by (auto simp: k interval_bounds_real[OF ⟨u ≤ v⟩]
field_simps)
  finally show norm (content K *R f' x - (f (Sup K) - f (Inf K))) ≤ e *
(Sup K - Inf K)
  using ⟨u ≤ v⟩ by (simp add: k)
qed
with ⟨a ≤ b⟩ show norm ((∑ (x, K) ∈ p. content K *R f' x) - (f b - f a)) ≤
e * content (cbox a b)
  by (auto simp: ba split_def sum_subtractf [symmetric] sum_distrib_left)
qed
qed

```

```

lemma ident_has_integral:
  fixes a::real
  assumes a ≤ b
  shows ((λx. x) has_integral (b2 - a2)/2) {a..b}
proof -
  have ((λx. x) has_integral inverse 2 * b2 - inverse 2 * a2) {a..b}
    unfolding power2_eq_square
    by (rule fundamental_theorem_of_calculus [OF assms] derivative_eq_intros |
simp)+
  then show ?thesis
    by (simp add: field_simps)
qed

```

```

lemma integral_ident [simp]:
  fixes a::real
  assumes a ≤ b
  shows integral {a..b} (λx. x) = (if a ≤ b then (b2 - a2)/2 else 0)
  by (metis assms ident_has_integral integral_unique)

```

```

lemma ident_integrable_on:
  fixes a::real
  shows (λx. x) integrable_on {a..b}
by (metis atLeastAtMost_empty_iff integrable_on_def has_integral_empty ident_has_integral)

```

```

lemma integral_sin [simp]:
  fixes a::real
  assumes  $a \leq b$  shows  $\text{integral } \{a..b\} \sin = \cos a - \cos b$ 
proof -
  have (sin has_integral (- cos b - - cos a)) {a..b}
  proof (rule fundamental_theorem_of_calculus)
    show (( $\lambda a. - \cos a$ ) has_vector_derivative sin x) (at x within {a..b}) for x
    unfolding has_field_derivative_iff_has_vector_derivative [symmetric]
    by (rule derivative_eq_intros | force)+
  qed (use assms in auto)
  then show ?thesis
    by (simp add: integral_unique)
qed

lemma integral_cos [simp]:
  fixes a::real
  assumes  $a \leq b$  shows  $\text{integral } \{a..b\} \cos = \sin b - \sin a$ 
proof -
  have (cos has_integral (sin b - sin a)) {a..b}
  proof (rule fundamental_theorem_of_calculus)
    show (sin has_vector_derivative cos x) (at x within {a..b}) for x
    unfolding has_field_derivative_iff_has_vector_derivative [symmetric]
    by (rule derivative_eq_intros | force)+
  qed (use assms in auto)
  then show ?thesis
    by (simp add: integral_unique)
qed

lemma has_integral_sin_nx: (( $\lambda x. \sin(\text{real\_of\_int } n * x)$ ) has_integral 0) {-pi..pi}
proof (cases  $n = 0$ )
  case False
  have (( $\lambda x. \sin (n * x)$ ) has_integral (- cos (n * pi)/n - - cos (n * - pi)/n))
  {-pi..pi}
  proof (rule fundamental_theorem_of_calculus)
    show (( $\lambda x. - \cos (n * x) / n$ ) has_vector_derivative sin (n * a)) (at a within
  {-pi..pi})
    if  $a \in \{-pi..pi\}$  for a :: real
    using that False
    unfolding has_vector_derivative_def
    by (intro derivative_eq_intros | force)+
  qed auto
  then show ?thesis
    by simp
qed auto

lemma integral_sin_nx:
  integral {-pi..pi} ( $\lambda x. \sin(x * \text{real\_of\_int } n)$ ) = 0
  using has_integral_sin_nx [of n] by (force simp: mult.commute)

```

```

lemma has_integral_cos_nx:
  (( $\lambda x. \cos(\text{real\_of\_int } n * x)$ ) has_integral (if  $n = 0$  then  $2 * \pi$  else  $0$ ))  $\{-\pi..pi\}$ 
proof (cases  $n = 0$ )
  case True
  then show ?thesis
    using has_integral_const_real [of  $1::\text{real } -\pi \pi$ ] by auto
next
  case False
  have (( $\lambda x. \cos (n * x)$ ) has_integral ( $\sin (n * \pi)/n - \sin (n * -\pi)/n$ ))
 $\{-\pi..pi\}$ 
  proof (rule fundamental_theorem_of_calculus)
    show (( $\lambda x. \sin (n * x) / n$ ) has_vector_derivative  $\cos (n * x)$ ) (at  $x$  within
 $\{-\pi..pi\}$ )
    if  $x \in \{-\pi..pi\}$ 
    for  $x :: \text{real}$ 
    using that False
    unfolding has_vector_derivative_def
    by (intro derivative_eq_intros | force)+
  qed auto
  with False show ?thesis
    by (simp add: mult.commute)
qed

```

```

lemma integral_cos_nx:
  integral  $\{-\pi..pi\}$  ( $\lambda x. \cos(x * \text{real\_of\_int } n)$ ) = (if  $n = 0$  then  $2 * \pi$  else  $0$ )
  using has_integral_cos_nx [of  $n$ ] by (force simp: mult.commute)

```

6.15.18 Taylor series expansion

```

lemma mvt_integral:
  fixes  $f::'a::\text{real\_normed\_vector} \Rightarrow 'b::\text{banach}$ 
  assumes  $f'[derivative\_intros]$ :
     $\bigwedge x. x \in S \implies (f \text{ has\_derivative } f' \ x) \text{ (at } x \text{ within } S)$ 
  assumes line_in:  $\bigwedge t. t \in \{0..1\} \implies x + t *_R y \in S$ 
  shows  $f(x + y) - f x = \text{integral } \{0..1\} (\lambda t. f' (x + t *_R y) \ y) \text{ (is ?th1)}$ 
proof –
  from assms have subset: ( $\lambda x a. x + x a *_R y$ ) ‘  $\{0..1\} \subseteq S$  by auto
  note [derivative_intros] =
    has_derivative_subset[OF subset]
    has_derivative_in_compose[where  $f=(\lambda x a. x + x a *_R y)$  and  $g = f$ ]
  note [continuous_intros] =
    continuous_on_compose2[where  $f=(\lambda x a. x + x a *_R y)$ ]
    continuous_on_subset[OF subset]
  have  $\bigwedge t. t \in \{0..1\} \implies$ 
    (( $\lambda t. f (x + t *_R y)$ ) has_vector_derivative  $f' (x + t *_R y) \ y$ )
    (at  $t$  within  $\{0..1\}$ )
  using assms
  by (auto simp: has_vector_derivative_def
    linear_cmul[OF has_derivative_linear[OF  $f'$ ], symmetric])

```

```

      intro!: derivative_eq_intros)
    from fundamental_theorem_of_calculus[rule_format, OF _ this]
  show ?th1
    by (auto intro!: integral_unique[symmetric])
qed

```

```

lemma (in bounded_bilinear) sum_prod_derivatives_has_vector_derivative:
  assumes p > 0
  and f0: Df 0 = f
  and Df:  $\bigwedge m t. m < p \implies a \leq t \implies t \leq b \implies$ 
    (Df m has_vector_derivative Df (Suc m) t) (at t within {a..b})
  and g0: Dg 0 = g
  and Dg:  $\bigwedge m t. m < p \implies a \leq t \implies t \leq b \implies$ 
    (Dg m has_vector_derivative Dg (Suc m) t) (at t within {a..b})
  and ivl:  $a \leq t \leq b$ 
  shows (( $\lambda t. \sum i < p. (-1)^i *_R \text{prod} (Df i t) (Dg (p - Suc i) t)$ )
    has_vector_derivative
    prod (f t) (Dg p t) - ( $-1$ )p *_R prod (Df p t) (g t))
    (at t within {a..b})
  using assms
proof cases
  assume p:  $p \neq 1$ 
  define p' where  $p' = p - 2$ 
  from assms p have p':  $\{..<p\} = \{..Suc\ p'\}$   $p = Suc (Suc\ p')$ 
  by (auto simp: p'_def)
  have *:  $\bigwedge i. i \leq p' \implies Suc (Suc\ p' - i) = (Suc (Suc\ p') - i)$ 
  by auto
  let ?f =  $\lambda i. (-1)^i *_R (\text{prod} (Df i t) (Dg ((p - i)) t))$ 
  have ( $\sum i < p. (-1)^i *_R (\text{prod} (Df i t) (Dg (Suc (p - Suc i)) t) +$ 
    prod (Df (Suc i) t) (Dg (p - Suc i) t))) =
    ( $\sum i \leq (Suc\ p'). ?f i - ?f (Suc i)$ )
  by (auto simp: algebra_simps p'(2) numeral_2_eq_2 * lessThan_Suc_atMost)
  also note sum_telescope
  finally
  have ( $\sum i < p. (-1)^i *_R (\text{prod} (Df i t) (Dg (Suc (p - Suc i)) t) +$ 
    prod (Df (Suc i) t) (Dg (p - Suc i) t)))
    = prod (f t) (Dg p t) - ( $-1$ )p *_R prod (Df p t) (g t)
  unfolding p'[symmetric]
  by (simp add: assms)
  thus ?thesis
  using assms
  by (auto intro!: derivative_eq_intros has_vector_derivative)
qed (auto intro!: derivative_eq_intros has_vector_derivative)

```

```

lemma
  fixes f::real=>'a::banach
  assumes p > 0
  and f0: Df 0 = f
  and Df:  $\bigwedge m t. m < p \implies a \leq t \implies t \leq b \implies$ 

```

```

  (Df m has_vector_derivative Df (Suc m) t) (at t within {a..b})
and ivl: a ≤ b
defines i ≡ λx. ((b - x) ^ (p - 1) / fact (p - 1)) *R Df p x
shows Taylor_has_integral:
  (i has_integral f b - (∑ i < p. ((b - a) ^ i / fact i) *R Df i a)) {a..b}
and Taylor_integral:
  f b = (∑ i < p. ((b - a) ^ i / fact i) *R Df i a) + integral {a..b} i
and Taylor_integrable:
  i integrable_on {a..b}
proof goal_cases
case 1
interpret bounded_bilinear scaleR::real⇒'a⇒'a
by (rule bounded_bilinear_scaleR)
define g where g s = (b - s) ^ (p - 1) / fact (p - 1) for s
define Dg where [abs_def]:
  Dg n s = (if n < p then (-1) ^ n * (b - s) ^ (p - 1 - n) / fact (p - 1 - n)
else 0) for n s
have g0: Dg 0 = g
using ⟨p > 0⟩
by (auto simp add: Dg_def field_split_simps g_def split: if_split_asm)
{
  fix m
  assume p > Suc m
  hence p - Suc m = Suc (p - Suc (Suc m))
  by auto
  hence real (p - Suc m) * fact (p - Suc (Suc m)) = fact (p - Suc m)
  by auto
}
note fact_eq = this
have Dg: ⋀m t. m < p ⟹ a ≤ t ⟹ t ≤ b ⟹
  (Dg m has_vector_derivative Dg (Suc m) t) (at t within {a..b})
unfolding Dg_def
by (auto intro!: derivative_eq_intros simp: has_vector_derivative_def fact_eq
field_split_simps)
let ?sum = λt. ∑ i < p. (- 1) ^ i *R Dg i t *R Df (p - Suc i) t
from sum_prod_derivatives_has_vector_derivative[of _ Dg _ _ Df,
OF ⟨p > 0⟩ g0 Dg f0 Df]
have deriv: ⋀t. a ≤ t ⟹ t ≤ b ⟹
  (?sum has_vector_derivative
    g t *R Df p t - (- 1) ^ p *R Dg p t *R f t) (at t within {a..b})
by auto
from fundamental_theorem_of_calculus[rule_format, OF ⟨a ≤ b⟩ deriv]
have (i has_integral ?sum b - ?sum a) {a..b}
using atLeastatMost_empty'[simp del]
by (simp add: i_def g_def Dg_def)
also
have one: (- 1) ^ p' * (- 1) ^ p' = (1::real)
and {..

```

```

    by (auto simp: power_mult_distrib[symmetric])
  then have ?sum b = f b
    using Suc_pred'[OF <p > 0>]
    by (simp add: diff_eq_eq Dg_def power_0_left le_Suc_eq if_distrib
        if_distribR sum.If_cases f0)
  also
  have {..

= ( $\lambda x. p - x - 1$ ) ' {..

proof safe
    fix x
    assume x < p
    thus x ∈ ( $\lambda x. p - x - 1$ ) ' {..

by (auto intro!: image_eqI[where x = p - x - 1])
  qed simp
  from _ this
  have ?sum a = ( $\sum i < p. ((b-a) ^ i / fact i) *_{\mathbb{R}} Df i a$ )
    by (rule sum.reindex_cong) (auto simp add: inj_on_def Dg_def one)
  finally show c: ?case .
  case 2 show ?case using c integral_unique
    by (metis (lifting) add commute diff_eq_eq integral_unique)
  case 3 show ?case using c by force
qed


```

6.15.19 Only need trivial subintervals if the interval itself is trivial

```

proposition division_of_nontrivial:
  fixes  $\mathcal{D} :: 'a::euclidean\_space \text{ set set}$ 
  assumes sdiv:  $\mathcal{D} \text{ division\_of } (cbox\ a\ b)$ 
  and cont0:  $content\ (cbox\ a\ b) \neq 0$ 
  shows  $\{k. k \in \mathcal{D} \wedge content\ k \neq 0\} \text{ division\_of } (cbox\ a\ b)$ 
  using sdiv
proof (induction card  $\mathcal{D}$  arbitrary;  $\mathcal{D}$  rule: less_induct)
  case less
  note  $\mathcal{D} = \text{division\_of } \mathcal{D}[OF\ less.premis]$ 
  {
    presume *:  $\{k \in \mathcal{D}. content\ k \neq 0\} \neq \mathcal{D} \implies ?case$ 
    then show ?case
      using less.premis by fastforce
  }
  assume noteq:  $\{k \in \mathcal{D}. content\ k \neq 0\} \neq \mathcal{D}$ 
  then obtain  $K\ c\ d$  where  $K \in \mathcal{D}$  and  $contk: content\ K = 0$  and  $keq: K =$ 
   $cbox\ c\ d$ 
    using  $\mathcal{D}(4)$  by blast
  then have card  $\mathcal{D} > 0$ 
    unfolding card_gt_0_iff using less by auto
  then have card:  $card\ (\mathcal{D} - \{K\}) < card\ \mathcal{D}$ 
    using less <K ∈  $\mathcal{D}$ > by (simp add:  $\mathcal{D}(1)$ )
  have closed:  $closed\ (\bigcup (\mathcal{D} - \{K\}))$ 
    using less.premis by auto

```



```

have  $x \text{ islimpt } \bigcup (\mathcal{D} - \{K\})$  if  $x \in K$  for  $x$ 
  unfolding islimpt_approachable
proof (intro allI impI)
  fix  $e :: \text{real}$ 
  assume  $e > 0$ 
  obtain  $i$  where  $i: c \cdot i = d \cdot i \ i \in \text{Basis}$ 
    using contk  $\mathcal{D}(3)$   $[OF \langle K \in \mathcal{D} \rangle]$  unfolding box_ne_empty keq
    by (meson content_eq_0 dual_order.antisym)
  then have  $xi: x \cdot i = d \cdot i$ 
    using  $\langle x \in K \rangle$  unfolding keq mem_box by (metis antisym)
  define  $y$  where  $y = (\sum_{j \in \text{Basis}. (if } j = i \text{ then if } c \cdot i \leq (a \cdot i + b \cdot i)/2 \text{ then } c \cdot i$ 
+
     $\min e (b \cdot i - c \cdot i)/2 \text{ else } c \cdot i - \min e (c \cdot i - a \cdot i)/2 \text{ else } x \cdot j) *_{\mathbb{R}} j)$ 
  show  $\exists x' \in \bigcup (\mathcal{D} - \{K\}). x' \neq x \wedge \text{dist } x' x < e$ 
proof (intro exI conjI)
  have  $d \in \text{cbox } c \ d$ 
  using  $\mathcal{D}(3)[OF \langle K \in \mathcal{D} \rangle]$  by (simp add: box_ne_empty(1) keq mem_box(2))
  then have  $d \in \text{cbox } a \ b$ 
    using  $\mathcal{D}(2)[OF \langle K \in \mathcal{D} \rangle]$  by (auto simp: keq)
  then have  $di: a \cdot i \leq d \cdot i \wedge d \cdot i \leq b \cdot i$ 
    using  $\langle i \in \text{Basis} \rangle \text{ mem\_box}(2)$  by blast
  then have  $xyi: y \cdot i \neq x \cdot i$ 
    unfolding  $y\_def \ i \ xi$  using  $\langle e > 0 \rangle \text{ cont0 } \langle i \in \text{Basis} \rangle$ 
    by (auto simp: content_eq_0 elim!: ballE[of _ _ i])
  then show  $y \neq x$ 
    unfolding euclidean_eq_iff[where 'a='a'] using  $i$  by auto
  have  $\text{norm } (y-x) \leq (\sum_{b \in \text{Basis}. |(y-x) \cdot b|)$ 
    by (rule norm_le_l1)
  also have  $\dots = |(y-x) \cdot i| + (\sum_{b \in \text{Basis} - \{i\}. |(y-x) \cdot b|)$ 
    by (meson finite_Basis  $i(2)$  sum.remove)
  also have  $\dots < e + \text{sum } (\lambda i. 0) \ \text{Basis}$ 
proof (rule add_less_le_mono)
  show  $|(y-x) \cdot i| < e$ 
    using  $di \ \langle e > 0 \rangle \ y\_def \ i \ xi$  by (auto simp: inner_simps)
  show  $(\sum_{i \in \text{Basis} - \{i\}. |(y-x) \cdot i|) \leq (\sum_{i \in \text{Basis}. 0)$ 
    unfolding  $y\_def$  by (auto simp: inner_simps)
qed
finally have  $\text{norm } (y-x) < e + \text{sum } (\lambda i. 0) \ \text{Basis} .$ 
then show  $\text{dist } y \ x < e$ 
  unfolding dist_norm by auto
have  $y \notin K$ 
  unfolding keq mem_box using  $i(1) \ i(2) \ xi \ xyi$  by fastforce
moreover have  $y \in \bigcup \mathcal{D}$ 
  using subsetD[ $OF \ \mathcal{D}(2)[OF \langle K \in \mathcal{D} \rangle] \ \langle x \in K \rangle \ \langle e > 0 \rangle \ di \ i$ 
    by (auto simp:  $\mathcal{D} \text{ mem\_box } y\_def \ \text{field\_simps} \ \text{elim!}: \text{ballE}[of \_ \_ i]$ )
ultimately show  $y \in \bigcup (\mathcal{D} - \{K\})$  by auto
qed
qed
then have  $K \subseteq \bigcup (\mathcal{D} - \{K\})$ 

```

```

    using closed closed_limpt by blast
  then have  $\bigcup (\mathcal{D} - \{K\}) = \text{cbox } a \ b$ 
    unfolding  $\mathcal{D}(6)[\text{symmetric}]$  by auto
  then have  $\mathcal{D} - \{K\}$  division_of cbox a b
    by (metis Diff_subset less.prems division_of_subset  $\mathcal{D}(6)$ )
  then have  $\{ka \in \mathcal{D} - \{K\}. \text{content } ka \neq 0\}$  division_of (cbox a b)
    using card less.hyps by blast
  moreover have  $\{ka \in \mathcal{D} - \{K\}. \text{content } ka \neq 0\} = \{K \in \mathcal{D}. \text{content } K \neq 0\}$ 
    using contk by auto
  ultimately show ?case by auto
qed

```

6.15.20 Integrability on subintervals

```

lemma operative_integrableI:
  fixes f :: 'b::euclidean_space  $\Rightarrow$  'a::banach
  assumes  $0 \leq e$ 
  shows operative conj True  $(\lambda i. f \text{ integrable\_on } i)$ 
proof -
  interpret comm_monoid conj True
  proof qed
  show ?thesis
  proof
    show  $\bigwedge a \ b. \text{box } a \ b = \{\} \implies (f \text{ integrable\_on cbox } a \ b) = \text{True}$ 
      by (simp add: content_eq_0_interior integrable_on_null)
    show  $\bigwedge a \ b \ c \ k.
      k \in \text{Basis} \implies
      (f \text{ integrable\_on cbox } a \ b) \longleftrightarrow
      (f \text{ integrable\_on cbox } a \ b \cap \{x. x \cdot k \leq c\} \wedge f \text{ integrable\_on cbox } a \ b \cap
      \{x. c \leq x \cdot k\})$ 
      unfolding integrable_on_def by (auto intro!: has_integral_split)
  qed
qed

```

```

lemma integrable_subinterval:
  fixes f :: 'b::euclidean_space  $\Rightarrow$  'a::banach
  assumes f: f integrable_on cbox a b
    and cd: cbox c d  $\subseteq$  cbox a b
  shows f integrable_on cbox c d
proof -
  interpret operative conj True  $\lambda i. f \text{ integrable\_on } i$ 
    using order_refl by (rule operative_integrableI)
  show ?thesis
  proof (cases cbox c d =  $\{\}$ )
    case True
    then show ?thesis
      using division [symmetric] f by (auto simp: comm_monoid_set_F_and)
  next
    case False

```

```

    then show ?thesis
      by (metis cd comm_monoid_set_F_and_division division_of_finite f partial_division_extend_1)
    qed
  qed

```

```

lemma integrable_subinterval_real:
  fixes f :: real  $\Rightarrow$  'a::banach
  assumes f integrable_on {a..b}
    and {c..d}  $\subseteq$  {a..b}
  shows f integrable_on {c..d}
  by (metis assms box_real(2) integrable_subinterval)

```

6.15.21 Combining adjacent intervals in 1 dimension

```

lemma has_integral_combine:
  fixes a b c :: real and j :: 'a::banach
  assumes a  $\leq$  c
    and c  $\leq$  b
    and ac: (f has_integral i) {a..c}
    and cb: (f has_integral j) {c..b}
  shows (f has_integral (i + j)) {a..b}
proof -
  interpret operative_real lift_option plus Some 0
     $\lambda i.$  if f integrable_on i then Some (integral i f) else None
  using operative_integralI by (rule operative_realI)
  from  $\langle a \leq c \rangle \langle c \leq b \rangle$  ac cb coalesce_less_eq
  have *: lift_option (+)
    (if f integrable_on {a..c} then Some (integral {a..c} f) else None)
    (if f integrable_on {c..b} then Some (integral {c..b} f) else None) =
    (if f integrable_on {a..b} then Some (integral {a..b} f) else None)
  by (auto simp: split: if_split_asm)
  then have f integrable_on cbox a b
    using ac cb by (auto split: if_split_asm)
  with *
  show ?thesis
    using ac cb by (auto simp add: integrable_on_def integral_unique split: if_split_asm)
  qed

```

```

lemma integral_combine:
  fixes f :: real  $\Rightarrow$  'a::banach
  assumes a  $\leq$  c
    and c  $\leq$  b
    and ab: f integrable_on {a..b}
  shows integral {a..c} f + integral {c..b} f = integral {a..b} f
proof -
  have (f has_integral integral {a..c} f) {a..c}
    using ab  $\langle c \leq b \rangle$  integrable_subinterval_real by fastforce

```

```

moreover
have (f has_integral integral {c..b} f) {c..b}
  using ab ⟨a ≤ c⟩ integrable_subinterval_real by fastforce
ultimately have (f has_integral integral {a..c} f + integral {c..b} f) {a..b}
  using ⟨a ≤ c⟩ ⟨c ≤ b⟩ has_integral_combine by blast
then show ?thesis
  by (simp add: has_integral_integrable_integral)
qed

```

```

lemma integrable_combine:
  fixes f :: real ⇒ 'a::banach
  assumes a ≤ c
    and c ≤ b
    and f integrable_on {a..c}
    and f integrable_on {c..b}
  shows f integrable_on {a..b}
  using assms
  unfolding integrable_on_def
  by (auto intro!: has_integral_combine)

```

```

lemma integral_minus_sets:
  fixes f::real ⇒ 'a::banach
  shows c ≤ a ⇒ c ≤ b ⇒ f integrable_on {c .. max a b} ⇒
    integral {c .. a} f - integral {c .. b} f =
      (if a ≤ b then - integral {a .. b} f else integral {b .. a} f)
  using integral_combine[of c a b f] integral_combine[of c b a f]
  by (auto simp: algebra_simps max_def)

```

```

lemma integral_minus_sets':
  fixes f::real ⇒ 'a::banach
  shows c ≥ a ⇒ c ≥ b ⇒ f integrable_on {min a b .. c} ⇒
    integral {a .. c} f - integral {b .. c} f =
      (if a ≤ b then integral {a .. b} f else - integral {b .. a} f)
  using integral_combine[of b a c f] integral_combine[of a b c f]
  by (auto simp: algebra_simps min_def)

```

6.15.22 Reduce integrability to "local" integrability

```

lemma integrable_on_little_subintervals:
  fixes f :: 'b::euclidean_space ⇒ 'a::banach
  assumes ∀ x ∈ cbox a b. ∃ d > 0. ∀ u v. x ∈ cbox u v ∧ cbox u v ⊆ ball x d ∧ cbox
u v ⊆ cbox a b ⟶
  f integrable_on cbox u v
  shows f integrable_on cbox a b
proof -
  interpret operative conj True λi. f integrable_on i
  using order_refl by (rule operative_integrableI)
  have ∀ x. ∃ d > 0. x ∈ cbox a b ⟶ (∀ u v. x ∈ cbox u v ∧ cbox u v ⊆ ball x d ∧
cbox u v ⊆ cbox a b ⟶

```

```

    f integrable_on cbox u v
  using assms by (metis zero_less_one)
  then obtain d where d:  $\bigwedge x. 0 < d \ x$ 
     $\bigwedge x \ u \ v. \llbracket x \in \text{cbox } a \ b; x \in \text{cbox } u \ v; \text{cbox } u \ v \subseteq \text{ball } x \ (d \ x); \text{cbox } u \ v \subseteq \text{cbox } a \ b \rrbracket$ 
     $\implies f \text{ integrable\_on } \text{cbox } u \ v$ 
  by metis
  obtain p where p: p tagged_division_of cbox a b ( $\lambda x. \text{ball } x \ (d \ x)$ ) fine p
  using fine_division_exists[OF gauge_ball_dependent, of d a b] d(1) by blast
  then have sndp: snd 'p division_of cbox a b
  by (metis division_of_tagged_division)
  have f integrable_on k if (x, k)  $\in$  p for x k
  using tagged_division_ofD(2-4)[OF p(1) that] fineD[OF p(2) that] d[of x]
  by auto
  then show ?thesis
  unfolding division [symmetric, OF sndp] comm_monoid_set_F_and
  by auto
qed

```

6.15.23 Second FTC or existence of antiderivative

```

lemma integrable_const[intro]: ( $\lambda x. c$ ) integrable_on cbox a b
  unfolding integrable_on_def by blast

```

```

lemma integral_has_vector_derivative_continuous_at:
  fixes f :: real  $\Rightarrow$  'a::banach
  assumes f: f integrable_on {a..b}
    and x:  $x \in \{a..b\} - S$ 
    and finite S
    and fx: continuous (at x within ( $\{a..b\} - S$ )) f
  shows (( $\lambda u. \text{integral } \{a..u\} f$ ) has_vector_derivative f x) (at x within ( $\{a..b\} - S$ ))
  proof -
    let ?I =  $\lambda a \ b. \text{integral } \{a..b\} f$ 
    { fix e::real
      assume e > 0
      obtain d where d>0 and d:  $\bigwedge x'. \llbracket x' \in \{a..b\} - S; |x' - x| < d \rrbracket \implies \text{norm}(f \ x' - f \ x) \leq e$ 
        using  $\langle e > 0 \rangle$  fx by (auto simp: continuous_within_eps_delta dist_norm less_imp_le)
      have norm (integral {a..y} f - integral {a..x} f - (y-x) *R f x)  $\leq e * |y - x|$  (is ?lhs  $\leq$  ?rhs)
        if y:  $y \in \{a..b\} - S$  and yx:  $|y - x| < d$  for y
      proof (cases y < x)
        case False
        have f integrable_on {a..y}
          using f y by (simp add: integrable_subinterval_real)
        then have Idiff: ?I a y - ?I a x = ?I x y
          using False x by (simp add: algebra_simps integral_combine)

```

```

    have fux_int: ((λu. f u - f x) has_integral integral {x..y} f - (y-x) *R f x)
    {x..y}
  proof (rule has_integral_diff)
    show (f has_integral integral {x..y} f) {x..y}
    using x y by (auto intro: integrable_integral [OF integrable_subinterval_real
[OF f]])
    show ((λu. f x) has_integral (y - x) *R f x) {x..y}
    using has_integral_const_real [of f x x y] False by simp
  qed
  have ?lhs ≤ e * content {x..y}
  using yx False d x y ⟨e>0⟩ assms
  by (intro has_integral_bound_real[where f=(λu. f u - f x)]) (auto simp:
Idiff fux_int)
  also have ... ≤ ?rhs
  using False by auto
  finally show ?thesis .
next
case True
have f_integrable_on {a..x}
  using f x by (simp add: integrable_subinterval_real)
then have Idiff: ?I a x - ?I a y = ?I y x
  using True x y by (simp add: algebra_simps integral_combine)
have fux_int: ((λu. f u - f x) has_integral integral {y..x} f - (x - y) *R f
x) {y..x}
  proof (rule has_integral_diff)
    show (f has_integral integral {y..x} f) {y..x}
    using x y by (auto intro: integrable_integral [OF integrable_subinterval_real
[OF f]])
    show ((λu. f x) has_integral (x - y) *R f x) {y..x}
    using has_integral_const_real [of f x y x] True by simp
  qed
  have norm (integral {a..x} f - integral {a..y} f - (x - y) *R f x) ≤ e *
content {y..x}
  using yx True d x y ⟨e>0⟩ assms
  by (intro has_integral_bound_real[where f=(λu. f u - f x)]) (auto simp:
Idiff fux_int)
  also have ... ≤ e * |y - x|
  using True by auto
  finally have norm (integral {a..x} f - integral {a..y} f - (x - y) *R f x) ≤
e * |y - x| .
  then show ?thesis
  by (simp add: algebra_simps norm_minus_commute)
  qed
  then have ∃ d>0. ∀ y∈{a..b} - S. |y - x| < d ⟶ norm (integral {a..y} f -
integral {a..x} f - (y-x) *R f x) ≤ e * |y - x|
  using ⟨d>0⟩ by blast
}
then show ?thesis
by (simp add: has_vector_derivative_def has_derivative_within_alt bounded_linear_scaleR_left)

```

qed

```
lemma integral_has_vector_derivative:
  fixes f :: real  $\Rightarrow$  'a::banach
  assumes continuous_on {a..b} f
    and x  $\in$  {a..b}
  shows (( $\lambda u$ . integral {a..u} f) has_vector_derivative f(x)) (at x within {a..b})
using assms integral_has_vector_derivative_continuous_at [OF integrable_continuous_real]
  by (fastforce simp: continuous_on_eq_continuous_within)
```

```
lemma integral_has_real_derivative:
  assumes continuous_on {a..b} g
  assumes t  $\in$  {a..b}
  shows (( $\lambda x$ . integral {a..x} g) has_real_derivative g t) (at t within {a..b})
  using integral_has_vector_derivative[of a b g t] assms
  by (auto simp: has_field_derivative_iff_has_vector_derivative)
```

```
lemma antiderivative_continuous:
  fixes q b :: real
  assumes continuous_on {a..b} f
  obtains g where  $\bigwedge x. x \in \{a..b\} \implies (g \text{ has\_vector\_derivative } (f x :: \_::\text{banach}))$ 
    (at x within {a..b})
  using integral_has_vector_derivative[OF assms] by auto
```

6.15.24 Combined fundamental theorem of calculus

```
lemma antiderivative_integral_continuous:
  fixes f :: real  $\Rightarrow$  'a::banach
  assumes continuous_on {a..b} f
  obtains g where  $\forall u \in \{a..b\}. \forall v \in \{a..b\}. u \leq v \implies (f \text{ has\_integral } (g v - g u)) \{u..v\}$ 
proof -
  obtain g
    where g:  $\bigwedge x. x \in \{a..b\} \implies (g \text{ has\_vector\_derivative } f x) (at x \text{ within } \{a..b\})$ 
    using antiderivative_continuous[OF assms] by metis
  have (f has_integral g v - g u) {u..v} if u  $\in$  {a..b} v  $\in$  {a..b} u  $\leq$  v for u v
  proof -
    have  $\bigwedge x. x \in \text{cbox } u \ v \implies (g \text{ has\_vector\_derivative } f x) (at x \text{ within } \text{cbox } u \ v)$ 
      by (metis atLeastAtMost_iff atLeastatMost_subset_iff box_real(2) g
        has_vector_derivative_within_subset subsetCE that(1,2))
    then show ?thesis
      by (metis box_real(2) that(3) fundamental_theorem_of_calculus)
  qed
  then show ?thesis
    using that by blast
qed
```

6.15.25 General "twiddling" for interval-to-interval function image

```

lemma has_integral_twiddle:
  assumes  $0 < r$ 
    and hg:  $\bigwedge x. h(g\ x) = x$ 
    and gh:  $\bigwedge x. g(h\ x) = x$ 
    and contg:  $\bigwedge x. \text{continuous}\ (at\ x)\ g$ 
    and g:  $\bigwedge u\ v. \exists w\ z. g\ ' \text{cbox}\ u\ v = \text{cbox}\ w\ z$ 
    and h:  $\bigwedge u\ v. \exists w\ z. h\ ' \text{cbox}\ u\ v = \text{cbox}\ w\ z$ 
    and r:  $\bigwedge u\ v. \text{content}(g\ ' \text{cbox}\ u\ v) = r * \text{content}\ (\text{cbox}\ u\ v)$ 
    and intfi:  $(f\ \text{has\_integral}\ i)\ (\text{cbox}\ a\ b)$ 
  shows  $((\lambda x. f(g\ x))\ \text{has\_integral}\ (1 / r) *_R i)\ (h\ ' \text{cbox}\ a\ b)$ 
proof (cases  $\text{cbox}\ a\ b = \{\}$ )
  case True
    then show ?thesis
      using intfi by auto
  next
  case False
    obtain w z where wz:  $h\ ' \text{cbox}\ a\ b = \text{cbox}\ w\ z$ 
      using h by blast
    have inj:  $\text{inj}\ g\ \text{inj}\ h$ 
      using hg gh injI by metis+
    from h obtain ha hb where h_eq:  $h\ ' \text{cbox}\ a\ b = \text{cbox}\ ha\ hb$  by blast
    have  $\exists d. \text{gauge}\ d \wedge (\forall p. p\ \text{tagged\_division\_of}\ h\ ' \text{cbox}\ a\ b \wedge d\ \text{fine}\ p$ 
       $\longrightarrow \text{norm}\ ((\sum (x, k) \in p. \text{content}\ k * f(g\ x)) - (1 / r) * i) < e)$ 
      if  $e > 0$  for e
    proof -
      have  $e * r > 0$  using that  $\langle 0 < r \rangle$  by simp
      with intfi[unfolded has_integral]
      obtain d where gauge d
        and d:  $\bigwedge p. p\ \text{tagged\_division\_of}\ \text{cbox}\ a\ b \wedge d\ \text{fine}\ p$ 
         $\implies \text{norm}\ ((\sum (x, k) \in p. \text{content}\ k * f\ x) - i) < e * r$ 
      by metis
      define d' where  $d'\ x = g\ -' d\ (g\ x)$  for x
      show ?thesis
        proof (rule_tac  $x=d'$  in exI, safe)
          show gauge d'
            using  $\langle \text{gauge}\ d \rangle\ \text{continuous\_open\_vimage}[OF\ \_\ \text{contg}]$  by (auto simp:
            gauge_def d'_def)
        next
        fix p
        assume ptag:  $p\ \text{tagged\_division\_of}\ h\ ' \text{cbox}\ a\ b$  and finep:  $d'\ \text{fine}\ p$ 
        note  $p = \text{tagged\_division\_of}\ D[OF\ ptag]$ 
        have gab:  $g\ y \in \text{cbox}\ a\ b$  if  $y \in K\ (x, K) \in p$  for  $x\ y\ K$ 
          by (metis hg inj(2) inj_image_mem_iff p(3) subsetCE that that)
        have gimp:  $(\lambda(x, K). (g\ x, g\ ' K))\ ' p\ \text{tagged\_division\_of}\ (\text{cbox}\ a\ b) \wedge$ 
           $d\ \text{fine}\ (\lambda(x, k). (g\ x, g\ ' k))\ ' p$ 
          unfolding tagged_division_of
        proof safe

```



```

show finite (( $\lambda(x, k). (g\ x, g\ 'k)$ ) '  $p$ )
  using ptag by auto
show d fine ( $\lambda(x, k). (g\ x, g\ 'k)$ ) '  $p$ 
  using finep unfolding fine_def d'_def by auto
next
fix x k
assume xk:  $(x, k) \in p$ 
show  $g\ x \in g\ 'k$ 
  using p(2)[OF xk] by auto
show  $\exists u\ v. g\ 'k = \text{cbox } u\ v$ 
  using p(4)[OF xk] using assms(5-6) by auto
fix x' K' u
assume xk':  $(x', K') \in p$  and u:  $u \in \text{interior } (g\ 'k) \wedge u \in \text{interior } (g\ 'K')$ 
have  $\text{interior } k \cap \text{interior } K' \neq \{\}$ 
proof
  assume  $\text{interior } k \cap \text{interior } K' = \{\}$ 
  moreover have  $u \in g\ '(\text{interior } k \cap \text{interior } K')$ 
    using interior_image_subset[OF inj g] contg u
    unfolding image_Int[OF inj(1)] by blast
  ultimately show False by blast
qed
then have same:  $(x, k) = (x', K')$ 
  using ptag xk' xk by blast
then show  $g\ x = g\ x'$ 
  by auto
show  $g\ u \in g\ 'K'$  if  $u \in k$  for u
  using that same by auto
show  $g\ u \in g\ 'k$  if  $u \in K'$  for u
  using that same by auto
next
fix x
assume  $x \in \text{cbox } a\ b$ 
then have  $h\ x \in \bigcup \{k. \exists x. (x, k) \in p\}$ 
  using p(6) by auto
then obtain X y where  $h\ x \in X \wedge (y, X) \in p$  by blast
then show  $x \in \bigcup \{k. \exists x. (x, k) \in (\lambda(x, k). (g\ x, g\ 'k))\ ' p\}$ 
  by clarsimp (metis (no_types, lifting) gh_image_eqI pair_imageI)
qed (use gab in auto)
have *:  $\text{inj\_on } (\lambda(x, k). (g\ x, g\ 'k))\ p$ 
  using inj(1) unfolding inj_on_def by fastforce
have  $(\sum_{(x,K) \in (\lambda(y,L). (g\ y, g\ 'L))\ ' p. \text{content } K *_R f\ x})$ 
  =  $(\sum_{u \in p. \text{case case } u \text{ of } (x,K) \Rightarrow (g\ x, g\ 'K) \text{ of } (y,L) \Rightarrow \text{content } L *_R$ 
 $f\ y)$ 
  by (metis (mono_tags, lifting) * sum.reindex_cong)
also have ... =  $(\sum_{(x,K) \in p. r *_R \text{content } K *_R f\ (g\ x)})$ 
  using r by (auto intro!: * sum.cong simp: bij_betw_def dest!: p(4))
finally
  have  $(\sum_{(x,K) \in (\lambda(x,K). (g\ x, g\ 'K))\ ' p. \text{content } K *_R f\ x}) - i = r *_R$ 
 $(\sum_{(x,K) \in p. \text{content } K *_R f\ (g\ x)}) - i$ 

```

```

    by (simp add: scaleR_right.sum split_def)
  also have ... = r *R (( $\sum (x,K) \in p.$  content  $K *_{\mathbb{R}} f (g x)$ ) - ( $1 / r$ ) *R i)
    using <0 < r> by (auto simp: scaleR_diff_right)
  finally show norm (( $\sum (x,K) \in p.$  content  $K *_{\mathbb{R}} f (g x)$ ) - ( $1 / r$ ) *R i) < e
    using d[OF gimp] <0 < r> by auto
qed
qed
then show ?thesis
  by (auto simp: h_eq has_integral)
qed

```

6.15.26 Special case of a basic affine transformation

```

lemma AE_lborel_inner_neg:
  assumes k: k ∈ Basis
  shows AE x in lborel. x • k ≠ c
proof -
  interpret finite_product_sigma_finite λ_. lborel Basis
  proof qed simp
  have emeasure lborel {x ∈ space lborel. x • k = c}
    = emeasure (ΠM j::'a ∈ Basis. lborel) (ΠE j ∈ Basis. if j = k then {c} else
UNIV)
    using k
  by (auto simp add: lborel_eq[where 'a='a] emeasure_distr intro!: arg_cong2[where
f=emeasure])
    (auto simp: space_PiM PiE_iff extensional_def split: if_split_asm)
  also have ... = (Π j ∈ Basis. emeasure lborel (if j = k then {c} else UNIV))
    by (intro measure_times) auto
  also have ... = 0
    by (intro prod_zero bexI[OF _ k]) auto
  finally show ?thesis
    by (subst AE_iff_measurable[OF _ refl]) auto
qed

```

```

lemma content_image_stretch_interval:
  fixes m :: 'a::euclidean_space ⇒ real
  defines s f x ≡ ( $\sum k::'a \in \text{Basis. } (f k * (x \cdot k)) *_{\mathbb{R}} k$ )
  shows content (s m ' cbox a b) = |Π k ∈ Basis. m k| * content (cbox a b)
proof cases
  have s[measurable]: s f ∈ borel →M borel for f
    by (auto simp: s_def[abs_def])
  assume m: ∀ k ∈ Basis. m k ≠ 0
  then have s_comp_s: s (λk. 1 / m k) ∘ s m = id s m ∘ s (λk. 1 / m k) = id
    by (auto simp: s_def[abs_def] fun_eq_iff euclidean_representation)
  then have inv (s (λk. 1 / m k)) = s m bij (s (λk. 1 / m k))
    by (auto intro: inv_unique_comp o_bij)
  then have eq: s m ' cbox a b = s (λk. 1 / m k) - ' cbox a b
    using bij_vimage_eq_inv_image[OF <bij (s (λk. 1 / m k))>, of cbox a b] by
auto

```

```

show ?thesis
  using m unfolding eq measure_def
  by (subst lborel_affine_euclidean[where c=m and t=0])
    (simp_all add: emeasure_density measurable_sets_borel[OF s] abs_prod
nn_integral_cmult
      s_def[symmetric] emeasure_distr vimage_comp s_comp_s
enn2real_mult prod_nonneg)
next
  assume  $\neg (\forall k \in \text{Basis}. m\ k \neq 0)$ 
  then obtain k where  $k \in \text{Basis}$   $m\ k = 0$  by auto
  then have [simp]:  $(\prod k \in \text{Basis}. m\ k) = 0$ 
    by (intro prod_zero) auto
  have emeasure_lborel  $\{x \in \text{space lborel}. x \in s\ m\ \text{cbox } a\ b\} = 0$ 
  proof (rule emeasure_eq_0_AE)
    show AE x in lborel.  $x \notin s\ m\ \text{cbox } a\ b$ 
      using AE_lborel_inner_neg[OF  $\langle k \in \text{Basis} \rangle$ ]
    proof eventually_elim
      show  $x \cdot k \neq 0 \implies x \notin s\ m\ \text{cbox } a\ b$  for x
        using k by (auto simp: s_def[abs_def] cbox_def)
    qed
  qed
qed
then show ?thesis
  by (simp add: measure_def)
qed

lemma interval_image_affinity_interval:
   $\exists u\ v. (\lambda x. m *_R (x::'a::\text{euclidean\_space}) + c)\ \text{cbox } a\ b = \text{cbox } u\ v$ 
  unfolding image_affinity_cbox
  by auto

lemma content_image_affinity_cbox:
   $\text{content}((\lambda x. 'a::\text{euclidean\_space}. m *_R x + c)\ \text{cbox } a\ b) =$ 
 $|m| \wedge \text{DIM}('a) * \text{content}(\text{cbox } a\ b)$  (is ?l = ?r)
proof (cases  $\text{cbox } a\ b = \{\}$ )
  case True then show ?thesis by simp
next
  case False
  show ?thesis
  proof (cases  $m \geq 0$ )
    case True
    with  $\langle \text{cbox } a\ b \neq \{\} \rangle$  have  $\text{cbox } (m *_R a + c)\ (m *_R b + c) \neq \{\}$ 
      by (simp add: box_ne_empty inner_left_distrib mult_left_mono)
    moreover from True have *:  $\bigwedge i. (m *_R b + c) \cdot i - (m *_R a + c) \cdot i = m$ 
 $*_R (b - a) \cdot i$ 
      by (simp add: inner_simps field_simps)
    ultimately show ?thesis
      by (simp add: image_affinity_cbox True content_cbox'
prod.distrib inner_diff_left)
  next

```

```

case False
with  $\langle \text{cbox } a \ b \neq \{\} \rangle$  have  $\text{cbox } (m *_R b + c) \ (m *_R a + c) \neq \{\}$ 
  by (simp add: box_ne_empty inner_left_distrib mult_left_mono)
moreover from False have  $\ast: \bigwedge i. (m *_R a + c) \cdot i - (m *_R b + c) \cdot i =$ 
 $(-m) *_R (b - a) \cdot i$ 
  by (simp add: inner_simps field_simps)
ultimately show ?thesis using False
  by (simp add: image_affinity_cbox content_cbox'
    prod.distrib[symmetric] inner_diff_left_flip: prod_constant)
qed
qed

```

```

lemma has_integral_affinity:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  assumes (f has_integral i) (cbox a b)
  and  $m \neq 0$ 
  shows  $((\lambda x. f(m *_R x + c)) \text{ has\_integral } (1 / (|m| \wedge \text{DIM}('a))) *_R i) ((\lambda x. (1$ 
 $/ m) *_R x + -((1 / m) *_R c)) \text{ ' cbox } a \ b)$ 
proof (rule has_integral_twiddle)
  show  $\exists w \ z. (\lambda x. (1 / m) *_R x + -((1 / m) *_R c)) \text{ ' cbox } u \ v = \text{cbox } w \ z$ 
     $\exists w \ z. (\lambda x. m *_R x + c) \text{ ' cbox } u \ v = \text{cbox } w \ z$  for  $u \ v$ 
  using interval_image_affinity_interval by blast+
  show  $\text{content } ((\lambda x. m *_R x + c) \text{ ' cbox } u \ v) = |m| \wedge \text{DIM}('a) * \text{content } (\text{cbox } u$ 
 $v)$  for  $u \ v$ 
  using content_image_affinity_cbox by blast
qed (use assms zero_less_power in <auto simp: field_simps>)

```

```

lemma integrable_affinity:
  assumes f integrable_on cbox a b
  and  $m \neq 0$ 
  shows  $(\lambda x. f(m *_R x + c)) \text{ integrable\_on } ((\lambda x. (1 / m) *_R x + -((1 / m) *_R$ 
 $c)) \text{ ' cbox } a \ b)$ 
  using has_integral_affinity assms
  unfolding integrable_on_def by blast

```

lemmas *has_integral_affinity01* = *has_integral_affinity* [*of _ _ 0 1::real, simplified*]

```

lemma integrable_on_affinity:
  assumes  $m \neq 0$  f integrable_on (cbox a b)
  shows  $(\lambda x. f(m *_R x + c)) \text{ integrable\_on } ((\lambda x. (1 / m) *_R x - ((1 / m) *_R$ 
 $c)) \text{ ' cbox } a \ b)$ 
proof –
  from assms obtain I where (f has_integral I) (cbox a b)
  by (auto simp: integrable_on_def)
  from has_integral_affinity[OF this assms(1), of c] show ?thesis
  by (auto simp: integrable_on_def)
qed

```

```

lemma has_integral_cmul_iff:
  assumes  $c \neq 0$ 
  shows  $((\lambda x. c *_{\mathbb{R}} f x) \text{ has\_integral } (c *_{\mathbb{R}} I)) \iff (f \text{ has\_integral } I) \wedge A$ 
  using assms has_integral_cmul[of  $f$   $I$   $A$   $c$ ]
        has_integral_cmul[of  $\lambda x. c *_{\mathbb{R}} f x$   $c *_{\mathbb{R}} I$   $A$   $\text{inverse } c$ ] by (auto simp:
field_simps)

lemma has_integral_affinity':
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  assumes  $(f \text{ has\_integral } i) \wedge (c \text{ box } a \ b) \text{ and } m > 0$ 
  shows  $((\lambda x. f(m *_{\mathbb{R}} x + c)) \text{ has\_integral } (i /_{\mathbb{R}} m \wedge \text{DIM}('a)))$ 
         $(c \text{ box } ((a - c) /_{\mathbb{R}} m) ((b - c) /_{\mathbb{R}} m))$ 
proof (cases  $c \text{ box } a \ b = \{\}$ )
  case True
  hence  $(c \text{ box } ((a - c) /_{\mathbb{R}} m) ((b - c) /_{\mathbb{R}} m)) = \{\}$ 
    using  $\langle m > 0 \rangle$  unfolding box_eq_empty by (auto simp: algebra_simps)
  with True and assms show ?thesis by simp
next
  case False
  have  $((\lambda x. f(m *_{\mathbb{R}} x + c)) \text{ has\_integral } (1 / |m| \wedge \text{DIM}('a)) *_{\mathbb{R}} i)$ 
         $((\lambda x. (1 / m) *_{\mathbb{R}} x + -((1 / m) *_{\mathbb{R}} c)) \text{ ' } c \text{ box } a \ b)$ 
    using assms by (intro has_integral_affinity) auto
  also have  $((\lambda x. (1 / m) *_{\mathbb{R}} x + -((1 / m) *_{\mathbb{R}} c)) \text{ ' } c \text{ box } a \ b) =$ 
         $((\lambda x. -((1 / m) *_{\mathbb{R}} c) + x) \text{ ' } (\lambda x. (1 / m) *_{\mathbb{R}} x) \text{ ' } c \text{ box } a \ b)$ 
    by (simp add: image_image algebra_simps)
  also have  $(\lambda x. (1 / m) *_{\mathbb{R}} x) \text{ ' } c \text{ box } a \ b = c \text{ box } ((1 / m) *_{\mathbb{R}} a) ((1 / m) *_{\mathbb{R}} b)$ 
using  $\langle m > 0 \rangle$  False
  by (subst image_smult_box) simp_all
  also have  $(\lambda x. -((1 / m) *_{\mathbb{R}} c) + x) \text{ ' } \dots = c \text{ box } ((a - c) /_{\mathbb{R}} m) ((b - c) /_{\mathbb{R}} m)$ 
    by (subst box_translation [symmetric]) (simp add: field_simps vector_add_divide_simps)
  finally show ?thesis using  $\langle m > 0 \rangle$  by (simp add: field_simps)
qed

lemma has_integral_affinity_iff:
  fixes  $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{real\_normed\_vector}$ 
  assumes  $m > 0$ 
  shows  $((\lambda x. f(m *_{\mathbb{R}} x + c)) \text{ has\_integral } (I /_{\mathbb{R}} m \wedge \text{DIM}('a)))$ 
         $(c \text{ box } ((a - c) /_{\mathbb{R}} m) ((b - c) /_{\mathbb{R}} m)) \iff$ 
         $(f \text{ has\_integral } I) \wedge (c \text{ box } a \ b) \text{ (is ?lhs = ?rhs)}$ 
proof
  assume ?lhs
  from has_integral_affinity'[OF this, of  $1 / m - c /_{\mathbb{R}} m$ ] and  $\langle m > 0 \rangle$ 
  show ?rhs by (simp add: vector_add_divide_simps) (simp add: field_simps)
next
  assume ?rhs
  from has_integral_affinity'[OF this, of  $m$   $c$ ] and  $\langle m > 0 \rangle$ 
  show ?lhs by simp
qed

```

6.15.27 Special case of stretching coordinate axes separately

lemma *has_integral_stretch*:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{real\_normed\_vector}$

assumes $(f \text{ has\_integral } i) \text{ (cbox } a \text{ } b)$

and $\forall k \in \text{Basis}. m \ k \neq 0$

shows $((\lambda x. f (\sum_{k \in \text{Basis}} (m \ k * (x \cdot k)) *_{\mathbb{R}} k)) \text{ has\_integral}$

$((1 / |\text{prod } m \text{ Basis}|) *_{\mathbb{R}} i)) ((\lambda x. (\sum_{k \in \text{Basis}} (1 / m \ k * (x \cdot k)) *_{\mathbb{R}} k)) \text{ ' cbox } a \text{ } b)$

apply $(\text{rule has\_integral\_twiddle}[\text{where } f=f])$

unfolding *zero_less_abs_iff content_image_stretch_interval*

unfolding *image_stretch_interval empty_as_interval euclidean_eq_iff* **[where** $'a = 'a]$

using *assms*

by *auto*

lemma *has_integral_stretch_real*:

fixes $f :: \text{real} \Rightarrow 'b::\text{real\_normed\_vector}$

assumes $(f \text{ has\_integral } i) \{a..b\}$ **and** $m \neq 0$

shows $((\lambda x. f (m * x)) \text{ has\_integral } (1 / |m|) *_{\mathbb{R}} i) ((\lambda x. x / m) \text{ ' } \{a..b\})$

using *has_integral_stretch* **[of** $f \ i \ a \ b \ \lambda b. m]$ **assms** **by** *simp*

lemma *integrable_stretch*:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{real\_normed\_vector}$

assumes $f \text{ integrable\_on cbox } a \text{ } b$

and $\forall k \in \text{Basis}. m \ k \neq 0$

shows $(\lambda x::'a. f (\sum_{k \in \text{Basis}} (m \ k * (x \cdot k)) *_{\mathbb{R}} k)) \text{ integrable\_on}$

$((\lambda x. \sum_{k \in \text{Basis}} (1 / m \ k * (x \cdot k)) *_{\mathbb{R}} k) \text{ ' cbox } a \text{ } b)$

using *assms* **unfolding** *integrable_on_def*

by $(\text{force dest: has\_integral\_stretch})$

lemma *vec_lambda_eq_sum*:

$(\chi \ k. f \ k (x \ \$ \ k)) = (\sum_{k \in \text{Basis}} (f (\text{axis\_index } k) (x \cdot k)) *_{\mathbb{R}} k) \text{ (is ?lhs = ?rhs)}$

proof –

have $?lhs = (\chi \ k. f \ k (x \cdot \text{axis } k \ 1))$

by $(\text{simp add: cart\_eq\_inner\_axis})$

also have $\dots = (\sum_{u \in \text{UNIV}. f \ u (x \cdot \text{axis } u \ 1) *_{\mathbb{R}} \text{axis } u \ 1)$

by $(\text{simp add: vec\_eq\_iff axis\_def if\_distrib cong: if\_cong})$

also have $\dots = ?rhs$

by $(\text{simp add: Basis\_vec\_def UNION\_singleton\_eq\_range sum.reindex axis\_eq\_axis inj\_on\_def})$

finally show $?thesis$.

qed

lemma *has_integral_stretch_cart*:

fixes $m :: 'n::\text{finite} \Rightarrow \text{real}$

assumes $f: (f \text{ has\_integral } i) \text{ (cbox } a \text{ } b)$ **and** $m: \bigwedge k. m \ k \neq 0$

shows $((\lambda x. f (\chi \ k. m \ k * x \$ k)) \text{ has\_integral } i /_{\mathbb{R}} |\text{prod } m \text{ UNIV}|)$

$((\lambda x. \chi \ k. x \$ k / m \ k) \text{ ' (cbox } a \text{ } b))$

```

proof –
  have *:  $\forall k:: \text{real}^n \in \text{Basis}. m (\text{axis\_index } k) \neq 0$ 
    using axis_index by (simp add: m)
  have eqp:  $(\prod k:: \text{real}^n \in \text{Basis}. m (\text{axis\_index } k)) = \text{prod } m \text{ UNIV}$ 
    by (simp add: Basis_vec_def UNION_singleton_eq_range prod.reindex axis_eq_axis
inj_on_def)
  show ?thesis
    using has_integral_stretch [OF f *] vec_lambda_eq_sum [where  $f = \lambda i x. m i$ 
 $* x$ ] vec_lambda_eq_sum [where  $f = \lambda i x. x / m i$ ]
    by (simp add: field_simps eqp)
qed

```

lemma *image_stretch_interval_cart*:

```

fixes  $m :: 'n::\text{finite} \Rightarrow \text{real}$ 
shows  $(\lambda x. \chi k. m k * x \$ k) ' \text{cbox } a b =$ 
   $(\text{if } \text{cbox } a b = \{\} \text{ then } \{\}$ 
     $\text{else } \text{cbox } (\chi k. \min (m k * a \$ k) (m k * b \$ k)) (\chi k. \max (m k * a \$ k) (m$ 
 $k * b \$ k)))$ 
proof –
  have *:  $(\sum k \in \text{Basis}. \min (m (\text{axis\_index } k) * (a \cdot k)) (m (\text{axis\_index } k) * (b \cdot$ 
 $k))) *_{\mathbb{R}} k)$ 
     $= (\chi k. \min (m k * a \$ k) (m k * b \$ k))$ 
     $(\sum k \in \text{Basis}. \max (m (\text{axis\_index } k) * (a \cdot k)) (m (\text{axis\_index } k) * (b \cdot$ 
 $k))) *_{\mathbb{R}} k)$ 
     $= (\chi k. \max (m k * a \$ k) (m k * b \$ k))$ 
  apply (simp_all add: Basis_vec_def cart_eq_inner_axis UNION_singleton_eq_range
sum.reindex axis_eq_axis inj_on_def)
  apply (simp_all add: vec_eq_iff axis_def if_distrib cong: if_cong)
  done
  show ?thesis
    by (simp add: vec_lambda_eq_sum [where  $f = \lambda i x. m i * x$ ] image_stretch_interval
eq_cbox *)
qed

```

6.15.28 even more special cases

lemma *uminus_interval_vector*[*simp*]:

```

fixes  $a b :: 'a::\text{euclidean\_space}$ 
shows uminus '  $\text{cbox } a b = \text{cbox } (-b) (-a)$ 

```

proof –

```

have  $x \in \text{uminus} ' \text{cbox } a b$  if  $x \in \text{cbox } (-b) (-a)$  for  $x$ 

```

proof –

```

have  $-x \in \text{cbox } a b$ 

```

```

  using that by (auto simp: mem_box)

```

```

then show ?thesis

```

```

  by force

```

qed

```

then show ?thesis

```

```

  by (auto simp: mem_box)

```

qed

```
lemma has_integral_reflect_lemma[intro]:
  assumes (f has_integral i) (cbox a b)
  shows (( $\lambda x. f(-x)$ ) has_integral i) (cbox (-b) (-a))
  using has_integral_affinity[OF assms, of -1 0]
  by auto
```

```
lemma has_integral_reflect_lemma_real[intro]:
  assumes (f has_integral i) {a..b::real}
  shows (( $\lambda x. f(-x)$ ) has_integral i) {-b .. -a}
  using assms
  unfolding box_real[symmetric]
  by (rule has_integral_reflect_lemma)
```

```
lemma has_integral_reflect[simp]:
  (( $\lambda x. f(-x)$ ) has_integral i) (cbox (-b) (-a))  $\longleftrightarrow$  (f has_integral i) (cbox a b)
  by (auto dest: has_integral_reflect_lemma)
```

```
lemma has_integral_reflect_real[simp]:
  fixes a b::real
  shows (( $\lambda x. f(-x)$ ) has_integral i) {-b..-a}  $\longleftrightarrow$  (f has_integral i) {a..b}
  by (metis has_integral_reflect interval_cbox)
```

```
lemma integrable_reflect[simp]: ( $\lambda x. f(-x)$ ) integrable_on cbox (-b) (-a)  $\longleftrightarrow$  f
  integrable_on cbox a b
  unfolding integrable_on_def by auto
```

```
lemma integrable_reflect_real[simp]: ( $\lambda x. f(-x)$ ) integrable_on {-b .. -a}  $\longleftrightarrow$ 
  f integrable_on {a..b::real}
  unfolding box_real[symmetric]
  by (rule integrable_reflect)
```

```
lemma integral_reflect[simp]: integral (cbox (-b) (-a)) ( $\lambda x. f(-x)$ ) = integral
  (cbox a b) f
  unfolding integral_def by auto
```

```
lemma integral_reflect_real[simp]: integral {-b .. -a} ( $\lambda x. f(-x)$ ) = integral
  {a..b::real} f
  unfolding box_real[symmetric]
  by (rule integral_reflect)
```

6.15.29 Stronger form of FCT; quite a tedious proof

```
lemma split_minus[simp]: ( $\lambda(x, k). f\ x\ k$ ) x - ( $\lambda(x, k). g\ x\ k$ ) x = ( $\lambda(x, k). f\ x\ k$ 
  - g x k) x
  by (simp add: split_def)
```

```
theorem fundamental_theorem_of_calculus_interior:
```



```

fixes  $f :: \text{real} \Rightarrow 'a::\text{real\_normed\_vector}$ 
assumes  $a \leq b$ 
  and  $\text{contf: continuous\_on } \{a..b\} f$ 
  and  $\text{derf: } \bigwedge x. x \in \{a <..< b\} \implies (f \text{ has\_vector\_derivative } f' x) \text{ (at } x)$ 
shows  $(f' \text{ has\_integral } (f b - f a)) \{a..b\}$ 
proof (cases  $a = b$ )
  case True
    then have  $∗: \text{cbox } a \ b = \{b\} \ f \ b - f \ a = 0$ 
    by (auto simp add: order\_antisym)
    with True show ?thesis by auto
  next
    case False
    with  $\langle a \leq b \rangle$  have  $ab: a < b$  by arith
    show ?thesis
      unfolding has\_integral\_factor\_content\_real
    proof (intro allI impI)
      fix  $e :: \text{real}$ 
      assume  $e: e > 0$ 
      then have  $eba8: (e * (b-a)) / 8 > 0$ 
      using  $ab$  by (auto simp add: field\_simps)
      note  $\text{derf\_exp} = \text{derf}[\text{unfolded has\_vector\_derivative\_def has\_derivative\_at\_alt,}$ 
 $\text{THEN conjunct2, rule\_format}]$ 
      have  $\text{bounded: } \bigwedge x. x \in \{a <..< b\} \implies \text{bounded\_linear } (\lambda u. u *_{\mathbb{R}} f' x)$ 
      by (simp add: bounded\_linear\_scaleR\_left)
      have  $\forall x \in \text{box } a \ b. \exists d > 0. \forall y. \text{norm } (y-x) < d \longrightarrow \text{norm } (f y - f x - (y-x) *_{\mathbb{R}} f' x) \leq e/2 * \text{norm } (y-x)$ 
      (is  $\forall x \in \text{box } a \ b. ?Q \ x$ ) — The explicit quantifier is required by the following
    step
      proof
        fix  $x$  assume  $x \in \text{box } a \ b$ 
        with  $e$  show ? $Q \ x$ 
        using  $\text{derf\_exp } [of \ x \ e/2]$  by auto
      qed
      then obtain  $d$  where  $d: \bigwedge x. 0 < d \ x$ 
       $\bigwedge x \ y. \llbracket x \in \text{box } a \ b; \text{norm } (y-x) < d \ x \rrbracket \implies \text{norm } (f y - f x - (y-x) *_{\mathbb{R}} f' x) \leq e/2 * \text{norm } (y-x)$ 
      unfolding bgauge\_existence\_lemma by metis
      have  $\text{bounded } (f' \text{ cbox } a \ b)$ 
      using compact\_cbox \text{assms} by (auto simp: compact\_imp\_bounded compact\_continuous\_image)
      then obtain  $B$ 
      where  $0 < B$  and  $B: \bigwedge x. x \in f' \text{ cbox } a \ b \implies \text{norm } x \leq B$ 
      unfolding bounded\_pos by metis
      obtain  $da$  where  $0 < da$ 
      and  $da: \bigwedge c. \llbracket a \leq c; \{a..c\} \subseteq \{a..b\}; \{a..c\} \subseteq \text{ball } a \ da \rrbracket$ 
       $\implies \text{norm } (\text{content } \{a..c\} *_{\mathbb{R}} f' a - (f c - f a)) \leq (e * (b-a))$ 
    / 4
    proof —
      have continuous (at  $a$  within  $\{a..b\}$ )  $f$ 

```

```

    using contf continuous_on_eq_continuous_within by force
  with eba8 obtain k where 0 < k
    and k:  $\bigwedge x. \llbracket x \in \{a..b\}; 0 < \text{norm } (x-a); \text{norm } (x-a) < k \rrbracket \implies \text{norm } (f x - f a) < e * (b-a) / 8$ 
    unfolding continuous_within Lim_within dist_norm by metis
  obtain l where l: 0 < l  $\text{norm } (l *_R f' a) \leq e * (b-a) / 8$ 
  proof (cases f' a = 0)
    case True with ab e that show ?thesis by auto
  next
    case False
    show ?thesis
    proof
      show  $\text{norm } ((e * (b-a) / 8 / \text{norm } (f' a)) *_R f' a) \leq e * (b-a) / 8$ 
        0 < e * (b-a) / 8 / norm (f' a)
      using False ab e by (auto simp add: field_simps)
    qed
  qed
  have  $\text{norm } (\text{content } \{a..c\} *_R f' a - (f c - f a)) \leq e * (b-a) / 4$ 
    if a ≤ c  $\{a..c\} \subseteq \{a..b\}$  and bmin:  $\{a..c\} \subseteq \text{ball } a (\min k l)$  for c
  proof -
    have minkl:  $|a - x| < \min k l$  if  $x \in \{a..c\}$  for x
      using bmin dist_real_def that by auto
    then have lel:  $|c - a| \leq |l|$ 
      using that by force
    have  $\text{norm } ((c - a) *_R f' a - (f c - f a)) \leq \text{norm } ((c - a) *_R f' a) +$ 
      norm (f c - f a)
      by (rule norm_triangle_ineq4)
    also have ... ≤ e * (b-a) / 8 + e * (b-a) / 8
    proof (rule add_mono)
      have  $\text{norm } ((c - a) *_R f' a) \leq \text{norm } (l *_R f' a)$ 
        by (auto intro: mult_right_mono [OF lel])
      also have ... ≤ e * (b-a) / 8
        by (rule l)
      finally show  $\text{norm } ((c - a) *_R f' a) \leq e * (b-a) / 8$  .
    next
      have  $\text{norm } (f c - f a) < e * (b-a) / 8$ 
      proof (cases a = c)
        case True then show ?thesis
          using eba8 by auto
      next
        case False show ?thesis
          by (rule k) (use minkl  $\langle a \leq c \rangle$  that False in auto)
      qed
      then show  $\text{norm } (f c - f a) \leq e * (b-a) / 8$  by simp
    qed
    finally show  $\text{norm } (\text{content } \{a..c\} *_R f' a - (f c - f a)) \leq e * (b-a) / 4$ 
      unfolding content_real[OF  $\langle a \leq c \rangle$ ] by auto
  qed
  then show ?thesis

```

```

    by (rule_tac da=min k l in that) (auto simp: l < 0 < k)
  qed
  obtain db where 0 < db
    and db:  $\bigwedge c. \llbracket c \leq b; \{c..b\} \subseteq \{a..b\}; \{c..b\} \subseteq \text{ball } b \text{ db} \rrbracket$ 
       $\implies \text{norm } (\text{content } \{c..b\} *_R f' b - (f b - f c)) \leq (e * (b-a))$ 
  / 4
  proof -
    have continuous (at b within {a..b}) f
      using contf continuous_on_eq_continuous_within by force
    with eba8 obtain k
      where 0 < k
      and k:  $\bigwedge x. \llbracket x \in \{a..b\}; 0 < \text{norm}(x-b); \text{norm}(x-b) < k \rrbracket$ 
         $\implies \text{norm } (f b - f x) < e * (b-a) / 8$ 
    unfolding continuous_within Lim_within dist_norm norm_minus_commute
  by metis
    obtain l where l: 0 < l norm (l *_R f' b)  $\leq (e * (b-a)) / 8$ 
    proof (cases f' b = 0)
      case True thus ?thesis
        using ab e that by auto
    next
      case False show ?thesis
        proof
          show norm ((e * (b - a) / 8 / norm (f' b)) *_R f' b)  $\leq e * (b - a) / 8$ 
            0 < e * (b - a) / 8 / norm (f' b)
          using False ab e by (auto simp add: field_simps)
        qed
      qed
    have norm (content {c..b} *_R f' b - (f b - f c))  $\leq e * (b-a) / 4$ 
      if c  $\leq b$   $\{c..b\} \subseteq \{a..b\}$  and bmin:  $\{c..b\} \subseteq \text{ball } b (\min k l)$  for c
    proof -
      have minkl:  $|b - x| < \min k l$  if  $x \in \{c..b\}$  for x
        using bmin dist_real_def that by auto
      then have lel:  $|b - c| \leq |l|$ 
        using that by force
      have norm ((b - c) *_R f' b - (f b - f c))  $\leq \text{norm } ((b - c) *_R f' b) +$ 
norm (f b - f c)
        by (rule norm_triangle_ineq4)
      also have ...  $\leq e * (b-a) / 8 + e * (b-a) / 8$ 
        proof (rule add_mono)
          have norm ((b - c) *_R f' b)  $\leq \text{norm } (l *_R f' b)$ 
            by (auto intro: mult_right_mono [OF lel])
          also have ...  $\leq e * (b-a) / 8$ 
            by (rule l)
          finally show norm ((b - c) *_R f' b)  $\leq e * (b-a) / 8$  .
        qed
    next
      have norm (f b - f c)  $< e * (b-a) / 8$ 
      proof (cases b = c)
        case True with eba8 show ?thesis
          by auto

```

```

next
  case False show ?thesis
  by (rule k) (use minkl  $\langle c \leq b \rangle$  that False in auto)
qed
then show norm (f b - f c)  $\leq e * (b - a) / 8$  by simp
qed
finally show norm (content {c..b} *R f' b - (f b - f c))  $\leq e * (b - a) / 4$ 
  unfolding content_real[OF  $\langle c \leq b \rangle$ ] by auto
qed
then show ?thesis
  by (rule_tac db=min k l in that) (auto simp: l  $\langle 0 < k \rangle$ )
qed
let ?d = ( $\lambda x$ . ball x (if x=a then da else if x=b then db else d x))
show  $\exists d$ . gauge d  $\wedge (\forall p$ . p tagged_division_of {a..b}  $\wedge$  d fine p  $\longrightarrow$ 
  norm (( $\sum (x,K) \in p$ . content K *R f' x) - (f b - f a))  $\leq e * \text{content}$ 
{a..b})
proof (rule exI, safe)
  show gauge ?d
    using ab  $\langle db > 0 \rangle$   $\langle da > 0 \rangle$  d(1) by (auto intro: gauge_ball_dependent)
next
  fix p
  assume ptag: p tagged_division_of {a..b} and fine: ?d fine p
  let ?A = {t. fst t  $\in$  {a, b}}
  note p = tagged_division_ofD[OF ptag]
  have pA: p = (p  $\cap$  ?A)  $\cup$  (p - ?A) finite (p  $\cap$  ?A) finite (p - ?A) (p  $\cap$  ?A)
 $\cap$  (p - ?A) = {}
    using ptag fine by auto
  have le_xz:  $\bigwedge w x y z :: \text{real}$ . y  $\leq z/2 \implies w - x \leq z/2 \implies w + y \leq x + z$ 
    by arith
  have non: False if xk: (x,K)  $\in$  p and x  $\neq a$  x  $\neq b$ 
    and less:  $e * (\text{Sup } K - \text{Inf } K)/2 < \text{norm} (\text{content } K *_{\text{R}} f' x - (f (\text{Sup } K) - f (\text{Inf } K)))$ 
    for x K
  proof -
    obtain u v where k: K = cbox u v
      using p(4) xk by blast
    then have u  $\leq v$  and uv: {u, v}  $\subseteq$  cbox u v
      using p(2)[OF xk] by auto
    then have result:  $e * (v - u)/2 < \text{norm} ((v - u) *_{\text{R}} f' x - (f v - f u))$ 
      using less[unfolded k box_real_interval_bounds_real content_real] by auto
    then have x  $\in$  box a b
      using p(2) p(3)  $\langle x \neq a \rangle \langle x \neq b \rangle$  xk by fastforce
    with d have *:  $\bigwedge y$ . norm (y-x)  $< d$  x
       $\implies \text{norm} (f y - f x - (y-x) *_{\text{R}} f' x) \leq e/2 * \text{norm} (y-x)$ 
    by metis
    have xd: norm (u - x)  $< d$  x norm (v - x)  $< d$  x
      using fineD[OF fine xk]  $\langle x \neq a \rangle \langle x \neq b \rangle$  uv
      by (auto simp add: k subset_eq dist_commute dist_real_def)
    have norm ((v - u) *R f' x - (f v - f u)) =

```

```

      norm ((f u - f x - (u - x) *R f' x) - (f v - f x - (v - x) *R f' x))
    by (rule arg_cong[where f=norm]) (auto simp: scaleR_left.diff)
  also have ... ≤ e/2 * norm (u - x) + e/2 * norm (v - x)
    by (metis norm_triangle_le_diff add_mono * xd)
  also have ... ≤ e/2 * norm (v - u)
    using p(2)[OF xk] by (auto simp add: field_simps k)
  also have ... < norm ((v - u) *R f' x - (f v - f u))
    using result by (simp add: ‹u ≤ v›)
  finally have e * (v - u)/2 < e * (v - u)/2
    using uv by auto
  then show False by auto
qed
have norm (∑ (x, K) ∈ p - ?A. content K *R f' x - (f (Sup K) - f (Inf
K)))
  ≤ (∑ (x, K) ∈ p - ?A. norm (content K *R f' x - (f (Sup K) - f (Inf
K))))
  by (auto intro: sum_norm_le)
  also have ... ≤ (∑ n ∈ p - ?A. e * (case n of (x, k) ⇒ Sup k - Inf k)/2)
    using non by (fastforce intro: sum_mono)
  finally have I: norm (∑ (x, k) ∈ p - ?A.
    content k *R f' x - (f (Sup k) - f (Inf k)))
    ≤ (∑ n ∈ p - ?A. e * (case n of (x, k) ⇒ Sup k - Inf k)/2)
    by (simp add: sum_divide_distrib)
  have II: norm (∑ (x, k) ∈ p ∩ ?A. content k *R f' x - (f (Sup k) - f (Inf
k))) -
    (∑ n ∈ p ∩ ?A. e * (case n of (x, k) ⇒ Sup k - Inf k))
    ≤ (∑ n ∈ p - ?A. e * (case n of (x, k) ⇒ Sup k - Inf k)/2)
  proof -
    have ge0: 0 ≤ e * (Sup k - Inf k) if xkp: (x, k) ∈ p ∩ ?A for x k
    proof -
      obtain u v where uv: k = cbox u v
      by (meson Int_iff xkp p(4))
      with p(2) that uv have cbox u v ≠ {}
      by blast
      then show 0 ≤ e * ((Sup k) - (Inf k))
        unfolding uv using e by (auto simp add: field_simps)
    qed
  let ?B = λx. {t ∈ p. fst t = x ∧ content (snd t) ≠ 0}
  let ?C = {t ∈ p. fst t ∈ {a, b} ∧ content (snd t) ≠ 0}
  have norm (∑ (x, k) ∈ p ∩ {t. fst t ∈ {a, b}}. content k *R f' x - (f (Sup
k) - f (Inf k))) ≤ e * (b-a)/2
  proof -
    have *: ⋀ S f e. sum f S = sum f (p ∩ ?C) ⇒ norm (sum f (p ∩ ?C))
      ≤ e ⇒ norm (sum f S) ≤ e
    by auto
    have 1: content K *R (f' x) - (f ((Sup K)) - f ((Inf K))) = 0
      if (x, K) ∈ p ∩ {t. fst t ∈ {a, b}} - p ∩ ?C for x K
    proof -
      have xk: (x, K) ∈ p and k0: content K = 0

```

```

    using that by auto
  then obtain u v where uv: K = cbox u v
    using p(4) by blast
  then have u = v
    using xk k0 p(2) by force
  then show content K *_R (f' x) - (f ((Sup K)) - f ((Inf K))) = 0
    using xk unfolding uv by auto
qed
have 2: norm( $\sum (x,K) \in p \cap ?C. \text{content } K *_R f' x - (f (\text{Sup } K) - f (\text{Inf } K))$ )  $\leq e * (b-a)/2$ 
proof -
  have norm_le: norm (sum f S)  $\leq e$ 
  if  $\bigwedge x y. \llbracket x \in S; y \in S \rrbracket \implies x = y \wedge x. x \in S \implies \text{norm } (f x) \leq e \ e > 0$ 
  for S f and e :: real
proof (cases S = {})
  case True
  with that show ?thesis by auto
next
  case False then obtain x where x  $\in S$ 
  by auto
  then have S = {x}
  using that(1) by auto
  then show ?thesis
  using  $\langle x \in S \rangle$  that(2) by auto
qed
have *:  $p \cap ?C = ?B a \cup ?B b$ 
  by blast
then have norm ( $\sum (x,K) \in p \cap ?C. \text{content } K *_R f' x - (f (\text{Sup } K) - f (\text{Inf } K))$ ) =
  norm ( $\sum (x,K) \in ?B a \cup ?B b. \text{content } K *_R f' x - (f (\text{Sup } K) - f (\text{Inf } K))$ )
  by simp
also have ... = norm (( $\sum (x,K) \in ?B a. \text{content } K *_R f' x - (f (\text{Sup } K) - f (\text{Inf } K))$ ) +
  ( $\sum (x,K) \in ?B b. \text{content } K *_R f' x - (f (\text{Sup } K) - f (\text{Inf } K))$ ))
  using p(1) ab e by (subst sum.union_disjoint) auto
also have ...  $\leq e * (b - a) / 4 + e * (b - a) / 4$ 
proof (rule norm_triangle_le [OF add_mono])
  have pa:  $\exists v. k = \text{cbox } a \ v \wedge a \leq v$  if  $(a, k) \in p$  for k
  using p(2) p(3) p(4) that by fastforce
  show norm ( $\sum (x,K) \in ?B a. \text{content } K *_R f' x - (f (\text{Sup } K) - f (\text{Inf } K))$ )  $\leq e * (b - a) / 4$ 
  proof (intro norm_le; clarsimp)
    fix K K'
    assume K:  $(a, K) \in p \ (a, K') \in p$  and ne0: content K  $\neq 0$  content K'  $\neq 0$ 
    with pa obtain v v' where v: K = cbox a v  $a \leq v$  and v': K' = cbox a v'  $a \leq v'$ 

```

```

      by blast
    let ?v = min v v'
    have box a ?v  $\subseteq$  K  $\cap$  K'
      unfolding v v' by (auto simp add: mem_box)
    then have interior (box a (min v v'))  $\subseteq$  interior K  $\cap$  interior K'
      using interior_Int interior_mono by blast
    moreover have (a + ?v)/2  $\in$  box a ?v
      using ne0 unfolding v v' content_eq_0 not_le
      by (auto simp add: mem_box)
    ultimately have (a + ?v)/2  $\in$  interior K  $\cap$  interior K'
      unfolding interior_open[OF open_box] by auto
    then show K = K'
      using p(5)[OF K] by auto
  next
  fix K
  assume K: (a, K)  $\in$  p and ne0: content K  $\neq$  0
  show norm (content c *R f' a - (f (Sup c) - f (Inf c))) * 4  $\leq$  e *
(b-a)
    if (a, c)  $\in$  p and ne0: content c  $\neq$  0 for c
  proof -
    obtain v where v: c = cbox a v and a  $\leq$  v
      using pa[OF  $\langle$ (a, c)  $\in$  p $\rangle$ ] by metis
    then have a  $\in$  {a.. $v$ } v  $\leq$  b
      using p(3)[OF  $\langle$ (a, c)  $\in$  p $\rangle$ ] by auto
    moreover have {a.. $v$ }  $\subseteq$  ball a da
      using fineD[OF  $\langle$ ?d fine p $\rangle$   $\langle$ (a, c)  $\in$  p $\rangle$ ] by (simp add: v split:
if_split_asm)
    ultimately show ?thesis
      unfolding v interval_bounds_real[OF  $\langle$ a  $\leq$  v $\rangle$ ] box_real
      using da  $\langle$ a  $\leq$  v $\rangle$  by auto
  qed
qed (use ab e in auto)
next
have pb:  $\exists v. k = cbox v b \wedge b \geq v$  if (b, k)  $\in$  p for k
  using p(2) p(3) p(4) that by fastforce
  show norm ( $\sum (x, K) \in ?B b. \text{content } K *_{\mathbb{R}} f' x - (f (\text{Sup } K) - f$ 
(Inf K)))  $\leq$  e * (b - a) / 4
  proof (intro norm_le; clarsimp)
    fix K K'
    assume K: (b, K)  $\in$  p (b, K')  $\in$  p and ne0: content K  $\neq$  0 content
K'  $\neq$  0
    with pb obtain v v' where v: K = cbox v b v  $\leq$  b and v': K' =
cbox v' b v'  $\leq$  b
    by blast
    let ?v = max v v'
    have box ?v b  $\subseteq$  K  $\cap$  K'
      unfolding v v' by (auto simp: mem_box)
    then have interior (box (max v v') b)  $\subseteq$  interior K  $\cap$  interior K'
      using interior_Int interior_mono by blast

```

```

    moreover have  $((b + ?v)/2) \in \text{box } ?v \ b$ 
      using ne0 unfolding v v' content_eq_0 not_le by (auto simp:
mem_box)
    ultimately have  $((b + ?v)/2) \in \text{interior } K \cap \text{interior } K'$ 
      unfolding interior_open[OF open_box] by auto
    then show  $K = K'$ 
      using p(5)[OF K] by auto
  next
  fix  $K$ 
  assume  $K: (b, K) \in p$  and ne0: content K  $\neq 0$ 
  show norm (content c *R f' b - (f (Sup c) - f (Inf c))) * 4  $\leq e$  *
    (b-a)
    if  $(b, c) \in p$  and ne0: content c  $\neq 0$  for  $c$ 
  proof -
  obtain  $v$  where  $v: c = \text{cbox } v \ b$  and  $v \leq b$ 
    using  $\langle (b, c) \in p \rangle$  pb by blast
  then have  $v \geq ab \in \{v..b\}$ 
    using p(3)[OF  $\langle (b, c) \in p \rangle$ ] by auto
  moreover have  $\{v..b\} \subseteq \text{ball } b \ db$ 
    using fineD[OF  $\langle ?d \text{ fine } p \rangle \langle (b, c) \in p \rangle$ ] box_real(2) v False by
force
  ultimately show ?thesis
    using db v by auto
  qed
  qed (use ab e in auto)
  qed
  also have  $\dots = e * (b-a)/2$ 
    by simp
  finally show norm ( $\sum_{(x,k) \in p \cap ?C} \text{content } k *_{\text{R}} f' x - (f (\text{Sup } k) - f (\text{Inf } k))) \leq e * (b-a)/2$ .
    qed
  show norm ( $\sum_{(x,k) \in p \cap ?A} \text{content } k *_{\text{R}} f' x - (f ((\text{Sup } k)) - f ((\text{Inf } k)))) \leq e * (b-a)/2$ 
    apply (rule * [OF sum.mono_neutral_right[OF pA(2)]])
    using 1 2 by (auto simp: split_paired_all)
  qed
  also have  $\dots = (\sum_{n \in p} e * (\text{case } n \text{ of } (x, k) \Rightarrow \text{Sup } k - \text{Inf } k))/2$ 
    unfolding sum_distrib_left[symmetric]
    by (subst additive_tagged_division_1[OF  $\langle a \leq b \rangle$  ptag]) auto
  finally have norm_le: norm ( $\sum_{(x,K) \in p \cap \{t. \text{fst } t \in \{a, b\}\}. \text{content } K$ 
    *R f' x - (f (Sup K) - f (Inf K)))
     $\leq (\sum_{n \in p} e * (\text{case } n \text{ of } (x, K) \Rightarrow \text{Sup } K - \text{Inf } K))/2$  .
  have le2:  $\bigwedge x \ s1 \ s2::\text{real}. 0 \leq s1 \implies x \leq (s1 + s2)/2 \implies x - s1 \leq s2/2$ 
    by auto
  show ?thesis
    apply (rule le2 [OF sum_nonneg])
    using ge0 apply force
    by (metis (no_types, lifting) Diff_Diff_Int Diff_subset norm_le p(1)
sum.subset_diff)

```



```

qed
note * = additive_tagged_division_1[OF assms(1) ptag, symmetric]
have norm ( $\sum (x,K) \in p \cap ?A \cup (p - ?A). \text{content } K *_R f' x - (f (\text{Sup } K) - f (\text{Inf } K))$ )
   $\leq e * (\sum (x,K) \in p \cap ?A \cup (p - ?A). \text{Sup } K - \text{Inf } K)$ 
  unfolding sum_distrib_left
  unfolding sum.union_disjoint[OF pA(2-)]
  using le_xz norm_triangle_le I II by blast
then
show norm ( $(\sum (x,K) \in p. \text{content } K *_R f' x) - (f b - f a)$ )  $\leq e * \text{content } \{a..b\}$ 
  by (simp only: content_real[OF ‹a ≤ b›] * [of ‹λx. x›] * [of f] sum_subtractf[symmetric]
    split_minus pA(1) [symmetric])
qed
qed
qed

```

6.15.30 Stronger form with finite number of exceptional points

```

lemma fundamental_theorem_of_calculus_interior_strong:
  fixes f :: real ⇒ 'a::banach
  assumes finite S
  and a ≤ b ∧ x. x ∈ {a <..b} - S ⇒ (f has_vector_derivative f'(x)) (at x)
  and continuous_on {a .. b} f
  shows (f' has_integral (f b - f a)) {a .. b}
  using assms
  proof (induction arbitrary: a b)
  case empty
  then show ?case
    using fundamental_theorem_of_calculus_interior by force
  next
  case (insert x S)
  show ?case
  proof (cases x ∈ {a <..b})
  case False then show ?thesis
    using insert by blast
  next
  case True then have a < x < b
    by auto
  have (f' has_integral f x - f a) {a..x} (f' has_integral f b - f x) {x..b}
    using ‹continuous_on {a..b} f› ‹a < x› ‹x < b› continuous_on_subset by
    (force simp: intro!: insert)+
  then have (f' has_integral f x - f a + (f b - f x)) {a..b}
    using ‹a < x› ‹x < b› has_integral_combine less_imp_le by blast
  then show ?thesis
    by simp
  qed
qed

```

corollary *fundamental_theorem_of_calculus_strong*:

fixes $f :: \text{real} \Rightarrow 'a::\text{banach}$
assumes *finite S*
and $a \leq b$
and *vec*: $\bigwedge x. x \in \{a..b\} - S \implies (f \text{ has\_vector\_derivative } f'(x)) \text{ (at } x)$
and *continuous_on* $\{a..b\}$ *f*
shows $(f' \text{ has\_integral } (f \text{ b} - f \text{ a})) \{a..b\}$
by (*rule fundamental_theorem_of_calculus_interior_strong* [*OF* $\langle \text{finite } S \rangle$]) (*force simp: assms*) $+$

proposition *indefinite_integral_continuous_left*:

fixes $f :: \text{real} \Rightarrow 'a::\text{banach}$
assumes *intf*: $f \text{ integrable\_on } \{a..b\}$ **and** $a < c \leq b$ $e > 0$
obtains d **where** $d > 0$
and $\forall t. c - d < t \wedge t \leq c \longrightarrow \text{norm } (\text{integral } \{a..c\} f - \text{integral } \{a..t\} f) < e$
proof –
obtain w **where** $w > 0$ **and** w : $\bigwedge t. \llbracket c - w < t; t < c \rrbracket \implies \text{norm } (f \text{ c}) * \text{norm } (c - t) < e/3$
proof (*cases* $f \text{ c} = 0$)
case *False*
hence $e/3$: $0 < e/3 / \text{norm } (f \text{ c})$ **using** $\langle e > 0 \rangle$ **by** *simp*
moreover have $\text{norm } (f \text{ c}) * \text{norm } (c - t) < e/3$
if $t < c$ **and** $c - e/3 / \text{norm } (f \text{ c}) < t$ **for** t
proof –
have $\text{norm } (c - t) < e/3 / \text{norm } (f \text{ c})$
using *that* **by** *auto*
then show $\text{norm } (f \text{ c}) * \text{norm } (c - t) < e/3$
by (*metis* $e/3 \text{ mult.commute norm\_not\_less\_zero pos\_less\_divide\_eq zero\_less\_divide\_iff}$)
qed
ultimately show *?thesis*
using *that* **by** *auto*
next
case *True* **then show** *?thesis*
using $\langle e > 0 \rangle$ **that** **by** *auto*
qed

let $?SUM = \lambda p. (\sum (x, K) \in p. \text{content } K *_{\text{R}} f \text{ x})$
have $e/3$: $e/3 > 0$
using $\langle e > 0 \rangle$ **by** *auto*
have $f \text{ integrable\_on } \{a..c\}$
using $\langle a < c \rangle \langle c \leq b \rangle$ **by** (*auto intro: integrable_subinterval_real*[*OF intf*])
then obtain $d1$ **where** *gauge* $d1$ **and** $d1$:
 $\bigwedge p. \llbracket p \text{ tagged\_division\_of } \{a..c\}; d1 \text{ fine } p \rrbracket \implies \text{norm } (?SUM \text{ p} - \text{integral } \{a..c\} f) < e/3$
using *integrable_integral has_integral_real* $e/3$ **by** *metis*
define d **where** [*abs_def*]: $d \text{ x} = \text{ball } x \text{ w} \cap d1 \text{ x}$ **for** x
have *gauge* d

```

    unfolding d_def using ‹w > 0› ‹gauge d1› by auto
  then obtain k where 0 < k and k: ball c k ⊆ d c
    by (meson gauge_def open_contains_ball)

  let ?d = min k (c - a)/2
  show thesis
  proof (intro that[of ?d] allI impI, safe)
    show ?d > 0
      using ‹0 < k› ‹a < c› by auto
  next
    fix t
    assume t: c - ?d < t ≤ c
    show norm (integral ({a..c}) f - integral ({a..t}) f) < e
    proof (cases t < c)
      case False with ‹t ≤ c› show ?thesis
        by (simp add: ‹e > 0›)
    next
      case True
      have f integrable_on {a..t}
        using ‹t < c› ‹c ≤ b› by (auto intro: integrable_subinterval_real[OF intf])
      then obtain d2 where d2: gauge d2
        ∧ p. p tagged_division_of {a..t} ∧ d2 fine p ⇒ norm (?SUM p - integral
        {a..t} f) < e/3
      using integrable_integral has_integral_real e3 by metis
      define d3 where d3 x = (if x ≤ t then d1 x ∩ d2 x else d1 x) for x
      have gauge d3
        using ‹gauge d1› ‹gauge d2› unfolding d3_def gauge_def by auto
      then obtain p where ptag: p tagged_division_of {a..t} and pfine: d3 fine p
        by (metis box_real(2) fine_division_exists)
      note p' = tagged_division_ofD[OF ptag]
      have pt: (x,K) ∈ p ⇒ x ≤ t for x K
        by (meson atLeastAtMost_iff p'(2) p'(3) subsetCE)
      with pfine have d2 fine p
        unfolding fine_def d3_def by fastforce
      then have d2_fin: norm (?SUM p - integral {a..t} f) < e/3
        using d2(2) ptag by auto
      have eqs: {a..c} ∩ {x. x ≤ t} = {a..t} {a..c} ∩ {x. x ≥ t} = {t..c}
        using t by (auto simp add: field_simps)
      have p ∪ {(c, {t..c})} tagged_division_of {a..c}
      proof (intro tagged_division_Un_interval_real)
        show {(c, {t..c})} tagged_division_of {a..c} ∩ {x. t ≤ x ≤ 1}
          using ‹t ≤ c› by (auto simp: eqs tagged_division_of_self_real)
      qed (auto simp: eqs ptag)
    moreover
      have d1 fine p ∪ {(c, {t..c})}
        unfolding fine_def
      proof safe
        fix x K y
        assume (x,K) ∈ p and y ∈ K then show y ∈ d1 x

```

```

    by (metis Int_iff d3_def subsetD fineD pfine)
  next
    fix x assume x ∈ {t..c}
    then have dist c x < k
      using t(1) by (auto simp add: field_simps dist_real_def)
    with k show x ∈ d1 c
      unfolding d_def by auto
  qed
  ultimately have d1_fin: norm (?SUM(p ∪ {(c, {t..c})}) - integral {a..c}
f) < e/3
    using d1 by metis
  have SUMEQ: ?SUM(p ∪ {(c, {t..c})}) = (c - t) *R f c + ?SUM p
  proof -
    have ?SUM(p ∪ {(c, {t..c})}) = (content{t..c} *R f c) + ?SUM p
    proof (subst sum.union_disjoint)
      show p ∩ {(c, {t..c})} = {}
        using ⟨t < c⟩ pt by force
    qed (use p'(1) in auto)
    also have ... = (c - t) *R f c + ?SUM p
      using ⟨t ≤ c⟩ by auto
    finally show ?thesis .
  qed
  have c - k < t
    using ⟨k > 0⟩ t(1) by (auto simp add: field_simps)
  moreover have k ≤ w
  proof (rule ccontr)
    assume ¬ k ≤ w
    then have c + (k + w) / 2 ∉ d c
      by (auto simp add: field_simps not_le not_less dist_real_def d_def)
    then have c + (k + w) / 2 ∉ ball c k
      using k by blast
    then show False
      using ⟨0 < w⟩ ⟨¬ k ≤ w⟩ dist_real_def by auto
  qed
  ultimately have cwt: c - w < t
    by (auto simp add: field_simps)
  have eq: integral {a..c} f - integral {a..t} f = -(((c - t) *R f c + ?SUM
p) -
    integral {a..c} f) + (?SUM p - integral {a..t} f) + (c - t) *R f c
    by auto
  have norm (integral {a..c} f - integral {a..t} f) < e/3 + e/3 + e/3
    unfolding eq
  proof (intro norm_triangle_lt add_strict_mono)
    show norm (-((c - t) *R f c + ?SUM p - integral {a..c} f)) < e/3
      by (metis SUMEQ d1_fin norm_minus_cancel)
    show norm (?SUM p - integral {a..t} f) < e/3
      using d2_fin by blast
    show norm ((c - t) *R f c) < e/3
      using w cwt ⟨t < c⟩ by simp (simp add: field_simps)
  qed

```

```

      qed
      then show ?thesis by simp
    qed
  qed
qed

lemma indefinite_integral_continuous_right:
  fixes f :: real  $\Rightarrow$  'a::banach
  assumes f integrable_on {a..b}
    and a  $\leq$  c
    and c < b
    and e > 0
  obtains d where 0 < d
    and  $\forall t. c \leq t \wedge t < c + d \longrightarrow \text{norm } (\text{integral } \{a..c\} f - \text{integral } \{a..t\} f) < e$ 
proof -
  have intm:  $(\lambda x. f (-x))$  integrable_on  $\{-b .. -a\} - b < -c - c \leq -a$ 
    using assms by auto
  from indefinite_integral_continuous_left[OF intm  $\langle e > 0 \rangle$ ]
  obtain d where 0 < d
    and d:  $\bigwedge t. \llbracket -c - d < t; t \leq -c \rrbracket \implies \text{norm } (\text{integral } \{-b..-c\} (\lambda x. f (-x)) - \text{integral } \{-b..t\} (\lambda x. f (-x))) < e$ 
    by metis
  let ?d = min d (b - c)
  show ?thesis
  proof (intro that[of ?d] allI impI)
    show 0 < ?d
      using  $\langle 0 < d \rangle \langle c < b \rangle$  by auto
    fix t :: real
    assume t:  $c \leq t \wedge t < c + ?d$ 
    have *:  $\text{integral } \{a..c\} f = \text{integral } \{a..b\} f - \text{integral } \{c..b\} f$ 
       $\text{integral } \{a..t\} f = \text{integral } \{a..b\} f - \text{integral } \{t..b\} f$ 
      using assms t by (auto simp: algebra_simps integral_combine)
    have  $(-c) - d < (-t) - t \leq -c$ 
      using t by auto
    from d[OF this] show  $\text{norm } (\text{integral } \{a..c\} f - \text{integral } \{a..t\} f) < e$ 
      by (auto simp add: algebra_simps norm_minus_commute *)
  qed
qed

lemma indefinite_integral_continuous_1:
  fixes f :: real  $\Rightarrow$  'a::banach
  assumes f integrable_on {a..b}
  shows continuous_on {a..b}  $(\lambda x. \text{integral } \{a..x\} f)$ 
proof -
  have  $\exists d > 0. \forall x' \in \{a..b\}. \text{dist } x' x < d \longrightarrow \text{dist } (\text{integral } \{a..x'\} f) (\text{integral } \{a..x\} f) < e$ 
    if  $x: x \in \{a..b\}$  and  $e > 0$  for  $x e :: real$ 

```

```

proof (cases a = b)
  case True
    with that show ?thesis by force
  next
    case False
      with x have a < b by force
      with x consider x = a | x = b | a < x x < b
        by force
      then show ?thesis
      proof cases
        case 1 then show ?thesis
          by (force simp: dist_norm algebra_simps intro: indefinite_integral_continuous_right
[OF assms _ ⟨a < b⟩ ⟨e > 0⟩])
        next
          case 2 then show ?thesis
            by (force simp: dist_norm norm_minus_commute algebra_simps intro:
indefinite_integral_continuous_left [OF assms ⟨a < b⟩ _ ⟨e > 0⟩])
          next
            case 3
              obtain d1 where 0 < d1
                and d1:  $\bigwedge t. \llbracket x - d1 < t; t \leq x \rrbracket \implies \text{norm } (\text{integral } \{a..x\} f - \text{integral } \{a..t\} f) < e$ 
                using 3 by (auto intro: indefinite_integral_continuous_left [OF assms ⟨a < x⟩ _ ⟨e > 0⟩])
              obtain d2 where 0 < d2
                and d2:  $\bigwedge t. \llbracket x \leq t; t < x + d2 \rrbracket \implies \text{norm } (\text{integral } \{a..x\} f - \text{integral } \{a..t\} f) < e$ 
                using 3 by (auto intro: indefinite_integral_continuous_right [OF assms _ ⟨x < b⟩ ⟨e > 0⟩])
              show ?thesis
              proof (intro exI ballI conjI impI)
                show 0 < min d1 d2
                  using ⟨0 < d1⟩ ⟨0 < d2⟩ by simp
                show dist (integral {a..y} f) (integral {a..x} f) < e
                  if y ∈ {a..b} dist y x < min d1 d2 for y
                proof (cases y < x)
                  case True
                    with that d1 show ?thesis by (auto simp: dist_commute dist_norm)
                  next
                    case False
                      with that d2 show ?thesis
                        by (auto simp: dist_commute dist_norm)
                qed
              qed
            qed
          then show ?thesis
            by (auto simp: continuous_on_iff)
        qed

```

```

lemma indefinite_integral_continuous_1':
  fixes f::real  $\Rightarrow$  'a::banach
  assumes f integrable_on {a..b}
  shows continuous_on {a..b} ( $\lambda x$ . integral {x..b} f)
proof -
  have integral {a..b} f - integral {a..x} f = integral {x..b} f if  $x \in \{a..b\}$  for x
    using integral_combine[OF _ _ assms, of x] that
    by (auto simp: algebra_simps)
  with _ show ?thesis
    by (rule continuous_on_eq) (auto intro!: continuous_intros indefinite_integral_continuous_1
    assms)
qed

```

```

theorem integral_has_vector_derivative':
  fixes f :: real  $\Rightarrow$  'b::banach
  assumes continuous_on {a..b} f
    and  $x \in \{a..b\}$ 
  shows (( $\lambda u$ . integral {u..b} f) has_vector_derivative - f x) (at x within {a..b})
proof -
  have *: integral {x..b} f = integral {a .. b} f - integral {a .. x} f if  $a \leq x \leq b$  for x
    using integral_combine[of a x b for x, OF that integrable_continuous_real[OF
    assms(1)]]
    by (simp add: algebra_simps)
  show ?thesis
    using  $\langle x \in \_ \rangle$  *
    by (rule has_vector_derivative_transform)
    (auto intro!: derivative_eq_intros assms integral_has_vector_derivative)
qed

```

```

lemma integral_has_real_derivative':
  assumes continuous_on {a..b} g
  assumes  $t \in \{a..b\}$ 
  shows (( $\lambda x$ . integral {x..b} g) has_real_derivative -g t) (at t within {a..b})
  using integral_has_vector_derivative'[OF assms]
  by (auto simp: has_field_derivative_iff_has_vector_derivative)

```

6.15.31 This doesn't directly involve integration, but that gives an easy proof

```

lemma has_derivative_zero_unique_strong_interval:
  fixes f :: real  $\Rightarrow$  'a::banach
  assumes finite k
    and conf: continuous_on {a..b} f
    and f a = y
    and fder:  $\bigwedge x. x \in \{a..b\} - k \implies (f \text{ has\_derivative } (\lambda h. 0))$  (at x within {a..b})
    and x:  $x \in \{a..b\}$ 
  shows f x = y

```

```

proof –
  have  $a \leq b$   $a \leq x$ 
    using assms by auto
  have  $((\lambda x. 0 :: 'a) \text{ has\_integral } f \, x - f \, a) \{a..x\}$ 
  proof (rule fundamental\_theorem\_of\_calculus\_interior\_strong [OF  $\langle \text{finite } k \rangle \langle a \leq x \rangle$ ]; clarify?)
    have  $\{a..x\} \subseteq \{a..b\}$ 
      using  $x$  by auto
    then show continuous\_on  $\{a..x\}$   $f$ 
      by (rule continuous\_on\_subset [OF contf])
    show  $(f \text{ has\_vector\_derivative } 0) \text{ (at } z \text{) if } z: z \in \{a <..< x\} \text{ and } \text{notin: } z \notin k$ 
for  $z$ 
  unfolding has\_vector\_derivative\_def
  proof (simp add: at\_within\_open [OF  $z$ , symmetric])
    show  $(f \text{ has\_derivative } (\lambda x. 0)) \text{ (at } z \text{ within } \{a <..< x\})$ 
      by (rule has\_derivative\_subset [OF fder]) (use  $x \, z$  notin in auto)
  qed
qed
from has\_integral\_unique [OF has\_integral\_0 this]
show ?thesis
unfolding assms by auto
qed

```

6.15.32 Generalize a bit to any convex set

```

lemma has\_derivative\_zero\_unique\_strong\_convex:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{banach}$ 
  assumes convex  $S$  finite  $K$ 
    and contf: continuous\_on  $S$   $f$ 
    and  $c \in S$   $f \, c = y$ 
    and derf:  $\bigwedge x. x \in S - K \implies (f \text{ has\_derivative } (\lambda h. 0)) \text{ (at } x \text{ within } S)$ 
    and  $x \in S$ 
  shows  $f \, x = y$ 
proof (cases  $x = c$ )
  case True with  $\langle f \, c = y \rangle$  show ?thesis
    by blast
next
  case False
  let  $? \varphi = \lambda u. (1 - u) *_R c + u *_R x$ 
  have contf': continuous\_on  $\{0..1\}$   $(f \circ ? \varphi)$ 
  proof (rule continuous\_intros continuous\_on\_subset [OF contf]) +
    show  $(\lambda u. (1 - u) *_R c + u *_R x) ' \{0..1\} \subseteq S$ 
      using  $\langle \text{convex } S \rangle \langle x \in S \rangle \langle c \in S \rangle$  by (auto simp add: convex\_alt algebra\_simps)
  qed
  have  $t = u$  if  $? \varphi \, t = ? \varphi \, u$  for  $t \, u$ 
proof –
  from that have  $(t - u) *_R x = (t - u) *_R c$ 
    by (auto simp add: algebra\_simps)

```



```

    then show ?thesis
      using ‹ $x \neq c$ › by auto
  qed
  then have eq: (SOME t. ? $\varphi$  t = ? $\varphi$  u) = u for u
    by blast
  then have (? $\varphi$  - 'K')  $\subseteq$  ( $\lambda z$ . SOME t. ? $\varphi$  t = z) 'K
    by (clarsimp simp: image_iff) (metis (no_types) eq)
  then have fin: finite (? $\varphi$  - 'K')
    by (rule finite_surj[OF ‹finite K›])

  have derf': (( $\lambda u$ . f (? $\varphi$  u)) has_derivative ( $\lambda h$ . 0)) (at t within {0..1})
    if t  $\in$  {0..1} - {t. ? $\varphi$  t  $\in$  K} for t
  proof -
    have df: (f has_derivative ( $\lambda h$ . 0)) (at (? $\varphi$  t) within ? $\varphi$  ' {0..1})
      using ‹convex S› ‹ $x \in S$ › ‹ $c \in S$ › that
      by (auto simp add: convex_alt algebra_simps intro: has_derivative_subset
[OF derf])
    have (f  $\circ$  ? $\varphi$  has_derivative ( $\lambda x$ . 0)  $\circ$  ( $\lambda z$ . (0 - z *R c) + z *R x)) (at t within {0..1})
      by (rule derivative_eq_intros df | simp)+
    then show ?thesis
      unfolding o_def .
  qed
  have (f  $\circ$  ? $\varphi$ ) 1 = y
    apply (rule has_derivative_zero_unique_strong_interval[OF fin contf'])
    unfolding o_def using ‹f c = y› derf' by auto
  then show ?thesis
    by auto
  qed

```

Also to any open connected set with finite set of exceptions. Could generalize to locally convex set with limpt-free set of exceptions.

lemma *has_derivative_zero_unique_strong_connected:*

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{banach}$

assumes *connected S*

and *open S*

and *finite K*

and *contf: continuous_on S f*

and $c \in S$

and $f\ c = y$

and *derf: $\bigwedge x. x \in S - K \implies (f \text{ has\_derivative } (\lambda h. 0)) \text{ (at } x \text{ within } S)$*

and $x \in S$

shows $f\ x = y$

proof -

have $\exists e > 0. \text{ball } x\ e \subseteq (S \cap f - \{f\ x\})$ **if** $x \in S$ **for** x

proof -

obtain e **where** $0 < e$ **and** $e: \text{ball } x\ e \subseteq S$

using ‹ $x \in S$ › ‹open S› *open_contains_ball* **by** *blast*

have $\text{ball } x\ e \subseteq \{u \in S. f\ u \in \{f\ x\}\}$

```

proof safe
  fix  $y$ 
  assume  $y: y \in \text{ball } x \ e$ 
  then show  $y \in S$ 
    using  $e$  by auto
  show  $f \ y = f \ x$ 
  proof (rule has_derivative_zero_unique_strong_convex [OF convex_ball  $\langle \text{finite } K \rangle$ ])
    show continuous_on ( $\text{ball } x \ e$ )  $f$ 
      using contf continuous_on_subset  $e$  by blast
    show ( $f \text{ has\_derivative } (\lambda h. 0)$ ) (at  $u$  within  $\text{ball } x \ e$ )
      if  $u \in \text{ball } x \ e - K$  for  $u$ 
      by (metis Diff_iff contra_subsetD derf e has_derivative_subset that)
    qed (use  $y \ e \ \langle 0 < e \rangle$  in auto)
  qed
  then show  $\exists e > 0. \text{ball } x \ e \subseteq (S \cap f^{-1} \{f \ x\})$ 
    using  $\langle 0 < e \rangle$  by blast
  qed
  then have openin (top_of_set  $S$ )  $(S \cap f^{-1} \{y\})$ 
    by (auto intro!: open_openin_trans [OF  $\langle \text{open } S \rangle$ ] simp: open_contains_ball)
  moreover have closedin (top_of_set  $S$ )  $(S \cap f^{-1} \{y\})$ 
    by (force intro!: continuous_closedin_preimage [OF contf])
  ultimately have  $(S \cap f^{-1} \{y\}) = \{y\} \vee (S \cap f^{-1} \{y\}) = S$ 
    using  $\langle \text{connected } S \rangle$  by (simp add: connected_clopen)
  then show ?thesis
    using  $\langle x \in S \rangle \ \langle f \ c = y \rangle \ \langle c \in S \rangle$  by auto
  qed

lemma has_derivative_zero_connected_constant:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{banach}$ 
  assumes connected  $S$ 
    and open  $S$ 
    and finite  $k$ 
    and continuous_on  $S \ f$ 
    and  $\forall x \in (S - k). (f \text{ has\_derivative } (\lambda h. 0)) \ (\text{at } x \text{ within } S)$ 
  obtains  $c$  where  $\bigwedge x. x \in S \implies f(x) = c$ 
proof (cases  $S = \{y\}$ )
  case True
    then show ?thesis
      by (metis empty_iff that)
  next
    case False
    then obtain  $c$  where  $c \in S$ 
      by (metis equals0I)
    then show ?thesis
      by (metis has_derivative_zero_unique_strong_connected assms that)
  qed

lemma DERIV_zero_connected_constant:

```

```

fixes  $f :: 'a::\{\text{real\_normed\_field}, \text{euclidean\_space}\} \Rightarrow 'a$ 
assumes connected S
  and open S
  and finite K
  and continuous_on S f
  and  $\forall x \in (S - K). \text{DERIV } f \, x :> 0$ 
obtains  $c$  where  $\bigwedge x. x \in S \implies f(x) = c$ 
using has_derivative_zero_connected_constant [OF assms(1-4)] assms
by (metis DERIV_const has_derivative_const Diff_iff at_within_open frechet_derivative_at
has_field_derivative_def)

```

6.15.33 Integrating characteristic function of an interval

```

lemma has_integral_restrict_open_subinterval:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{banach}$ 
  assumes intf: (f has_integral i) (cbox c d)
  and cb: cbox c d  $\subseteq$  cbox a b
  shows  $((\lambda x. \text{if } x \in \text{box } c \, d \text{ then } f \, x \text{ else } 0) \text{ has\_integral } i) (\text{cbox } a \, b)$ 
proof (cases cbox c d = {})
  case True
  then have box c d = {}
  by (metis bot.extremum_uniqueI box_subset_cbox)
  then show ?thesis
  using True intf by auto
next
  case False
  then obtain  $p$  where pdiv: p division_of cbox a b and inp: cbox c d  $\in$  p
  using cb partial_division_extend_1 by blast
  define  $g$  where  $[\text{abs\_def}]: g \, x = (\text{if } x \in \text{box } c \, d \text{ then } f \, x \text{ else } 0)$  for  $x$ 
  interpret operative lift_option plus Some (0 :: 'b)
   $\lambda i. \text{if } g \text{ integrable\_on } i \text{ then } \text{Some } (\text{integral } i \, g) \text{ else } \text{None}$ 
  by (fact operative_integralI)
  note operat = division [OF pdiv, symmetric]
  show ?thesis
  proof (cases (g has_integral i) (cbox a b))
  case True then show ?thesis
  by (simp add: g_def)
  next
  case False
  have iterate:F ( $\lambda i. \text{if } g \text{ integrable\_on } i \text{ then } \text{Some } (\text{integral } i \, g) \text{ else } \text{None}$ ) (p
  - {cbox c d}) = Some 0
  proof (intro neutral ballI)
  fix  $x$ 
  assume  $x: x \in p - \{\text{cbox } c \, d\}$ 
  then have  $x \in p$ 
  by auto
  then obtain  $u \, v$  where  $uv: x = \text{cbox } u \, v$ 
  using pdiv by blast
  have  $\text{interior } x \cap \text{interior } (\text{cbox } c \, d) = \{\}$ 

```

```

    using pdiv inp x by blast
  then have (g has_integral 0) x
    unfolding uv using has_integral_spike_interior[where f= $\lambda x. 0$ ]
    by (metis (no_types, lifting) disjoint_iff_not_equal g_def has_integral_0_eq
interior_cbox)
  then show (if g integrable_on x then Some (integral x g) else None) = Some
0
    by auto
qed
interpret comm_monoid_set lift_option plus Some (0 :: 'b)
by (intro comm_monoid_set.intro comm_monoid_lift_option add.comm_monoid_axioms)
have intg: g integrable_on cbox c d
  using integrable_spike_interior[where f=f]
  by (meson g_def has_integral_integrable intf)
moreover have integral (cbox c d) g = i
proof (rule has_integral_unique[OF has_integral_spike_interior intf])
  show  $\bigwedge x. x \in \text{box } c \ d \implies f \ x = g \ x$ 
    by (auto simp: g_def)
  show (g has_integral integral (cbox c d) g) (cbox c d)
    by (rule integrable_integral[OF intg])
qed
ultimately have F ( $\lambda A. \text{if } g \text{ integrable\_on } A \text{ then Some (integral } A \ g) \text{ else None}$ ) p = Some i
  by (metis (full_types, lifting) division_of_finite inp iterate pdiv remove
right_neutral)
then
  have (g has_integral i) (cbox a b)
  by (metis integrable_on_def integral_unique operat_option.inject option.simps(3))
with False show ?thesis
  by blast
qed
qed

```

```

lemma has_integral_restrict_closed_subinterval:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::banach
  assumes (f has_integral i) (cbox c d)
    and cbox c d  $\subseteq$  cbox a b
  shows (( $\lambda x. \text{if } x \in \text{cbox } c \ d \text{ then } f \ x \text{ else } 0$ ) has_integral i) (cbox a b)
proof -
  note has_integral_restrict_open_subinterval[OF assms]
  note * = has_integral_spike[OF negligible_frontier_interval _ this]
  show ?thesis
    by (rule *[of c d]) (use box_subset_cbox[of c d] in auto)
qed

```

```

lemma has_integral_restrict_closed_subintervals_eq:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::banach
  assumes cbox c d  $\subseteq$  cbox a b

```

```

shows (( $\lambda x.$  if  $x \in \text{cbox } c \ d$  then  $f \ x$  else  $0$ )  $\text{has\_integral } i$ ) ( $\text{cbox } a \ b$ )  $\longleftrightarrow$  ( $f$ 
 $\text{has\_integral } i$ ) ( $\text{cbox } c \ d$ )
(is ?l = ?r)
proof (cases  $\text{cbox } c \ d = \{\}$ )
  case False
    let  $?g = \lambda x.$  if  $x \in \text{cbox } c \ d$  then  $f \ x$  else  $0$ 
    show  $?thesis$ 
    proof
      assume  $?l$ 
      then have  $?g \text{ integrable\_on } \text{cbox } c \ d$ 
        using  $\text{assms has\_integral\_integrable\_subinterval}$  by blast
      then have  $f \text{ integrable\_on } \text{cbox } c \ d$ 
        by (rule  $\text{integrable\_eq}$ ) auto
      moreover then have  $i = \text{integral } (\text{cbox } c \ d) \ f$ 
        by (meson  $\langle (\lambda x.$  if  $x \in \text{cbox } c \ d$  then  $f \ x$  else  $0$ )  $\text{has\_integral } i$ ) ( $\text{cbox } a \ b \rangle$ 
 $\text{assms has\_integral\_restrict\_closed\_subinterval has\_integral\_unique integrable\_integral}$ )
      ultimately show  $?r$  by auto
    next
      assume  $?r$  then show  $?l$ 
        by (rule  $\text{has\_integral\_restrict\_closed\_subinterval[OF\_assms]}$ )
    qed
qed auto

```

Hence we can apply the limit process uniformly to all integrals.

```

lemma  $\text{has\_integral'}$ :
  fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$ 
  shows ( $f \text{ has\_integral } i$ )  $S \longleftrightarrow$ 
    ( $\forall e > 0. \exists B > 0. \forall a \ b. \text{ball } 0 \ B \subseteq \text{cbox } a \ b \longrightarrow$ 
      ( $\exists z. ((\lambda x.$  if  $x \in S$  then  $f(x)$  else  $0$ )  $\text{has\_integral } z$ ) ( $\text{cbox } a \ b$ )  $\wedge \text{norm}(z -$ 
 $i) < e$ )
    (is  $?l \longleftrightarrow (\forall e > 0. ?r \ e)$ )
proof (cases  $\exists a \ b. S = \text{cbox } a \ b$ )
  case False then show  $?thesis$ 
    by (simp add:  $\text{has\_integral\_alt}$ )
next
  case True
    then obtain  $a \ b$  where  $S: S = \text{cbox } a \ b$ 
      by blast
    obtain  $B$  where  $0 < B$  and  $B: \bigwedge x. x \in \text{cbox } a \ b \implies \text{norm } x \leq B$ 
      using  $\text{bounded\_cbox[unfolded bounded\_pos]}$  by blast
    show  $?thesis$ 
    proof safe
      fix  $e :: \text{real}$ 
      assume  $?l$  and  $e > 0$ 
      have (( $\lambda x.$  if  $x \in S$  then  $f \ x$  else  $0$ )  $\text{has\_integral } i$ ) ( $\text{cbox } c \ d$ )
        if  $\text{ball } 0 \ (B+1) \subseteq \text{cbox } c \ d$  for  $c \ d$ 
          unfolding  $S$  using  $B$  that
          by (force intro:  $\langle ?l \rangle[\text{unfolded } S] \text{ has\_integral\_restrict\_closed\_subinterval}$ )
    qed

```

```

then show ?r e
  by (meson ⟨0 < B⟩ ⟨0 < e⟩ add_pos_pos le_less_trans zero_less_one
norm_pths(2))
next
  assume as: ∀ e > 0. ?r e
  then obtain C
    where C: ∧ a b. ball 0 C ⊆ cbox a b ⇒
      ∃ z. ((λ x. if x ∈ S then f x else 0) has_integral z) (cbox a b)
    by (meson zero_less_one)
  define c :: 'n where c = (∑ i ∈ Basis. (− max B C) *R i)
  define d :: 'n where d = (∑ i ∈ Basis. max B C *R i)
  have c · i ≤ x · i ∧ x · i ≤ d · i if norm x ≤ B i ∈ Basis for x i
    using that and Basis_le_norm[OF ⟨i ∈ Basis⟩, of x]
    by (auto simp add: field_simps sum_negf c_def d_def)
  then have c_d: cbox a b ⊆ cbox c d
    by (meson B mem_box(2) subsetI)
  have c · i ≤ x · i ∧ x · i ≤ d · i
    if x: norm (0 − x) < C and i: i ∈ Basis for x i
    using Basis_le_norm[OF i, of x] x i by (auto simp: sum_negf c_def d_def)
  then have ball 0 C ⊆ cbox c d
    by (auto simp: mem_box dist_norm)
  with C obtain y where y: (f has_integral y) (cbox a b)
    using c_d has_integral_restrict_closed_subintervals_eq S by blast
  have y = i
  proof (rule ccontr)
    assume y ≠ i
    then have 0 < norm (y − i)
      by auto
    from as[rule_format, OF this]
    obtain C where C: ∧ a b. ball 0 C ⊆ cbox a b ⇒
      ∃ z. ((λ x. if x ∈ S then f x else 0) has_integral z) (cbox a b) ∧ norm (z − i)
      < norm (y − i)
    by auto
    define c :: 'n where c = (∑ i ∈ Basis. (− max B C) *R i)
    define d :: 'n where d = (∑ i ∈ Basis. max B C *R i)
    have c · i ≤ x · i ∧ x · i ≤ d · i
      if norm x ≤ B and i ∈ Basis for x i
      using that Basis_le_norm[of i x] by (auto simp add: field_simps sum_negf
c_def d_def)
    then have c_d: cbox a b ⊆ cbox c d
      by (simp add: B mem_box(2) subset_eq)
    have c · i ≤ x · i ∧ x · i ≤ d · i if norm (0 − x) < C and i ∈ Basis for x i
      using Basis_le_norm[of i x] that by (auto simp: sum_negf c_def d_def)
    then have ball 0 C ⊆ cbox c d
      by (auto simp: mem_box dist_norm)
    with C obtain z where z: (f has_integral z) (cbox a b) norm (z − i) < norm
(y − i)
      using has_integral_restrict_closed_subintervals_eq[OF c_d] S by blast
    moreover then have z = y

```

```

      by (blast intro: has_integral_unique[OF _ y])
    ultimately show False
      by auto
  qed
  then show ?l
    using y by (auto simp: S)
  qed
qed

lemma has_integral_le:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  real
  assumes fg: (f has_integral i) S (g has_integral j) S
    and le:  $\bigwedge x. x \in S \implies f\ x \leq g\ x$ 
  shows  $i \leq j$ 
  using has_integral_component_le[OF _ fg, of 1] le by auto

lemma integral_le:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  real
  assumes f integrable_on S
    and g integrable_on S
    and  $\bigwedge x. x \in S \implies f\ x \leq g\ x$ 
  shows  $\text{integral } S\ f \leq \text{integral } S\ g$ 
  by (rule has_integral_le[OF assms(1,2)[unfolded has_integral_integral] assms(3)])

lemma has_integral_nonneg:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  real
  assumes (f has_integral i) S
    and  $\bigwedge x. x \in S \implies 0 \leq f\ x$ 
  shows  $0 \leq i$ 
  using has_integral_component_nonneg[of 1 f i S]
  unfolding o_def
  using assms
  by auto

lemma integral_nonneg:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  real
  assumes f: f integrable_on S and 0:  $\bigwedge x. x \in S \implies 0 \leq f\ x$ 
  shows  $0 \leq \text{integral } S\ f$ 
  by (rule has_integral_nonneg[OF f[unfolded has_integral_integral] 0])

Hence a general restriction property.

lemma has_integral_restrict [simp]:
  fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: banach
  assumes  $S \subseteq T$ 
  shows  $((\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) \text{ has\_integral } i)\ T \longleftrightarrow (f \text{ has\_integral } i)\ S$ 
proof -
  have *:  $\bigwedge x. (\text{if } x \in T \text{ then if } x \in S \text{ then } f\ x \text{ else } 0 \text{ else } 0) = (\text{if } x \in S \text{ then } f\ x \text{ else } 0)$ 
  using assms by auto

```

```

show ?thesis
  apply (subst(2) has_integral')
  apply (subst has_integral')
  apply (simp add: *)
  done
qed

corollary has_integral_restrict_UNIV:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  shows (( $\lambda x$ . if  $x \in s$  then  $f x$  else 0) has_integral i) UNIV  $\longleftrightarrow$  (f has_integral i) s
  by auto

lemma has_integral_restrict_Int:
  fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: banach
  shows (( $\lambda x$ . if  $x \in S$  then  $f x$  else 0) has_integral i) T  $\longleftrightarrow$  (f has_integral i) (S  $\cap$  T)
proof -
  have (( $\lambda x$ . if  $x \in T$  then if  $x \in S$  then  $f x$  else 0 else 0) has_integral i) UNIV =
    (( $\lambda x$ . if  $x \in S \cap T$  then  $f x$  else 0) has_integral i) UNIV
  by (rule has_integral_cong) auto
  then show ?thesis
  using has_integral_restrict_UNIV by fastforce
qed

lemma integral_restrict_Int:
  fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: banach
  shows integral T ( $\lambda x$ . if  $x \in S$  then  $f x$  else 0) = integral (S  $\cap$  T) f
  by (metis (no_types, lifting) has_integral_cong has_integral_restrict_Int integrable_integral integral_unique not_integrable_integral)

lemma integrable_restrict_Int:
  fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: banach
  shows ( $\lambda x$ . if  $x \in S$  then  $f x$  else 0) integrable_on T  $\longleftrightarrow$  f integrable_on (S  $\cap$  T)
  using has_integral_restrict_Int by fastforce

lemma has_integral_on_superset:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes f: (f has_integral i) S
  and  $\bigwedge x. x \notin S \implies f x = 0$ 
  and  $S \subseteq T$ 
  shows (f has_integral i) T
proof -
  have ( $\lambda x$ . if  $x \in S$  then  $f x$  else 0) = ( $\lambda x$ . if  $x \in T$  then  $f x$  else 0)
  using assms by fastforce
  with f show ?thesis
  by (simp only: has_integral_restrict_UNIV [symmetric, of f])
qed

```



```

lemma integrable_on_superset:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes f integrable_on S
    and  $\bigwedge x. x \notin S \implies f x = 0$ 
    and  $S \subseteq t$ 
  shows f integrable_on t
  using assms
  unfolding integrable_on_def
  by (auto intro: has_integral_on_superset)

lemma integral_restrict_UNIV:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  shows integral UNIV ( $\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0$ ) = integral S f
  by (simp add: integral_restrict_Int)

lemma integrable_restrict_UNIV:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  shows ( $\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0$ ) integrable_on UNIV  $\longleftrightarrow$  f integrable_on s
  unfolding integrable_on_def
  by auto

lemma has_integral_subset_component_le:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'm::euclidean_space
  assumes k: k  $\in$  Basis
    and as:  $S \subseteq T$  ( $f$  has_integral i) S ( $f$  has_integral j) T  $\bigwedge x. x \in T \implies 0 \leq f(x) \cdot k$ 
  shows  $i \cdot k \leq j \cdot k$ 
proof -
  have  $\S$ : (( $\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0$ ) has_integral i) UNIV
    (( $\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0$ ) has_integral j) UNIV
  by (simp_all add: assms)
  show ?thesis
    using as by (force intro!: has_integral_component_le[OF k  $\S$ ])
qed

```

6.15.34 Integrals on set differences

```

lemma has_integral_setdiff:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::banach
  assumes S: (f has_integral i) S and T: (f has_integral j) T
    and neg: negligible (T - S)
  shows (f has_integral (i - j)) (S - T)
proof -
  show ?thesis
    unfolding has_integral_restrict_UNIV [symmetric, of f]
  proof (rule has_integral_spike [OF neg])
    have eq: ( $\lambda x. (\text{if } x \in S \text{ then } f x \text{ else } 0) - (\text{if } x \in T \text{ then } f x \text{ else } 0)$ ) =
      ( $\lambda x. \text{if } x \in T - S \text{ then } -f x \text{ else if } x \in S - T \text{ then } f x \text{ else } 0$ )

```

```

    by (force simp add: )
  have (( $\lambda x$ . if  $x \in S$  then  $f x$  else 0) has_integral i) UNIV
    (( $\lambda x$ . if  $x \in T$  then  $f x$  else 0) has_integral j) UNIV
    using  $S \ T$  has_integral_restrict_UNIV by auto
  from has_integral_diff [OF this]
  show (( $\lambda x$ . if  $x \in T - S$  then  $-f x$  else if  $x \in S - T$  then  $f x$  else 0)
        has_integral i-j) UNIV
    by (simp add: eq)
  qed force
qed

```

```

lemma integral_setdiff:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::banach$ 
  assumes  $f$  integrable_on  $S$   $f$  integrable_on  $T$  negligible( $T - S$ )
  shows  $\text{integral } (S - T) f = \text{integral } S f - \text{integral } T f$ 
  by (rule integral_unique) (simp add: assms has_integral_setdiff integrable_integral)

```

```

lemma integrable_setdiff:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::banach$ 
  assumes ( $f$  has_integral i)  $S$  ( $f$  has_integral j)  $T$  negligible ( $T - S$ )
  shows  $f$  integrable_on ( $S - T$ )
  using has_integral_setdiff [OF assms]
  by (simp add: has_integral_iff)

```

```

lemma negligible_setdiff [simp]:  $T \subseteq S \implies \text{negligible } (T - S)$ 
  by (metis Diff_eq_empty_iff negligible_empty)

```

```

lemma negligible_on_intervals:  $\text{negligible } s \longleftrightarrow (\forall a \ b. \text{negligible}(s \cap \text{cbox } a \ b))$ 
(is ?l  $\longleftrightarrow$  ?r)

```

```

proof
  assume  $R$ : ?r
  show ?l
    unfolding negligible_def
  proof safe
    fix  $a \ b$ 
    have negligible ( $s \cap \text{cbox } a \ b$ )
      by (simp add: R)
    then show (indicator  $s$  has_integral 0) ( $\text{cbox } a \ b$ )
      by (meson Diff_iff Int_iff has_integral_negligible indicator_simps(2))
  qed
qed (simp add: negligible_Int)

```

```

lemma negligible_translation:
  assumes negligible  $S$ 
  shows negligible ((+)  $c$  '  $S$ )
proof -
  have inj: inj ((+)  $c$ )
    by simp
  show ?thesis

```

```

using assms
proof (clarsimp simp: negligible_def)
  fix a b
  assume  $\forall x y. (\text{indicator } S \text{ has\_integral } 0) (cbox\ x\ y)$ 
  then have *:  $(\text{indicator } S \text{ has\_integral } 0) (cbox\ (a-c)\ (b-c))$ 
    by (meson Diff_iff assms has_integral_negligible indicator_simps(2))
  have eq:  $\text{indicator } ((+) c\ 'S) = (\lambda x. \text{indicator } S\ (x - c))$ 
    by (force simp add: indicator_def)
  show  $(\text{indicator } ((+) c\ 'S) \text{ has\_integral } 0) (cbox\ a\ b)$ 
    using has_integral_affinity [OF *, of 1 -c]
      cbox_translation [of c -c+a -c+b]
    by (simp add: eq) (simp add: ac_simps)
qed
qed

lemma negligible_translation_rev:
  assumes negligible  $((+) c\ 'S)$ 
  shows negligible S
by (metis negligible_translation [OF assms, of -c] translation_galois)

lemma negligible_atLeastAtMostI:  $b \leq a \implies \text{negligible } \{a..(b::\text{real})\}$ 
  using negligible_insert by fastforce

lemma has_integral_spike_set_eq:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes negligible  $\{x \in S - T. f\ x \neq 0\}$  negligible  $\{x \in T - S. f\ x \neq 0\}$ 
  shows  $(f \text{ has\_integral } y)\ S \longleftrightarrow (f \text{ has\_integral } y)\ T$ 
proof -
  have  $((\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) \text{ has\_integral } y)\ UNIV =$ 
     $((\lambda x. \text{if } x \in T \text{ then } f\ x \text{ else } 0) \text{ has\_integral } y)\ UNIV$ 
  proof (rule has_integral_spike_eq)
    show negligible  $(\{x \in S - T. f\ x \neq 0\} \cup \{x \in T - S. f\ x \neq 0\})$ 
      by (rule negligible_Un [OF assms])
  qed auto
  then show ?thesis
    by (simp add: has_integral_restrict_UNIV)
qed

corollary integral_spike_set:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes negligible  $\{x \in S - T. f\ x \neq 0\}$  negligible  $\{x \in T - S. f\ x \neq 0\}$ 
  shows  $\text{integral } S\ f = \text{integral } T\ f$ 
  using has_integral_spike_set_eq [OF assms]
  by (metis eq_integralD integral_unique)

lemma integrable_spike_set:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes f:  $f \text{ integrable\_on } S$  and neg: negligible  $\{x \in S - T. f\ x \neq 0\}$  negligible  $\{x \in T - S. f\ x \neq 0\}$ 

```

shows f integrable_on T
using *has_integral_spike_set_eq* [*OF neg*] f **by** *blast*

lemma *integrable_spike_set_eq*:
fixes $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$
assumes *negligible* $((S - T) \cup (T - S))$
shows f integrable_on $S \longleftrightarrow f$ integrable_on T
by (*blast intro: integrable_spike_set assms negligible_subset*)

lemma *integrable_on_insert_iff*: f integrable_on $(\text{insert } x \ X) \longleftrightarrow f$ integrable_on X
for $f :: \_ \Rightarrow 'a::\text{banach}$
by (*rule integrable_spike_set_eq (auto simp: insert_Diff_if)*)

lemma *has_integral_interior*:
fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{banach}$
shows *negligible*(*frontier* S) $\implies (f$ *has_integral* $y) (\text{interior } S) \longleftrightarrow (f$ *has_integral* $y) S$
by (*rule has_integral_spike_set_eq [OF empty_imp_negligible negligible_subset]*)
(use interior_subset in (auto simp: frontier_def closure_def))

lemma *has_integral_closure*:
fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{banach}$
shows *negligible*(*frontier* S) $\implies (f$ *has_integral* $y) (\text{closure } S) \longleftrightarrow (f$ *has_integral* $y) S$
by (*rule has_integral_spike_set_eq [OF negligible_subset empty_imp_negligible]*)
(auto simp: closure_Un_frontier)

lemma *has_integral_open_interval*:
fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{banach}$
shows $(f$ *has_integral* $y) (\text{box } a \ b) \longleftrightarrow (f$ *has_integral* $y) (\text{cbox } a \ b)$
unfolding *interior_cbox* [*symmetric*]
by (*metis frontier_cbox has_integral_interior negligible_frontier_interval*)

lemma *integrable_on_open_interval*:
fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{banach}$
shows f integrable_on $\text{box } a \ b \longleftrightarrow f$ integrable_on $\text{cbox } a \ b$
by (*simp add: has_integral_open_interval integrable_on_def*)

lemma *integral_open_interval*:
fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{banach}$
shows *integral*($\text{box } a \ b$) $f = \text{integral}(\text{cbox } a \ b) f$
by (*metis has_integral_integrable_integral has_integral_open_interval not_integrable_integral*)

6.15.35 More lemmas that are useful later

lemma *has_integral_subset_le*:
fixes $f :: 'n::\text{euclidean\_space} \Rightarrow \text{real}$
assumes $s \subseteq t$

```

    and (f has_integral i) s
    and (f has_integral j) t
    and  $\forall x \in t. 0 \leq f x$ 
  shows  $i \leq j$ 
  using has_integral_subset_component_le[OF __ assms(1), of 1 f i j]
  using assms
  by auto

```

```

lemma integral_subset_component_le:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'm::euclidean_space
  assumes k  $\in$  Basis
    and  $s \subseteq t$ 
    and f integrable_on s
    and f integrable_on t
    and  $\forall x \in t. 0 \leq f x \cdot k$ 
  shows  $(\text{integral } s f) \cdot k \leq (\text{integral } t f) \cdot k$ 
  by (meson assms has_integral_subset_component_le integrable_integral)

```

```

lemma integral_subset_le:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  real
  assumes  $s \subseteq t$ 
    and f integrable_on s
    and f integrable_on t
    and  $\forall x \in t. 0 \leq f x$ 
  shows  $\text{integral } s f \leq \text{integral } t f$ 
  using assms has_integral_subset_le by blast

```

```

lemma has_integral_alt':
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  shows (f has_integral i) s  $\longleftrightarrow$ 
    ( $\forall a b. (\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0) \text{ integrable\_on } \text{cbox } a b) \wedge$ 
    ( $\forall e > 0. \exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow$ 
       $\text{norm } (\text{integral } (\text{cbox } a b) (\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0) - i) < e$ )
    (is ?l = ?r)

```

```

proof
  assume rhs: ?r
  show ?l
  proof (subst has_integral', intro allI impI)
    fix e::real
    assume e > 0
    from rhs[THEN conjunct2, rule_format, OF this]
    show  $\exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow$ 
      ( $\exists z. ((\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0) \text{ has\_integral } z)$ 
         $(\text{cbox } a b) \wedge \text{norm } (z - i) < e)$ )
    by (simp add: has_integral_iff rhs)
  qed
next
  let ? $\Phi = \lambda e a b. \exists z. ((\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0) \text{ has\_integral } z) (\text{cbox } a b) \wedge$ 
     $\text{norm } (z - i) < e$ 

```

```

assume ?l
then have lhs:  $\exists B > 0. \forall a \ b. \text{ball } 0 \ B \subseteq \text{cbox } a \ b \longrightarrow ?\Phi \ e \ a \ b \text{ if } e > 0 \text{ for } e$ 
  using that has_integral'[of f] by auto
let ?f =  $\lambda x. \text{if } x \in s \text{ then } f \ x \text{ else } 0$ 
show ?r
proof (intro conjI allI impI)
  fix a b :: 'n
  from lhs[OF zero_less_one]
  obtain B where  $0 < B$  and B:  $\bigwedge a \ b. \text{ball } 0 \ B \subseteq \text{cbox } a \ b \implies ?\Phi \ 1 \ a \ b$ 
  by blast
  let ?a =  $\sum_{i \in \text{Basis}. \min (a \cdot i) (-B) *_R i :: 'n}$ 
  let ?b =  $\sum_{i \in \text{Basis}. \max (b \cdot i) B *_R i :: 'n}$ 
  show ?f integrable_on cbox a b
  proof (rule integrable_subinterval[of _ ?a ?b])
    have  $?a \cdot i \leq x \cdot i \wedge x \cdot i \leq ?b \cdot i$  if  $\text{norm } (0 - x) < B$   $i \in \text{Basis}$  for  $x \ i$ 
    using Basis_le_norm[of i x] that by (auto simp add:field_simps)
    then have  $\text{ball } 0 \ B \subseteq \text{cbox } ?a \ ?b$ 
    by (auto simp: mem_box dist_norm)
    then show ?f integrable_on cbox ?a ?b
    unfolding integrable_on_def using B by blast
    show  $\text{cbox } a \ b \subseteq \text{cbox } ?a \ ?b$ 
    by (force simp: mem_box)
  qed
qed

fix e :: real
assume  $e > 0$ 
with lhs show  $\exists B > 0. \forall a \ b. \text{ball } 0 \ B \subseteq \text{cbox } a \ b \longrightarrow$ 
   $\text{norm } (\text{integral } (\text{cbox } a \ b) (\lambda x. \text{if } x \in s \text{ then } f \ x \text{ else } 0) - i) < e$ 
  by (metis (no_types, lifting) has_integral_integrable_integral)
qed
qed

```

6.15.36 Continuity of the integral (for a 1-dimensional interval)

```

lemma integrable_alt:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  shows  $f \text{ integrable\_on } s \longleftrightarrow$ 
     $(\forall a \ b. (\lambda x. \text{if } x \in s \text{ then } f \ x \text{ else } 0) \text{ integrable\_on } \text{cbox } a \ b) \wedge$ 
     $(\forall e > 0. \exists B > 0. \forall a \ b \ c \ d. \text{ball } 0 \ B \subseteq \text{cbox } a \ b \wedge \text{ball } 0 \ B \subseteq \text{cbox } c \ d \longrightarrow$ 
     $\text{norm } (\text{integral } (\text{cbox } a \ b) (\lambda x. \text{if } x \in s \text{ then } f \ x \text{ else } 0) -$ 
     $\text{integral } (\text{cbox } c \ d) (\lambda x. \text{if } x \in s \text{ then } f \ x \text{ else } 0)) < e)$ 
    (is ?l = ?r)
proof
  let ?F =  $\lambda x. \text{if } x \in s \text{ then } f \ x \text{ else } 0$ 
  assume ?l
  then obtain y where  $\text{intF}: \bigwedge a \ b. ?F \text{ integrable\_on } \text{cbox } a \ b$ 
  and y:  $0 < e \implies$ 
     $\exists B > 0. \forall a \ b. \text{ball } 0 \ B \subseteq \text{cbox } a \ b \longrightarrow \text{norm } (\text{integral } (\text{cbox } a \ b) ?F -$ 

```

```

y) < e
  unfolding integrable_on_def has_integral_alt'[of f] by auto
show ?r
proof (intro conjI allI impI intF)
  fix e::real
  assume e > 0
  then have e/2 > 0
    by auto
  obtain B where 0 < B
    and B:  $\bigwedge a b. \text{ball } 0 B \subseteq \text{cbox } a b \implies \text{norm } (\text{integral } (\text{cbox } a b) ?F - y) < e/2$ 
  using <0 < e/2> y by blast
  show  $\exists B > 0. \forall a b c d. \text{ball } 0 B \subseteq \text{cbox } a b \wedge \text{ball } 0 B \subseteq \text{cbox } c d \longrightarrow$ 
     $\text{norm } (\text{integral } (\text{cbox } a b) ?F - \text{integral } (\text{cbox } c d) ?F) < e$ 
  proof (intro conjI exI impI allI, rule <0 < B>)
    fix a b c d::'n
    assume sub:  $\text{ball } 0 B \subseteq \text{cbox } a b \wedge \text{ball } 0 B \subseteq \text{cbox } c d$ 
    show  $\text{norm } (\text{integral } (\text{cbox } a b) ?F - \text{integral } (\text{cbox } c d) ?F) < e$ 
      using sub by (auto intro: norm_triangle_half_l dest: B)
  qed
qed
next
let ?F =  $\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0$ 
assume rhs: ?r
let ?cube =  $\lambda n. \text{cbox } (\sum i \in \text{Basis}. - \text{real } n *_{\mathbb{R}} i::'n) (\sum i \in \text{Basis}. \text{real } n *_{\mathbb{R}} i)$ 
have Cauchy ( $\lambda n. \text{integral } (?cube n) ?F$ )
  unfolding Cauchy_def
proof (intro allI impI)
  fix e::real
  assume e > 0
  with rhs obtain B where 0 < B
    and B:  $\bigwedge a b c d. \text{ball } 0 B \subseteq \text{cbox } a b \wedge \text{ball } 0 B \subseteq \text{cbox } c d$ 
       $\implies \text{norm } (\text{integral } (\text{cbox } a b) ?F - \text{integral } (\text{cbox } c d) ?F) < e$ 
  by blast
  obtain N where B  $\leq$  real N
    using real_arch_simple by blast
  have  $\text{ball } 0 B \subseteq ?cube n$  if n:  $n \geq N$  for n
  proof -
    have  $\text{sum } ((*_{\mathbb{R}}) (- \text{real } n)) \text{Basis} \cdot i \leq x \cdot i \wedge$ 
       $x \cdot i \leq \text{sum } ((*_{\mathbb{R}}) (\text{real } n)) \text{Basis} \cdot i$ 
    if  $\text{norm } x < B$   $i \in \text{Basis}$  for  $x i::'n$ 
      using Basis_le_norm[of i x] n N that by (auto simp add: field_simps
sum_negf)
    then show ?thesis
      by (auto simp: mem_box dist_norm)
  qed
  then show  $\exists M. \forall m \geq M. \forall n \geq M. \text{dist } (\text{integral } (?cube m) ?F) (\text{integral } (?cube$ 
n) ?F) < e
    by (fastforce simp add: dist_norm intro!: B)

```

```

qed
then obtain i where i: (λn. integral (?cube n) ?F) → i
  using convergent_eq_Cauchy by blast
have ∃ B>0. ∀ a b. ball 0 B ⊆ cbox a b → norm (integral (cbox a b) ?F - i)
< e
  if e > 0 for e
proof -
  have *: e/2 > 0 using that by auto
  then obtain N where N: ∧n. N ≤ n ⇒ norm (i - integral (?cube n) ?F)
< e/2
  using i[THEN LIMSEQ_D, simplified norm_minus_commute] by meson
obtain B where 0 < B
  and B: ∧a b c d. [ball 0 B ⊆ cbox a b; ball 0 B ⊆ cbox c d] ⇒
    norm (integral (cbox a b) ?F - integral (cbox c d) ?F) < e/2
  using rhs * by meson
let ?B = max (real N) B
show ?thesis
proof (intro exI conjI allI impI)
  show 0 < ?B
    using ⟨B > 0⟩ by auto
  fix a b :: 'n
  assume ball 0 ?B ⊆ cbox a b
  moreover obtain n where n: max (real N) B ≤ real n
    using real_arch_simple by blast
  moreover have ball 0 B ⊆ ?cube n
  proof
    fix x :: 'n
    assume x: x ∈ ball 0 B
    have [norm (0 - x) < B; i ∈ Basis]
      ⇒ sum ((*_R) (-n)) Basis · i ≤ x · i ∧ x · i ≤ sum ((*_R) n) Basis · i
  for i
    using Basis_le_norm[of i x] n by (auto simp add: field_simps sum_negf)
  then show x ∈ ?cube n
    using x by (auto simp: mem_box dist_norm)
  qed
  ultimately show norm (integral (cbox a b) ?F - i) < e
    using norm_triangle_half_l [OF B N] by force
  qed
qed
then show ?l unfolding integrable_on_def has_integral_alt'[of f]
  using rhs by blast
qed

lemma integrable_altD:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  assumes f integrable_on s
  shows ∧a b. (λx. if x ∈ s then f x else 0) integrable_on cbox a b
    and ∧e. e > 0 ⇒ ∃ B>0. ∀ a b c d. ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c
d →

```


$\text{norm } (\text{integral } (\text{cbox } a \ b) \ (\lambda x. \text{ if } x \in s \text{ then } f \ x \text{ else } 0) - \text{integral } (\text{cbox } c \ d) \ (\lambda x. \text{ if } x \in s \text{ then } f \ x \text{ else } 0)) < e$
using *assms[unfolded integrable_alt[of f]]* **by** *auto*

lemma *integrable_alt_subset*:

fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{banach}$

shows

$f \text{ integrable\_on } S \longleftrightarrow$

$(\forall a \ b. (\lambda x. \text{ if } x \in S \text{ then } f \ x \text{ else } 0) \text{ integrable\_on } \text{cbox } a \ b) \wedge$

$(\forall e > 0. \exists B > 0. \forall a \ b \ c \ d.$

$\text{ball } 0 \ B \subseteq \text{cbox } a \ b \wedge \text{cbox } a \ b \subseteq \text{cbox } c \ d$

$\longrightarrow \text{norm}(\text{integral } (\text{cbox } a \ b) \ (\lambda x. \text{ if } x \in S \text{ then } f \ x \text{ else } 0) - \text{integral } (\text{cbox } c \ d) \ (\lambda x. \text{ if } x \in S \text{ then } f \ x \text{ else } 0)) < e)$

$(\text{is } \_ = ?rhs)$

proof $-$

let $?g = \lambda x. \text{ if } x \in S \text{ then } f \ x \text{ else } 0$

have $f \text{ integrable\_on } S \longleftrightarrow$

$(\forall a \ b. ?g \text{ integrable\_on } \text{cbox } a \ b) \wedge$

$(\forall e > 0. \exists B > 0. \forall a \ b \ c \ d. \text{ball } 0 \ B \subseteq \text{cbox } a \ b \wedge \text{ball } 0 \ B \subseteq \text{cbox } c \ d \longrightarrow \text{norm}(\text{integral } (\text{cbox } a \ b) \ ?g - \text{integral } (\text{cbox } c \ d) \ ?g) < e)$

by $(\text{rule integrable\_alt})$

also have $\dots = ?rhs$

proof $-$

{ fix $e :: \text{real}$

assume $e: \bigwedge e. e > 0 \implies \exists B > 0. \forall a \ b \ c \ d. \text{ball } 0 \ B \subseteq \text{cbox } a \ b \wedge \text{cbox } a \ b \subseteq \text{cbox } c \ d \longrightarrow$

$\text{norm}(\text{integral } (\text{cbox } a \ b) \ ?g - \text{integral } (\text{cbox } c \ d) \ ?g)$

$< e$

and $e > 0$

obtain B **where** $B > 0$

and $B: \bigwedge a \ b \ c \ d. [\text{ball } 0 \ B \subseteq \text{cbox } a \ b; \text{cbox } a \ b \subseteq \text{cbox } c \ d] \implies$

$\text{norm}(\text{integral } (\text{cbox } a \ b) \ ?g - \text{integral } (\text{cbox } c \ d) \ ?g) < e/2$

using $\langle e > 0 \rangle e \text{ [of } e/2 \text{]} \text{ by force}$

have $\exists B > 0. \forall a \ b \ c \ d.$

$\text{ball } 0 \ B \subseteq \text{cbox } a \ b \wedge \text{ball } 0 \ B \subseteq \text{cbox } c \ d \longrightarrow$

$\text{norm}(\text{integral } (\text{cbox } a \ b) \ ?g - \text{integral } (\text{cbox } c \ d) \ ?g) < e$

proof $(\text{intro exI allI conjI impI})$

fix $a \ b \ c \ d :: 'a$

let $? \alpha = \sum_{i \in \text{Basis}} i \cdot a \cdot i \ (c \cdot i) *_{\mathbb{R}} i$

let $? \beta = \sum_{i \in \text{Basis}} i \cdot b \cdot i \ (d \cdot i) *_{\mathbb{R}} i$

show $\text{norm}(\text{integral } (\text{cbox } a \ b) \ ?g - \text{integral } (\text{cbox } c \ d) \ ?g) < e$

if $\text{ball } 0 \ B \subseteq \text{cbox } a \ b \wedge \text{ball } 0 \ B \subseteq \text{cbox } c \ d$

proof $-$

have $B': \text{norm}(\text{integral } (\text{cbox } a \ b \cap \text{cbox } c \ d) \ ?g - \text{integral } (\text{cbox } x \ y) \ ?g) < e/2$

if $\text{cbox } a \ b \cap \text{cbox } c \ d \subseteq \text{cbox } x \ y$ **for** $x \ y$

using $B \text{ [of } ? \alpha \ ? \beta \ x \ y \text{]} \text{ ball that by (simp add: Int\_interval [symmetric])}$

show $?thesis$

using $B' \text{ [of } a \ b \text{]} B' \text{ [of } c \ d \text{]} \text{ norm\_triangle\_half\_r by blast}$

```

      qed
    qed (use <B > 0> in auto)}
  then show ?thesis
    by force
  qed
  finally show ?thesis .
qed

```

```

lemma integrable_on_subcbx:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes intf: f integrable_on S
  and sub: cbx a b  $\subseteq$  S
  shows f integrable_on cbx a b
proof -
  have ( $\lambda x$ . if  $x \in S$  then f x else 0) integrable_on cbx a b
    by (simp add: intf integrable_altD(1))
  then show ?thesis
    by (metis (mono_tags) sub integrable_restrict_Int le_inf_iff order_refl sub-
    set_antisym)
qed

```

6.15.37 A straddling criterion for integrability

```

lemma integrable_straddle_interval:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  real
  assumes  $\bigwedge e. e > 0 \implies \exists g \ h \ i \ j. (g \text{ has\_integral } i) (cbx \ a \ b) \wedge (h \text{ has\_integral } j) (cbx \ a \ b) \wedge$ 
 $|i - j| < e \wedge (\forall x \in cbx \ a \ b. (g \ x) \leq f \ x \wedge f \ x \leq h \ x)$ 
  shows f integrable_on cbx a b
proof -
  have  $\exists d. \text{gauge } d \wedge$ 
 $(\forall p1 \ p2. p1 \text{ tagged\_division\_of } cbx \ a \ b \wedge d \text{ fine } p1 \wedge$ 
 $p2 \text{ tagged\_division\_of } cbx \ a \ b \wedge d \text{ fine } p2 \implies$ 
 $|(\sum (x,K) \in p1. \text{content } K *_{\mathbb{R}} f \ x) - (\sum (x,K) \in p2. \text{content } K *_{\mathbb{R}}$ 
 $f \ x)| < e)$ 
    if  $e > 0$  for e
  proof -
    have  $e/3 > 0$ 
      using that by auto
    then obtain g h i j where  $ij: |i - j| < e/3$ 
      and  $(g \text{ has\_integral } i) (cbx \ a \ b)$ 
      and  $(h \text{ has\_integral } j) (cbx \ a \ b)$ 
      and fgh:  $\bigwedge x. x \in cbx \ a \ b \implies g \ x \leq f \ x \wedge f \ x \leq h \ x$ 
      using assms real_norm_def by metis
    then obtain d1 d2 where gauge d1 gauge d2
      and d1:  $\bigwedge p. \llbracket p \text{ tagged\_division\_of } cbx \ a \ b; d1 \text{ fine } p \rrbracket \implies$ 
 $|(\sum (x,K) \in p. \text{content } K *_{\mathbb{R}} g \ x) - i| < e/3$ 
      and d2:  $\bigwedge p. \llbracket p \text{ tagged\_division\_of } cbx \ a \ b; d2 \text{ fine } p \rrbracket \implies$ 
 $|(\sum (x,K) \in p. \text{content } K *_{\mathbb{R}} h \ x) - j| < e/3$ 

```

```

    by (metis e has_integral_real_norm_def)
  have |( $\sum (x,K) \in p1. \text{content } K *_R f x$ ) - ( $\sum (x,K) \in p2. \text{content } K *_R f x$ )|
  < e
    if p1: p1 tagged_division_of_cbox a b and 11: d1 fine p1 and 21: d2 fine p1
    and p2: p2 tagged_division_of_cbox a b and 12: d1 fine p2 and 22: d2 fine
p2 for p1 p2
  proof -
    have *:  $\bigwedge g1\ g2\ h1\ h2\ f1\ f2. \llbracket |g2 - i| < e/3; |g1 - i| < e/3; |h2 - j| < e/3; |h1 - j| < e/3; \\ g1 - h2 \leq f1 - f2; f1 - f2 \leq h1 - g2 \rrbracket \\ \implies |f1 - f2| < e$ 
    using  $\langle e > 0 \rangle$  ij by arith
    have 0: ( $\sum (x, k) \in p1. \text{content } k *_R f x$ ) - ( $\sum (x, k) \in p1. \text{content } k *_R g x$ )
  ≥ 0
      0 ≤ ( $\sum (x, k) \in p2. \text{content } k *_R h x$ ) - ( $\sum (x, k) \in p2. \text{content } k *_R f x$ )
      ( $\sum (x, k) \in p2. \text{content } k *_R f x$ ) - ( $\sum (x, k) \in p2. \text{content } k *_R g x$ ) ≥ 0
      0 ≤ ( $\sum (x, k) \in p1. \text{content } k *_R h x$ ) - ( $\sum (x, k) \in p1. \text{content } k *_R f x$ )
    unfolding sum_subtractf[symmetric]
    apply (auto intro!: sum_nonneg)
    apply (meson fgh measure_nonneg mult_left_mono tag_in_interval that
sum_nonneg)+
    done
  show ?thesis
  proof (rule *)
    show |( $\sum (x,K) \in p2. \text{content } K *_R g x$ ) - i| < e/3
      by (rule d1[OF p2 12])
    show |( $\sum (x,K) \in p1. \text{content } K *_R g x$ ) - i| < e/3
      by (rule d1[OF p1 11])
    show |( $\sum (x,K) \in p2. \text{content } K *_R h x$ ) - j| < e/3
      by (rule d2[OF p2 22])
    show |( $\sum (x,K) \in p1. \text{content } K *_R h x$ ) - j| < e/3
      by (rule d2[OF p1 21])
    qed (use 0 in auto)
  qed
  then show ?thesis
    by (rule_tac x= $\lambda x. d1\ x \cap d2\ x$  in exI)
      (auto simp: fine_Int intro:  $\langle \text{gauge } d1 \rangle \langle \text{gauge } d2 \rangle d1\ d2$ )
  qed
  then show ?thesis
    by (simp add: integrable_Cauchy)
  qed

lemma integrable_straddle:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  real
  assumes  $\bigwedge e. e > 0 \implies \exists g\ h\ i\ j. (g \text{ has\_integral } i) \wedge (h \text{ has\_integral } j) \wedge \\ |i - j| < e \wedge (\forall x \in s. g\ x \leq f\ x \wedge f\ x \leq h\ x)$ 
  shows f integrable_on s
proof -
  let ?fs = ( $\lambda x. \text{if } x \in s \text{ then } f\ x \text{ else } 0$ )

```

```

have ?fs integrable_on cbox a b for a b
proof (rule integrable_straddle_interval)
  fix e::real
  assume e > 0
  then have *: e/4 > 0
  by auto
  with assms obtain g h i j where g: (g has_integral i) s and h: (h has_integral
j) s
    and ij: |i - j| < e/4
    and fgh:  $\bigwedge x. x \in s \implies g\ x \leq f\ x \wedge f\ x \leq h\ x$ 
  by metis
  let ?gs = ( $\lambda x. \text{if } x \in s \text{ then } g\ x \text{ else } 0$ )
  let ?hs = ( $\lambda x. \text{if } x \in s \text{ then } h\ x \text{ else } 0$ )
  obtain Bg where Bg:  $\bigwedge a\ b. \text{ball } 0\ Bg \subseteq \text{cbox } a\ b \implies |\text{integral } (\text{cbox } a\ b)\ ?gs - i| < e/4$ 
    and int_g:  $\bigwedge a\ b. ?gs \text{ integrable\_on } \text{cbox } a\ b$ 
  using g * unfolding has_integral_alt' real_norm_def by meson
  obtain Bh where
    Bh:  $\bigwedge a\ b. \text{ball } 0\ Bh \subseteq \text{cbox } a\ b \implies |\text{integral } (\text{cbox } a\ b)\ ?hs - j| < e/4$ 
    and int_h:  $\bigwedge a\ b. ?hs \text{ integrable\_on } \text{cbox } a\ b$ 
  using h * unfolding has_integral_alt' real_norm_def by meson
  define c where c = ( $\sum i \in \text{Basis}. \min (a \cdot i) (- (\max Bg\ Bh)) *_R i$ )
  define d where d = ( $\sum i \in \text{Basis}. \max (b \cdot i) (\max Bg\ Bh) *_R i$ )
  have  $\llbracket \text{norm } (0 - x) < Bg; i \in \text{Basis} \rrbracket \implies c \cdot i \leq x \cdot i \wedge x \cdot i \leq d \cdot i \text{ for } x\ i$ 
  using Basis_le_norm[of i x] unfolding c_def d_def by auto
  then have ballBg:  $\text{ball } 0\ Bg \subseteq \text{cbox } c\ d$ 
  by (auto simp: mem_box dist_norm)
  have  $\llbracket \text{norm } (0 - x) < Bh; i \in \text{Basis} \rrbracket \implies c \cdot i \leq x \cdot i \wedge x \cdot i \leq d \cdot i \text{ for } x\ i$ 
  using Basis_le_norm[of i x] unfolding c_def d_def by auto
  then have ballBh:  $\text{ball } 0\ Bh \subseteq \text{cbox } c\ d$ 
  by (auto simp: mem_box dist_norm)
  have ab_cd:  $\text{cbox } a\ b \subseteq \text{cbox } c\ d$ 
  by (auto simp: c_def d_def subset_box_imp)
  have **:  $\bigwedge ch\ cg\ ag\ ah::\text{real}. \llbracket |ah - ag| \leq |ch - cg|; |cg - i| < e/4; |ch - j| < e/4 \rrbracket$ 
     $\implies |ag - ah| < e$ 
  using ij by arith
  show  $\exists g\ h\ i\ j. (g \text{ has\_integral } i) (\text{cbox } a\ b) \wedge (h \text{ has\_integral } j) (\text{cbox } a\ b) \wedge$ 
     $|i - j| < e \wedge$ 
     $(\forall x \in \text{cbox } a\ b. g\ x \leq (\text{if } x \in s \text{ then } f\ x \text{ else } 0) \wedge$ 
     $(\text{if } x \in s \text{ then } f\ x \text{ else } 0) \leq h\ x)$ 
  proof (intro exI ballI conjI)
    have eq:  $\bigwedge x\ f\ g. (\text{if } x \in s \text{ then } f\ x \text{ else } 0) - (\text{if } x \in s \text{ then } g\ x \text{ else } 0) =$ 
     $(\text{if } x \in s \text{ then } f\ x - g\ x \text{ else } (0::\text{real}))$ 
    by auto
    have int_hg: ( $\lambda x. \text{if } x \in s \text{ then } h\ x - g\ x \text{ else } 0$ ) integrable_on cbox a b
    ( $\lambda x. \text{if } x \in s \text{ then } h\ x - g\ x \text{ else } 0$ ) integrable_on cbox c d
    by (metis (no_types) integrable_diff g h has_integral_integrable integrable_altD(1))+

```

```

show (?gs has_integral integral (cbox a b) ?gs) (cbox a b)
  (?hs has_integral integral (cbox a b) ?hs) (cbox a b)
  by (intro integrable_integral int_g int_h)+
then have integral (cbox a b) ?gs ≤ integral (cbox a b) ?hs
  using fgh by (force intro: has_integral_le)
then have 0 ≤ integral (cbox a b) ?hs - integral (cbox a b) ?gs
  by simp
then have |integral (cbox a b) ?hs - integral (cbox a b) ?gs|
  ≤ |integral (cbox c d) ?hs - integral (cbox c d) ?gs|
  apply (simp add: integral_diff [symmetric] int_g int_h)
  apply (subst abs_of_nonneg[OF integral_nonneg[OF integrable_diff, OF
int_h int_g]])
  using fgh apply (force simp: eq intro!: integral_subset_le [OF ab_cd
int_hg])+)
done
then show |integral (cbox a b) ?gs - integral (cbox a b) ?hs| < e
  using ** Bg ballBg Bh ballBh by blast
show  $\bigwedge x. x \in \text{cbox } a \ b \implies ?gs \ x \leq ?fs \ x \bigwedge x. x \in \text{cbox } a \ b \implies ?fs \ x \leq ?hs \ x$ 
  using fgh by auto
qed
qed
then have int_f: ?fs integrable_on cbox a b for a b
  by simp
have  $\exists B > 0. \forall a \ b \ c \ d.$ 
  ball 0 B  $\subseteq \text{cbox } a \ b \wedge \text{ball } 0 \ B \subseteq \text{cbox } c \ d \implies$ 
  abs (integral (cbox a b) ?fs - integral (cbox c d) ?fs) < e
  if 0 < e for e
proof -
  have *:  $e/3 > 0$ 
  using that by auto
  with assms obtain g h i j where g: (g has_integral i) s and h: (h has_integral
j) s
    and ij:  $|i - j| < e/3$ 
    and fgh:  $\bigwedge x. x \in s \implies g \ x \leq f \ x \wedge f \ x \leq h \ x$ 
  by metis
  let ?gs = ( $\lambda x. \text{if } x \in s \text{ then } g \ x \text{ else } 0$ )
  let ?hs = ( $\lambda x. \text{if } x \in s \text{ then } h \ x \text{ else } 0$ )
  obtain Bg where Bg > 0
    and Bg:  $\bigwedge a \ b. \text{ball } 0 \ Bg \subseteq \text{cbox } a \ b \implies |\text{integral (cbox } a \ b) ?gs - i| <$ 
e/3
    and int_g:  $\bigwedge a \ b. ?gs \text{ integrable\_on } \text{cbox } a \ b$ 
  using g * unfolding has_integral_alt' real_norm_def by meson
  obtain Bh where Bh > 0
    and Bh:  $\bigwedge a \ b. \text{ball } 0 \ Bh \subseteq \text{cbox } a \ b \implies |\text{integral (cbox } a \ b) ?hs - j|$ 
< e/3
    and int_h:  $\bigwedge a \ b. ?hs \text{ integrable\_on } \text{cbox } a \ b$ 
  using h * unfolding has_integral_alt' real_norm_def by meson
  { fix a b c d :: 'n
    assume as: ball 0 (max Bg Bh)  $\subseteq \text{cbox } a \ b \text{ ball } 0 \ (\max Bg Bh) \subseteq \text{cbox } c \ d$ 

```

```

    have **: ball 0 Bg  $\subseteq$  ball (0::'n) (max Bg Bh) ball 0 Bh  $\subseteq$  ball (0::'n) (max
Bg Bh)
    by auto
    have *:  $\bigwedge ga\ gc\ ha\ hc\ fa\ fc. [|ga - i| < e/3; |gc - i| < e/3; |ha - j| < e/3;$ 
 $|hc - j| < e/3; ga \leq fa; fa \leq ha; gc \leq fc; fc \leq hc] \implies$ 
 $|fa - fc| < e$ 
    using ij by arith
    have abs (integral (cbox a b) ( $\lambda x. \text{if } x \in s \text{ then } f\ x \text{ else } 0$ ) - integral (cbox c
d)
( $\lambda x. \text{if } x \in s \text{ then } f\ x \text{ else } 0$ )) < e
    proof (rule *)
    show |integral (cbox a b) ?gs - i| < e/3
    using ** Bg as by blast
    show |integral (cbox c d) ?gs - i| < e/3
    using ** Bg as by blast
    show |integral (cbox a b) ?hs - j| < e/3
    using ** Bh as by blast
    show |integral (cbox c d) ?hs - j| < e/3
    using ** Bh as by blast
    qed (use int_f int_g int_h fgh in <simp_all add: integral_le>)
  }
  then show ?thesis
  apply (rule_tac x=max Bg Bh in exI)
  using <Bg > 0> by auto
  qed
  then show ?thesis
  unfolding integrable_alt[of f] real_norm_def by (blast intro: int_f)
  qed

```

6.15.38 Adding integrals over several sets

lemma has_integral_Un:

fixes $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$

assumes $f: (f \text{ has\_integral } i) \ S \ (f \text{ has\_integral } j) \ T$

and $neg: \text{negligible } (S \cap T)$

shows $(f \text{ has\_integral } (i + j)) \ (S \cup T)$

unfolding has_integral_restrict_UNIV[symmetric, of f]

proof (rule has_integral_spike[OF neg])

let $?f = \lambda x. (\text{if } x \in S \text{ then } f\ x \text{ else } 0) + (\text{if } x \in T \text{ then } f\ x \text{ else } 0)$

show $(?f \text{ has\_integral } i + j) \ UNIV$

by (simp add: f has_integral_add)

qed auto

lemma integral_Un [simp]:

fixes $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$

assumes $f \text{ integrable\_on } S \ f \text{ integrable\_on } T \ \text{negligible } (S \cap T)$

shows $\text{integral } (S \cup T) \ f = \text{integral } S \ f + \text{integral } T \ f$

by (simp add: has_integral_Un assms integrable_integral integral_unique)

```

lemma integrable_Un:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::banach
  assumes negligible (A  $\cap$  B) f integrable_on A f integrable_on B
  shows f integrable_on (A  $\cup$  B)
proof -
  from assms obtain y z where (f has_integral y) A (f has_integral z) B
  by (auto simp: integrable_on_def)
  from has_integral_Un[OF this assms(1)] show ?thesis by (auto simp: integrable_on_def)
qed

lemma integrable_Un':
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::banach
  assumes f integrable_on A f integrable_on B negligible (A  $\cap$  B) C = A  $\cup$  B
  shows f integrable_on C
  using integrable_Un[of A B f] assms by simp

lemma has_integral_Union:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes  $\mathcal{T}$ : finite  $\mathcal{T}$ 
  and int:  $\bigwedge S. S \in \mathcal{T} \implies (f \text{ has\_integral } (i \ S)) \ S$ 
  and neg: pairwise ( $\lambda S \ S'. \text{negligible } (S \cap S') \ \mathcal{T}$ )
  shows (f has_integral (sum i  $\mathcal{T}$ )) ( $\bigcup \mathcal{T}$ )
proof -
  let  $\mathcal{U} = ((\lambda(a,b). a \cap b) \cdot \{(a,b). a \in \mathcal{T} \wedge b \in \{y. y \in \mathcal{T} \wedge a \neq y\}\})$ 
  have (( $\lambda x. \text{if } x \in \bigcup \mathcal{T} \text{ then } f \ x \text{ else } 0$ ) has_integral sum i  $\mathcal{T}$ ) UNIV
  proof (rule has_integral_spike)
    show negligible ( $\bigcup \mathcal{U}$ )
  proof (rule negligible_Union)
    have finite ( $\mathcal{T} \times \mathcal{T}$ )
    by (simp add:  $\mathcal{T}$ )
    moreover have  $\{(a, b). a \in \mathcal{T} \wedge b \in \{y \in \mathcal{T}. a \neq y\}\} \subseteq \mathcal{T} \times \mathcal{T}$ 
    by auto
    ultimately show finite  $\mathcal{U}$ 
    by (blast intro: finite_subset[of _  $\mathcal{T} \times \mathcal{T}$ ])
    show  $\bigwedge t. t \in \mathcal{U} \implies \text{negligible } t$ 
    using neg unfolding pairwise_def by auto
  qed
  qed
next
  show (if  $x \in \bigcup \mathcal{T}$  then  $f \ x$  else 0) = ( $\sum A \in \mathcal{T}. \text{if } x \in A \text{ then } f \ x \text{ else } 0$ )
  if  $x \in \text{UNIV} - (\bigcup \mathcal{U})$  for  $x$ 
  proof clarsimp
    fix S assume  $S \in \mathcal{T} \ x \in S$ 
    moreover then have  $\forall b \in \mathcal{T}. x \in b \longleftrightarrow b = S$ 
    using that by blast
    ultimately show  $f \ x = (\sum A \in \mathcal{T}. \text{if } x \in A \text{ then } f \ x \text{ else } 0)$ 
    by (simp add: sum.delta[OF  $\mathcal{T}$ ])
  qed
next

```

1900

```

show (( $\lambda x. \sum A \in \mathcal{T}. \text{if } x \in A \text{ then } f \ x \text{ else } 0$ ) has_integral ( $\sum A \in \mathcal{T}. i \ A$ ))
UNIV
using int by (simp add: has_integral_restrict_UNIV has_integral_sum [OF
 $\mathcal{T}$ ])
qed
then show ?thesis
using has_integral_restrict_UNIV by blast
qed

```

In particular adding integrals over a division, maybe not of an interval.

```

lemma has_integral_combine_division:
fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$ 
assumes  $\mathcal{D}$  division_of  $S$ 
and  $\bigwedge k. k \in \mathcal{D} \implies (f \text{ has\_integral } (i \ k)) \ k$ 
shows ( $f \text{ has\_integral } (\text{sum } i \ \mathcal{D})) \ S$ 
proof –
note  $\mathcal{D} = \text{division\_of } D[\text{OF } \text{assms}(1)]$ 
have neg: negligible ( $S \cap s'$ ) if  $S \in \mathcal{D} \ s' \in \mathcal{D} \ S \neq s'$  for  $S \ s'$ 
proof –
obtain  $a \ c \ b \ \mathcal{D}$  where obt:  $S = \text{cbox } a \ b \ s' = \text{cbox } c \ \mathcal{D}$ 
by (meson  $\langle S \in \mathcal{D} \rangle \langle s' \in \mathcal{D} \rangle \mathcal{D}(4)$ )
from  $\mathcal{D}(5)[\text{OF } \text{that}]$  show ?thesis
unfolding obt interior_cbox
by (metis (no_types, lifting) Diff_empty Int_interval box_Int_box negligible_frontier_interval)
qed
show ?thesis
unfolding  $\mathcal{D}(6)[\text{symmetric}]$ 
by (auto intro: D neg assms has_integral_Union pairwiseI)
qed

lemma integral_combine_division_bottomup:
fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$ 
assumes  $\mathcal{D}$  division_of  $S \ \bigwedge k. k \in \mathcal{D} \implies f \text{ integrable\_on } k$ 
shows  $\text{integral } S \ f = \text{sum } (\lambda i. \text{integral } i \ f) \ \mathcal{D}$ 
by (meson assms integral_unique has_integral_combine_division has_integral_integrable_integral)

```

```

lemma has_integral_combine_division_topdown:
fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$ 
assumes  $f: f \text{ integrable\_on } S$ 
and  $\mathcal{D}: \mathcal{D} \text{ division\_of } K$ 
and  $K \subseteq S$ 
shows ( $f \text{ has\_integral } (\text{sum } (\lambda i. \text{integral } i \ f) \ \mathcal{D})) \ K$ 
proof –
have  $f \text{ integrable\_on } L$  if  $L \in \mathcal{D}$  for  $L$ 
proof –
have  $L \subseteq S$ 
using  $\langle K \subseteq S \rangle \ \mathcal{D}$  that by blast
then show  $f \text{ integrable\_on } L$ 

```



```

    using that by (metis (no_types) f  $\mathcal{D}$  division_ofD(4) integrable_on_subcbx)
  qed
  then show ?thesis
    by (meson  $\mathcal{D}$  has_integral_combine_division has_integral_integrable_integral)
  qed

```

```

lemma integral_combine_division_topdown:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes f integrable_on S
    and  $\mathcal{D}$  division_of S
  shows integral S f = sum ( $\lambda i.$  integral i f)  $\mathcal{D}$ 
  using assms has_integral_combine_division_topdown by blast

```

```

lemma integrable_combine_division:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes  $\mathcal{D}$ :  $\mathcal{D}$  division_of S
    and f:  $\bigwedge i. i \in \mathcal{D} \implies f$  integrable_on i
  shows f integrable_on S
  using f unfolding integrable_on_def by (metis has_integral_combine_division[OF  $\mathcal{D}$ ])

```

```

lemma integrable_on_subdivision:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes  $\mathcal{D}$ :  $\mathcal{D}$  division_of i
    and f: f integrable_on S
    and  $i \subseteq S$ 
  shows f integrable_on i
proof -
  have f integrable_on i if  $i \in \mathcal{D}$  for i
proof -
  have  $i \subseteq S$ 
    using assms that by auto
  then show f integrable_on i
    using that by (metis (no_types)  $\mathcal{D}$  f division_ofD(4) integrable_on_subcbx)
  qed
  then show ?thesis
    using  $\mathcal{D}$  integrable_combine_division by blast
  qed

```

6.15.39 Also tagged divisions

```

lemma has_integral_combine_tagged_division:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes p tagged_division_of S
    and  $\bigwedge x k. (x,k) \in p \implies (f$  has_integral (i k)) k
  shows (f has_integral ( $\sum (x,k) \in p. i k$ )) S
proof -
  have *: (f has_integral ( $\sum k \in \text{snd} p. integral k f$ )) S
proof -

```

```

    have snd 'p division_of S
      by (simp add: assms(1) division_of_tagged_division)
    with assms show ?thesis
      by (metis (mono_tags, lifting) has_integral_combine_division has_integral_integrable_integral
        imageE prod.collapse)
    qed
    also have  $(\sum k \in \text{snd } p. \text{integral } k f) = (\sum (x, k) \in p. \text{integral } k f)$ 
      by (intro sum.over_tagged_division_lemma[OF assms(1), symmetric] integral_null)
      (simp add: content_eq_0_interior)
    finally show ?thesis
      using assms by (auto simp add: has_integral_iff intro!: sum.cong)
  qed

```

```

lemma integral_combine_tagged_division_bottomup:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes p: p tagged_division_of (cbox a b)
    and f:  $\bigwedge x k. (x, k) \in p \implies f \text{ integrable\_on } k$ 
  shows integral (cbox a b) f = sum  $(\lambda(x, k). \text{integral } k f)$  p
    by (simp add: has_integral_combine_tagged_division[OF p] integral_unique f
      integrable_integral)

```

```

lemma has_integral_combine_tagged_division_topdown:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes f: f integrable_on cbox a b
    and p: p tagged_division_of (cbox a b)
  shows (f has_integral (sum  $(\lambda(x, K). \text{integral } K f)$  p)) (cbox a b)
proof -
  have (f has_integral integral K f) K if  $(x, K) \in p$  for x K
    by (metis assms integrable_integral integrable_on_subcbox tagged_division_ofD(3,4)
      that)
  then show ?thesis
    by (simp add: has_integral_combine_tagged_division p)
qed

```

```

lemma integral_combine_tagged_division_topdown:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes f integrable_on cbox a b
    and p tagged_division_of (cbox a b)
  shows integral (cbox a b) f = sum  $(\lambda(x, k). \text{integral } k f)$  p
  using assms by (auto intro: integral_unique [OF has_integral_combine_tagged_division_topdown])

```

6.15.40 Henstock's lemma

```

lemma Henstock_lemma_part1:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes intf: f integrable_on cbox a b
    and e > 0
    and gauge d

```

```

    and less_e:  $\bigwedge p. \llbracket p \text{ tagged\_division\_of } (cbox\ a\ b); d \text{ fine } p \rrbracket \implies$ 
      norm (sum ( $\lambda(x,K). \text{content } K *_R f\ x$ )  $p$  - integral( $cbox\ a\ b$ )  $f$ )
    < e
    and p:  $p \text{ tagged\_partial\_division\_of } (cbox\ a\ b) \text{ d fine } p$ 
    shows norm (sum ( $\lambda(x,K). \text{content } K *_R f\ x$  - integral  $K\ f$ )  $p$ )  $\leq e$  (is ?lhs  $\leq$ 
    e)
  proof (rule field_le_epsilon)
    fix k :: real
    assume k > 0
    let ?SUM =  $\lambda p. (\sum (x,K) \in p. \text{content } K *_R f\ x)$ 
    note p' = tagged_partial_division_ofD[OF p(1)]
    have  $\bigcup (snd\ 'p) \subseteq cbox\ a\ b$ 
    using p'(3) by fastforce
    then obtain q where q:  $snd\ 'p \subseteq q$  and qdiv:  $q \text{ division\_of } cbox\ a\ b$ 
    by (meson p(1) partial_division_extend_interval partial_division_of_tagged_division)
    note q' = division_ofD[OF qdiv]
    define r where  $r = q - snd\ 'p$ 
    have  $snd\ 'p \cap r = \{\}$ 
    unfolding r_def by auto
    have finite r
    using q' unfolding r_def by auto
    have  $\exists p. p \text{ tagged\_division\_of } i \wedge d \text{ fine } p \wedge$ 
      norm (?SUM  $p$  - integral  $i\ f$ ) <  $k / (\text{real } (card\ r) + 1)$ 
    if  $i \in r$  for i
  proof -
    have gt0:  $k / (\text{real } (card\ r) + 1) > 0$  using  $\langle k > 0 \rangle$  by simp
    have i:  $i \in q$ 
    using that unfolding r_def by auto
    then obtain u v where uv:  $i = cbox\ u\ v$ 
    using q'(4) by blast
    then have  $cbox\ u\ v \subseteq cbox\ a\ b$ 
    using i q'(2) by auto
    then have f integrable_on  $cbox\ u\ v$ 
    by (rule integrable_subinterval[OF intf])
    with integrable_integral[OF this, unfolded has_integral[of f]]
    obtain dd where gauge dd and dd:
       $\bigwedge \mathcal{D}. \llbracket \mathcal{D} \text{ tagged\_division\_of } cbox\ u\ v; dd \text{ fine } \mathcal{D} \rrbracket \implies$ 
      norm (?SUM  $\mathcal{D}$  - integral ( $cbox\ u\ v$ )  $f$ ) <  $k / (\text{real } (card\ r) + 1)$ 
    using gt0 by auto
    with gauge_Int[OF  $\langle \text{gauge } d \rangle \langle \text{gauge } dd \rangle$ ]
    obtain qq where qq:  $qq \text{ tagged\_division\_of } cbox\ u\ v (\lambda x. d\ x \cap dd\ x) \text{ fine } qq$ 
    using fine_division_exists by blast
    with dd[of qq] show ?thesis
    by (auto simp: fine_Int uv)
  qed
  then obtain qq where qq:  $\bigwedge i. i \in r \implies qq\ i \text{ tagged\_division\_of } i \wedge$ 
    d fine qq i  $\wedge$  norm (?SUM (qq i) - integral i f) <  $k / (\text{real } (card\ r) + 1)$ 
  by metis

```

```

let ?p = p ∪ ⋃(qq ' r)
have norm (?SUM ?p - integral (cbox a b) f) < e
proof (rule less_e)
  show d fine ?p
  by (metis (mono_tags, opaque_lifting) qq fine_Un fine_Union imageE p(2))
note ptag = tagged_partial_division_of_Union_self[OF p(1)]
have p ∪ ⋃(qq ' r) tagged_division_of ⋃(snd ' p) ∪ ⋃ r
proof (rule tagged_division_Un[OF ptag tagged_division_Union [OF ⟨finite
r⟩]])
  show ∧i. i ∈ r ⇒ qq i tagged_division_of i
  using qq by auto
  show ∧i1 i2. [i1 ∈ r; i2 ∈ r; i1 ≠ i2] ⇒ interior i1 ∩ interior i2 = {}
  by (simp add: q'(5) r_def)
  show interior (⋃(snd ' p)) ∩ interior (⋃ r) = {}
  proof (rule Int_interior_Union_intervals [OF ⟨finite r⟩])
    show open (interior (⋃(snd ' p)))
    by blast
    show ∧T. T ∈ r ⇒ ∃ a b. T = cbox a b
    by (simp add: q'(4) r_def)
    have interior T ∩ interior (⋃(snd ' p)) = {} if T ∈ r for T
    proof (rule Int_interior_Union_intervals)
      show ∧U. U ∈ snd ' p ⇒ ∃ a b. U = cbox a b
      using q q'(4) by blast
      show ∧U. U ∈ snd ' p ⇒ interior T ∩ interior U = {}
      by (metis DiffE q q'(5) r_def subsetD that)
    qed (use p' in auto)
    then show ∧T. T ∈ r ⇒ interior (⋃(snd ' p)) ∩ interior T = {}
    by (metis Int_commute)
  qed
  qed
  moreover have ⋃(snd ' p) ∪ ⋃ r = cbox a b and {qq i | i. i ∈ r} = qq ' r
  using qdiv q unfolding Union_Un_distrib[symmetric] r_def by auto
  ultimately show ?p tagged_division_of (cbox a b)
  by fastforce
qed
then have norm (?SUM p + (?SUM (⋃(qq ' r))) - integral (cbox a b) f) < e
proof (subst sum.union_inter_neutral[symmetric, OF ⟨finite p⟩], safe)
  show content L *R f x = 0 if (x, L) ∈ p (x, L) ∈ qq K K ∈ r for x K L
  proof -
    obtain u v where uv: L = cbox u v
    using ⟨(x,L) ∈ p⟩ p'(4) by blast
    have L ⊆ K
    using qq[OF that(3)] tagged_division_ofD(3) ⟨(x,L) ∈ qq K⟩ by metis
    have L ∈ snd ' p
    using ⟨(x,L) ∈ p⟩ image_iff by fastforce
    then have L ∈ q K ∈ q L ≠ K
    using that(1,3) q(1) unfolding r_def by auto
    with q'(5) have interior L = {}
    using interior_mono[OF ⟨L ⊆ K⟩] by blast
  qed

```

```

    then show content  $L *_R f x = 0$ 
      unfolding uv content_eq_0_interior[symmetric] by auto
    qed
    show finite  $(\bigcup (qq \text{ ' } r))$ 
      by (meson finite_UN qq ⟨finite r⟩ tagged_division_of_finite)
    qed
    moreover have content  $M *_R f x = 0$ 
      if  $x: (x, M) \in qq \ K \ (x, M) \in qq \ L$  and  $KL: qq \ K \neq qq \ L$  and  $r: K \in r \ L \in r$ 
      for  $x \ M \ K \ L$ 
    proof -
      note kl = tagged_division_ofD(3,4)[OF qq[THEN conjunct1]]
      obtain  $u \ v$  where  $uv: M = cbox \ u \ v$ 
        using ⟨ $(x, M) \in qq \ L \rangle \langle L \in r \rangle$  kl(2) by blast
      have empty: interior  $(K \cap L) = \{\}$ 
        by (metis DiffD1 interior_Int q'(5) r_def KL r)
      have interior  $M = \{\}$ 
        by (metis (no_types, lifting) Int_assoc empty inf.absorb_iff2 interior_Int
kl(1) subset_empty x r)
      then show content  $M *_R f x = 0$ 
        unfolding uv content_eq_0_interior[symmetric]
        by auto
    qed
    ultimately have norm  $(?SUM \ p + \text{sum } ?SUM \ (qq \text{ ' } r) - \text{integral } (cbox \ a \ b) \ f)$ 
<  $e$ 
      apply (subst (asm) sum.Union_comp)
      using qq by (force simp: split_paired_all)+
    moreover have content  $M *_R f x = 0$ 
      if  $K \in r \ L \in r \ K \neq L \ qq \ K = qq \ L \ (x, M) \in qq \ K$  for  $K \ L \ x \ M$ 
      using tagged_division_ofD(6) qq that by (metis (no_types, lifting))
    ultimately have less_e: norm  $(?SUM \ p + \text{sum } (?SUM \circ qq) \ r - \text{integral } (cbox$ 
 $a \ b) \ f) < e$ 
      proof (subst (asm) sum.reindex_nontrivial [OF ⟨finite r⟩])
        qed (auto simp: split_paired_all sum.neutral)
      have norm_le: norm  $(cp - ip) \leq e + k$ 
        if norm  $((cp + cr) - i) < e$  norm  $(cr - ir) < k$   $ip + ir = i$ 
        for  $ir \ ip \ i \ cr \ cp::'a$ 
      proof -
        from that show ?thesis
          using norm_triangle_le[of  $cp + cr - i - (cr - ir)$ ]
          unfolding that(3)[symmetric] norm_minus_cancel
          by (auto simp add: algebra_simps)
      qed

    have ?lhs = norm  $(?SUM \ p - (\sum_{(x, k) \in p}. \text{integral } k \ f))$ 
      unfolding split_def sum_subtractf ..
    also have  $\dots \leq e + k$ 
    proof (rule norm_le[OF less_e])
      have lessk:  $k * \text{real } (\text{card } r) / (1 + \text{real } (\text{card } r)) < k$ 
        using ⟨ $k > 0$ ⟩ by (auto simp add: field_simps)

```

```

    have norm (sum (?SUM ∘ qq) r - (∑ k∈r. integral k f)) ≤ (∑ x∈r. k / (real
(card r) + 1))
    unfolding sum_subtractf[symmetric] by (force dest: qq intro!: sum_norm_le)
    also have ... < k
    by (simp add: lessk add.commute mult.commute)
    finally show norm (sum (?SUM ∘ qq) r - (∑ k∈r. integral k f)) < k .
next
from q(1) have [simp]: snd ' p ∪ q = q by auto
have integral l f = 0
  if inp: (x, l) ∈ p (y, m) ∈ p and ne: (x, l) ≠ (y, m) and l = m for x l y m
proof -
  obtain u v where uv: l = cbox u v
  using inp p'(4) by blast
  have content (cbox u v) = 0
  unfolding content_eq_0_interior using that p(1) uv
  by (auto dest: tagged_partial_division_ofD)
  then show ?thesis
  using uv by blast
qed
then have (∑ (x, K)∈p. integral K f) = (∑ K∈snd ' p. integral K f)
  apply (subst sum.reindex_nontrivial [OF ⟨finite p⟩])
  unfolding split_paired_all split_def by auto
then show (∑ (x, k)∈p. integral k f) + (∑ k∈r. integral k f) = integral (cbox
a b) f
  unfolding integral_combine_division_topdown[OF intf qdiv] r_def
  using q'(1) p'(1) sum.union_disjoint [of snd ' p q - snd ' p, symmetric]
  by simp
qed
finally show ?lhs ≤ e + k .
qed

lemma Henstock_lemma_part2:
  fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
  assumes fed: f integrable_on cbox a b e > 0 gauge d
  and less_e: ⋀ D. [D tagged_division_of (cbox a b); d fine D] ⇒
    norm (sum (λ(x,k). content k *R f x) D - integral (cbox a b) f)
  < e
  and tag: p tagged_partial_division_of (cbox a b)
  and d fine p
  shows sum (λ(x,k). norm (content k *R f x - integral k f)) p ≤ 2 * real
(DIM('n)) * e
proof -
  have finite p
  using tag tagged_partial_division_ofD by blast
  then show ?thesis
  unfolding split_def
  proof (rule sum_norm_allsubsets_bound)
    fix Q
    assume Q: Q ⊆ p

```

```

then have fine: d fine Q
  by (simp add: ⟨d fine p⟩ fine_subset)
show norm (∑ x∈Q. content (snd x) *R f (fst x) - integral (snd x) f) ≤ e
  apply (rule Henstock_lemma_part1[OF fed_less_e, unfolded split_def])
  using Q tag tagged_partial_division_subset by (force simp add: fine)+
qed
qed

```

lemma *Henstock_lemma*:

fixes $f :: 'm::euclidean\_space \Rightarrow 'n::euclidean\_space$

assumes $\text{intf}: f \text{ integrable\_on } \text{cbox } a \ b$

and $e > 0$

obtains γ **where** $\text{gauge } \gamma$

and $\bigwedge p. \llbracket p \text{ tagged\_partial\_division\_of } (\text{cbox } a \ b); \gamma \text{ fine } p \rrbracket \implies$
 $\text{sum } (\lambda(x,k). \text{norm}(\text{content } k *_{\mathbb{R}} f \ x - \text{integral } k \ f)) \ p < e$

proof –

have $*$: $e/(2 * (\text{real } \text{DIM}('n) + 1)) > 0$ **using** $\langle e > 0 \rangle$ **by** *simp*

with *integrable_integral*[OF *intf*, *unfolded has_integral*]

obtain γ **where** $\text{gauge } \gamma$

and $\gamma: \bigwedge \mathcal{D}. \llbracket \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b; \gamma \text{ fine } \mathcal{D} \rrbracket \implies$
 $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f \ x) - \text{integral } (\text{cbox } a \ b) \ f)$
 $< e/(2 * (\text{real } \text{DIM}('n) + 1))$

by *metis*

show *thesis*

proof (*rule that* [OF $\langle \text{gauge } \gamma \rangle$])

fix p

assume p : $p \text{ tagged\_partial\_division\_of } \text{cbox } a \ b \ \gamma \text{ fine } p$

have $(\sum (x,K) \in p. \text{norm } (\text{content } K *_{\mathbb{R}} f \ x - \text{integral } K \ f))$
 $\leq 2 * \text{real } \text{DIM}('n) * (e/(2 * (\text{real } \text{DIM}('n) + 1)))$

using *Henstock_lemma_part2*[OF *intf* * $\langle \text{gauge } \gamma \rangle \ \gamma \ p$] **by** *metis*

also have $\dots < e$

using $\langle e > 0 \rangle$ **by** (*auto simp add: field_simps*)

finally

show $(\sum (x,K) \in p. \text{norm } (\text{content } K *_{\mathbb{R}} f \ x - \text{integral } K \ f)) < e$.

qed

qed

6.15.41 Monotone convergence (bounded interval first)

lemma *bounded_increasing_convergent*:

fixes $f :: \text{nat} \Rightarrow \text{real}$

shows $\llbracket \text{bounded } (\text{range } f); \bigwedge n. f \ n \leq f \ (\text{Suc } n) \rrbracket \implies \exists l. f \longrightarrow l$

using *Bseq_mono_convergent*[of *f*] *incseq_Suc_iff*[of *f*]

by (*auto simp: image_def Bseq_eq_bounded convergent_def incseq_def*)

lemma *monotone_convergence_interval*:

fixes $f :: \text{nat} \Rightarrow 'n::euclidean\_space \Rightarrow \text{real}$

assumes $\text{intf}: \bigwedge k. (f \ k) \text{ integrable\_on } \text{cbox } a \ b$

and $le: \bigwedge k \ x. x \in \text{cbox } a \ b \implies (f \ k \ x) \leq f \ (\text{Suc } k) \ x$

```

    and fg:  $\bigwedge x. x \in \text{cbox } a \ b \implies ((\lambda k. f \ k \ x) \longrightarrow g \ x) \text{ sequentially}$ 
    and bou: bounded (range ( $\lambda k. \text{integral } (\text{cbox } a \ b) (f \ k)$ ))
    shows  $g \text{ integrable\_on } \text{cbox } a \ b \wedge ((\lambda k. \text{integral } (\text{cbox } a \ b) (f \ k)) \longrightarrow \text{integral } (\text{cbox } a \ b) \ g) \text{ sequentially}$ 
  proof (cases content ( $\text{cbox } a \ b$ ) = 0)
    case True then show ?thesis
      by auto
  next
    case False
    have fg1:  $(f \ k \ x) \leq (g \ x)$  if  $x: x \in \text{cbox } a \ b$  for  $x \ k$ 
    proof -
      have  $\forall_F j \text{ in sequentially. } f \ k \ x \leq f \ j \ x$ 
      proof (rule eventually_sequentiallyI [of  $k$ ])
        show  $\bigwedge j. k \leq j \implies f \ k \ x \leq f \ j \ x$ 
        using  $le \ x$  by (force intro: transitive_stepwise_le)
      qed
      then show  $f \ k \ x \leq g \ x$ 
        using tendsto_lowerbound [OF fg] x trivial_limit_sequentially by blast
    qed
    have int_inc:  $\bigwedge n. \text{integral } (\text{cbox } a \ b) (f \ n) \leq \text{integral } (\text{cbox } a \ b) (f \ (\text{Suc } n))$ 
      by (metis integral_le intf le)
    then obtain  $i$  where  $i: (\lambda k. \text{integral } (\text{cbox } a \ b) (f \ k)) \longrightarrow i$ 
      using bounded_increasing_convergent bou by blast
    have  $\bigwedge k. \forall_F x \text{ in sequentially. } \text{integral } (\text{cbox } a \ b) (f \ k) \leq \text{integral } (\text{cbox } a \ b) (f \ x)$ 
      unfolding eventually_sequentially
      by (force intro: transitive_stepwise_le int_inc)
    then have  $i': \bigwedge k. (\text{integral } (\text{cbox } a \ b) (f \ k)) \leq i$ 
      using tendsto_le [OF trivial_limit_sequentially  $i$ ] by blast
    have ( $g \text{ has\_integral } i$ ) ( $\text{cbox } a \ b$ )
      unfolding has_integral real_norm_def
    proof clarify
      fix  $e::\text{real}$ 
      assume  $e: e > 0$ 
      have  $\bigwedge k. (\exists \gamma. \text{gauge } \gamma \wedge (\forall \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } (\text{cbox } a \ b) \wedge \gamma \text{ fine } \mathcal{D} \longrightarrow$ 
 $\text{abs } ((\sum_{(x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f \ k \ x) - \text{integral } (\text{cbox } a \ b) (f \ k)) < e/2 \wedge (k + 2))))$ 
      using intf e by (auto simp: has_integral_integral has_integral)
      then obtain  $c$  where  $c: \bigwedge x. \text{gauge } (c \ x)$ 
       $\bigwedge x \ \mathcal{D}. \llbracket \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b; c \ x \text{ fine } \mathcal{D} \rrbracket \implies$ 
 $\text{abs } ((\sum_{(u,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f \ x \ u) - \text{integral } (\text{cbox } a \ b) (f \ x)) < e/2 \wedge (x + 2))$ 
      by metis

      have  $\exists r. \forall k \geq r. 0 \leq i - (\text{integral } (\text{cbox } a \ b) (f \ k)) \wedge i - (\text{integral } (\text{cbox } a \ b) (f \ k)) < e/4$ 
    proof -
      have  $e/4 > 0$ 

```



```

    using e by auto
  show ?thesis
    using LIMSEQ_D [OF i < e/4 > 0] i' by auto
qed
then obtain r where r:  $\bigwedge k. r \leq k \implies 0 \leq i - \text{integral } (\text{cbox } a \ b) (f \ k)$ 
 $\bigwedge k. r \leq k \implies i - \text{integral } (\text{cbox } a \ b) (f \ k) < e/4$ 
  by metis
have  $\exists n \geq r. \forall k \geq n. 0 \leq (g \ x) - (f \ k \ x) \wedge (g \ x) - (f \ k \ x) < e/(4 * \text{content}(\text{cbox } a \ b))$ 
  if  $x \in \text{cbox } a \ b$  for x
proof -
  have  $e/(4 * \text{content } (\text{cbox } a \ b)) > 0$ 
    by (simp add: False content_lt_nz e)
  with fg that LIMSEQ_D
  obtain N where  $\forall n \geq N. \text{norm } (f \ n \ x - g \ x) < e/(4 * \text{content } (\text{cbox } a \ b))$ 
    by metis
  then show  $\exists n \geq r. \forall k \geq n. 0 \leq g \ x - f \ k \ x \wedge g \ x - f \ k \ x < e/(4 * \text{content } (\text{cbox } a \ b))$ 
    apply (rule_tac x=N + r in exI)
    using fg1 [OF that] by (auto simp add: field_simps)
qed
then obtain m where r_le_m:  $\bigwedge x. x \in \text{cbox } a \ b \implies r \leq m \ x$ 
  and m:  $\bigwedge x \ k. \llbracket x \in \text{cbox } a \ b; m \ x \leq k \rrbracket$ 
 $\implies 0 \leq g \ x - f \ k \ x \wedge g \ x - f \ k \ x < e/(4 * \text{content } (\text{cbox } a \ b))$ 
  by metis
define d where d x = c (m x) x for x
show  $\exists \gamma. \text{gauge } \gamma \wedge$ 
 $(\forall \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b \wedge$ 
 $\gamma \text{ fine } \mathcal{D} \longrightarrow \text{abs } ((\sum (x, K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} g \ x) - i) < e)$ 
proof (rule exI, safe)
  show gauge d
    using c(1) unfolding gauge_def d_def by auto
next
fix  $\mathcal{D}$ 
assume ptag:  $\mathcal{D} \text{ tagged\_division\_of } (\text{cbox } a \ b)$  and d fine  $\mathcal{D}$ 
note p'=tagged_division_ofD [OF ptag]
obtain s where s:  $\bigwedge x. x \in \mathcal{D} \implies m (fst \ x) \leq s$ 
by (metis finite_imageI finite_nat_set_iff_bounded_le p'(1) rev_image_eqI)
have *:  $|a - d| < e$  if  $|a - b| \leq e/4 \ |b - c| < e/2 \ |c - d| < e/4$  for a b c d
  using that norm_triangle_lt [of a - b b - c 3 * e/4]
  norm_triangle_lt [of a - b + (b - c) c - d e]
  by (auto simp add: algebra_simps)
show  $|(\sum (x, k) \in \mathcal{D}. \text{content } k *_{\mathbb{R}} g \ x) - i| < e$ 
proof (rule *)
  have  $|(\sum (x, K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} g \ x) - (\sum (x, K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f (m \ x) \ x)|$ 
 $\leq (\sum i \in \mathcal{D}. |(case \ i \ of \ (x, K) \Rightarrow \text{content } K *_{\mathbb{R}} g \ x) - (case \ i \ of \ (x, K)$ 
 $\Rightarrow \text{content } K *_{\mathbb{R}} f (m \ x) \ x)|)$ 
    by (metis (mono_tags) sum_subtractf sum_abs)

```

```

also have ... ≤ (∑ (x, k) ∈  $\mathcal{D}$ . content k * (e/(4 * content (cbox a b))))
proof (rule sum_mono, simp add: split_paired_all)
  fix x K
  assume xk: (x, K) ∈  $\mathcal{D}$ 
  with ptag have x: x ∈ cbox a b
  by blast
  then have abs (content K * (g x - f (m x) x)) ≤ content K * (e/(4 *
content (cbox a b)))
  by (metis m[OF x] mult_nonneg_nonneg abs_of_nonneg less_eq_real_def
measure_nonneg mult_left_mono order_refl)
  then show |content K * g x - content K * f (m x) x| ≤ content K * e/(4
* content (cbox a b))
  by (simp add: algebra_simps)
qed
also have ... = (e/(4 * content (cbox a b))) * (∑ (x, k) ∈  $\mathcal{D}$ . content k)
by (simp add: sum_distrib_left sum_divide_distrib split_def mult.commute)
also have ... ≤ e/4
by (metis False additive_content_tagged_division [OF ptag] nonzero_mult_divide_mult_cancel_r
order_refl times_divide_eq_left)
  finally show |(∑ (x, K) ∈  $\mathcal{D}$ . content K *R g x) - (∑ (x, K) ∈  $\mathcal{D}$ . content K
*_R f (m x) x)| ≤ e/4 .

next
have norm ((∑ (x, K) ∈  $\mathcal{D}$ . content K *_R f (m x) x) - (∑ (x, K) ∈  $\mathcal{D}$ . integral
K (f (m x))))
  ≤ norm (∑ j = 0..s. ∑ (x, K) ∈ {xk ∈  $\mathcal{D}$ . m (fst xk) = j}. content K *_R
f (m x) x - integral K (f (m x)))
  apply (subst sum.group)
  using s by (auto simp: sum_subtractf split_def p'(1))
also have ... < e/2
proof -
  have norm (∑ j = 0..s. ∑ (x, k) ∈ {xk ∈  $\mathcal{D}$ . m (fst xk) = j}. content k *_R
f (m x) x - integral k (f (m x)))
  ≤ (∑ i = 0..s. e/2 ^ (i + 2))
  proof (rule sum_norm_le)
    fix t
    assume t ∈ {0..s}
    have norm (∑ (x, k) ∈ {xk ∈  $\mathcal{D}$ . m (fst xk) = t}. content k *_R f (m x) x
- integral k (f (m x))) =
      norm (∑ (x, k) ∈ {xk ∈  $\mathcal{D}$ . m (fst xk) = t}. content k *_R f t x -
integral k (f t))
    by (force intro!: sum.cong arg_cong[where f=norm])
    also have ... ≤ e/2 ^ (t + 2)
    proof (rule Henstock_lemma_part1 [OF intf])
      show {xk ∈  $\mathcal{D}$ . m (fst xk) = t} tagged_partial_division_of cbox a b
      proof (rule tagged_partial_division_subset[of  $\mathcal{D}$ ])
        show  $\mathcal{D}$  tagged_partial_division_of cbox a b
        using ptag tagged_division_of_def by blast
      qed auto
  
```

```

    show c t fine {xk ∈ D. m (fst xk) = t}
    using ⟨d fine D⟩ by (auto simp: fine_def d_def)
  qed (use c e in auto)
  finally show norm (∑ (x,K)∈{xk ∈ D. m (fst xk) = t}. content K *R f
(m x) x -
      integral K (f (m x))) ≤ e/2 ^ (t + 2) .

  qed
  also have ... = (e/2/2) * (∑ i = 0..s. (1/2) ^ i)
    by (simp add: sum_distrib_left field_simps)
  also have ... < e/2
    by (simp add: sum_gp_mult_strict_left_mono[OF _ e])
  finally show norm (∑ j = 0..s. ∑ (x, k)∈{xk ∈ D.
m (fst xk) = j}. content k *R f (m x) x - integral k (f (m x))) < e/2 .
  qed
  finally show |(∑ (x,K)∈D. content K *R f (m x) x) - (∑ (x,K)∈D. integral
K (f (m x)))| < e/2
    by simp
next
  have comb: integral (cbox a b) (f y) = (∑ (x, k)∈D. integral k (f y)) for y
    using integral_combine_tagged_division_topdown[OF intf ptag] by metis
  have f_le: ⋀ y m n. [y ∈ cbox a b; n ≥ m] ⇒ f m y ≤ f n y
    using le by (auto intro: transitive_stepwise_le)
  have (∑ (x, k)∈D. integral k (f r)) ≤ (∑ (x, K)∈D. integral K (f (m x)))
  proof (rule sum_mono, simp add: split_paired_all)
    fix x K
    assume xK: (x, K) ∈ D
    show integral K (f r) ≤ integral K (f (m x))
    proof (rule integral_le)
      show f r integrable_on K
        by (metis integrable_on_subcbox intf p'(3) p'(4) xK)
      show f (m x) integrable_on K
        by (metis elementary_interval integrable_on_subdivision intf p'(3)
p'(4) xK)
      show f r y ≤ f (m x) y if y ∈ K for y
        using that r_le_m[of x] p'(2-3)[OF xK] f_le by auto
    qed
  qed
  moreover have (∑ (x, K)∈D. integral K (f (m x))) ≤ (∑ (x, k)∈D. integral
k (f s))
  proof (rule sum_mono, simp add: split_paired_all)
    fix x K
    assume xK: (x, K) ∈ D
    show integral K (f (m x)) ≤ integral K (f s)
    proof (rule integral_le)
      show f (m x) integrable_on K
        by (metis elementary_interval integrable_on_subdivision intf p'(3)
p'(4) xK)
      show f s integrable_on K
        by (metis integrable_on_subcbox intf p'(3) p'(4) xK)

```

```

      show  $f (m\ x)\ y \leq f\ s\ y$  if  $y \in K$  for  $y$ 
      using that  $s\ xK\ f\_le\ p'(\mathcal{B})$  by fastforce
    qed
  qed
  moreover have  $0 \leq i - \text{integral}\ (cbox\ a\ b)\ (f\ r)\ i - \text{integral}\ (cbox\ a\ b)\ (f\ r) < e/4$ 
    using  $r$  by auto
  ultimately show  $|(\sum_{(x,K) \in \mathcal{D}} \text{integral}\ K\ (f\ (m\ x))) - i| < e/4$ 
    using comb  $i'[of\ s]$  by auto
  qed
  qed
  qed
  with  $i\ \text{integral\_unique}$  show ?thesis
    by blast
  qed

lemma monotone_convergence_increasing:
  fixes  $f :: \text{nat} \Rightarrow 'n::\text{euclidean\_space} \Rightarrow \text{real}$ 
  assumes  $\text{int\_f}: \bigwedge k. (f\ k)\ \text{integrable\_on}\ S$ 
    and  $\bigwedge k\ x. x \in S \implies (f\ k\ x) \leq (f\ (\text{Suc}\ k)\ x)$ 
    and  $\text{fg}: \bigwedge x. x \in S \implies ((\lambda k. f\ k\ x) \longrightarrow g\ x)\ \text{sequentially}$ 
    and  $\text{bou}: \text{bounded}\ (\text{range}\ (\lambda k. \text{integral}\ S\ (f\ k)))$ 
  shows  $g\ \text{integrable\_on}\ S \wedge ((\lambda k. \text{integral}\ S\ (f\ k)) \longrightarrow \text{integral}\ S\ g)\ \text{sequentially}$ 
  proof -
    have lem:  $g\ \text{integrable\_on}\ S \wedge ((\lambda k. \text{integral}\ S\ (f\ k)) \longrightarrow \text{integral}\ S\ g)\ \text{sequentially}$ 
    if  $f0: \bigwedge k\ x. x \in S \implies 0 \leq f\ k\ x$ 
    and  $\text{int\_f}: \bigwedge k. (f\ k)\ \text{integrable\_on}\ S$ 
    and  $\text{le}: \bigwedge k\ x. x \in S \implies f\ k\ x \leq f\ (\text{Suc}\ k)\ x$ 
    and  $\text{lim}: \bigwedge x. x \in S \implies ((\lambda k. f\ k\ x) \longrightarrow g\ x)\ \text{sequentially}$ 
    and  $\text{bou}: \text{bounded}\ (\text{range}\ (\lambda k. \text{integral}\ S\ (f\ k)))$ 
    for  $f :: \text{nat} \Rightarrow 'n::\text{euclidean\_space} \Rightarrow \text{real}$  and  $g\ S$ 
  proof -
    have  $\text{fg}: (f\ k\ x) \leq (g\ x)$  if  $x \in S$  for  $x\ k$ 
  proof -
    have  $\bigwedge xa. k \leq xa \implies f\ k\ x \leq f\ xa\ x$ 
      using le by (force intro: transitive_stepwise_le that)
    then show ?thesis
      using tendsto_lowerbound [OF lim [OF that]] eventually_sequentiallyI by
force
  qed
  obtain  $i$  where  $i: (\lambda k. \text{integral}\ S\ (f\ k)) \longrightarrow i$ 
    using bounded_increasing_convergent [OF bou] le int_f integral_le by blast
  have  $i': (\text{integral}\ S\ (f\ k)) \leq i$  for  $k$ 
  proof -
    have  $\bigwedge k. \bigwedge x. x \in S \implies \forall n \geq k. f\ k\ x \leq f\ n\ x$ 
      using le by (force intro: transitive_stepwise_le)
    then show ?thesis
      using tendsto_lowerbound [OF i eventually_sequentiallyI trivial_limit_sequentially]

```

```

    by (meson int_f integral_le)
  qed
  let ?f = ( $\lambda k x. \text{if } x \in S \text{ then } f k x \text{ else } 0$ )
  let ?g = ( $\lambda x. \text{if } x \in S \text{ then } g x \text{ else } 0$ )
  have int: ?f k integrable_on cbox a b for a b k
    by (simp add: int_f integrable_altD(1))
  have int':  $\bigwedge k a b. f k \text{ integrable\_on } cbox a b \cap S$ 
    using int by (simp add: Int_commute integrable_restrict_Int)
  have g: ?g integrable_on cbox a b  $\wedge$ 
    ( $\lambda k. \text{integral } (cbox a b) (?f k) \longrightarrow \text{integral } (cbox a b) ?g$ ) for a b
  proof (rule monotone_convergence_interval)
    have norm ( $\text{integral } (cbox a b) (?f k)$ )  $\leq$  norm ( $\text{integral } S (f k)$ ) for k
    proof -
      have  $0 \leq \text{integral } (cbox a b) (?f k)$ 
        by (metis (no_types) integral_nonneg Int_iff f0 inf_commute integrable_restrict_Int int')
      moreover have  $0 \leq \text{integral } S (f k)$ 
        by (simp add: integral_nonneg f0 int_f)
      moreover have  $\text{integral } (S \cap cbox a b) (f k) \leq \text{integral } S (f k)$ 
        by (metis f0 inf_commute int' int_f integral_subset_le le_inf_iff order_refl)
      ultimately show ?thesis
        by (simp add: integral_restrict_Int)
    qed
    moreover obtain B where  $\bigwedge x. x \in \text{range } (\lambda k. \text{integral } S (f k)) \implies \text{norm } x \leq B$ 
      using bou unfolding bounded_iff by blast
    ultimately show bounded (range ( $\lambda k. \text{integral } (cbox a b) (?f k)$ ))
      unfolding bounded_iff by (blast intro: order_trans)
  qed (use int le lim in auto)
  moreover have  $\exists B > 0. \forall a b. \text{ball } 0 B \subseteq cbox a b \longrightarrow \text{norm } (\text{integral } (cbox a b) ?g - i) < e$ 
    if  $0 < e$  for e
  proof -
    have  $e/4 > 0$ 
      using that by auto
    with LIMSEQ_D [OF i] obtain N where  $N: \bigwedge n. n \geq N \implies \text{norm } (\text{integral } S (f n) - i) < e/4$ 
      by metis
    with int_f[of N, unfolded has_integral_integral has_integral_alt'[of f N]]
    obtain B where  $0 < B$  and B:
       $\bigwedge a b. \text{ball } 0 B \subseteq cbox a b \implies \text{norm } (\text{integral } (cbox a b) (?f N) - \text{integral } S (f N)) < e/4$ 
    by (meson  $\langle 0 < e/4 \rangle$ )
    have norm ( $\text{integral } (cbox a b) ?g - i$ )  $< e$  if  $ab: \text{ball } 0 B \subseteq cbox a b$  for a b
    proof -
      obtain M where  $M: \bigwedge n. n \geq M \implies \text{abs } (\text{integral } (cbox a b) (?f n) - \text{integral } (cbox a b) ?g) < e/2$ 
        using  $\langle e > 0 \rangle g$  by (fastforce simp add: dest!: LIMSEQ_D [where r =
```

```

e/2])
have *:  $\bigwedge \alpha \beta g. \llbracket |\alpha - i| < e/2; |\beta - g| < e/2; \alpha \leq \beta; \beta \leq i \rrbracket \implies |g - i|$ 
< e
  unfolding real_inner_1_right by arith
show norm (integral (cbox a b) ?g - i) < e
  unfolding real_norm_def
proof (rule *)
  show |integral (cbox a b) (?f N) - i| < e/2
  proof (rule abs_triangle_half_l)
    show |integral (cbox a b) (?f N) - integral S (f N)| < e/2/2
    using B[OF ab] by simp
    show abs (i - integral S (f N)) < e/2/2
    using N by (simp add: abs_minus_commute)
  qed
  show |integral (cbox a b) (?f (M + N)) - integral (cbox a b) ?g| < e/2
  by (metis le_add1 M[of M + N])
  show integral (cbox a b) (?f N)  $\leq$  integral (cbox a b) (?f (M + N))
  proof (intro ballI integral_le[OF int int])
    fix x assume x  $\in$  cbox a b
    have (f m x)  $\leq$  (f n x) if x  $\in$  S n  $\geq$  m for m n
    proof (rule transitive_stepwise_le [OF <n  $\geq$  m> order_refl])
      show  $\bigwedge u y z. \llbracket f u x \leq f y x; f y x \leq f z x \rrbracket \implies f u x \leq f z x$ 
      using dual_order.trans by blast
    qed (simp add: le <x  $\in$  S>)
    then show (?f N)x  $\leq$  (?f (M+N))x
    by auto
  qed
  have integral (cbox a b  $\cap$  S) (f (M + N))  $\leq$  integral S (f (M + N))
  by (metis Int_lower1 f0 inf_commute int' int_f integral_subset_le)
  then have integral (cbox a b) (?f (M + N))  $\leq$  integral S (f (M + N))
  by (metis (no_types) inf_commute integral_restrict_Int)
  also have ...  $\leq$  i
  using i'[of M + N] by auto
  finally show integral (cbox a b) (?f (M + N))  $\leq$  i .
qed
qed
then show ?thesis
  using <0 < B> by blast
qed
ultimately have (g has_integral i) S
  unfolding has_integral_alt' by auto
then show ?thesis
  using has_integral_integrable_integral i integral_unique by metis
qed

have sub:  $\bigwedge k. \text{integral } S (\lambda x. f k x - f 0 x) = \text{integral } S (f k) - \text{integral } S (f 0)$ 
  by (simp add: integral_diff int_f)
have *:  $\bigwedge x m n. x \in S \implies n \geq m \implies f m x \leq f n x$ 
  using asms(2) by (force intro: transitive_stepwise_le)

```

```

have gf: ( $\lambda x. g\ x - f\ 0\ x$ ) integrable_on S  $\wedge$  (( $\lambda k. \text{integral } S\ (\lambda x. f\ (\text{Suc } k)\ x - f\ 0\ x)$ )  $\longrightarrow$ 
  integral S ( $\lambda x. g\ x - f\ 0\ x$ )) sequentially
proof (rule lem)
  show  $\bigwedge k. (\lambda x. f\ (\text{Suc } k)\ x - f\ 0\ x)$  integrable_on S
    by (simp add: integrable_diff_int_f)
  show ( $\lambda k. f\ (\text{Suc } k)\ x - f\ 0\ x$ )  $\longrightarrow$   $g\ x - f\ 0\ x$  if  $x \in S$  for  $x$ 
  proof -
    have ( $\lambda n. f\ (\text{Suc } n)\ x$ )  $\longrightarrow$   $g\ x$ 
      using LIMSEQ_ignore_initial_segment[OF fg[OF  $\langle x \in S \rangle$ , of 1] by simp
    then show ?thesis
      by (simp add: tendsto_diff)
  qed
  show bounded (range ( $\lambda k. \text{integral } S\ (\lambda x. f\ (\text{Suc } k)\ x - f\ 0\ x)$ ))
  proof -
    obtain B where B:  $\bigwedge k. \text{norm } (\text{integral } S\ (f\ k)) \leq B$ 
      using bou by (auto simp: bounded_iff)
    then have norm ( $\text{integral } S\ (\lambda x. f\ (\text{Suc } k)\ x - f\ 0\ x)$ )
       $\leq B + \text{norm } (\text{integral } S\ (f\ 0))$  for  $k$ 
      unfolding sub by (meson add_le_cancel_right norm_triangle_le_diff)
    then show ?thesis
      unfolding bounded_iff by blast
  qed
qed
qed (use * in auto)
then have ( $\lambda x. \text{integral } S\ (\lambda xa. f\ (\text{Suc } x)\ xa - f\ 0\ xa) + \text{integral } S\ (f\ 0)$ )
   $\longrightarrow$   $\text{integral } S\ (\lambda x. g\ x - f\ 0\ x) + \text{integral } S\ (f\ 0)$ 
  by (auto simp add: tendsto_add)
moreover have ( $\lambda x. g\ x - f\ 0\ x + f\ 0\ x$ ) integrable_on S
  using gf integrable_add_int_f [of 0] by metis
ultimately show ?thesis
  by (simp add: integral_diff_int_f LIMSEQ_imp_Suc sub)
qed

```

lemma has_integral_monotone_convergence_increasing:

```

fixes f :: nat  $\Rightarrow$  'a::euclidean_space  $\Rightarrow$  real
assumes f:  $\bigwedge k. (f\ k \text{ has\_integral } x\ k)\ s$ 
assumes  $\bigwedge k\ x. x \in s \implies f\ k\ x \leq f\ (\text{Suc } k)\ x$ 
assumes  $\bigwedge x. x \in s \implies (\lambda k. f\ k\ x) \longrightarrow g\ x$ 
assumes  $x \longrightarrow x'$ 
shows ( $g \text{ has\_integral } x'$ ) s

```

```

proof -
  have x_eq:  $x = (\lambda i. \text{integral } s\ (f\ i))$ 
    by (simp add: integral_unique[OF f])
  then have x:  $\text{range } (\lambda k. \text{integral } s\ (f\ k)) = \text{range } x$ 
    by auto
  have *:  $g \text{ integrable\_on } s \wedge (\lambda k. \text{integral } s\ (f\ k)) \longrightarrow \text{integral } s\ g$ 
  proof (intro monotone_convergence_increasing allI ballI assms)
    show bounded (range ( $\lambda k. \text{integral } s\ (f\ k)$ ))
      using x convergent_imp_bounded assms by metis
  qed

```

```

qed (use f in auto)
then have integral s g = x'
  by (intro LIMSEQ_unique[OF _ ⟨x ⟶ x'⟩]) (simp add: x_eq)
with * show ?thesis
  by (simp add: has_integral_integral)
qed

```

```

lemma monotone_convergence_decreasing:
  fixes f :: nat ⇒ 'n::euclidean_space ⇒ real
  assumes intf:  $\bigwedge k. (f\ k) \text{ integrable\_on } S$ 
    and le:  $\bigwedge k\ x. x \in S \implies f\ (Suc\ k)\ x \leq f\ k\ x$ 
    and fg:  $\bigwedge x. x \in S \implies ((\lambda k. f\ k\ x) \longrightarrow g\ x) \text{ sequentially}$ 
    and bou: bounded (range( $\lambda k. \text{integral } S\ (f\ k)$ ))
  shows g integrable_on S  $\wedge (\lambda k. \text{integral } S\ (f\ k)) \longrightarrow \text{integral } S\ g$ 
proof -
  have *: range( $\lambda k. \text{integral } S\ (\lambda x. - f\ k\ x)$ ) = (*R) (- 1) ` (range( $\lambda k. \text{integral } S\ (f\ k)$ ))
  by force
  have ( $\lambda x. - g\ x$ ) integrable_on S  $\wedge (\lambda k. \text{integral } S\ (\lambda x. - f\ k\ x)) \longrightarrow \text{integral } S\ (\lambda x. - g\ x)$ 
  proof (rule monotone_convergence_increasing)
    show  $\bigwedge k. (\lambda x. - f\ k\ x) \text{ integrable\_on } S$ 
    by (blast intro: integrable_neg intf)
    show  $\bigwedge k\ x. x \in S \implies - f\ k\ x \leq - f\ (Suc\ k)\ x$ 
    by (simp add: le)
    show  $\bigwedge x. x \in S \implies (\lambda k. - f\ k\ x) \longrightarrow - g\ x$ 
    by (simp add: fg tendsto_minus)
    show bounded (range( $\lambda k. \text{integral } S\ (\lambda x. - f\ k\ x)$ ))
    using * bou bounded_scaling by auto
  qed
  then show ?thesis
  by (force dest: integrable_neg tendsto_minus)
qed

```

```

lemma integral_norm_bound_integral:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  assumes int_f: f integrable_on S
    and int_g: g integrable_on S
    and le_g:  $\bigwedge x. x \in S \implies \text{norm } (f\ x) \leq g\ x$ 
  shows norm (integral S f) ≤ integral S g
proof -
  have norm: norm η ≤ y + e
  if norm ζ ≤ x and |x - y| < e/2 and norm (ζ - η) < e/2
  for e x y and ζ η :: 'a
  proof -
    have norm (η - ζ) < e/2
    by (metis norm_minus_commute that(3))
    moreover have x ≤ y + e/2
    using that(2) by linarith

```



```

ultimately show ?thesis
  using that(1) le_less_trans[OF norm_triangle_sub[of  $\eta$   $\zeta$ ]] by (auto simp:
less_imp_le)
qed
have lem: norm (integral (cbox a b) f)  $\leq$  integral (cbox a b) g
  if f: f integrable_on cbox a b
  and g: g integrable_on cbox a b
  and nle:  $\bigwedge x. x \in \text{cbox } a \ b \implies \text{norm } (f \ x) \leq g \ x$ 
  for f :: 'n  $\Rightarrow$  'a and g a b
proof (rule field_le_epsilon)
  fix e :: real
  assume e > 0
  then have e: e/2 > 0
    by auto
  with integrable_integral[OF f,unfolded has_integral[of f]]
  obtain  $\gamma$  where  $\gamma$ : gauge  $\gamma$ 
     $\bigwedge \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b \wedge \gamma \text{ fine } \mathcal{D}$ 
     $\implies \text{norm } ((\sum (x, k) \in \mathcal{D}. \text{content } k *_R f \ x) - \text{integral } (\text{cbox } a \ b) f) < e/2$ 
  by meson
  moreover
  from integrable_integral[OF g,unfolded has_integral[of g]] e
  obtain  $\delta$  where  $\delta$ : gauge  $\delta$ 
     $\bigwedge \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b \wedge \delta \text{ fine } \mathcal{D}$ 
     $\implies \text{norm } ((\sum (x, k) \in \mathcal{D}. \text{content } k *_R g \ x) - \text{integral } (\text{cbox } a \ b) g) < e/2$ 
  by meson
  ultimately have gauge ( $\lambda x. \gamma \ x \cap \delta \ x$ )
    using gauge_Int by blast
  with fine_division_exists obtain  $\mathcal{D}$ 
    where p:  $\mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b \ (\lambda x. \gamma \ x \cap \delta \ x) \text{ fine } \mathcal{D}$ 
  by metis
  have  $\gamma \text{ fine } \mathcal{D} \ \delta \text{ fine } \mathcal{D}$ 
    using fine_Int p(2) by blast+
  show norm (integral (cbox a b) f)  $\leq$  integral (cbox a b) g + e
  proof (rule norm)
    have norm (content K *_R f x)  $\leq$  content K *_R g x if (x, K)  $\in \mathcal{D}$  for x K
    proof-
      have K: x  $\in$  K K  $\subseteq$  cbox a b
        using  $\langle (x, K) \in \mathcal{D} \rangle$  p(1) by blast+
      obtain u v where K = cbox u v
        using  $\langle (x, K) \in \mathcal{D} \rangle$  p(1) by blast
      moreover have content K * norm (f x)  $\leq$  content K * g x
        by (meson K(1) K(2) content_pos_le mult_left_mono nle subsetD)
      then show ?thesis
        by simp
    qed
  qed
  then show norm ( $\sum (x, k) \in \mathcal{D}. \text{content } k *_R f \ x$ )  $\leq$  ( $\sum (x, k) \in \mathcal{D}. \text{content } k$ 
*_R g x)
    by (simp add: sum_norm_le split_def)
  show norm (( $\sum (x, k) \in \mathcal{D}. \text{content } k *_R f \ x$ ) - integral (cbox a b) f) < e/2

```

```

      using ⟨ $\gamma$  fine  $\mathcal{D}$ ⟩  $\gamma$   $p(1)$  by simp
      show  $|(\sum_{(x, k) \in \mathcal{D}} \text{content } k *_R g \ x) - \text{integral } (\text{cbox } a \ b) \ g| < e/2$ 
      using ⟨ $\delta$  fine  $\mathcal{D}$ ⟩  $\delta$   $p(1)$  by simp
    qed
  qed
  show ?thesis
  proof (rule field_le_epsilon)
    fix e :: real
    assume e > 0
    then have e:  $e/2 > 0$ 
    by auto
    let ?f = ( $\lambda x. \text{if } x \in S \text{ then } f \ x \text{ else } 0$ )
    let ?g = ( $\lambda x. \text{if } x \in S \text{ then } g \ x \text{ else } 0$ )
    have f: ?f integrable_on cbox a b and g: ?g integrable_on cbox a b for a b
      using int_f int_g integrable_altD by auto
    obtain Bf where  $0 < Bf$ 
    and Bf:  $\bigwedge a \ b. \text{ball } 0 \ Bf \subseteq \text{cbox } a \ b \implies$ 
       $\exists z. (?f \text{ has\_integral } z) (\text{cbox } a \ b) \wedge \text{norm } (z - \text{integral } S \ f) < e/2$ 
    using integrable_integral [OF int_f, unfolded has_integral[of f]] e that by
  blast
    obtain Bg where  $0 < Bg$ 
    and Bg:  $\bigwedge a \ b. \text{ball } 0 \ Bg \subseteq \text{cbox } a \ b \implies$ 
       $\exists z. (?g \text{ has\_integral } z) (\text{cbox } a \ b) \wedge \text{norm } (z - \text{integral } S \ g) < e/2$ 
    using integrable_integral [OF int_g, unfolded has_integral[of g]] e that by
  blast
    obtain a b::'n where ab:  $\text{ball } 0 \ Bf \cup \text{ball } 0 \ Bg \subseteq \text{cbox } a \ b$ 
    using ball_max_Un by (metis bounded_ball bounded_subset_cbox_symmetric)
    have ball 0 Bf  $\subseteq \text{cbox } a \ b$ 
    using ab by auto
    with Bf obtain z where int_fz:  $(?f \text{ has\_integral } z) (\text{cbox } a \ b)$  and z:  $\text{norm } (z - \text{integral } S \ f) < e/2$ 
    by meson
    have ball 0 Bg  $\subseteq \text{cbox } a \ b$ 
    using ab by auto
    with Bg obtain w where int_gw:  $(?g \text{ has\_integral } w) (\text{cbox } a \ b)$  and w:  $\text{norm } (w - \text{integral } S \ g) < e/2$ 
    by meson
    show  $\text{norm } (\text{integral } S \ f) \leq \text{integral } S \ g + e$ 
    proof (rule norm)
      show  $\text{norm } (\text{integral } (\text{cbox } a \ b) \ ?f) \leq \text{integral } (\text{cbox } a \ b) \ ?g$ 
      by (simp add: le_g lem[OF f g, of a b])
      show  $|\text{integral } (\text{cbox } a \ b) \ ?g - \text{integral } S \ g| < e/2$ 
      using int_gw integral_unique w by auto
      show  $\text{norm } (\text{integral } (\text{cbox } a \ b) \ ?f - \text{integral } S \ f) < e/2$ 
      using int_fz integral_unique z by blast
    qed
  qed
  qed
  qed

```

```

lemma continuous_on_imp_absolutely_integrable_on:
  fixes f::real  $\Rightarrow$  'a::banach
  shows continuous_on {a..b} f  $\implies$ 
    norm (integral {a..b} f)  $\leq$  integral {a..b} ( $\lambda x$ . norm (f x))
  by (intro integral_norm_bound_integral integrable_continuous_real continuous_on_norm)
  auto

```

```

lemma integral_bound:
  fixes f::real  $\Rightarrow$  'a::banach
  assumes a  $\leq$  b
  assumes continuous_on {a .. b} f
  assumes  $\bigwedge t. t \in \{a .. b\} \implies \text{norm } (f t) \leq B$ 
  shows norm (integral {a .. b} f)  $\leq B * (b - a)$ 
proof -
  note continuous_on_imp_absolutely_integrable_on[OF assms(2)]
  also have integral {a..b} ( $\lambda x$ . norm (f x))  $\leq$  integral {a..b} ( $\lambda \_.$  B)
    by (rule integral_le)
    (auto intro!: integrable_continuous_real continuous_intros assms)
  also have ... = B * (b - a) using assms by simp
  finally show ?thesis .
qed

```

```

lemma integral_norm_bound_integral_component:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  fixes g :: 'n  $\Rightarrow$  'b::euclidean_space
  assumes f: f integrable_on S and g: g integrable_on S
    and fg:  $\bigwedge x. x \in S \implies \text{norm}(f x) \leq (g x) \cdot k$ 
  shows norm (integral S f)  $\leq$  (integral S g)  $\cdot k$ 
proof -
  have norm (integral S f)  $\leq$  integral S (( $\lambda x$ . x  $\cdot$  k)  $\circ$  g)
    using integral_norm_bound_integral[OF f integrable_linear[OF g]]
    by (simp add: bounded_linear_inner_left fg)
  then show ?thesis
    unfolding o_def integral_component_eq[OF g] .
qed

```

```

lemma has_integral_norm_bound_integral_component:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  fixes g :: 'n  $\Rightarrow$  'b::euclidean_space
  assumes f: (f has_integral i) S
    and g: (g has_integral j) S
    and  $\bigwedge x. x \in S \implies \text{norm } (f x) \leq (g x) \cdot k$ 
  shows norm i  $\leq j \cdot k$ 
using integral_norm_bound_integral_component[of f S g k]
unfolding integral_unique[OF f] integral_unique[OF g]
using assms
by auto

```

```

lemma uniformly_convergent_improper_integral:
  fixes f :: 'b  $\Rightarrow$  real  $\Rightarrow$  'a :: {banach}
  assumes deriv:  $\bigwedge x. x \geq a \implies (G \text{ has\_field\_derivative } g \ x) \text{ (at } x \text{ within } \{a..\})$ 
  assumes integrable:  $\bigwedge a' b x. x \in A \implies a' \geq a \implies b \geq a' \implies f \ x \text{ integrable\_on } \{a'..b\}$ 
  assumes G: convergent G
  assumes le:  $\bigwedge y x. y \in A \implies x \geq a \implies \text{norm } (f \ y \ x) \leq g \ x$ 
  shows uniformly_convergent_on A ( $\lambda b x. \text{integral } \{a..b\} (f \ x)$ )
proof (intro Cauchy_uniformly_convergent uniformly_Cauchy_onI', goal_cases)
  case (1  $\varepsilon$ )
  from G have Cauchy G
  by (auto intro!: convergent_Cauchy)
  with 1 obtain M where M:  $\text{dist } (G \ (\text{real } m)) (G \ (\text{real } n)) < \varepsilon \text{ if } m \geq M \ n \geq M$  for m n
  by (force simp: Cauchy_def)
  define M' where  $M' = \max (\text{nat } \lceil a \rceil) M$ 

  show ?case
proof (rule exI[of _ M'], safe, goal_cases)
  case (1 x m n)
  have M':  $M' \geq a \ M' \geq M$  unfolding M'_def by linarith+
  have int_g:  $(g \text{ has\_integral } (G \ (\text{real } n) - G \ (\text{real } m))) \ \{ \text{real } m.. \text{real } n \}$ 
    using 1 M' by (intro fundamental_theorem_of_calculus)
    (auto simp: has_field_derivative_iff_has_vector_derivative
[symmetric]
intro!: DERIV_subset[OF deriv])
  have int_f:  $f \ x \text{ integrable\_on } \{a'.. \text{real } n\} \text{ if } a' \geq a \text{ for } a'$ 
    using that 1 by (cases  $a' \leq \text{real } n$ ) (auto intro: integrable)

  have  $\text{dist } (\text{integral } \{a.. \text{real } m\} (f \ x)) (\text{integral } \{a.. \text{real } n\} (f \ x)) =$ 
     $\text{norm } (\text{integral } \{a.. \text{real } n\} (f \ x) - \text{integral } \{a.. \text{real } m\} (f \ x))$ 
    by (simp add: dist_norm norm_minus_commute)
  also have  $\text{integral } \{a.. \text{real } m\} (f \ x) + \text{integral } \{ \text{real } m.. \text{real } n\} (f \ x) =$ 
     $\text{integral } \{a.. \text{real } n\} (f \ x)$ 
    using M' and 1 by (intro integral_combine int_f) auto
  hence  $\text{integral } \{a.. \text{real } n\} (f \ x) - \text{integral } \{a.. \text{real } m\} (f \ x) =$ 
     $\text{integral } \{ \text{real } m.. \text{real } n\} (f \ x)$ 
    by (simp add: algebra_simps)
  also have  $\text{norm } \dots \leq \text{integral } \{ \text{real } m.. \text{real } n\} g$ 
    using le 1 M' int_f int_g by (intro integral_norm_bound_integral) auto
  also from int_g have  $\text{integral } \{ \text{real } m.. \text{real } n\} g = G \ (\text{real } n) - G \ (\text{real } m)$ 
    by (simp add: has_integral_iff)
  also have  $\dots \leq \text{dist } (G \ m) (G \ n)$ 
    by (simp add: dist_norm)
  also from 1 and M' have  $\dots < \varepsilon$ 
    by (intro M) auto
  finally show ?case .
qed
qed

```

```

lemma uniformly_convergent_improper_integral':
  fixes  $f :: 'b \Rightarrow \text{real} \Rightarrow 'a :: \{\text{banach}, \text{real\_normed\_algebra}\}$ 
  assumes deriv:  $\bigwedge x. x \geq a \implies (G \text{ has\_field\_derivative } g \ x) \text{ (at } x \text{ within } \{a..\})$ 
  assumes integrable:  $\bigwedge a' b x. x \in A \implies a' \geq a \implies b \geq a' \implies f \ x \text{ integrable\_on } \{a'..b\}$ 
  assumes G: convergent G
  assumes le: eventually  $(\lambda x. \forall y \in A. \text{norm } (f \ y \ x) \leq g \ x) \text{ at\_top}$ 
  shows uniformly_convergent_on A  $(\lambda b x. \text{integral } \{a..b\} (f \ x))$ 
proof -
  from le obtain  $a''$  where  $\bigwedge y x. y \in A \implies x \geq a'' \implies \text{norm } (f \ y \ x) \leq g \ x$ 
    by (auto simp: eventually_at_top_linorder)
  define  $a'$  where  $a' = \max a \ a''$ 

  have uniformly_convergent_on A  $(\lambda b x. \text{integral } \{a'.. \text{real } b\} (f \ x))$ 
proof (rule uniformly_convergent_improper_integral)
  fix  $t$  assume  $t: t \geq a'$ 
  hence  $(G \text{ has\_field\_derivative } g \ t) \text{ (at } t \text{ within } \{a..\})$ 
    by (intro deriv) (auto simp: a'_def)
  moreover have  $\{a'..\} \subseteq \{a..\}$  unfolding a'_def by auto
  ultimately show  $(G \text{ has\_field\_derivative } g \ t) \text{ (at } t \text{ within } \{a'..\})$ 
    by (rule DERIV_subset)
qed (insert le, auto simp: a'_def intro: integrable G)
hence uniformly_convergent_on A  $(\lambda b x. \text{integral } \{a..a'\} (f \ x) + \text{integral } \{a'.. \text{real } b\} (f \ x))$ 
  (is ?P) by (intro uniformly_convergent_add) auto
  also have eventually  $(\lambda x. \forall y \in A. \text{integral } \{a..a'\} (f \ y) + \text{integral } \{a'..x\} (f \ y) = \text{integral } \{a..x\} (f \ y)) \text{ sequentially}$ 
    by (intro eventually_mono [OF eventually_ge_at_top[of nat ⌈a'⌉]] ballI integral_combine)
    (auto simp: a'_def intro: integrable)
  hence ?P  $\longleftrightarrow$  ?thesis
    by (intro uniformly_convergent_cong) simp_all
  finally show ?thesis .
qed

```

6.15.42 differentiation under the integral sign

```

lemma integral_continuous_on_param:
  fixes  $f :: 'a :: \text{topological\_space} \Rightarrow 'b :: \text{euclidean\_space} \Rightarrow 'c :: \text{banach}$ 
  assumes cont_fx: continuous_on  $(U \times \text{cbox } a \ b) (\lambda(x, t). f \ x \ t)$ 
  shows continuous_on U  $(\lambda x. \text{integral } (\text{cbox } a \ b) (f \ x))$ 
proof cases
  assume content  $(\text{cbox } a \ b) \neq 0$ 
  then have  $ne: \text{cbox } a \ b \neq \{\}$  by auto

  note  $[\text{continuous\_intros}] =$ 
    continuous_on_compose2 [OF cont_fx, where  $f = \lambda y. \text{Pair } x \ y$  for  $x$ ,
      unfolded split_beta fst_conv snd_conv]

```

```

show ?thesis
  unfolding continuous_on_def
proof (safe intro!: tendstoI)
  fix e'::real and x
  assume e' > 0
  define e where e = e' / (content (cbox a b) + 1)
  have e > 0 using ‹e' > 0› by (auto simp: e_def intro!: divide_pos_pos
add_nonneg_pos)
  assume x ∈ U
  from continuous_on_prod_compactE[OF cont_fx compact_cbox ‹x ∈ U› ‹0
< e›]
  obtain X0 where X0: x ∈ X0 open X0
  and fx_bound:  $\bigwedge y t. y \in X0 \cap U \implies t \in \text{cbox } a \text{ } b \implies \text{norm } (f y \text{ } t - f x \text{ } t) \leq e$ 
  unfolding split_beta fst_conv snd_conv dist_norm
  by metis
  have  $\forall_F y$  in at x within U.  $y \in X0 \cap U$ 
  using X0(1) X0(2) eventually_at_topological by auto
  then show  $\forall_F y$  in at x within U.  $\text{dist } (\text{integral } (\text{cbox } a \text{ } b) (f y)) (\text{integral } (\text{cbox } a \text{ } b) (f x)) < e'$ 
  proof eventually_elim
    case (elim y)
    have  $\text{dist } (\text{integral } (\text{cbox } a \text{ } b) (f y)) (\text{integral } (\text{cbox } a \text{ } b) (f x)) = \text{norm } (\text{integral } (\text{cbox } a \text{ } b) (\lambda t. f y \text{ } t - f x \text{ } t))$ 
    using elim ‹x ∈ U›
    unfolding dist_norm
    by (subst integral_diff)
    (auto intro!: integrable_continuous continuous_intros)
    also have  $\dots \leq e * \text{content } (\text{cbox } a \text{ } b)$ 
    using elim ‹x ∈ U›
    by (intro integrable_bound)
    (auto intro!: fx_bound ‹x ∈ U› less_imp_le[OF ‹0 < e›]
integrable_continuous continuous_intros)
    also have  $\dots < e'$ 
    using ‹0 < e'› ‹e > 0›
    by (auto simp: e_def field_split_simps)
  finally show  $\text{dist } (\text{integral } (\text{cbox } a \text{ } b) (f y)) (\text{integral } (\text{cbox } a \text{ } b) (f x)) < e'$ .
qed
qed
qed (auto intro!: continuous_on_const)

```

lemma leibniz_rule:

```

fixes f::'a::banach  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'c::banach
assumes fx:  $\bigwedge x t. x \in U \implies t \in \text{cbox } a \text{ } b \implies ((\lambda x. f x \text{ } t) \text{ has\_derivative } \text{blinfun\_apply } (f x \text{ } t)) \text{ (at } x \text{ within } U)$ 
assumes integrable_f2:  $\bigwedge x. x \in U \implies f x \text{ integrable\_on } \text{cbox } a \text{ } b$ 
assumes cont_fx: continuous_on (U  $\times$  (cbox a b)) ( $\lambda(x, t). f x \text{ } t$ )
assumes [intro]: x0 ∈ U

```

```

assumes convex U
shows
  (( $\lambda x. \text{integral } (cbox\ a\ b)\ (f\ x)$ ) has_derivative integral (cbox a b) (fx x0)) (at x0
within U)
  (is (?F has_derivative ?dF) __)
proof cases
  assume content (cbox a b)  $\neq 0$ 
  then have ne: cbox a b  $\neq \{\}$  by auto
  note [continuous_intros] =
    continuous_on_compose2[OF cont_fx, where f= $\lambda y. \text{Pair } x\ y$  for x,
    unfolded split_beta fst_conv snd_conv]
  show ?thesis
proof (intro has_derivativeI bounded_linear_scaleR_left tendstoI, fold norm_conv_dist)
  have cont_f1:  $\bigwedge t. t \in cbox\ a\ b \implies \text{continuous\_on } U\ (\lambda x. f\ x\ t)$ 
  by (auto simp: continuous_on_eq continuous_within intro!: has_derivative_continuous
fx)
  note [continuous_intros] = continuous_on_compose2[OF cont_f1]
  fix e'::real
  assume e' > 0
  define e where e = e' / (content (cbox a b) + 1)
  have e > 0 using <e' > 0> by (auto simp: e_def intro!: divide_pos_pos
add_nonneg_pos)
  from continuous_on_prod_compactE[OF cont_fx compact_cbox <x0  $\in U$ > <e
> 0>]
  obtain X0 where X0: x0  $\in X0$  open X0
  and fx_bound:  $\bigwedge x\ t. x \in X0 \cap U \implies t \in cbox\ a\ b \implies \text{norm } (fx\ x\ t - fx\ x0)
t) \leq e$ 
  unfolding split_beta fst_conv snd_conv
  by (metis dist_norm)

  note eventually_closed_segment[OF <open X0> <x0  $\in X0$ >, of U]
  moreover
  have  $\forall_F\ x\ \text{in at } x0\ \text{within } U. x \in X0$ 
  using <open X0> <x0  $\in X0$ > eventually_at_topological by blast
  moreover have  $\forall_F\ x\ \text{in at } x0\ \text{within } U. x \neq x0$ 
  by (auto simp: eventually_at_filter)
  moreover have  $\forall_F\ x\ \text{in at } x0\ \text{within } U. x \in U$ 
  by (auto simp: eventually_at_filter)
  ultimately
  show  $\forall_F\ x\ \text{in at } x0\ \text{within } U. \text{norm } ((?F\ x - ?F\ x0 - ?dF\ (x - x0)) /_R \text{norm }
(x - x0)) < e'$ 
  proof eventually_elim
  case (elim x)
  from elim have 0 < norm (x - x0) by simp
  have closed_segment x0 x  $\subseteq U$ 
  by (rule <convex U>[unfolded convex_contains_segment, rule_format, OF
<x0  $\in U$ > <x  $\in U$ >])
  from elim have [intro]: x  $\in U$  by auto
  have ?F x - ?F x0 - ?dF (x - x0) =

```

```

integral (cbox a b) (λy. f x y - f x0 y - fx x0 y (x - x0))
(is _ = ?id)
using ⟨x ≠ x0⟩
by (subst blinfun_apply_integral integral_diff,
    auto intro!: integrable_diff integrable_f2 continuous_intros
    intro: integrable_continuous)+
also
{
  fix t assume t: t ∈ (cbox a b)
  have seg: ∧t. t ∈ {0..1} ⇒ x0 + t *R (x - x0) ∈ X0 ∩ U
    using ⟨closed_segment x0 x ⊆ U⟩
    ⟨closed_segment x0 x ⊆ X0⟩
  by (force simp: closed_segment_def algebra_simps)
  from t have deriv:
    ((λx. f x t) has_derivative (fx y t)) (at y within X0 ∩ U)
  if y ∈ X0 ∩ U for y
  unfolding has_vector_derivative_def[symmetric]
  using that ⟨x ∈ X0⟩
  by (intro has_derivative_subset[OF fx]) auto
  have ∧x. x ∈ X0 ∩ U ⇒ onorm (blinfun_apply (fx x t) - (fx x0 t)) ≤ e
    using fx_bound t
  by (auto simp add: norm_blinfun_def fun_diff_def blinfun.bilinear_simps[symmetric])
  from differentiable_bound_linearization[OF seg deriv this] X0
  have norm (f x t - f x0 t - fx x0 t (x - x0)) ≤ e * norm (x - x0)
    by (auto simp add: ac_simps)
}
then have norm ?id ≤ integral (cbox a b) (λ_. e * norm (x - x0))
  by (intro integral_norm_bound_integral)
  (auto intro!: continuous_intros integrable_diff integrable_f2
  intro: integrable_continuous)
also have ... = content (cbox a b) * e * norm (x - x0)
  by simp
also have ... < e' * norm (x - x0)
proof (intro mult_strict_right_mono[OF _ ⟨0 < norm (x - x0)⟩])
  show content (cbox a b) * e < e'
    using ⟨e' > 0⟩ by (simp add: divide_simps e_def not_less)
qed
finally have norm (?F x - ?F x0 - ?dF (x - x0)) < e' * norm (x - x0) .
then show ?case
  by (auto simp: divide_simps)
qed
qed (rule blinfun.bounded_linear_right)
qed (auto intro!: derivative_eq_intros simp: blinfun.bilinear_simps)

lemma has_vector_derivative_eq_has_derivative_blinfun:
  (f has_vector_derivative f') (at x within U) ⟷
  (f has_derivative blinfun_scaleR_left f') (at x within U)
  by (simp add: has_vector_derivative_def)

```



```

lemma leibniz_rule_vector_derivative:
  fixes f::real  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'c::banach
  assumes fx:  $\bigwedge x t. x \in U \Rightarrow t \in \text{cbox } a \ b \Rightarrow$ 
     $((\lambda x. f \ x \ t) \text{ has\_vector\_derivative } (f \ x \ t)) \text{ (at } x \text{ within } U)$ 
  assumes integrable_f2:  $\bigwedge x. x \in U \Rightarrow (f \ x) \text{ integrable\_on } \text{cbox } a \ b$ 
  assumes cont_fx: continuous_on  $(U \times \text{cbox } a \ b) (\lambda(x, t). f \ x \ t)$ 
  assumes U:  $x0 \in U \text{ convex } U$ 
  shows  $((\lambda x. \text{integral } (\text{cbox } a \ b) (f \ x)) \text{ has\_vector\_derivative integral } (\text{cbox } a \ b)$ 
 $(f \ x0))$ 
     $(\text{at } x0 \text{ within } U)$ 
proof -
  note [continuous_intros] =
    continuous_on_compose2[OF cont_fx, where  $f = \lambda y. \text{Pair } x \ y$  for  $x$ ,
    unfolded split_beta fst_conv snd_conv]
  show ?thesis
    unfolding has_vector_derivative_eq_has_derivative_blinfun
  proof (rule has_derivative_eq_rhs [OF leibniz_rule[OF integrable_f2 U]])
    show continuous_on  $(U \times \text{cbox } a \ b) (\lambda(x, t). \text{blinfun\_scaleR\_left } (f \ x \ t))$ 
      using cont_fx by (auto simp: split_beta intro!: continuous_intros)
    show blinfun_apply  $(\text{integral } (\text{cbox } a \ b) (\lambda t. \text{blinfun\_scaleR\_left } (f \ x0 \ t))) =$ 
      blinfun_apply  $(\text{blinfun\_scaleR\_left } (\text{integral } (\text{cbox } a \ b) (f \ x0)))$ 
      by (subst integral_linear[symmetric])
      (auto simp: has_vector_derivative_def o_def
        intro!: integrable_continuous U continuous_intros bounded_linear_intros)
    qed (use fx in  $\langle \text{auto simp: has\_vector\_derivative\_def} \rangle$ )
  qed

```

```

lemma has_field_derivative_eq_has_derivative_blinfun:
   $(f \text{ has\_field\_derivative } f') \text{ (at } x \text{ within } U) \longleftrightarrow (f \text{ has\_derivative blinfun\_mult\_right } f') \text{ (at } x \text{ within } U)$ 
  by (simp add: has_field_derivative_def)

```

```

lemma leibniz_rule_field_derivative:
  fixes f::'a::{real_normed_field, banach}  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'a
  assumes fx:  $\bigwedge x t. x \in U \Rightarrow t \in \text{cbox } a \ b \Rightarrow ((\lambda x. f \ x \ t) \text{ has\_field\_derivative } f \ x \ t) \text{ (at } x \text{ within } U)$ 
  assumes integrable_f2:  $\bigwedge x. x \in U \Rightarrow (f \ x) \text{ integrable\_on } \text{cbox } a \ b$ 
  assumes cont_fx: continuous_on  $(U \times (\text{cbox } a \ b)) (\lambda(x, t). f \ x \ t)$ 
  assumes U:  $x0 \in U \text{ convex } U$ 
  shows  $((\lambda x. \text{integral } (\text{cbox } a \ b) (f \ x)) \text{ has\_field\_derivative integral } (\text{cbox } a \ b) (f \ x0)) \text{ (at } x0 \text{ within } U)$ 
proof -
  note [continuous_intros] =
    continuous_on_compose2[OF cont_fx, where  $f = \lambda y. \text{Pair } x \ y$  for  $x$ ,
    unfolded split_beta fst_conv snd_conv]
  have *:  $\text{blinfun\_mult\_right } (\text{integral } (\text{cbox } a \ b) (f \ x0)) =$ 
    integral  $(\text{cbox } a \ b) (\lambda t. \text{blinfun\_mult\_right } (f \ x0 \ t))$ 
    by (subst integral_linear[symmetric])
    (auto simp: has_vector_derivative_def o_def)

```

```

      intro!: integrable_continuous U continuous_intros bounded_linear_intros)
show ?thesis
  unfolding has_field_derivative_eq_has_derivative_blinfun
proof (rule has_derivative_eq_rhs [OF leibniz_rule[OF _ integrable_f2 _ U,
where fx= $\lambda x t. \text{blinfun\_mult\_right } (fx \ x \ t)$ ]])
  show continuous_on (U  $\times$  cbox a b) ( $\lambda(x, t). \text{blinfun\_mult\_right } (fx \ x \ t)$ )
  using cont_fx by (auto simp: split_beta intro!: continuous_intros)
  show blinfun_apply (integral (cbox a b) ( $\lambda t. \text{blinfun\_mult\_right } (fx \ x0 \ t)$ )) =
    blinfun_apply (blinfun_mult_right (integral (cbox a b) (fx x0)))
  by (subst integral_linear[symmetric])
    (auto simp: has_vector_derivative_def o_def
      intro!: integrable_continuous U continuous_intros bounded_linear_intros)
qed (use fx in <auto simp: has_field_derivative_def>)
qed

```

```

lemma leibniz_rule_field_differentiable:
  fixes f::'a::{real_normed_field, banach}  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'a
  assumes  $\bigwedge x t. x \in U \implies t \in \text{cbox } a \ b \implies ((\lambda x. f \ x \ t) \text{ has\_field\_derivative } fx$ 
 $x \ t)$  (at x within U)
  assumes  $\bigwedge x. x \in U \implies (f \ x) \text{ integrable\_on } \text{cbox } a \ b$ 
  assumes continuous_on (U  $\times$  (cbox a b)) ( $\lambda(x, t). f \ x \ t$ )
  assumes  $x0 \in U$  convex U
  shows ( $\lambda x. \text{integral } (\text{cbox } a \ b) (f \ x)$ ) field_differentiable at x0 within U
  using leibniz_rule_field_derivative[OF assms] by (auto simp: field_differentiable_def)

```

6.15.43 Exchange uniform limit and integral

```

lemma uniform_limit_integral_cbox:
  fixes f::'a  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'c::banach
  assumes u: uniform_limit (cbox a b) f g F
  assumes c:  $\bigwedge n. \text{continuous\_on } (\text{cbox } a \ b) (f \ n)$ 
  assumes [simp]:  $F \neq \text{bot}$ 
  obtains I J where
     $\bigwedge n. (f \ n \text{ has\_integral } I \ n) (\text{cbox } a \ b)$ 
     $(g \text{ has\_integral } J) (\text{cbox } a \ b)$ 
     $(I \longrightarrow J) F$ 
proof -
  have fi[simp]:  $f \ n \text{ integrable\_on } (\text{cbox } a \ b)$  for n
  by (auto intro!: integrable_continuous assms)
  then obtain I where I:  $\bigwedge n. (f \ n \text{ has\_integral } I \ n) (\text{cbox } a \ b)$ 
  by atomize_elim (auto simp: integrable_on_def intro!: choice)

  moreover
  have gi[simp]:  $g \text{ integrable\_on } (\text{cbox } a \ b)$ 
  by (auto intro!: integrable_continuous uniform_limit_theorem[OF _ u] eventuallyI c)
  then obtain J where J:  $(g \text{ has\_integral } J) (\text{cbox } a \ b)$ 
  by blast

```

```

moreover
have  $(I \longrightarrow J) F$ 
proof cases
  assume  $\text{content } (\text{cbox } a \ b) = 0$ 
  hence  $I = (\lambda \_. 0) \ J = 0$ 
  by  $(\text{auto intro!}; \text{has\_integral\_unique } I \ J)$ 
  thus ?thesis by simp
next
assume  $\text{content\_nonzero}: \text{content } (\text{cbox } a \ b) \neq 0$ 
show ?thesis
proof  $(\text{rule tendstoI})$ 
  fix  $e::\text{real}$ 
  assume  $e > 0$ 
  define  $e'$  where  $e' = e/2$ 
  with  $\langle e > 0 \rangle$  have  $e' > 0$  by simp
  then have  $\forall_F n \text{ in } F. \forall x \in \text{cbox } a \ b. \text{norm } (f \ n \ x - g \ x) < e' / \text{content } (\text{cbox } a \ b)$ 
    using  $u \ \text{content\_nonzero}$  by  $(\text{auto simp: uniform\_limit\_iff dist\_norm zero\_less\_measure\_iff})$ 
  then show  $\forall_F n \text{ in } F. \text{dist } (I \ n) \ J < e$ 
  proof eventually_elim
    case  $(\text{elim } n)$ 
    have  $I \ n = \text{integral } (\text{cbox } a \ b) (f \ n)$ 
     $J = \text{integral } (\text{cbox } a \ b) g$ 
    using  $I[\text{of } n] \ J$  by  $(\text{simp\_all add: integral\_unique})$ 
    then have  $\text{dist } (I \ n) \ J = \text{norm } (\text{integral } (\text{cbox } a \ b) (\lambda x. f \ n \ x - g \ x))$ 
    by  $(\text{simp add: integral\_diff dist\_norm})$ 
    also have  $\dots \leq \text{integral } (\text{cbox } a \ b) (\lambda x. (e' / \text{content } (\text{cbox } a \ b)))$ 
    using elim
    by  $(\text{intro integral\_norm\_bound\_integral}) (\text{auto intro!}; \text{integrable\_diff})$ 
    also have  $\dots < e$ 
    using  $\langle 0 < e \rangle$ 
    by  $(\text{simp add: } e'\text{\_def})$ 
    finally show ?case .
  qed
qed
qed
ultimately show ?thesis ..
qed

```

```

lemma uniform_limit_integral:
  fixes  $f::'a \Rightarrow 'b::\text{ordered\_euclidean\_space} \Rightarrow 'c::\text{banach}$ 
  assumes  $u: \text{uniform\_limit } \{a..b\} \ f \ g \ F$ 
  assumes  $c: \bigwedge n. \text{continuous\_on } \{a..b\} (f \ n)$ 
  assumes  $[\text{simp}]: F \neq \text{bot}$ 
  obtains  $I \ J$  where
     $\bigwedge n. (f \ n \ \text{has\_integral } I \ n) \ \{a..b\}$ 
     $(g \ \text{has\_integral } J) \ \{a..b\}$ 
     $(I \longrightarrow J) \ F$ 

```

by (metis interval_cbox assms uniform_limit_integral_cbox)

6.15.44 Integration by parts

lemma *integration_by_parts_interior_strong*:
fixes *prod* :: $\_ \Rightarrow \_ \Rightarrow 'b :: \text{banach}$
assumes *bilinear*: *bounded_bilinear* (*prod*)
assumes *s*: *finite s* **and** *le*: $a \leq b$
assumes *cont* [*continuous_intros*]: *continuous_on* $\{a..b\}$ *f* *continuous_on* $\{a..b\}$
g
assumes *deriv*: $\bigwedge x. x \in \{a <..< b\} - s \implies (f \text{ has\_vector\_derivative } f' x) (at x)$
 $\bigwedge x. x \in \{a <..< b\} - s \implies (g \text{ has\_vector\_derivative } g' x) (at x)$
assumes *int*: $((\lambda x. \text{prod } (f x) (g' x)) \text{ has\_integral } (prod (f b) (g b) - prod (f a) (g a) - y)) \{a..b\}$
shows $((\lambda x. \text{prod } (f' x) (g x)) \text{ has\_integral } y) \{a..b\}$
proof –
interpret *bounded_bilinear prod* **by** *fact*
have $((\lambda x. \text{prod } (f x) (g' x) + \text{prod } (f' x) (g x)) \text{ has\_integral } (prod (f b) (g b) - prod (f a) (g a))) \{a..b\}$
using *deriv* **by** (*intro fundamental_theorem_of_calculus_interior_strong*[*OF s le*])
 $(\text{auto intro! : continuous_intros continuous\_on has\_vector\_derivative})$
from *has_integral_diff*[*OF this int*] **show** ?thesis **by** (*simp add: algebra_simps*)
qed

lemma *integration_by_parts_interior*:
fixes *prod* :: $\_ \Rightarrow \_ \Rightarrow 'b :: \text{banach}$
assumes *bounded_bilinear* (*prod*) $a \leq b$
 $\text{continuous\_on } \{a..b\} f \text{ continuous\_on } \{a..b\} g$
assumes $\bigwedge x. x \in \{a <..< b\} \implies (f \text{ has\_vector\_derivative } f' x) (at x)$
 $\bigwedge x. x \in \{a <..< b\} \implies (g \text{ has\_vector\_derivative } g' x) (at x)$
assumes $((\lambda x. \text{prod } (f x) (g' x)) \text{ has\_integral } (prod (f b) (g b) - prod (f a) (g a) - y)) \{a..b\}$
shows $((\lambda x. \text{prod } (f' x) (g x)) \text{ has\_integral } y) \{a..b\}$
by (*rule integration_by_parts_interior_strong*[*of* $\_ \{ \} \_ \_ f g f' g'$]) (*insert assms, simp_all*)

lemma *integration_by_parts*:
fixes *prod* :: $\_ \Rightarrow \_ \Rightarrow 'b :: \text{banach}$
assumes *bounded_bilinear* (*prod*) $a \leq b$
 $\text{continuous\_on } \{a..b\} f \text{ continuous\_on } \{a..b\} g$
assumes $\bigwedge x. x \in \{a..b\} \implies (f \text{ has\_vector\_derivative } f' x) (at x)$
 $\bigwedge x. x \in \{a..b\} \implies (g \text{ has\_vector\_derivative } g' x) (at x)$
assumes $((\lambda x. \text{prod } (f x) (g' x)) \text{ has\_integral } (prod (f b) (g b) - prod (f a) (g a) - y)) \{a..b\}$
shows $((\lambda x. \text{prod } (f' x) (g x)) \text{ has\_integral } y) \{a..b\}$
by (*rule integration_by_parts_interior*[*of* $\_ \_ \_ f g f' g'$]) (*insert assms, simp_all*)

lemma *integrable_by_parts_interior_strong*:

```

fixes prod ::  $\_ \Rightarrow \_ \Rightarrow 'b :: \text{banach}$ 
assumes bilinear: bounded_bilinear (prod)
assumes s: finite s and le:  $a \leq b$ 
assumes cont [continuous_intros]: continuous_on {a..b} f continuous_on {a..b}
g
assumes deriv:  $\bigwedge x. x \in \{a <..<b\} - s \implies (f \text{ has\_vector\_derivative } f' x) (at x)$ 
 $\bigwedge x. x \in \{a <..<b\} - s \implies (g \text{ has\_vector\_derivative } g' x) (at x)$ 
assumes int:  $(\lambda x. \text{prod } (f x) (g' x)) \text{ integrable\_on } \{a..b\}$ 
shows  $(\lambda x. \text{prod } (f' x) (g x)) \text{ integrable\_on } \{a..b\}$ 
proof -
from int obtain I where  $((\lambda x. \text{prod } (f x) (g' x)) \text{ has\_integral } I) \{a..b\}$ 
unfolding integrable_on_def by blast
hence  $((\lambda x. \text{prod } (f x) (g' x)) \text{ has\_integral } (\text{prod } (f b) (g b) - \text{prod } (f a) (g a) -$ 
 $(\text{prod } (f b) (g b) - \text{prod } (f a) (g a) - I))) \{a..b\}$  by simp
from integration_by_parts_interior_strong[OF assms(1-7) this]
show ?thesis unfolding integrable_on_def by blast
qed

```

```

lemma integrable_by_parts_interior:
fixes prod ::  $\_ \Rightarrow \_ \Rightarrow 'b :: \text{banach}$ 
assumes bounded_bilinear (prod)  $a \leq b$ 
 $\text{continuous\_on } \{a..b\} f \text{ continuous\_on } \{a..b\} g$ 
assumes  $\bigwedge x. x \in \{a <..<b\} \implies (f \text{ has\_vector\_derivative } f' x) (at x)$ 
 $\bigwedge x. x \in \{a <..<b\} \implies (g \text{ has\_vector\_derivative } g' x) (at x)$ 
assumes  $(\lambda x. \text{prod } (f x) (g' x)) \text{ integrable\_on } \{a..b\}$ 
shows  $(\lambda x. \text{prod } (f' x) (g x)) \text{ integrable\_on } \{a..b\}$ 
by (rule integrable_by_parts_interior_strong[of _ {} _ _ f g f' g']) (insert
assms, simp_all)

```

```

lemma integrable_by_parts:
fixes prod ::  $\_ \Rightarrow \_ \Rightarrow 'b :: \text{banach}$ 
assumes bounded_bilinear (prod)  $a \leq b$ 
 $\text{continuous\_on } \{a..b\} f \text{ continuous\_on } \{a..b\} g$ 
assumes  $\bigwedge x. x \in \{a..b\} \implies (f \text{ has\_vector\_derivative } f' x) (at x)$ 
 $\bigwedge x. x \in \{a..b\} \implies (g \text{ has\_vector\_derivative } g' x) (at x)$ 
assumes  $(\lambda x. \text{prod } (f x) (g' x)) \text{ integrable\_on } \{a..b\}$ 
shows  $(\lambda x. \text{prod } (f' x) (g x)) \text{ integrable\_on } \{a..b\}$ 
by (rule integrable_by_parts_interior_strong[of _ {} _ _ f g f' g']) (insert
assms, simp_all)

```

6.15.45 Integration by substitution

```

lemma has_integral_substitution_general:
fixes f ::  $\text{real} \Rightarrow 'a :: \text{euclidean\_space}$  and g ::  $\text{real} \Rightarrow \text{real}$ 
assumes s: finite s and le:  $a \leq b$ 
and subset:  $g \text{ ' } \{a..b\} \subseteq \{c..d\}$ 
and f [continuous_intros]: continuous_on {c..d} f
and g [continuous_intros]: continuous_on {a..b} g
and deriv [derivative_intros]:

```

```

       $\bigwedge x. x \in \{a..b\} - s \implies (g \text{ has\_field\_derivative } g' x) \text{ (at } x \text{ within } \{a..b\})$ 
    shows  $((\lambda x. g' x *_R f (g x)) \text{ has\_integral } (\text{integral } \{g \text{ a..g } b\} f - \text{integral } \{g$ 
 $b..g a\} f)) \{a..b\}$ 
  proof -
    let ?F =  $\lambda x. \text{integral } \{c..g x\} f$ 
    have cont_int: continuous_on  $\{a..b\}$  ?F
    by (rule continuous_on_compose2[OF _ g subset] indefinite_integral_continuous_1
        f integrable_continuous_real)+
    have deriv:  $((\lambda x. \text{integral } \{c..x\} f) \circ g) \text{ has\_vector\_derivative } g' x *_R f (g x)$ 
 $(\text{at } x \text{ within } \{a..b\})$  if  $x \in \{a..b\} - s$  for  $x$ 
  proof (rule has_vector_derivative_eq_rhs [OF vector_diff_chain_within refl])
    show  $(g \text{ has\_vector\_derivative } g' x) \text{ (at } x \text{ within } \{a..b\})$ 
      using deriv has_field_derivative_iff_has_vector_derivative that by blast
    show  $((\lambda x. \text{integral } \{c..x\} f) \text{ has\_vector\_derivative } f (g x))$ 
 $(\text{at } (g x) \text{ within } g^{-1} \{a..b\})$ 
      using that le subset
    by (blast intro: has_vector_derivative_within_subset integral_has_vector_derivative
f)
  qed
  have deriv:  $(?F \text{ has\_vector\_derivative } g' x *_R f (g x))$ 
 $(\text{at } x) \text{ if } x \in \{a..b\} - (s \cup \{a, b\})$  for  $x$ 
    using deriv[of x] that by (simp add: at_within_Icc_at_o_def)
  have  $((\lambda x. g' x *_R f (g x)) \text{ has\_integral } (?F b - ?F a)) \{a..b\}$ 
    using le cont_int s deriv cont_int
    by (intro fundamental_theorem_of_calculus_interior_strong[of s  $\cup \{a, b\}$ ])
simp_all
  also
    from subset have  $g x \in \{c..d\}$  if  $x \in \{a..b\}$  for  $x$  using that by blast
    from this[of a] this[of b] le have  $cd: c \leq g a \wedge g b \leq d \wedge c \leq g b \wedge g a \leq d$  by auto
    have  $\text{integral } \{c..g b\} f - \text{integral } \{c..g a\} f = \text{integral } \{g a..g b\} f - \text{integral}$ 
 $\{g b..g a\} f$ 
  proof cases
    assume  $g a \leq g b$ 
    note le = le this
    from cd have  $\text{integral } \{c..g a\} f + \text{integral } \{g a..g b\} f = \text{integral } \{c..g b\} f$ 
    by (intro integral_combine integrable_continuous_real continuous_on_subset[OF
f] le) simp_all
    with le show ?thesis
      by (cases  $g a = g b$ ) (simp_all add: algebra_simps)
  next
    assume less:  $\neg g a \leq g b$ 
    then have  $g a \geq g b$  by simp
    note le = le this
    from cd have  $\text{integral } \{c..g b\} f + \text{integral } \{g b..g a\} f = \text{integral } \{c..g a\} f$ 
    by (intro integral_combine integrable_continuous_real continuous_on_subset[OF
f] le) simp_all
    with less show ?thesis
      by (simp_all add: algebra_simps)
  qed

```

finally show ?thesis .
qed

lemma has_integral_substitution_strong:

fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ and $g :: \text{real} \Rightarrow \text{real}$
 assumes s : finite s and le : $a \leq b$ $g\ a \leq g\ b$
 and subset: $g\ ' \{a..b\} \subseteq \{c..d\}$
 and f [continuous_intros]: continuous_on $\{c..d\}$ f
 and g [continuous_intros]: continuous_on $\{a..b\}$ g
 and deriv [derivative_intros]:
 $\bigwedge x. x \in \{a..b\} - s \implies (g\ \text{has\_field\_derivative}\ g'\ x)\ (\text{at } x\ \text{within } \{a..b\})$
 shows $((\lambda x. g'\ x *_R f\ (g\ x))\ \text{has\_integral}\ (\text{integral}\ \{g\ a..g\ b\}\ f))\ \{a..b\}$
 using has_integral_substitution_general[OF $s\ le(1)\ \text{subset}\ f\ g\ \text{deriv}\ le(2)$]
 by (cases $g\ a = g\ b$) auto

lemma has_integral_substitution:

fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ and $g :: \text{real} \Rightarrow \text{real}$
 assumes $a \leq b$ $g\ a \leq g\ b$ $g\ ' \{a..b\} \subseteq \{c..d\}$
 and continuous_on $\{c..d\}$ f
 and $\bigwedge x. x \in \{a..b\} \implies (g\ \text{has\_field\_derivative}\ g'\ x)\ (\text{at } x\ \text{within } \{a..b\})$
 shows $((\lambda x. g'\ x *_R f\ (g\ x))\ \text{has\_integral}\ (\text{integral}\ \{g\ a..g\ b\}\ f))\ \{a..b\}$
 by (intro has_integral_substitution_strong[of $\{ \}$ $a\ b\ g\ c\ d$] assms)
 (auto intro: DERIV_continuous_on assms)

lemma integral_shift:

fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$
 assumes cont: continuous_on $\{a + c..b + c\}$ f
 shows $\text{integral}\ \{a..b\}\ (f \circ (\lambda x. x + c)) = \text{integral}\ \{a + c..b + c\}\ f$
 proof (cases $a \leq b$)
 case True
 have $((\lambda x. 1 *_R f\ (x + c))\ \text{has\_integral}\ \text{integral}\ \{a+c..b+c\}\ f)\ \{a..b\}$
 using True cont
 by (intro has_integral_substitution[where $c = a + c$ and $d = b + c$])
 (auto intro!: derivative_eq_intros)
 thus ?thesis by (simp add: has_integral_iff o_def)
 qed auto

6.15.46 Compute a double integral using iterated integrals and switching the order of integration

lemma continuous_on_imp_integrable_on_Pair1:

fixes $f :: \_ \Rightarrow 'b::\text{banach}$
 assumes con: continuous_on (cbox $(a,c)\ (b,d)$) f and x : $x \in \text{cbox}\ a\ b$
 shows $(\lambda y. f\ (x, y))\ \text{integrable\_on}\ (\text{cbox}\ c\ d)$
 proof –
 have $f \circ (\lambda y. (x, y))\ \text{integrable\_on}\ (\text{cbox}\ c\ d)$
 proof (intro integrable_continuous continuous_on_compose [OF _ continuous_on_subset [OF con]])
 show continuous_on (cbox $c\ d$) (Pair x)

```

      by (simp add: continuous_on_Pair)
    show Pair x ' cbox c d  $\subseteq$  cbox (a,c) (b,d)
      using x by blast
  qed
  then show ?thesis
    by (simp add: o_def)
qed

lemma integral_integrable_2dim:
  fixes f :: ('a::euclidean_space * 'b::euclidean_space)  $\Rightarrow$  'c::banach
  assumes continuous_on (cbox (a,c) (b,d)) f
  shows ( $\lambda x$ . integral (cbox c d) ( $\lambda y$ . f (x,y))) integrable_on cbox a b
proof (cases content(cbox c d) = 0)
case True
  then show ?thesis
    by (simp add: True integrable_const)
next
case False
  have uc: uniformly_continuous_on (cbox (a,c) (b,d)) f
    by (simp add: assms compact_cbox compact_uniformly_continuous)
  { fix x::'a and e::real
    assume x: x  $\in$  cbox a b and e: 0 < e
    then have e2_gt: 0 < e/2 / content (cbox c d) and e2_less: e/2 / content
      (cbox c d) * content (cbox c d) < e
      by (auto simp: False content_lt_nz e)
    then obtain dd
      where dd:  $\bigwedge x x'. \llbracket x \in \text{cbox } (a, c) (b, d); x' \in \text{cbox } (a, c) (b, d); \text{norm } (x' - x) < dd \rrbracket$ 
       $\implies \text{norm } (f x' - f x) \leq e/(2 * \text{content } (\text{cbox } c d)) \text{ } dd > 0$ 
      using uc [unfolded uniformly_continuous_on_def, THEN spec, of e/(2 *
        content (cbox c d))]
      by (auto simp: dist_norm intro: less_imp_le)
    have  $\exists \text{delta} > 0. \forall x' \in \text{cbox } a b. \text{norm } (x' - x) < \text{delta} \longrightarrow \text{norm } (\text{integral } (\text{cbox } c d) (\lambda u. f (x', u) - f (x, u))) < e$ 
      using dd e2_gt assms x
      apply (rule_tac x=dd in exI)
      apply clarify
      apply (rule le_less_trans [OF integrable_bound e2_less])
      apply (auto intro: integrable_diff continuous_on_imp_integrable_on_Pair1)
      done
    } note * = this
  show ?thesis
proof (rule integrable_continuous)
  show continuous_on (cbox a b) ( $\lambda x$ . integral (cbox c d) ( $\lambda y$ . f (x, y)))
    by (simp add: * continuous_on_iff dist_norm integral_diff [symmetric]
      continuous_on_imp_integrable_on_Pair1 [OF assms])
  qed
qed

```


lemma *integral_split*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::\{real\_normed\_vector, complete\_space\}$
assumes $f: f \text{ integrable\_on } (cbox\ a\ b)$
and $k: k \in Basis$
shows $integral\ (cbox\ a\ b)\ f =$
 $integral\ (cbox\ a\ b \cap \{x. x \cdot k \leq c\})\ f +$
 $integral\ (cbox\ a\ b \cap \{x. x \cdot k \geq c\})\ f$
using $k\ f$
by (*auto simp: has_integral_integral intro: integral_unique [OF has_integral_split]*)

lemma *integral_swap_operativeI*:
fixes $f :: ('a::euclidean\_space * 'b::euclidean\_space) \Rightarrow 'c::banach$
assumes $f: continuous\_on\ s\ f$ **and** $e: 0 < e$
shows *operative conj True*
 $(\lambda k. \forall a\ b\ c\ d.$
 $cbox\ (a,c)\ (b,d) \subseteq k \wedge cbox\ (a,c)\ (b,d) \subseteq s$
 $\longrightarrow norm(integral\ (cbox\ (a,c)\ (b,d))\ f -$
 $integral\ (cbox\ a\ b)\ (\lambda x. integral\ (cbox\ c\ d)\ (\lambda y. f((x,y))))$
 $\leq e * content\ (cbox\ (a,c)\ (b,d)))$

proof (*standard, auto*)
fix $a::'a$ **and** $c::'b$ **and** $b::'a$ **and** $d::'b$ **and** $u::'a$ **and** $v::'a$ **and** $w::'b$ **and** $z::'b$
assume $*$: $box\ (a, c)\ (b, d) = \{\}$
and $cb1$: $cbox\ (u, w)\ (v, z) \subseteq cbox\ (a, c)\ (b, d)$
and $cb2$: $cbox\ (u, w)\ (v, z) \subseteq s$
then have $c0$: $content\ (cbox\ (a, c)\ (b, d)) = 0$
using $*$ **unfolding** *content_eq_0_interior* **by** *simp*
have $c0'$: $content\ (cbox\ (u, w)\ (v, z)) = 0$
by (*fact content_0_subset [OF c0 cb1]*)
show $norm\ (integral\ (cbox\ (u,w)\ (v,z))\ f - integral\ (cbox\ u\ v)\ (\lambda x. integral\ (cbox\ w\ z)\ (\lambda y. f\ (x, y))))$
 $\leq e * content\ (cbox\ (u,w)\ (v,z))$
using *content_cbox_pair_eq0_D* [*OF c0'*]
by (*force simp add: c0'*)

next

fix $a::'a$ **and** $c::'b$ **and** $b::'a$ **and** $d::'b$
and $M::real$ **and** $i::'a$ **and** $j::'b$
and $u::'a$ **and** $v::'a$ **and** $w::'b$ **and** $z::'b$
assume ij : $(i,j) \in Basis$
and $n1$: $\forall a'\ b'\ c'\ d'.$
 $cbox\ (a',c')\ (b',d') \subseteq cbox\ (a,c)\ (b,d) \wedge$
 $cbox\ (a',c')\ (b',d') \subseteq \{x. x \cdot (i,j) \leq M\} \wedge cbox\ (a',c')\ (b',d') \subseteq s \longrightarrow$
 $norm\ (integral\ (cbox\ (a',c')\ (b',d'))\ f - integral\ (cbox\ a'\ b')\ (\lambda x.$
 $integral\ (cbox\ c'\ d')\ (\lambda y. f\ (x,y))))$
 $\leq e * content\ (cbox\ (a',c')\ (b',d'))$
and $n2$: $\forall a'\ b'\ c'\ d'.$
 $cbox\ (a',c')\ (b',d') \subseteq cbox\ (a,c)\ (b,d) \wedge$
 $cbox\ (a',c')\ (b',d') \subseteq \{x. M \leq x \cdot (i,j)\} \wedge cbox\ (a',c')\ (b',d') \subseteq s \longrightarrow$
 $norm\ (integral\ (cbox\ (a',c')\ (b',d'))\ f - integral\ (cbox\ a'\ b')\ (\lambda x.$
 $integral\ (cbox\ c'\ d')\ (\lambda y. f\ (x,y))))$

```

      ≤ e * content (cbox (a',c') (b',d'))
    and subs: cbox (u,w) (v,z) ⊆ cbox (a,c) (b,d) cbox (u,w) (v,z) ⊆ s
  have fcont: continuous_on (cbox (u, w) (v, z)) f
    using assms(1) continuous_on_subset subs(2) by blast
  then have fint: f integrable_on cbox (u, w) (v, z)
    by (metis integrable_continuous)
  consider i ∈ Basis j=0 | j ∈ Basis i=0 using ij
    by (auto simp: Euclidean_Space.Basis_prod_def)
  then show norm (integral (cbox (u,w) (v,z)) f - integral (cbox u v) (λx. integral
(cbox w z) (λy. f (x,y))))
    ≤ e * content (cbox (u,w) (v,z)) (is ?normle)
  proof cases
  case 1
  then have e: e * content (cbox (u, w) (v, z)) =
    e * (content (cbox u v ∩ {x. x • i ≤ M}) * content (cbox w z)) +
    e * (content (cbox u v ∩ {x. M ≤ x • i}) * content (cbox w z))
    by (simp add: content_split [where c=M] content_Pair algebra_simps)
  have *: integral (cbox u v) (λx. integral (cbox w z) (λy. f (x, y))) =
    integral (cbox u v ∩ {x. x • i ≤ M}) (λx. integral (cbox w z) (λy. f
(x, y))) +
    integral (cbox u v ∩ {x. M ≤ x • i}) (λx. integral (cbox w z) (λy. f
(x, y)))
    using 1 f subs integral_integrable_2dim continuous_on_subset
    by (blast intro: integral_split)
  show ?normle
    apply (rule norm_diff2 [OF integral_split [where c=M, OF fint ij] * e])
    using 1 subs
    apply (simp_all add: cbox_Pair_eq setcomp_dot1 [of λu. M≤u] setcomp_dot1
[of λu. u≤M] Sigma_Int_Paircomp1)
    apply (simp_all add: interval_split ij flip: cbox_Pair_eq content_Pair)
    apply (force simp flip: interval_split intro!: n1 [rule_format])
    apply (force simp flip: interval_split intro!: n2 [rule_format])
    done
  next
  case 2
  then have e: e * content (cbox (u, w) (v, z)) =
    e * (content (cbox u v) * content (cbox w z ∩ {x. x • j ≤ M})) +
    e * (content (cbox u v) * content (cbox w z ∩ {x. M ≤ x • j}))
    by (simp add: content_split [where c=M] content_Pair algebra_simps)
  have (λx. integral (cbox w z ∩ {x. x • j ≤ M}) (λy. f (x, y))) integrable_on
cbox u v
    (λx. integral (cbox w z ∩ {x. M ≤ x • j}) (λy. f (x, y))) integrable_on cbox
u v
    using 2 subs
    apply (simp_all add: interval_split)
    apply (rule integral_integrable_2dim [OF continuous_on_subset [OF f]]);
  auto simp: cbox_Pair_eq interval_split [symmetric])+
  done
  with 2 have *: integral (cbox u v) (λx. integral (cbox w z) (λy. f (x, y))) =

```

```

      integral (cbox u v) (λx. integral (cbox w z ∩ {x. x • j ≤ M}) (λy. f
(x, y))) +
      integral (cbox u v) (λx. integral (cbox w z ∩ {x. M ≤ x • j}) (λy. f
(x, y)))
    by (simp add: integral_add [symmetric] integral_split [symmetric]
        continuous_on_imp_integrable_on_Pair1 [OF fcont] cong:
integral_cong)
    show ?normle
    apply (rule norm_diff2 [OF integral_split [where c=M, OF fint ij] * e])
    using 2 subs
    apply (simp_all add: cbox_Pair_eq setcomp_dot2 [of λu. M≤u] setcomp_dot2
[of λu. u≤M] Sigma_Int_Paircomp2)
    apply (simp_all add: interval_split ij flip: cbox_Pair_eq content_Pair)
    apply (force simp flip: interval_split intro!: n1 [rule_format])
    apply (force simp flip: interval_split intro!: n2 [rule_format])
    done
  qed
qed

lemma integral_Pair_const:
  integral (cbox (a,c) (b,d)) (λx. k) =
  integral (cbox a b) (λx. integral (cbox c d) (λy. k))
  by (simp add: content_Pair)

lemma integral_prod_continuous:
  fixes f :: ('a::euclidean_space * 'b::euclidean_space) ⇒ 'c::banach
  assumes continuous_on (cbox (a, c) (b, d)) f (is continuous_on ?CBOX f)
  shows integral (cbox (a, c) (b, d)) f = integral (cbox a b) (λx. integral (cbox c
d) (λy. f (x, y)))
  proof (cases content ?CBOX = 0)
    case True
    then show ?thesis
    by (auto simp: content_Pair)
  next
    case False
    then have cbp: content ?CBOX > 0
    using content_lt_nz by blast
    have norm (integral ?CBOX f - integral (cbox a b) (λx. integral (cbox c d) (λy.
f (x,y)))) = 0
    proof (rule dense_eq0_I, simp)
      fix e :: real
      assume 0 < e
      with ‹content ?CBOX > 0› have 0 < e/content ?CBOX
      by simp
      have f_int_acbd: f integrable_on ?CBOX
      by (rule integrable_continuous [OF assms])
      { fix p
        assume p: p division_of ?CBOX
        then have finite p

```

```

    by blast
  define e' where e' = e/content ?CBOX
  with ⟨0 < e⟩ ⟨0 < e/content ?CBOX⟩
  have 0 < e'
    by simp
  interpret operative conj True
    λk. ∀ a' b' c' d'.
      cbox (a', c') (b', d') ⊆ k ∧ cbox (a', c') (b', d') ⊆ ?CBOX
      ⟶ norm (integral (cbox (a', c') (b', d')) f -
        integral (cbox a' b') (λx. integral (cbox c' d') (λy. f ((x, y)))))
        ≤ e' * content (cbox (a', c') (b', d'))
    using assms ⟨0 < e'⟩ by (rule integral_swap_operativeI)
  have norm (integral ?CBOX f - integral (cbox a b) (λx. integral (cbox c d)
    (λy. f (x, y))))
    ≤ e' * content ?CBOX
    if ∧ t u v w z. t ∈ p ⟹ cbox (u, w) (v, z) ⊆ t ⟹ cbox (u, w) (v, z) ⊆
    ?CBOX
    ⟹ norm (integral (cbox (u, w) (v, z)) f -
      integral (cbox u v) (λx. integral (cbox w z) (λy. f (x, y))))
      ≤ e' * content (cbox (u, w) (v, z))
    using that division [of p (a, c) (b, d)] p ⟨finite p⟩ by (auto simp add:
    comm_monoid_set_F_and)
  with False have norm (integral ?CBOX f - integral (cbox a b) (λx. integral
    (cbox c d) (λy. f (x, y))))
    ≤ e
    if ∧ t u v w z. t ∈ p ⟹ cbox (u, w) (v, z) ⊆ t ⟹ cbox (u, w) (v, z) ⊆
    ?CBOX
    ⟹ norm (integral (cbox (u, w) (v, z)) f -
      integral (cbox u v) (λx. integral (cbox w z) (λy. f (x, y))))
      ≤ e * content (cbox (u, w) (v, z)) / content ?CBOX
    using that by (simp add: e'_def)
  } note op_acbd = this
  { fix k::real and D and u::'a and v w and z::'b and t1 t2 l
    assume k: 0 < k
    and nf: ∧ x y u v.
      ⟦x ∈ cbox a b; y ∈ cbox c d; u ∈ cbox a b; v ∈ cbox c d; norm (u-x,
      v-y) < k⟧
      ⟹ norm (f(u,v) - f(x,y)) < e/(2 * (content ?CBOX))
    and p_acbd: D tagged_division_of cbox (a,c) (b,d)
    and fine: (λx. ball x k) fine D ((t1,t2), l) ∈ D
    and uvwz_sub: cbox (u,w) (v,z) ⊆ l cbox (u,w) (v,z) ⊆ cbox (a,c) (b,d)
    have t: t1 ∈ cbox a b t2 ∈ cbox c d
      by (meson fine p_acbd cbox_Pair_iff tag_in_interval)+
    have ls: l ⊆ ball (t1,t2) k
      using fine by (simp add: fine_def Ball_def)
    { fix x1 x2
      assume xuvwz: x1 ∈ cbox u v x2 ∈ cbox w z
      then have x: x1 ∈ cbox a b x2 ∈ cbox c d
        using uvwz_sub by auto
    }
  }

```

```

    have norm (x1 - t1, x2 - t2) = norm (t1 - x1, t2 - x2)
      by (simp add: norm_Pair norm_minus_commute)
    also have norm (t1 - x1, t2 - x2) < k
      using xuvwz ls uvwz_sub unfolding ball_def
      by (force simp add: cbox_Pair_eq dist_norm )
    finally have norm (f (x1,x2) - f (t1,t2)) ≤ e/content ?CBOX/2
      using nf [OF t x] by simp
  } note nf' = this
  have f_int_uvwz: f integrable_on cbox (u,w) (v,z)
    using f_int_acbd uvwz_sub integrable_on_subcbox by blast
  have f_int_uv:  $\bigwedge x. x \in \text{cbox } u \ v \implies (\lambda y. f (x,y)) \text{ integrable\_on } \text{cbox } w \ z$ 
    using assms continuous_on_subset uvwz_sub
    by (blast intro: continuous_on_imp_integrable_on_Pair1)
  have 1: norm (integral (cbox (u,w) (v,z)) f - integral (cbox (u,w) (v,z)) ( $\lambda x. f (t1,t2)$ ))
    ≤ e * content (cbox (u,w) (v,z)) / content ?CBOX/2
    using cbp ⟨0 < e/content ?CBOX⟩ nf'
    apply (simp only: integral_diff [symmetric] f_int_uvwz integrable_const)
    apply (auto simp: integrable_diff f_int_uvwz integrable_const intro: order_trans [OF integrable_bound [of e/content ?CBOX/2]])
    done
  have int_integrable: ( $\lambda x. \text{integral } (\text{cbox } w \ z) (\lambda y. f (x, y))$ ) integrable_on cbox u v
    using assms integral_integrable_2dim continuous_on_subset uvwz_sub(2)
  by blast
  have normint_wz:
     $\bigwedge x. x \in \text{cbox } u \ v \implies$ 
    norm (integral (cbox w z) ( $\lambda y. f (x, y)$ ) - integral (cbox w z) ( $\lambda y. f (t1, t2)$ ))
    ≤ e * content (cbox w z) / content (cbox (a, c) (b, d))/2
    using cbp ⟨0 < e/content ?CBOX⟩ nf'
    apply (simp only: integral_diff [symmetric] f_int_uv integrable_const)
    apply (auto simp: integrable_diff f_int_uv integrable_const intro: order_trans [OF integrable_bound [of e/content ?CBOX/2]])
    done
  have norm (integral (cbox u v)
    ( $\lambda x. \text{integral } (\text{cbox } w \ z) (\lambda y. f (x,y)) - \text{integral } (\text{cbox } w \ z) (\lambda y. f (t1,t2))$ ))
    ≤ e * content (cbox w z) / content ?CBOX/2 * content (cbox u v)
    using cbp ⟨0 < e/content ?CBOX⟩
    apply (intro integrable_bound [OF __ normint_wz])
    apply (auto simp: field_split_simps integrable_diff int_integrable integrable_const)
    done
  also have ... ≤ e * content (cbox (u,w) (v,z)) / content ?CBOX/2
    by (simp add: content_Pair field_split_simps)
  finally have 2: norm (integral (cbox u v) ( $\lambda x. \text{integral } (\text{cbox } w \ z) (\lambda y. f (x,y)) -$ 
    integral (cbox u v) ( $\lambda x. \text{integral } (\text{cbox } w \ z) (\lambda y. f (t1,t2))$ ))

```

```

      ≤ e * content (cbox (u,w) (v,z)) / content ?CBOX/2
    by (simp only: integral_diff [symmetric] int_integrable integrable_const)
    have norm_xx:  $\llbracket x' = y'; \text{norm}(x - x') \leq e/2; \text{norm}(y - y') \leq e/2 \rrbracket \implies$ 
 $\text{norm}(x - y) \leq e$  for  $x::'c$  and  $y \ x' \ y' \ e$ 
    using norm_triangle_mono [of  $x - y' \ e/2 \ y' - y \ e/2$ ] field_sum_of_halves
    by (simp add: norm_minus_commute)
    have norm (integral (cbox (u,w) (v,z)) f - integral (cbox u v) ( $\lambda x.$  integral
(cbox w z) ( $\lambda y.$  f (x,y))))
      ≤ e * content (cbox (u,w) (v,z)) / content ?CBOX
    by (rule norm_xx [OF integral_Pair_const 1 2])
  } note * = this
  have norm (integral ?CBOX f - integral (cbox a b) ( $\lambda x.$  integral (cbox c d)
( $\lambda y.$  f (x,y)))) ≤ e
    if  $\forall x \in ?CBOX. \forall x' \in ?CBOX. \text{norm}(x' - x) < k \longrightarrow \text{norm}(f x' - f x) < e$ 
/(2 * content (?CBOX))  $0 < k$  for k
  proof -
    obtain p where ptag: p tagged_division_of cbox (a, c) (b, d)
      and fine: ( $\lambda x.$  ball x k) fine p
    using fine_division_exists  $\langle 0 < k \rangle$  by blast
    show ?thesis
      using that fine ptag  $\langle 0 < k \rangle$ 
      by (auto simp: * intro: op_acbd [OF division_of_tagged_division [OF ptag]])
    qed
    then show norm (integral ?CBOX f - integral (cbox a b) ( $\lambda x.$  integral (cbox
c d) ( $\lambda y.$  f (x,y)))) ≤ e
      using compact_uniformly_continuous [OF assms compact_cbox]
      apply (simp add: uniformly_continuous_on_def dist_norm)
      apply (drule_tac  $x = e/2$  / content?CBOX in spec)
      using cbp  $\langle 0 < e \rangle$  by (auto simp: zero_less_mult_iff)
    qed
    then show ?thesis
      by simp
    qed
  lemma integral_swap_2dim:
    fixes f :: [ $'a::\text{euclidean\_space}$ ,  $'b::\text{euclidean\_space}$ ]  $\Rightarrow$   $'c::\text{banach}$ 
    assumes continuous_on (cbox (a,c) (b,d)) ( $\lambda(x,y).$  f x y)
    shows integral (cbox (a, c) (b, d)) ( $\lambda(x, y).$  f x y) = integral (cbox (c, a) (d,
b)) ( $\lambda(x, y).$  f y x)
  proof -
    have (( $\lambda(x, y).$  f x y) has_integral integral (cbox (c, a) (d, b)) ( $\lambda(x, y).$  f y x))
    (prod.swap ' (cbox (c, a) (d, b)))
    proof (rule has_integral_twiddle [of 1 prod.swap prod.swap  $\lambda(x,y).$  f y x integral
(cbox (c, a) (d, b)) ( $\lambda(x, y).$  f y x), simplified])
      show  $\bigwedge u \ v. \text{content}(\text{prod.swap ' cbox } u \ v) = \text{content}(\text{cbox } u \ v)$ 
        by (metis content_Pair mult.commute old.prod.exhaust swap_cbox_Pair)
      show (( $\lambda(x, y).$  f y x) has_integral integral (cbox (c, a) (d, b)) ( $\lambda(x, y).$  f y x))
        (cbox (c, a) (d, b))
        by (simp add: assms integrable_continuous integrable_integral swap_continuous)
    qed
  end

```

```

qed (use isCont_swap in ⟨fastforce+⟩)
then show ?thesis
  by force
qed

```

theorem *integral_swap_continuous:*

```

fixes f :: ['a::euclidean_space, 'b::euclidean_space] ⇒ 'c::banach
assumes continuous_on (cbox (a,c) (b,d)) (λ(x,y). f x y)
shows integral (cbox a b) (λx. integral (cbox c d) (f x)) =
      integral (cbox c d) (λy. integral (cbox a b) (λx. f x y))
proof -
  have integral (cbox a b) (λx. integral (cbox c d) (f x)) = integral (cbox (a, c) (b,
d)) (λ(x, y). f x y)
  using integral_prod_continuous [OF assms] by auto
  also have ... = integral (cbox (c, a) (d, b)) (λ(x, y). f y x)
  by (rule integral_swap_2dim [OF assms])
  also have ... = integral (cbox c d) (λy. integral (cbox a b) (λx. f x y))
  using integral_prod_continuous [OF swap_continuous] assms
  by auto
  finally show ?thesis .
qed

```

6.15.47 Definite integrals for exponential and power function

lemma *has_integral_exp_minus_to_infinity:*

```

assumes a: a > 0
shows ((λx::real. exp (-a*x)) has_integral exp (-a*c)/a) {c..}
proof -
  define f where f = (λk x. if x ∈ {c..real k} then exp (-a*x) else 0)
  {
    fix k :: nat assume k: of_nat k ≥ c
    from k a
    have ((λx. exp (-a*x)) has_integral (-exp (-a*real k)/a - (-exp (-a*c)/a)))
{c..real k}
    by (intro fundamental_theorem_of_calculus)
      (auto intro!: derivative_eq_intros
        simp: has_field_derivative_iff_has_vector_derivative [symmetric])
    hence (f k has_integral (exp (-a*c)/a - exp (-a*real k)/a)) {c..} unfolding
f_def
    by (subst has_integral_restrict) simp_all
  } note has_integral_f = this

```

```

  have [simp]: f k = (λ_. 0) if of_nat k < c for k using that by (auto simp:
fun_eq_iff f_def)

```

```

  have integral_f: integral {c..} (f k) =
    (if real k ≥ c then exp (-a*c)/a - exp (-a*real k)/a else 0)
    for k using integral_unique[OF has_integral_f[of k]] by simp

```

```

  have A: (λx. exp (-a*x)) integrable_on {c..} ∧

```

```

      (λk. integral {c..} (f k)) → integral {c..} (λx. exp (-a*x))
proof (intro monotone_convergence_increasing allI ballI)
  fix k :: nat
  have (λx. exp (-a*x)) integrable_on {c..of_real k} (is ?P)
  unfolding f_def by (auto intro!: continuous_intros integrable_continuous_real)
  hence (f k) integrable_on {c..of_real k}
    by (rule integrable_eq) (simp add: f_def)
  then show f k integrable_on {c..}
    by (rule integrable_on_superset) (auto simp: f_def)
next
  fix x assume x: x ∈ {c..}
  have sequentially ≤ principal {nat ⌈x⌉..} unfolding at_top_def by (simp add:
Inf_lower)
  also have {nat ⌈x⌉..} ⊆ {k. x ≤ real k} by auto
  also have inf (principal ...) (principal {k. ¬x ≤ real k}) =
    principal ({k. x ≤ real k} ∩ {k. ¬x ≤ real k}) by simp
  also have {k. x ≤ real k} ∩ {k. ¬x ≤ real k} = {} by blast
  finally have inf sequentially (principal {k. ¬x ≤ real k}) = bot
    by (simp add: inf.coboundedI1 bot_unique)
  with x show (λk. f k x) → exp (-a*x) unfolding f_def
    by (intro filterlim_IIf) auto
next
  have |integral {c..} (f k)| ≤ exp (-a*c)/a for k
proof (cases c > of_nat k)
  case False
  hence abs (integral {c..} (f k)) = abs (exp (- (a * c)) / a - exp (- (a * real
k)) / a)
    by (simp add: integral_f)
  also have abs (exp (- (a * c)) / a - exp (- (a * real k)) / a) =
    exp (- (a * c)) / a - exp (- (a * real k)) / a
    using False a by (intro abs_of_nonneg) (simp_all add: field_simps)
  also have ... ≤ exp (- a * c) / a using a by simp
  finally show ?thesis .
qed (insert a, simp_all add: integral_f)
thus bounded (range(λk. integral {c..} (f k)))
  by (intro boundedI[of _ exp (-a*c)/a]) auto
qed (auto simp: f_def)
have (λk. exp (-a*c)/a - exp (-a * of_nat k)/a) → exp (-a*c)/a - 0/a
  by (intro tendsto_intros filterlim_compose[OF exp_at_bot]
    filterlim_tendsto_neg_mult_at_bot[OF tendsto_const] filterlim_real_sequentially)+
    (insert a, simp_all)
moreover
  from eventually_gt_at_top[of nat ⌈c⌉] have eventually (λk. of_nat k > c)
    sequentially
    by eventually_elim linarith
  hence eventually (λk. exp (-a*c)/a - exp (-a * of_nat k)/a = integral {c..}
(f k)) sequentially
    by eventually_elim (simp add: integral_f)
  ultimately have (λk. integral {c..} (f k)) → exp (-a*c)/a - 0/a

```



```

    by (rule Lim_transform_eventually)
  from LIMSEQ_unique[OF conjunct2[OF A] this]
  have integral {c..} ( $\lambda x. \exp(-a*x) = \exp(-a*c)/a$ ) by simp
  with conjunct1[OF A] show ?thesis
    by (simp add: has_integral_integral)
qed

lemma integrable_on_exp_minus_to_infinity:  $a > 0 \implies (\lambda x. \exp(-a*x) :: \text{real})$ 
  integrable_on {c..}
  using has_integral_exp_minus_to_infinity[of a c] unfolding integrable_on_def
  by blast

lemma has_integral_powr_from_0:
  assumes  $a: a > (-1::\text{real})$  and  $c: c \geq 0$ 
  shows  $((\lambda x. x \text{ powr } a) \text{ has\_integral } (c \text{ powr } (a+1) / (a+1))) \{0..c\}$ 
proof (cases  $c = 0$ )
  case False
  define f where  $f = (\lambda k x. \text{if } x \in \{\text{inverse}(\text{of\_nat}(\text{Suc } k))..c\} \text{ then } x \text{ powr } a$ 
    else 0)
  define F where  $F = (\lambda k. \text{if } \text{inverse}(\text{of\_nat}(\text{Suc } k)) \leq c \text{ then } c \text{ powr } (a+1)/(a+1) - \text{inverse}(\text{real}(\text{Suc } k)) \text{ powr } (a+1)/(a+1) \text{ else } 0)$ 
  {
    fix k :: nat
    have (f k has_integral F k) {0..c}
    proof (cases  $\text{inverse}(\text{of\_nat}(\text{Suc } k)) \leq c$ )
      case True
      {
        fix x assume  $x: x \geq \text{inverse}(1 + \text{real } k)$ 
        have  $0 < \text{inverse}(1 + \text{real } k)$  by simp
        also note x
        finally have  $x > 0$  .
      } note x = this
      hence  $((\lambda x. x \text{ powr } a) \text{ has\_integral } c \text{ powr } (a+1) / (a+1) - \text{inverse}(\text{real}(\text{Suc } k)) \text{ powr } (a+1) / (a+1)) \{\text{inverse}(\text{real}(\text{Suc } k))..c\}$ 
        using True a by (intro fundamental_theorem_of_calculus)
        (auto intro!: derivative_eq_intros continuous_on_powr' continuous_on_const
          simp: has_field_derivative_iff_has_vector_derivative [symmetric])
      with True show ?thesis unfolding f_def F_def by (subst has_integral_restrict)
    simp_all
  } next
  case False
  thus ?thesis unfolding f_def F_def by (subst has_integral_restrict) auto
qed
} note has_integral_f = this
have integral_f: integral {0..c} (f k) = F k for k
  using has_integral_f[of k] by (rule integral_unique)

```

```

have A: (λx. x powr a) integrable_on {0..c} ∧
  (λk. integral {0..c} (f k)) → integral {0..c} (λx. x powr a)
proof (intro monotone_convergence_increasing ballI allI)
  fix k from has_integral_f[of k] show f k integrable_on {0..c}
  by (auto simp: integrable_on_def)
next
fix k :: nat and x :: real
{
  assume x: inverse (real (Suc k)) ≤ x
  have inverse (real (Suc (Suc k))) ≤ inverse (real (Suc k)) by (simp add:
field_simps)
  also note x
  finally have inverse (real (Suc (Suc k))) ≤ x .
}
thus f k x ≤ f (Suc k) x by (auto simp: f_def simp del: of_nat_Suc)
next
fix x assume x: x ∈ {0..c}
show (λk. f k x) → x powr a
proof (cases x = 0)
  case False
  with x have x > 0 by simp
  from order_tendstoD(2)[OF LIMSEQ_inverse_real_of_nat this]
  have eventually (λk. x powr a = f k x) sequentially
  by eventually_elim (insert x, simp add: f_def)
  moreover have (λ_. x powr a) → x powr a by simp
  ultimately show ?thesis by (blast intro: Lim_transform_eventually)
qed (simp_all add: f_def)
next
{
  fix k
  from a have F k ≤ c powr (a + 1) / (a + 1)
  by (auto simp add: F_def divide_simps)
  also from a have F k ≥ 0
  by (auto simp: F_def divide_simps simp del: of_nat_Suc intro!: powr_mono2)
  hence F k = abs (F k) by simp
  finally have abs (F k) ≤ c powr (a + 1) / (a + 1) .
}
thus bounded (range(λk. integral {0..c} (f k)))
  by (intro boundedI[of _ c powr (a+1) / (a+1)]) (auto simp: integral_f)
qed

from False c have c > 0 by simp
from order_tendstoD(2)[OF LIMSEQ_inverse_real_of_nat this]
  have eventually (λk. c powr (a + 1) / (a + 1) - inverse (real (Suc k)) powr
(a+1) / (a+1) =
  integral {0..c} (f k)) sequentially
  by eventually_elim (simp add: integral_f F_def)
  moreover have (λk. c powr (a + 1) / (a + 1) - inverse (real (Suc k)) powr
(a + 1) / (a + 1))

```

```

       $\longrightarrow c \text{ powr } (a + 1) / (a + 1) - 0 \text{ powr } (a + 1) / (a + 1)$ 
    using a by (intro tendsto_intros LIMSEQ_inverse_real_of_nat) auto
    hence  $(\lambda k. c \text{ powr } (a + 1) / (a + 1) - \text{inverse } (\text{real } (\text{Suc } k)) \text{ powr } (a + 1) / (a + 1))$ 
       $\longrightarrow c \text{ powr } (a + 1) / (a + 1)$  by simp
    ultimately have  $(\lambda k. \text{integral } \{0..c\} (f k)) \longrightarrow c \text{ powr } (a+1) / (a+1)$ 
      by (blast intro: Lim_transform_eventually)
    with A have  $\text{integral } \{0..c\} (\lambda x. x \text{ powr } a) = c \text{ powr } (a+1) / (a+1)$ 
      by (blast intro: LIMSEQ_unique)
    with A show ?thesis by (simp add: has_integral_integral)
  qed (simp_all add: has_integral_refl)

```

```

lemma integrable_on_powr_from_0:
  assumes a:  $a > (-1::\text{real})$  and c:  $c \geq 0$ 
  shows  $(\lambda x. x \text{ powr } a) \text{ integrable\_on } \{0..c\}$ 
  using has_integral_powr_from_0[OF assms] unfolding integrable_on_def by
  blast

```

```

lemma has_integral_powr_to_inf:
  fixes a e :: real
  assumes e < -1 a > 0
  shows  $((\lambda x. x \text{ powr } e) \text{ has\_integral } -(a \text{ powr } (e + 1)) / (e + 1)) \{a..\}$ 
proof -
  define f :: nat  $\Rightarrow$  real  $\Rightarrow$  real where  $f = (\lambda n x. \text{if } x \in \{a..n\} \text{ then } x \text{ powr } e \text{ else } 0)$ 
  define F where  $F = (\lambda x. x \text{ powr } (e + 1) / (e + 1))$ 

```

```

  have has_integral_f:  $(f n \text{ has\_integral } (F n - F a)) \{a..\}$ 
    if n:  $n \geq a$  for n :: nat
  proof -
    from n assms have  $((\lambda x. x \text{ powr } e) \text{ has\_integral } (F n - F a)) \{a..n\}$ 
    by (intro fundamental_theorem_of_calculus) (auto intro!: derivative_eq_intros
      simp: has_field_derivative_iff_has_vector_derivative [symmetric] F_def)
    hence  $(f n \text{ has\_integral } (F n - F a)) \{a..n\}$ 
    by (rule has_integral_eq [rotated]) (simp add: f_def)
    thus  $(f n \text{ has\_integral } (F n - F a)) \{a..\}$ 
    by (rule has_integral_on_superset) (auto simp: f_def)
  qed
  have integral_f:  $\text{integral } \{a..\} (f n) = (\text{if } n \geq a \text{ then } F n - F a \text{ else } 0)$  for n :: nat
  proof (cases  $n \geq a$ )
    case True
    with has_integral_f[OF this] show ?thesis by (simp add: integral_unique)
  next
    case False
    have  $(f n \text{ has\_integral } 0) \{a\}$  by (rule has_integral_refl)
    hence  $(f n \text{ has\_integral } 0) \{a..\}$ 
    by (rule has_integral_on_superset) (insert False, simp_all add: f_def)
    with False show ?thesis by (simp add: integral_unique)
  qed

```

qed

```

have *: (λx. x powr e) integrable_on {a..} ∧
  (λn. integral {a..} (f n)) → integral {a..} (λx. x powr e)
proof (intro monotone_convergence_increasing allI ballI)
  fix n :: nat
  from assms have (λx. x powr e) integrable_on {a..n}
    by (auto intro!: integrable_continuous_real continuous_intros)
  hence f n integrable_on {a..n}
    by (rule integrable_eq) (auto simp: f_def)
  thus f n integrable_on {a..}
    by (rule integrable_on_superset) (auto simp: f_def)
next
fix n :: nat and x :: real
show f n x ≤ f (Suc n) x by (auto simp: f_def)
next
fix x :: real assume x: x ∈ {a..}
from filterlim_real_sequentially
  have eventually (λn. real n ≥ x) at_top
    by (simp add: filterlim_at_top)
with x have eventually (λn. f n x = x powr e) at_top
  by (auto elim!: eventually_mono simp: f_def)
thus (λn. f n x) → x powr e by (simp add: tendsto_eventually)
next
have norm (integral {a..} (f n)) ≤ -F a for n :: nat
proof (cases n ≥ a)
  case True
  with assms have a powr (e + 1) ≥ n powr (e + 1)
    by (intro powr_mono2') simp_all
  with assms show ?thesis by (auto simp: divide_simps F_def integral_f)
qed (insert assms, simp add: integral_f F_def field_split_simps)
thus bounded (range(λk. integral {a..} (f k)))
  unfolding bounded_iff by (intro exI[of _ -F a]) auto
qed

from filterlim_real_sequentially
  have eventually (λn. real n ≥ a) at_top
    by (simp add: filterlim_at_top)
hence eventually (λn. F n - F a = integral {a..} (f n)) at_top
  by eventually_elim (simp add: integral_f)
moreover have (λn. F n - F a) → 0 / (e + 1) - F a unfolding F_def
  by (insert assms, (rule tendsto_intros filterlim_compose[OF tendsto_neg_powr]
    filterlim_ident filterlim_real_sequentially | simp)+)
hence (λn. F n - F a) → -F a by simp
ultimately have (λn. integral {a..} (f n)) → -F a by (blast intro: Lim_transform_eventually)
from conjunct2[OF *] and this
  have integral {a..} (λx. x powr e) = -F a by (rule LIMSEQ_unique)
with conjunct1[OF *] show ?thesis
  by (simp add: has_integral_integral F_def)

```

qed

```

lemma has_integral_inverse_power_to_inf:
  fixes a :: real and n :: nat
  assumes n > 1 a > 0
  shows (( $\lambda x. 1 / x^n$ ) has_integral 1 / (real (n - 1) * a(n - 1))) {a..}
proof -
  from assms have real_of_int (-int n) < -1 by simp
  note has_integral_pow_to_inf[OF this ‹a > 0›]
  also have - (a powr (real_of_int (- int n) + 1)) / (real_of_int (- int n) + 1) =
    1 / (real (n - 1) * a powr (real (n - 1))) using assms
  by (simp add: field_split_simps powr_add [symmetric] of_nat_diff)
  also from assms have a powr (real (n - 1)) = a(n - 1)
  by (intro powr_realpow)
  finally show ?thesis
  by (rule has_integral_eq [rotated])
    (insert assms, simp_all add: powr_minus powr_realpow field_split_simps)
qed

```

Adaption to ordered Euclidean spaces and the Cartesian Euclidean space

```

lemma integral_component_eq_cart[simp]:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  realm
  assumes f integrable_on s
  shows integral s ( $\lambda x. f\ x\ \$\ k$ ) = integral s f $ k
  using integral_linear[OF assms(1) bounded_linear_vec_nth,unfolded o_def] .

```

```

lemma content_closed_interval:
  fixes a :: 'a::ordered_euclidean_space
  assumes a ≤ b
  shows content {a..b} = ( $\prod i \in \text{Basis}. b \cdot i - a \cdot i$ )
  using content_cbox[of a b] assms by (simp add: cbox_interval eucl_le[where 'a='a])

```

```

lemma integrable_const_ivl[intro]:
  fixes a :: 'a::ordered_euclidean_space
  shows ( $\lambda x. c$ ) integrable_on {a..b}
  unfolding cbox_interval[symmetric] by (rule integrable_const)

```

```

lemma integrable_on_subinterval:
  fixes f :: 'n::ordered_euclidean_space  $\Rightarrow$  'a::banach
  assumes f integrable_on S {a..b}  $\subseteq$  S
  shows f integrable_on {a..b}
  using integrable_on_subcbox[of f S a b] assms by (simp add: cbox_interval)

```

end

6.16 Radon-Nikodým Derivative

```
theory Radon_Nikodym
imports Bochner_Integration
begin
```

definition *diff_measure* :: 'a measure $\Rightarrow$ 'a measure $\Rightarrow$ 'a measure

where

diff_measure *M N* = *measure_of* (*space* *M*) (*sets* *M*) ($\lambda A. \text{emeasure } M \ A - \text{emeasure } N \ A$)

lemma

shows *space_diff_measure*[*simp*]: *space* (*diff_measure* *M N*) = *space* *M*
and *sets_diff_measure*[*simp*]: *sets* (*diff_measure* *M N*) = *sets* *M*
by (*auto simp: diff_measure_def*)

lemma *emeasure_diff_measure*:

assumes *fin*: *finite_measure* *M* *finite_measure* *N* **and** *sets_eq*: *sets* *M* = *sets* *N*
assumes *pos*: $\bigwedge A. A \in \text{sets } M \implies \text{emeasure } N \ A \leq \text{emeasure } M \ A$ **and** *A*: *A* $\in \text{sets } M$

shows *emeasure* (*diff_measure* *M N*) *A* = *emeasure* *M* *A* - *emeasure* *N* *A* (**is** $\_ = ?\mu \ A$)

unfolding *diff_measure_def*

proof (*rule* *emeasure_measure_of_sigma*)

show *sigma_algebra* (*space* *M*) (*sets* *M*) ..

show *positive* (*sets* *M*) $?\mu$

using *pos* **by** (*simp add: positive_def*)

show *countably_additive* (*sets* *M*) $?\mu$

proof (*rule* *countably_additiveI*)

fix *A* :: *nat* $\Rightarrow$ $\_$ **assume** *A*: *range* *A* $\subseteq \text{sets } M$ **and** *disjoint_family* *A*

then have *suminf*:

$(\sum i. \text{emeasure } M \ (A \ i)) = \text{emeasure } M \ (\bigcup i. A \ i)$

$(\sum i. \text{emeasure } N \ (A \ i)) = \text{emeasure } N \ (\bigcup i. A \ i)$

by (*simp_all add: suminf_emeasure_sets_eq*)

with *A* **have** $(\sum i. \text{emeasure } M \ (A \ i) - \text{emeasure } N \ (A \ i)) =$

$(\sum i. \text{emeasure } M \ (A \ i)) - (\sum i. \text{emeasure } N \ (A \ i))$

using *fin pos* [*of* *A* $\_$]

by (*intro ennreal_suminf_minus*)

(*auto simp: sets_eq finite_measure.emeasure_eq_measure suminf_emeasure*)

then show $(\sum i. \text{emeasure } M \ (A \ i) - \text{emeasure } N \ (A \ i)) =$

$\text{emeasure } M \ (\bigcup i. A \ i) - \text{emeasure } N \ (\bigcup i. A \ i)$

by (*simp add: suminf*)

qed

qed *fact*

An equivalent characterization of sigma-finite spaces is the existence of integrable positive functions (or, still equivalently, the existence of a probability measure which is in the same measure class as the original measure).

proposition (**in** *sigma_finite_measure*) *obtain_positive_integrable_function*:

```

obtains f::'a  $\Rightarrow$  real where
  f  $\in$  borel_measurable M
   $\bigwedge x. f\ x > 0$ 
   $\bigwedge x. f\ x \leq 1$ 
  integrable M f
proof -
  obtain A :: nat  $\Rightarrow$  'a set where range A  $\subseteq$  sets M  $(\bigcup i. A\ i) = \text{space } M \bigwedge i.$ 
    emeasure M (A i)  $\neq \infty$ 
    using sigma_finite by auto
    then have [measurable]: A n  $\in$  sets M for n by auto
    define g where g = ( $\lambda x. (\sum n. (1/2)^\wedge(\text{Suc } n) * \text{indicator } (A\ n)\ x / (1 + \text{measure } M\ (A\ n)))$ )
    have [measurable]: g  $\in$  borel_measurable M unfolding g_def by auto
    have *: summable ( $\lambda n. (1/2)^\wedge(\text{Suc } n) * \text{indicator } (A\ n)\ x / (1 + \text{measure } M\ (A\ n))$ ) for x
      apply (rule summable_comparison_test' [of  $\lambda n. (1/2)^\wedge(\text{Suc } n)\ 0$ ])
      using power_half_series summable_def by (auto simp add: indicator_def divide_simps)
    have g x  $\leq (\sum n. (1/2)^\wedge(\text{Suc } n))$  for x unfolding g_def
      apply (rule suminf_le) using * power_half_series summable_def by (auto simp add: indicator_def divide_simps)
    then have g_le_1: g x  $\leq 1$  for x using power_half_series sums_unique by fastforce

    have g_pos: g x  $> 0$  if x  $\in$  space M for x
    unfolding g_def proof (subst suminf_pos_iff [OF *[of x]], auto)
      obtain i where x  $\in$  A i using  $(\bigcup i. A\ i) = \text{space } M$   $\langle x \in \text{space } M \rangle$  by auto
      then have 0 <  $(1 / 2)^\wedge \text{Suc } i * \text{indicator } (A\ i)\ x / (1 + \text{Sigma\_Algebra.measure } M\ (A\ i))$ 
        unfolding indicator_def apply (auto simp add: divide_simps) using measure_nonneg [of M A i]
        by (auto, meson add_nonneg_nonneg linorder_not_le mult_nonneg_nonneg zero_le_numeral zero_le_one zero_le_power)
      then show  $\exists i. 0 < (1 / 2)^\wedge i * \text{indicator } (A\ i)\ x / (2 + 2 * \text{Sigma\_Algebra.measure } M\ (A\ i))$ 
        by auto
    qed

    have integrable M g
    unfolding g_def proof (rule integrable_suminf)
      fix n
      show integrable M ( $\lambda x. (1 / 2)^\wedge \text{Suc } n * \text{indicator } (A\ n)\ x / (1 + \text{Sigma\_Algebra.measure } M\ (A\ n))$ )
        using  $\langle \text{emeasure } M\ (A\ n) \neq \infty \rangle$ 
        by (auto intro!: integrable_mult_right integrable_divide_zero integrable_real_indicator simp add: top.not_eq_extremum)
      next
        show AE x in M. summable ( $\lambda n. \text{norm } ((1 / 2)^\wedge \text{Suc } n * \text{indicator } (A\ n)\ x / (1 + \text{Sigma\_Algebra.measure } M\ (A\ n)))$ )

```

```

using * by auto
show summable (λn. (∫ x. norm ((1 / 2) ^ Suc n * indicator (A n) x / (1 +
Sigma_Algebra.measure M (A n))) ∂M))
  apply (rule summable_comparison_test' [of λn. (1/2) ^ (Suc n) 0], auto)
  using power_half_series summable_def apply auto[1]
  apply (auto simp add: field_split_simps) using measure_nonneg [of M]
not_less by fastforce
qed

```

```

define f where f = (λx. if x ∈ space M then g x else 1)
have f x > 0 for x unfolding f_def using g_pos by auto
moreover have f x ≤ 1 for x unfolding f_def using g_le_1 by auto
moreover have [measurable]: f ∈ borel_measurable M unfolding f_def by auto
moreover have integrable M f
  apply (subst integrable_cong [of _ _ _ g]) unfolding f_def using ⟨integrable
M g⟩ by auto
ultimately show (∧f. f ∈ borel_measurable M ⇒ (∧x. 0 < f x) ⇒ (∧x. f
x ≤ 1) ⇒ integrable M f ⇒ thesis) ⇒ thesis
  by (meson that)
qed

```

```

lemma (in sigma_finite_measure) Ex_finite_integrable_function:
  ∃ h ∈ borel_measurable M. integralN M h ≠ ∞ ∧ (∀ x ∈ space M. 0 < h x ∧ h x <
∞)
proof -
  obtain A :: nat ⇒ 'a set where
    range [measurable]: range A ⊆ sets M and
    space: (⋃ i. A i) = space M and
    measure: ∧ i. emeasure M (A i) ≠ ∞ and
    disjoint: disjoint_family A
  using sigma_finite_disjoint by blast
  let ?B = λi. 2Suc i * emeasure M (A i)
  have [measurable]: ∧ i. A i ∈ sets M
    using range by fastforce+
  have ∀ i. ∃ x. 0 < x ∧ x < inverse (?B i)
  proof
    fix i show ∃ x. 0 < x ∧ x < inverse (?B i)
      using measure [of i]
    by (auto intro!: dense simp: ennreal_inverse_positive ennreal_mult_eq_top_iff
power_eq_top_ennreal)
  qed
  from choice[OF this] obtain n where ∧ i. 0 < n i
    ∧ i. n i < inverse (2Suc i * emeasure M (A i)) by auto
  { fix i have 0 ≤ n i using n(1) [of i] by auto } note pos = this
  let ?h = λx. ∑ i. n i * indicator (A i) x
  show ?thesis
  proof (safe intro!: bexI [of _ ?h] del: notI)
    have integralN M ?h = (∑ i. n i * emeasure M (A i)) using pos
    by (simp add: nn_integral_suminf nn_integral_cmult_indicator)
  qed

```



```

also have ...  $\leq (\sum i. \text{ennreal } ((1/2)^{\text{Suc } i}))$ 
proof (intro suminf_le allI)
  fix N
  have  $n \ N * \text{emeasure } M \ (A \ N) \leq \text{inverse } (2^{\text{Suc } N} * \text{emeasure } M \ (A \ N))$ 
*  $\text{emeasure } M \ (A \ N)$ 
  using n[of N] by (intro mult_right_mono) auto
  also have ...  $= (1/2)^{\text{Suc } N} * (\text{inverse } (\text{emeasure } M \ (A \ N)) * \text{emeasure } M \ (A \ N))$ 
  using measure[of N]
  by (simp add: ennreal_inverse_power divide_ennreal_def ennreal_inverse_mult
    power_eq_top_ennreal less_top[symmetric] mult_ac
    del: power_Suc)
  also have ...  $\leq \text{inverse } (\text{ennreal } 2)^{\text{Suc } N}$ 
  using measure[of N]
  by (cases  $\text{emeasure } M \ (A \ N)$ ; cases  $\text{emeasure } M \ (A \ N) = 0$ )
    (auto simp: inverse_ennreal ennreal_mult[symmetric] divide_ennreal_def
    simp del: power_Suc)
  also have ...  $= \text{ennreal } (\text{inverse } 2)^{\text{Suc } N}$ 
  by (subst ennreal_power[symmetric], simp) (simp add: inverse_ennreal)
  finally show  $n \ N * \text{emeasure } M \ (A \ N) \leq \text{ennreal } ((1/2)^{\text{Suc } N})$ 
  by simp
qed auto
also have ...  $< \text{top}$ 
  unfolding less_top[symmetric]
  by (rule ennreal_suminf_neq_top)
    (auto simp: summable_geometric summable_Suc_iff simp del: power_Suc)
finally show  $\text{integral}^N M \neq \infty$ 
  by (auto simp: top_unique)
next
  { fix x assume  $x \in \text{space } M$ 
    then obtain i where  $x \in A \ i$  using space[symmetric] by auto
    with disjoint n have  $?h \ x = n \ i$ 
    by (auto intro!: suminf_cmult_indicator intro: less_imp_le)
    then show  $0 < ?h \ x$  and  $?h \ x < \infty$  using n[of i] by (auto simp: less_top[symmetric])
  }
  note pos = this
qed measurable
qed

```

6.16.1 Absolutely continuous

definition *absolutely_continuous* :: 'a measure $\Rightarrow$ 'a measure $\Rightarrow$ bool **where**
absolutely_continuous M N $\longleftrightarrow$ null_sets M $\subseteq$ null_sets N

lemma *absolutely_continuousI_count_space*: *absolutely_continuous* (count_space A) M
unfolding *absolutely_continuous_def* **by** (auto simp: null_sets_count_space)

lemma *absolutely_continuousI_density*:

$f \in \text{borel\_measurable } M \implies \text{absolutely\_continuous } M \text{ (density } M f)$
by (force simp add: absolutely_continuous_def null_sets_density_iff dest: AE_not_in)

lemma *absolutely_continuousI_point_measure_finite*:
 $(\bigwedge x. \llbracket x \in A ; f x \leq 0 \rrbracket \implies g x \leq 0) \implies \text{absolutely\_continuous } (\text{point\_measure } A f) \text{ (point\_measure } A g)$
unfolding absolutely_continuous_def **by** (force simp: null_sets_point_measure_iff)

lemma *absolutely_continuousD*:
 $\text{absolutely\_continuous } M N \implies A \in \text{sets } M \implies \text{emeasure } M A = 0 \implies \text{emeasure } N A = 0$
by (auto simp: absolutely_continuous_def null_sets_def)

lemma *absolutely_continuous_AE*:
assumes sets_eq: $\text{sets } M' = \text{sets } M$
and absolutely_continuous $M M'$ AE x in M . $P x$
shows AE x in M' . $P x$
proof –
from $\langle \text{AE } x \text{ in } M. P x \rangle$ **obtain** N **where** N : $N \in \text{null\_sets } M \{x \in \text{space } M. \neg P x\} \subseteq N$
unfolding eventually_ae_filter **by** auto
show AE x in M' . $P x$
proof (rule AE_I')
show $\{x \in \text{space } M'. \neg P x\} \subseteq N$ **using** sets_eq_imp_space_eq[OF sets_eq]
 $N(2)$ **by** simp
from $\langle \text{absolutely\_continuous } M M' \rangle$ **show** $N \in \text{null\_sets } M'$
using N **unfolding** absolutely_continuous_def sets_eq null_sets_def **by** auto
qed
qed

6.16.2 Existence of the Radon-Nikodym derivative

proposition
(in finite_measure) Radon_Nikodym_finite_measure:
assumes finite_measure N **and** sets_eq[simp]: $\text{sets } N = \text{sets } M$
assumes absolutely_continuous $M N$
shows $\exists f \in \text{borel\_measurable } M. \text{density } M f = N$
proof –
interpret N : finite_measure N **by** fact
define G **where** $G = \{g \in \text{borel\_measurable } M. \forall A \in \text{sets } M. (\int^+ x. g x * \text{indicator } A x \partial M) \leq N A\}$
have [measurable_dest]: $f \in G \implies f \in \text{borel\_measurable } M$
and G_D : $\bigwedge A. f \in G \implies A \in \text{sets } M \implies (\int^+ x. f x * \text{indicator } A x \partial M) \leq N A$ **for** f
by (auto simp: G_def)
note this[measurable_dest]
have $(\lambda x. 0) \in G$ **unfolding** G_def **by** auto
hence $G \neq \{\}$ **by** auto

```

{ fix f g assume f[measurable]: f ∈ G and g[measurable]: g ∈ G
  have (λx. max (g x) (f x)) ∈ G (is ?max ∈ G) unfolding G_def
proof safe
  let ?A = {x ∈ space M. f x ≤ g x}
  have ?A ∈ sets M using f g unfolding G_def by auto
  fix A assume [measurable]: A ∈ sets M
  have union: ((?A ∩ A) ∪ ((space M - ?A) ∩ A)) = A
    using sets.sets_into_space[OF ‹A ∈ sets M›] by auto
  have ∧x. x ∈ space M ⇒ max (g x) (f x) * indicator A x =
    g x * indicator (?A ∩ A) x + f x * indicator ((space M - ?A) ∩ A) x
    by (auto simp: indicator_def max_def)
  hence (∫+x. max (g x) (f x) * indicator A x ∂M) =
    (∫+x. g x * indicator (?A ∩ A) x ∂M) +
    (∫+x. f x * indicator ((space M - ?A) ∩ A) x ∂M)
    by (auto cong: nn_integral_cong intro!: nn_integral_add)
  also have ... ≤ N (?A ∩ A) + N ((space M - ?A) ∩ A)
    using f g unfolding G_def by (auto intro!: add_mono)
  also have ... = N A
    using union by (subst plus_emeasure) auto
  finally show (∫+x. max (g x) (f x) * indicator A x ∂M) ≤ N A .
qed auto }
note max_in_G = this
{ fix f assume incseq f and f: ∧i. f i ∈ G
  then have [measurable]: ∧i. f i ∈ borel_measurable M by (auto simp: G_def)
  have (λx. SUP i. f i x) ∈ G unfolding G_def
proof safe
  show (λx. SUP i. f i x) ∈ borel_measurable M by measurable
next
  fix A assume A ∈ sets M
  have (∫+x. (SUP i. f i x) * indicator A x ∂M) =
    (∫+x. (SUP i. f i x * indicator A x) ∂M)
    by (intro nn_integral_cong) (simp split: split_indicator)
  also have ... = (SUP i. (∫+x. f i x * indicator A x ∂M))
    using ‹incseq f› f ‹A ∈ sets M›
    by (intro nn_integral_monotone_convergence_SUP)
    (auto simp: G_def incseq_Suc_iff le_fun_def split: split_indicator)
  finally show (∫+x. (SUP i. f i x) * indicator A x ∂M) ≤ N A
    using f ‹A ∈ sets M› by (auto intro!: SUP_least simp: G_D)
qed }
note SUP_in_G = this
let ?y = SUP g ∈ G. integralN M g
have y_le: ?y ≤ N (space M) unfolding G_def
proof (safe intro!: SUP_least)
  fix g assume ∀ A ∈ sets M. (∫+x. g x * indicator A x ∂M) ≤ N A
  from this[THEN bspec, OF sets.top] show integralN M g ≤ N (space M)
    by (simp cong: nn_integral_cong)
qed
from ennreal_SUP_countable_SUP [OF ‹G ≠ {}›, of integralN M]
obtain ys :: nat ⇒ ennreal

```

```

    where  $ys$ :  $\text{range } ys \subseteq \text{integral}^N M \text{ ' } G \wedge \text{Sup } (\text{integral}^N M \text{ ' } G) = \text{Sup } (\text{range } ys)$ 
  by auto
  then have  $\forall n. \exists g. g \in G \wedge \text{integral}^N M g = ys\ n$ 
  proof safe
    fix  $n$  assume  $\text{range } ys \subseteq \text{integral}^N M \text{ ' } G$ 
    hence  $ys\ n \in \text{integral}^N M \text{ ' } G$  by auto
    thus  $\exists g. g \in G \wedge \text{integral}^N M g = ys\ n$  by auto
  qed
  from choice[OF this] obtain  $gs$  where  $\bigwedge i. gs\ i \in G \wedge \bigwedge n. \text{integral}^N M (gs\ n) = ys\ n$  by auto
  hence  $y\_eq$ :  $?y = (\text{SUP } i. \text{integral}^N M (gs\ i))$  using  $ys$  by auto
  let  $?g = \lambda i\ x. \text{Max } ((\lambda n. gs\ n\ x) \text{ ' } \{..i\})$ 
  define  $f$  where [abs_def]:  $f\ x = (\text{SUP } i. ?g\ i\ x)$  for  $x$ 
  let  $?F = \lambda A\ x. f\ x * \text{indicator } A\ x$ 
  have  $gs\_not\_empty$ :  $\bigwedge i\ x. (\lambda n. gs\ n\ x) \text{ ' } \{..i\} \neq \{\}$  by auto
  { fix  $i$  have  $?g\ i \in G$ 
    proof (induct i)
      case 0 thus  $?case$  by simp fact
    next
      case (Suc i)
      with  $Suc\ gs\_not\_empty \langle gs\ (Suc\ i) \in G \rangle$  show  $?case$ 
        by (auto simp add: atMost_Suc intro!: max_in_G)
    qed }
  note  $g\_in\_G = this$ 
  have incseq  $?g$  using  $gs\_not\_empty$ 
    by (auto intro!: incseq_SucI le_funI simp add: atMost_Suc)

  from  $SUP\_in\_G$ [OF this g_in_G] have [measurable]:  $f \in G$  unfolding  $f\_def$ 
  .
  then have [measurable]:  $f \in \text{borel\_measurable } M$  unfolding  $G\_def$  by auto

  have  $\text{integral}^N M f = (\text{SUP } i. \text{integral}^N M (?g\ i))$  unfolding  $f\_def$ 
    using  $g\_in\_G \langle \text{incseq } ?g \rangle$  by (auto intro!: nn_integral_monotone_convergence_SUP simp: G_def)
  also have  $\dots = ?y$ 
  proof (rule antisym)
    show  $(\text{SUP } i. \text{integral}^N M (?g\ i)) \leq ?y$ 
      using  $g\_in\_G$  by (auto intro: SUP_mono)
    show  $?y \leq (\text{SUP } i. \text{integral}^N M (?g\ i))$  unfolding  $y\_eq$ 
      by (auto intro!: SUP_mono nn_integral_mono Max_ge)
  qed
  finally have  $\text{int\_f\_eq\_y}$ :  $\text{integral}^N M f = ?y$  .

  have upper_bound:  $\forall A \in \text{sets } M. N\ A \leq \text{density } M f\ A$ 
  proof (rule ccontr)
    assume  $\neg ?thesis$ 
    then obtain  $A$  where  $A$ [measurable]:  $A \in \text{sets } M$  and  $f\_less\_N$ :  $\text{density } M f\ A < N\ A$ 

```

```

    by (auto simp: not_le)
  then have pos_A:  $0 < M\ A$ 
    using ‹absolutely_continuous M N›[THEN absolutely_continuousD, OF A]
    by (auto simp: zero_less_iff_neq_zero)

  define b where  $b = (N\ A - \text{density } M\ f\ A) / M\ A / 2$ 
  with f_less_N pos_A have  $0 < b$   $b \neq \text{top}$ 
  by (auto intro!: diff_gr0 ennreal simp: zero_less_iff_neq_zero diff_eq_0_iff_ennreal
    ennreal_divide_eq_top_iff)

  let ?f =  $\lambda x. f\ x + b$ 
  have nn_integral_M_f_neq_top
    using ‹f ∈ G›[THEN G_D, of space M] by (auto simp: top_unique cong:
    nn_integral_cong)
  with ‹b ≠ top› interpret Mf: finite_measure_density M ?f
    by (intro finite_measureI)
    (auto simp: field_simps mult_indicator_subset ennreal_mult_eq_top_iff
      emeasure_density nn_integral_cmult_indicator nn_integral_add
      cong: nn_integral_cong)

  from unsigned_Hahn_decomposition[of density M ?f N A]
  obtain Y where [measurable]:  $Y \in \text{sets } M$  and [simp]:  $Y \subseteq A$ 
    and Y1:  $\bigwedge C. C \in \text{sets } M \implies C \subseteq Y \implies \text{density } M\ ?f\ C \leq N\ C$ 
    and Y2:  $\bigwedge C. C \in \text{sets } M \implies C \subseteq A \implies C \cap Y = \{\} \implies N\ C \leq \text{density}$ 
    M ?f C
    by auto

  let ?f' =  $\lambda x. f\ x + b * \text{indicator } Y\ x$ 
  have M_Y_neq_0
  proof
    assume M_Y = 0
    then have N_Y = 0
      using ‹absolutely_continuous M N›[THEN absolutely_continuousD, of Y]
    by auto
    then have N_A = N (A - Y)
      by (subst emeasure_Diff) auto
    also have ... ≤ density M ?f (A - Y)
      by (rule Y2) auto
    also have ... ≤ density M ?f A - density M ?f Y
      by (subst emeasure_Diff) auto
    also have ... ≤ density M ?f A - 0
      by (intro ennreal_minus_mono) auto
    also have density M ?f A = b * M A + density M f A
    by (simp add: emeasure_density field_simps mult_indicator_subset nn_integral_add
    nn_integral_cmult_indicator)
    also have ... < N A
      using f_less_N pos_A
    by (cases density M f A; cases M A; cases N A)
      (auto simp: b_def ennreal_less_iff ennreal_minus divide_ennreal en-

```

```

nreal_numeral[symmetric]
      ennreal_plus[symmetric] ennreal_mult[symmetric] field_simps
      simp del: ennreal_numeral ennreal_plus)
  finally show False
    by simp
qed
then have nn_integral M f < nn_integral M ?f'
  using ⟨0 < b⟩ ⟨nn_integral M f ≠ top⟩
  by (simp add: nn_integral_add nn_integral_cmult_indicator ennreal_zero_less_mult_iff
zero_less_iff_neq_zero)
moreover
  have ?f' ∈ G
    unfolding G_def
  proof safe
    fix X assume [measurable]: X ∈ sets M
    have (∫+ x. ?f' x * indicator X x ∂M) = density M f (X - Y) + density
M ?f (X ∩ Y)
      by (auto simp add: emeasure_density nn_integral_add[symmetric] split:
split_indicator intro!: nn_integral_cong)
    also have ... ≤ N (X - Y) + N (X ∩ Y)
      using G_D[OF ⟨f ∈ G⟩] by (intro add_mono Y1) (auto simp: emea-
sure_density)
    also have ... = N X
      by (subst plus_emeasure) (auto intro!: arg_cong2[where f=emeasure])
    finally show (∫+ x. ?f' x * indicator X x ∂M) ≤ N X .
  qed simp
then have nn_integral M ?f' ≤ ?y
  by (rule SUP_upper)
ultimately show False
  by (simp add: int_f_eq_y)
qed
show ?thesis
proof (intro bexI[of _ f] measure_eqI conjI antisym)
  fix A assume A ∈ sets (density M f) then show emeasure (density M f) A ≤
emeasure N A
    by (auto simp: emeasure_density intro!: G_D[OF ⟨f ∈ G⟩])
  next
    fix A assume A: A ∈ sets (density M f) then show emeasure N A ≤ emeasure
(density M f) A
      using upper_bound by auto
  qed auto
qed

```

lemma (in finite_measure) split_space_into_finite_sets_and_rest:

assumes *ac*: absolutely_continuous M N **and** sets_eq[simp]: sets N = sets M

shows $\exists B::\text{nat} \Rightarrow \text{'a set. disjoint\_family } B \wedge \text{range } B \subseteq \text{sets } M \wedge (\forall i. N (B i) \neq \infty) \wedge$

$(\forall A \in \text{sets } M. A \cap (\bigcup i. B i) = \{\} \longrightarrow (\text{emeasure } M A = 0 \wedge N A = 0) \vee$

$(\text{emeasure } M A > 0 \wedge N A = \infty))$

```

proof -
  let ?Q = {Q ∈ sets M. N Q ≠ ∞}
  let ?a = SUP Q ∈ ?Q. emeasure M Q
  have {} ∈ ?Q by auto
  then have Q_not_empty: ?Q ≠ {} by blast
  have ?a ≤ emeasure M (space M) using sets.sets_into_space
    by (auto intro!: SUP_least emeasure_mono)
  then have ?a ≠ ∞
    using finite_emeasure_space
    by (auto simp: less_top[symmetric] top_unique simp del: SUP_eq_top_iff
      Sup_eq_top_iff)
  from ennreal_SUP_countable_SUP [OF Q_not_empty, of emeasure M]
  obtain Q'' where range Q'' ⊆ emeasure M ' ?Q and a: ?a = (SUP i::nat. Q''
i)
    by auto
  then have ∀ i. ∃ Q'. Q'' i = emeasure M Q' ∧ Q' ∈ ?Q by auto
  from choice[OF this] obtain Q' where Q': ∧ i. Q'' i = emeasure M (Q' i) ∧ i.
Q' i ∈ ?Q
    by auto
  then have a_Lim: ?a = (SUP i. emeasure M (Q' i)) using a by simp
  let ?O = λn. ⋃ i ≤ n. Q' i
  have Union: (SUP i. emeasure M (?O i)) = emeasure M (⋃ i. ?O i)
  proof (rule SUP_emeasure_incseq[of ?O])
    show range ?O ⊆ sets M using Q' by auto
    show incseq ?O by (fastforce intro!: incseq_SucI)
  qed
  have Q'_sets[measurable]: ∧ i. Q' i ∈ sets M using Q' by auto
  have O_sets: ∧ i. ?O i ∈ sets M using Q' by auto
  then have O_in_G: ∧ i. ?O i ∈ ?Q
  proof (safe del: notI)
    fix i have Q' ' {...i} ⊆ sets M using Q' by auto
    then have N (?O i) ≤ (∑ i ≤ i. N (Q' i))
      by (simp add: emeasure_subadditive_finite)
    also have ... < ∞ using Q' by (simp add: less_top)
    finally show N (?O i) ≠ ∞ by simp
  qed auto
  have O_mono: ∧ n. ?O n ⊆ ?O (Suc n) by fastforce
  have a_eq: ?a = emeasure M (⋃ i. ?O i) unfolding Union[symmetric]
  proof (rule antisym)
    show ?a ≤ (SUP i. emeasure M (?O i)) unfolding a_Lim
      using Q' by (auto intro!: SUP_mono emeasure_mono)
    show (SUP i. emeasure M (?O i)) ≤ ?a
  proof (safe intro!: Sup_mono, unfold bex_simps)
    fix i
    have *: (⋃ (Q' ' {...i})) = ?O i by auto
    then show ∃ x. (x ∈ sets M ∧ N x ≠ ∞) ∧
      emeasure M (⋃ (Q' ' {...i})) ≤ emeasure M x
      using O_in_G[of i] by (auto intro!: exI[of _ ?O i])
    qed
  qed

```

```

qed
let ?O_0 = (⋃ i. ?O i)
have ?O_0 ∈ sets M using Q' by auto
have disjointed Q' i ∈ sets M for i
  using sets.range_disjointed_sets[of Q' M] using Q'_sets by (auto simp: sub-
set_eq)
note Q_sets = this
show ?thesis
proof (intro bexI exI conjI ballI impI allI)
  show disjoint_family (disjointed Q')
    by (rule disjoint_family_disjointed)
  show range (disjointed Q') ⊆ sets M
    using Q'_sets by (intro sets.range_disjointed_sets) auto
  { fix A assume A: A ∈ sets M A ∩ (⋃ i. disjointed Q' i) = {}
    then have A1: A ∩ (⋃ i. Q' i) = {}
      unfolding UN_disjointed_eq by auto
    show emeasure M A = 0 ∧ N A = 0 ∨ 0 < emeasure M A ∧ N A = ∞
      proof (rule disjCI, simp)
        assume *: emeasure M A = 0 ∨ N A ≠ top
        show emeasure M A = 0 ∧ N A = 0
          proof (cases emeasure M A = 0)
            case True
              with ac A have N A = 0
                unfolding absolutely_continuous_def by auto
              with True show ?thesis by simp
            next
              case False
                with * have N A ≠ ∞ by auto
                with A have emeasure M ?O_0 + emeasure M A = emeasure M (?O_0
∪ A)
                  using Q' A1 by (auto intro!: plus_emeasure simp: set_eq_iff)
                also have ... = (SUP i. emeasure M (?O i ∪ A))
                  proof (rule SUP_emeasure_incseq[of λi. ?O i ∪ A, symmetric, simplified])
                    show range (λi. ?O i ∪ A) ⊆ sets M
                      using ⟨N A ≠ ∞⟩ O_sets A by auto
                    qed (fastforce intro!: incseq_SucI)
                    also have ... ≤ ?a
                      proof (safe intro!: SUP_least)
                        fix i have ?O i ∪ A ∈ ?Q
                          proof (safe del: notI)
                            show ?O i ∪ A ∈ sets M using O_sets A by auto
                            from O_in_G[of i] have N (?O i ∪ A) ≤ N (?O i) + N A
                              using emeasure_subadditive[of ?O i N A] A O_sets by auto
                            with O_in_G[of i] show N (?O i ∪ A) ≠ ∞
                              using ⟨N A ≠ ∞⟩ by (auto simp: top_unique)
                          qed
                        qed
                      then show emeasure M (?O i ∪ A) ≤ ?a by (rule SUP_upper)
                    qed
                  finally have emeasure M A = 0

```



```

      unfolding a_eq using measure_nonneg[of M A] by (simp add: emea-
sure_eq_measure)
    with ‹emeasure M A  $\neq$  0› show ?thesis by auto
  qed
qed }
{ fix i
  have N (disjointed Q' i)  $\leq$  N (Q' i)
    by (auto intro!: emeasure_mono simp: disjointed_def)
  then show N (disjointed Q' i)  $\neq$   $\infty$ 
    using Q'(2)[of i] by (auto simp: top_unique) }
qed
qed

```

proposition (in finite_measure) Radon_Nikodym_finite_measure_infinite:

assumes absolutely_continuous M N and sets_eq: sets N = sets M

shows $\exists f \in \text{borel\_measurable } M. \text{density } M f = N$

proof –

from split_space_into_finite_sets_and_rest[OF assms]

obtain Q :: nat $\Rightarrow$ 'a set

where Q: disjoint_family Q range Q $\subseteq$ sets M

and in_Q0: $\bigwedge A. A \in \text{sets } M \implies A \cap (\bigcup i. Q i) = \{\} \implies \text{emeasure } M A = 0 \wedge N A = 0 \vee 0 < \text{emeasure } M A \wedge N A = \infty$

and Q_fin: $\bigwedge i. N (Q i) \neq \infty$ by force

from Q have Q_sets: $\bigwedge i. Q i \in \text{sets } M$ by auto

let ?N = $\lambda i. \text{density } N (\text{indicator } (Q i))$ and ?M = $\lambda i. \text{density } M (\text{indicator } (Q i))$

have $\forall i. \exists f \in \text{borel\_measurable } (?M i). \text{density } (?M i) f = ?N i$

proof (intro allI finite_measure.Radon_Nikodym_finite_measure)

fix i

from Q show finite_measure (?M i)

by (auto intro!: finite_measureI cong: nn_integral_cong

simp add: emeasure_density_subset_eq sets_eq)

from Q have emeasure (?N i) (space N) = emeasure N (Q i)

by (simp add: sets_eq[symmetric] emeasure_density_subset_eq cong: nn_integral_cong)

with Q_fin show finite_measure (?N i)

by (auto intro!: finite_measureI)

show sets (?N i) = sets (?M i) by (simp add: sets_eq)

have [measurable]: $\bigwedge A. A \in \text{sets } M \implies A \in \text{sets } N$ by (simp add: sets_eq)

show absolutely_continuous (?M i) (?N i)

using ‹absolutely_continuous M N› ‹Q i $\in$ sets M›

by (auto simp: absolutely_continuous_def null_sets_density_iff sets_eq

intro!: absolutely_continuous_AE[OF sets_eq])

qed

from choice[OF this[unfolded Bex_def]]

obtain f where borel: $\bigwedge i. f i \in \text{borel\_measurable } M \wedge i x. 0 \leq f i x$

and f_density: $\bigwedge i. \text{density } (?M i) (f i) = ?N i$

by force

{ fix A i assume A: A $\in$ sets M

with Q borel have $(\int^+ x. f i x * \text{indicator } (Q i \cap A) x \, \partial M) = \text{emeasure}$

```

(density (?M i) (f i)) A
  by (auto simp add: emeasure_density nn_integral_density subset_eq
      intro!: nn_integral_cong split: split_indicator)
  also have ... = emeasure N (Q i ∩ A)
    using A Q by (simp add: f_density emeasure_restricted subset_eq sets_eq)
  finally have emeasure N (Q i ∩ A) = (∫+x. f i x * indicator (Q i ∩ A) x
    ∂M) .. }
  note integral_eq = this
  let ?f = λx. (∑ i. f i x * indicator (Q i) x) + ∞ * indicator (space M - (∪ i.
    Q i)) x
  show ?thesis
  proof (safe intro!: bexI[of _ ?f])
    show ?f ∈ borel_measurable M using borel Q_sets
      by (auto intro!: measurable_Iff)
    show density M ?f = N
    proof (rule measure_eqI)
      fix A assume A ∈ sets (density M ?f)
      then have A ∈ sets M by simp
      have Qi: ∧ i. Q i ∈ sets M using Q by auto
      have [intro,simp]: ∧ i. (λx. f i x * indicator (Q i ∩ A) x) ∈ borel_measurable
        M
        (
          ∧ i. AE x in M. 0 ≤ f i x * indicator (Q i ∩ A) x
          using borel Qi ⟨A ∈ sets M⟩ by auto
          have (∫+x. ?f x * indicator A x ∂M) = (∫+x. (∑ i. f i x * indicator (Q i
            ∩ A) x) + ∞ * indicator ((space M - (∪ i. Q i)) ∩ A) x ∂M)
            using borel by (intro nn_integral_cong) (auto simp: indicator_def)
          also have ... = (∫+x. (∑ i. f i x * indicator (Q i ∩ A) x) ∂M) + ∞ *
            emeasure M ((space M - (∪ i. Q i)) ∩ A)
            using borel Qi ⟨A ∈ sets M⟩
            by (subst nn_integral_add)
            (auto simp add: nn_integral_cmult_indicator sets.Int intro!: suminf_0_le)
          also have ... = (∑ i. N (Q i ∩ A)) + ∞ * emeasure M ((space M - (∪ i.
            Q i)) ∩ A)
            by (subst integral_eq[OF ⟨A ∈ sets M⟩], subst nn_integral_suminf) auto
          finally have (∫+x. ?f x * indicator A x ∂M) = (∑ i. N (Q i ∩ A)) + ∞ *
            emeasure M ((space M - (∪ i. Q i)) ∩ A) .
          moreover have (∑ i. N (Q i ∩ A)) = N ((∪ i. Q i) ∩ A)
            using Q Q_sets ⟨A ∈ sets M⟩
            by (subst suminf_emeasure) (auto simp: disjoint_family_on_def sets_eq)
          moreover
          have (space M - (∪ x. Q x)) ∩ A ∩ (∪ x. Q x) = {}
            by auto
          then have ∞ * emeasure M ((space M - (∪ i. Q i)) ∩ A) = N ((space M
            - (∪ i. Q i)) ∩ A)
            using in_Q0[of (space M - (∪ i. Q i)) ∩ A] ⟨A ∈ sets M⟩ Q by (auto
            simp: ennreal_top_mult)
          moreover have (space M - (∪ i. Q i)) ∩ A ∈ sets M ((∪ i. Q i) ∩ A) ∈
            sets M
            using Q_sets ⟨A ∈ sets M⟩ by auto
        )
    end
  end

```

```

    moreover have  $((\bigcup i. Q\ i) \cap A) \cup ((space\ M - (\bigcup i. Q\ i)) \cap A) = A$   $((\bigcup i. Q\ i) \cap A) \cap ((space\ M - (\bigcup i. Q\ i)) \cap A) = \{\}$ 
    using  $\langle A \in sets\ M \rangle sets.sets\_into\_space$  by auto
    ultimately have  $N\ A = (\int^+ x. ?f\ x * indicator\ A\ x\ \partial M)$ 
    using plus_emeasure[of  $(\bigcup i. Q\ i) \cap A\ N$   $(space\ M - (\bigcup i. Q\ i)) \cap A]$  by
    (simp add: sets_eq)
    with  $\langle A \in sets\ M \rangle$  borel Q show emeasure (density M ?f) A = N A
    by (auto simp: subset_eq emeasure_density)
  qed (simp add: sets_eq)
qed
qed

```

theorem (in sigma_finite_measure) Radon_Nikodym:

assumes ac: absolutely_continuous M N assumes sets_eq: sets N = sets M
 shows $\exists f \in borel\_measurable\ M. density\ M\ f = N$

proof –

from Ex_finite_integrable_function

obtain h where finite: $integral^N\ M\ h \neq \infty$ and

borel: $h \in borel\_measurable\ M$ and

nn: $\bigwedge x. 0 \leq h\ x$ and

pos: $\bigwedge x. x \in space\ M \implies 0 < h\ x$ and

$\bigwedge x. x \in space\ M \implies h\ x < \infty$ by auto

let ?T = $\lambda A. (\int^+ x. h\ x * indicator\ A\ x\ \partial M)$

let ?MT = density M h

from borel finite nn interpret T: finite_measure ?MT

by (auto intro!: finite_measureI cong: nn_integral_cong simp: emeasure_density)

have absolutely_continuous ?MT N sets N = sets ?MT

proof (unfold absolutely_continuous_def, safe)

fix A assume $A \in null\_sets\ ?MT$

with borel have $A \in sets\ M$ AE x in M. $x \in A \implies h\ x \leq 0$

by (auto simp add: null_sets_density_iff)

with pos sets.sets_into_space have AE x in M. $x \notin A$

by (elim eventually_mono) (auto simp: not_le[symmetric])

then have $A \in null\_sets\ M$

using $\langle A \in sets\ M \rangle$ by (simp add: AE_iff_null_sets)

with ac show $A \in null\_sets\ N$

by (auto simp: absolutely_continuous_def)

qed (auto simp add: sets_eq)

from T.Radon_Nikodym_finite_measure_infinite[OF this]

obtain f where f_borel: $f \in borel\_measurable\ M$ density ?MT f = N by auto

with nn borel show ?thesis

by (auto intro!: bexI[of $\lambda x. h\ x * f\ x$] simp: density_density_eq)

qed

6.16.3 Uniqueness of densities

lemma finite_density_unique:

assumes borel: $f \in borel\_measurable\ M$ $g \in borel\_measurable\ M$

assumes pos: AE x in M. $0 \leq f\ x$ AE x in M. $0 \leq g\ x$

```

and fin: integralN M f ≠ ∞
shows density M f = density M g ↔ (AE x in M. f x = g x)
proof (intro iffI ballI)
  fix A assume eq: AE x in M. f x = g x
  with borel show density M f = density M g
    by (auto intro: density_cong)
next
let ?P = λf A. ∫+ x. f x * indicator A x ∂M
assume density M f = density M g
with borel have eq: ∀ A ∈ sets M. ?P f A = ?P g A
  by (simp add: emeasure_density[symmetric])
from this[THEN bspec, OF sets.top] fin
have g_fin: integralN M g ≠ ∞ by (simp cong: nn_integral_cong)
{ fix f g assume borel: f ∈ borel_measurable M g ∈ borel_measurable M
  and pos: AE x in M. 0 ≤ f x AE x in M. 0 ≤ g x
  and g_fin: integralN M g ≠ ∞ and eq: ∀ A ∈ sets M. ?P f A = ?P g A
  let ?N = {x ∈ space M. g x < f x}
  have N: ?N ∈ sets M using borel by simp
  have ?P g ?N ≤ integralN M g using pos
    by (intro nn_integral_mono_AE) (auto split: split_indicator)
  then have Pg_fin: ?P g ?N ≠ ∞ using g_fin by (auto simp: top_unique)
  have ?P (λx. (f x - g x)) ?N = (∫+ x. f x * indicator ?N x - g x * indicator
?N x ∂M)
    by (auto intro!: nn_integral_cong simp: indicator_def)
  also have ... = ?P f ?N - ?P g ?N
  proof (rule nn_integral_diff)
    show (λx. f x * indicator ?N x) ∈ borel_measurable M (λx. g x * indicator
?N x) ∈ borel_measurable M
      using borel N by auto
    show AE x in M. g x * indicator ?N x ≤ f x * indicator ?N x
      using pos by (auto split: split_indicator)
  qed fact
  also have ... = 0
  unfolding eq[THEN bspec, OF N] using Pg_fin by auto
  finally have AE x in M. f x ≤ g x
    using pos borel nn_integral_PInf_AE[OF borel(2) g_fin]
    by (subst (asm) nn_integral_0_iff_AE)
    (auto split: split_indicator simp: not_less ennreal_minus_eq_0) }
from this[OF borel pos g_fin eq] this[OF borel(2,1) pos(2,1) fin] eq
show AE x in M. f x = g x by auto
qed

```

```

lemma (in finite_measure) density_unique_finite_measure:
  assumes borel: f ∈ borel_measurable M f' ∈ borel_measurable M
  assumes pos: AE x in M. 0 ≤ f x AE x in M. 0 ≤ f' x
  assumes f: ∧ A. A ∈ sets M ⇒ (∫+ x. f x * indicator A x ∂M) = (∫+ x. f' x
* indicator A x ∂M)
  (is ∧ A. A ∈ sets M ⇒ ?P f A = ?P f' A)
  shows AE x in M. f x = f' x

```

proof –

```

let ?D =  $\lambda f.$  density M f
let ?N =  $\lambda A.$  ?P f A and ?N' =  $\lambda A.$  ?P f' A
let ?f =  $\lambda A\ x.$  f x * indicator A x and ?f' =  $\lambda A\ x.$  f' x * indicator A x

have ac: absolutely_continuous M (density M f) sets (density M f) = sets M
using borel by (auto intro!: absolutely_continuousI_density)
from split_space_into_finite_sets_and_rest[OF this]
obtain Q :: nat  $\Rightarrow$  'a set
  where Q: disjoint_family Q range Q  $\subseteq$  sets M
  and in_Q0:  $\bigwedge A. A \in \text{sets } M \implies A \cap (\bigcup i. Q\ i) = \{\} \implies \text{emeasure } M\ A =$ 
   $0 \wedge ?D\ f\ A = 0 \vee 0 < \text{emeasure } M\ A \wedge ?D\ f\ A = \infty$ 
  and Q_fin:  $\bigwedge i. ?D\ f\ (Q\ i) \neq \infty$  by force
  with borel_pos have in_Q0:  $\bigwedge A. A \in \text{sets } M \implies A \cap (\bigcup i. Q\ i) = \{\} \implies$ 
   $\text{emeasure } M\ A = 0 \wedge ?N\ A = 0 \vee 0 < \text{emeasure } M\ A \wedge ?N\ A = \infty$ 
  and Q_fin:  $\bigwedge i. ?N\ (Q\ i) \neq \infty$  by (auto simp: emeasure_density_subset_eq)

from Q have Q_sets[measurable]:  $\bigwedge i. Q\ i \in \text{sets } M$  by auto
let ?D = {x  $\in$  space M. f x  $\neq$  f' x}
have ?D  $\in$  sets M using borel by auto
have *:  $\bigwedge i\ x\ A. \bigwedge y::\text{ennreal}. y * \text{indicator } (Q\ i)\ x * \text{indicator } A\ x = y * \text{indicator}$ 
   $(Q\ i \cap A)\ x$ 
unfolding indicator_def by auto
have  $\forall i. AE\ x\ \text{in } M. ?f\ (Q\ i)\ x = ?f'\ (Q\ i)\ x$  using borel Q_fin Q_pos
by (intro finite_density_unique[THEN iffD1] allI)
  (auto intro!: f_measure_eqI simp: emeasure_density * subset_eq)
moreover have AE x in M. ?f (space M - ( $\bigcup i. Q\ i$ )) x = ?f' (space M -
  ( $\bigcup i. Q\ i$ )) x
proof (rule AE_I')
  { fix f :: 'a  $\Rightarrow$  ennreal assume borel: f  $\in$  borel_measurable M
    and eq:  $\bigwedge A. A \in \text{sets } M \implies ?N\ A = (\int^+ x. f\ x * \text{indicator } A\ x\ \partial M)$ 
    let ?A =  $\lambda i. (\text{space } M - (\bigcup i. Q\ i)) \cap \{x \in \text{space } M. f\ x < (i::\text{nat})\}$ 
    have ( $\bigcup i. ?A\ i$ )  $\in$  null_sets M
    proof (rule null_sets_UN)
      fix i :: nat have ?A i  $\in$  sets M
      using borel by auto
      have ?N (?A i)  $\leq (\int^+ x. (i::\text{ennreal}) * \text{indicator } (?A\ i)\ x\ \partial M)$ 
      unfolding eq[OF  $\langle ?A\ i \in \text{sets } M \rangle$ ]
      by (auto intro!: nn_integral_mono simp: indicator_def)
      also have ... = i * emeasure M (?A i)
      using  $\langle ?A\ i \in \text{sets } M \rangle$  by (auto intro!: nn_integral_cmult_indicator)
      also have ...  $< \infty$  using emeasure_real[of ?A i] by (auto simp: en-
      nreal_mult_less_top of_nat_less_top)
      finally have ?N (?A i)  $\neq \infty$  by simp
      then show ?A i  $\in$  null_sets M using in_Q0[OF  $\langle ?A\ i \in \text{sets } M \rangle$ ]  $\langle ?A\ i$ 
       $\in \text{sets } M \rangle$  by auto
    }
qed
also have ( $\bigcup i. ?A\ i$ ) = (space M - ( $\bigcup i. Q\ i$ ))  $\cap \{x \in \text{space } M. f\ x \neq \infty\}$ 
by (auto simp: ennreal_Ex_less_of_nat_less_top[symmetric])

```

finally have $(\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f x \neq \infty\} \in \text{null\_sets } M$ **by** *simp* }
from *this*[*OF borel(1) refl*] *this*[*OF borel(2) f*]
have $(\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f x \neq \infty\} \in \text{null\_sets } M$ $(\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f' x \neq \infty\} \in \text{null\_sets } M$ **by** *simp_all*
then show $((\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f x \neq \infty\}) \cup ((\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f' x \neq \infty\}) \in \text{null\_sets } M$ **by** (*rule null_sets.Un*)
show $\{x \in \text{space } M. ?f (\text{space } M - (\bigcup i. Q i)) x \neq ?f' (\text{space } M - (\bigcup i. Q i)) x\} \subseteq$
 $((\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f x \neq \infty\}) \cup ((\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f' x \neq \infty\})$ **by** (*auto simp: indicator_def*)
qed
moreover have $AE x \text{ in } M. (?f (\text{space } M - (\bigcup i. Q i)) x = ?f' (\text{space } M - (\bigcup i. Q i)) x) \longrightarrow (\forall i. ?f (Q i) x = ?f' (Q i) x) \longrightarrow$
 $?f (\text{space } M) x = ?f' (\text{space } M) x$
by (*auto simp: indicator_def*)
ultimately have $AE x \text{ in } M. ?f (\text{space } M) x = ?f' (\text{space } M) x$
unfolding *AE_all_countable[symmetric]*
by *eventually_elim (auto split: if_split_asm simp: indicator_def)*
then show $AE x \text{ in } M. f x = f' x$ **by** *auto*
qed

proposition (*in sigma_finite_measure*) *density_unique*:

assumes $f: f \in \text{borel\_measurable } M$
assumes $f': f' \in \text{borel\_measurable } M$
assumes *density_eq*: $\text{density } M f = \text{density } M f'$
shows $AE x \text{ in } M. f x = f' x$

proof –

obtain h **where** $h\_borel: h \in \text{borel\_measurable } M$
and $fin: \text{integral}^N M h \neq \infty$ **and** $pos: \bigwedge x. x \in \text{space } M \implies 0 < h x \wedge h x < \infty \wedge x. 0 \leq h x$
using *Ex_finite_integrable_function* **by** *auto*
then have $h\_nn: AE x \text{ in } M. 0 \leq h x$ **by** *auto*
let $?H = \text{density } M h$
interpret $h: \text{finite\_measure } ?H$
using $fin h\_borel pos$
by (*intro finite_measureI*) (*simp cong: nn_integral_cong emeasure_density add: fin*)
let $?fM = \text{density } M f$
let $?f'M = \text{density } M f'$
{ fix $A \in \text{sets } M$
then have $\{x \in \text{space } M. h x * \text{indicator } A x \neq 0\} = A$
using *pos(1) sets.sets_into_space* **by** (*force simp: indicator_def*)
then have $(\int^+ x. h x * \text{indicator } A x \partial M) = 0 \iff A \in \text{null\_sets } M$
using $h\_borel \langle A \in \text{sets } M \rangle h\_nn$ **by** (*subst nn_integral_0_iff*) *auto* }
note $h\_null\_sets = \text{this}$
{ fix $A \in \text{sets } M$
have $(\int^+ x. f x * (h x * \text{indicator } A x) \partial M) = (\int^+ x. h x * \text{indicator } A x \partial ?fM)$

```

    using ‹A ∈ sets M› h_borel h_nn f f'
  by (intro nn_integral_density[symmetric]) auto
also have ... = (∫+x. h x * indicator A x ∂?f'M)
  by (simp_all add: density_eq)
also have ... = (∫+x. f' x * (h x * indicator A x) ∂M)
  using ‹A ∈ sets M› h_borel h_nn f f'
  by (intro nn_integral_density) auto
finally have (∫+x. h x * (f x * indicator A x) ∂M) = (∫+x. h x * (f' x *
indicator A x) ∂M)
  by (simp add: ac_simps)
then have (∫+x. (f x * indicator A x) ∂?H) = (∫+x. (f' x * indicator A x)
∂?H)
  using ‹A ∈ sets M› h_borel h_nn f f'
  by (subst (asm) (1 2) nn_integral_density[symmetric]) auto }
then have AE x in ?H. f x = f' x using h_borel h_nn f f'
  by (intro h.density_unique_finite_measure absolutely_continuous_AE[of M])
auto
with AE_space[of M] pos show AE x in M. f x = f' x
  unfolding AE_density[OF h_borel] by auto
qed

```

```

lemma (in sigma_finite_measure) density_unique_iff:
  assumes f: f ∈ borel_measurable M and f': f' ∈ borel_measurable M
  shows density M f = density M f' ⟷ (AE x in M. f x = f' x)
  using density_unique[OF assms] density_cong[OF f f'] by auto

```

```

lemma sigma_finite_density_unique:
  assumes borel: f ∈ borel_measurable M g ∈ borel_measurable M
  and fin: sigma_finite_measure (density M f)
  shows density M f = density M g ⟷ (AE x in M. f x = g x)
proof
  assume AE x in M. f x = g x with borel show density M f = density M g
    by (auto intro: density_cong)
next
  assume eq: density M f = density M g
  interpret f: sigma_finite_measure density M f by fact
  from f.sigma_finite_incseq obtain A where cover: range A ⊆ sets (density M
f)
  ⋃ (range A) = space (density M f)
  ⋀ i. emeasure (density M f) (A i) ≠ ∞
  incseq A
  by auto
  have AE x in M. ∀ i. x ∈ A i ⟶ f x = g x
    unfolding AE_all_countable
  proof
    fix i
    have density (density M f) (indicator (A i)) = density (density M g) (indicator
(A i))
      unfolding eq ..

```

```

moreover have  $(\int^+ x. f\ x * \text{indicator}\ (A\ i)\ x\ \partial M) \neq \infty$ 
using cover(1) cover(3)[of i] borel by (auto simp: emeasure_density_subset_eq)
ultimately have  $AE\ x\ in\ M. f\ x * \text{indicator}\ (A\ i)\ x = g\ x * \text{indicator}\ (A\ i)\ x$ 
using borel cover(1)
by (intro finite_density_unique[THEN iffD1]) (auto simp: density_density_eq_subset_eq)
then show  $AE\ x\ in\ M. x \in A\ i \longrightarrow f\ x = g\ x$ 
by auto
qed
with AE_space show  $AE\ x\ in\ M. f\ x = g\ x$ 
apply eventually_elim
using cover(2)[symmetric]
apply auto
done
qed

```

```

lemma (in sigma_finite_measure) sigma_finite_iff_density_finite':
assumes  $f: f \in \text{borel\_measurable}\ M$ 
shows  $\text{sigma\_finite\_measure}\ (\text{density}\ M\ f) \longleftrightarrow (AE\ x\ in\ M. f\ x \neq \infty)$ 
(is sigma_finite_measure ?N  $\longleftrightarrow$  _)
proof
assume sigma_finite_measure ?N
then interpret  $N: \text{sigma\_finite\_measure}\ ?N$  .
from  $N.Ex\_finite\_integrable\_function$  obtain  $h$  where
 $h: h \in \text{borel\_measurable}\ M\ \text{integral}^N\ ?N\ h \neq \infty$  and
 $fin: \forall x \in \text{space}\ M. 0 < h\ x \wedge h\ x < \infty$ 
by auto
have  $AE\ x\ in\ M. f\ x * h\ x \neq \infty$ 
proof (rule AE_I')
have  $\text{integral}^N\ ?N\ h = (\int^+ x. f\ x * h\ x\ \partial M)$ 
using  $f\ h$  by (auto intro!: nn_integral_density)
then have  $(\int^+ x. f\ x * h\ x\ \partial M) \neq \infty$ 
using  $h(2)$  by simp
then show  $(\lambda x. f\ x * h\ x) - \{\infty\} \cap \text{space}\ M \in \text{null\_sets}\ M$ 
using  $f\ h(1)$  by (auto intro!: nn_integral_PInf[unfolded infinity_ennreal_def]
borel_measurable_vimage)
qed auto
then show  $AE\ x\ in\ M. f\ x \neq \infty$ 
using  $fin$  by (auto elim!: AE_Ball_mp simp: less_top ennreal_mult_less_top)
next
assume  $AE: AE\ x\ in\ M. f\ x \neq \infty$ 
from sigma_finite obtain  $Q :: \text{nat} \Rightarrow 'a\ \text{set}$ 
where  $Q: \text{range}\ Q \subseteq \text{sets}\ M \cup (\text{range}\ Q) = \text{space}\ M \wedge i. \text{emeasure}\ M\ (Q\ i) \neq \infty$ 
by auto
define  $A$  where  $A\ i =$ 
 $f - \{(case\ i\ of\ 0 \Rightarrow \{\infty\} \mid Suc\ n \Rightarrow \{.. \text{ennreal}(\text{of\_nat}\ (Suc\ n))\}) \cap \text{space}\ M$ 
for  $i$ 

```



```

{ fix i j have  $A\ i \cap Q\ j \in \text{sets } M$ 
  unfolding  $A\_def$  using  $f\ Q$ 
  apply (rule_tac sets.Int)
  by (cases i) (auto intro: measurable_sets[OF  $f(1)$ ]) }
note  $A\_in\_sets = this$ 

show sigma_finite_measure ?N
proof (standard, intro exI conjI ballI)
  show countable (range ( $\lambda(i, j). A\ i \cap Q\ j$ ))
    by auto
  show range ( $\lambda(i, j). A\ i \cap Q\ j$ )  $\subseteq$  sets (density  $M\ f$ )
    using  $A\_in\_sets$  by auto
next
have  $\bigcup (\text{range } (\lambda(i, j). A\ i \cap Q\ j)) = (\bigcup i\ j. A\ i \cap Q\ j)$ 
  by auto
also have  $\dots = (\bigcup i. A\ i) \cap \text{space } M$  using  $Q$  by auto
also have  $(\bigcup i. A\ i) = \text{space } M$ 
proof safe
  fix x assume  $x \in \text{space } M$ 
  show  $x \in (\bigcup i. A\ i)$ 
  proof (cases f x rule: ennreal_cases)
    case top with x show ?thesis unfolding  $A\_def$  by (auto intro: exI[of _
0])
  next
    case (real r)
    with ennreal_Ex_less_of_nat[of f x] obtain  $n :: nat$  where  $f\ x < n$ 
      by auto
    also have  $n < (\text{Suc } n :: ennreal)$ 
      by simp
    finally show ?thesis
      using x real by (auto simp:  $A\_def$  ennreal_of_nat_eq_real_of_nat intro!:
exI[of _ Suc n])
  qed
qed (auto simp:  $A\_def$ )
finally show  $\bigcup (\text{range } (\lambda(i, j). A\ i \cap Q\ j)) = \text{space } ?N$  by simp
next
fix X assume  $X \in \text{range } (\lambda(i, j). A\ i \cap Q\ j)$ 
then obtain i j where [simp]:  $X = A\ i \cap Q\ j$  by auto
have  $(\int^+ x. f\ x * \text{indicator } (A\ i \cap Q\ j)\ x\ \partial M) \neq \infty$ 
proof (cases i)
  case 0
  have  $AE\ x\ \text{in } M. f\ x * \text{indicator } (A\ i \cap Q\ j)\ x = 0$ 
    using AE by (auto simp:  $A\_def$   $\langle i = 0 \rangle$ )
  from nn_integral_cong_AE[OF this] show ?thesis by simp
next
  case (Suc n)
  then have  $(\int^+ x. f\ x * \text{indicator } (A\ i \cap Q\ j)\ x\ \partial M) \leq$ 
     $(\int^+ x. (\text{Suc } n :: ennreal) * \text{indicator } (Q\ j)\ x\ \partial M)$ 
  by (auto intro!: nn_integral_mono simp: indicator_def  $A\_def$  ennreal_of_nat_eq_real_of_nat)

```

```

    also have ... = Suc n * emeasure M (Q j)
    using Q by (auto intro!: nn_integral_cmult_indicator)
    also have ... < ∞
    using Q by (auto simp: ennreal_mult_less_top less_top of_nat_less_top)
    finally show ?thesis by simp
qed
then show emeasure ?N X ≠ ∞
using A_in_sets Q f by (auto simp: emeasure_density)
qed
qed

lemma (in sigma_finite_measure) sigma_finite_iff_density_finite:
  f ∈ borel_measurable M ⇒ sigma_finite_measure (density M f) ⟷ (AE x in
M. f x ≠ ∞)
  by (subst sigma_finite_iff_density_finite')
    (auto simp: max_def intro!: measurable_Iif)

```

6.16.4 Radon-Nikodym derivative

definition $RN\_deriv :: 'a\ measure \Rightarrow 'a\ measure \Rightarrow 'a \Rightarrow ennreal$ **where**

```

RN_deriv M N =
  (if ∃ f. f ∈ borel_measurable M ∧ density M f = N
   then SOME f. f ∈ borel_measurable M ∧ density M f = N
   else (λ_. 0))

```

lemma RN_derivI :

```

assumes f ∈ borel_measurable M density M f = N
shows density M (RN_deriv M N) = N

```

proof –

```

have *: ∃ f. f ∈ borel_measurable M ∧ density M f = N
using assms by auto
then have density M (SOME f. f ∈ borel_measurable M ∧ density M f = N)
= N
  by (rule someI2_ex) auto
with * show ?thesis
  by (auto simp: RN_deriv_def)
qed

```

lemma $borel\_measurable\_RN\_deriv[measurable]$: $RN\_deriv\ M\ N \in borel\_measurable\ M$

proof –

```

{ assume ex: ∃ f. f ∈ borel_measurable M ∧ density M f = N
  have 1: (SOME f. f ∈ borel_measurable M ∧ density M f = N) ∈ borel_measurable
M
    using ex by (rule someI2_ex) auto }
from this show ?thesis
  by (auto simp: RN_deriv_def)
qed

```

lemma *density_RN_deriv_density:*

assumes $f: f \in \text{borel\_measurable } M$

shows $\text{density } M (\text{RN\_deriv } M (\text{density } M f)) = \text{density } M f$

by (rule RN_derivI[OF f]) simp

lemma (in *sigma_finite_measure*) *density_RN_deriv:*

absolutely_continuous $M \ N \implies \text{sets } N = \text{sets } M \implies \text{density } M (\text{RN\_deriv } M \ N) = N$

by (metis RN_derivI Radon_Nikodym)

lemma (in *sigma_finite_measure*) *RN_deriv_nn_integral:*

assumes $N: \text{absolutely\_continuous } M \ N \ \text{sets } N = \text{sets } M$

and $f: f \in \text{borel\_measurable } M$

shows $\text{integral}^N N f = (\int^+ x. \text{RN\_deriv } M \ N \ x * f \ x \ \partial M)$

proof –

have $\text{integral}^N N f = \text{integral}^N (\text{density } M (\text{RN\_deriv } M \ N)) f$

using N **by** (simp add: density_RN_deriv)

also have $\dots = (\int^+ x. \text{RN\_deriv } M \ N \ x * f \ x \ \partial M)$

using f **by** (simp add: nn_integral_density)

finally show ?thesis **by** simp

qed

lemma (in *sigma_finite_measure*) *RN_deriv_unique:*

assumes $f: f \in \text{borel\_measurable } M$

and $eq: \text{density } M f = N$

shows $\text{AE } x \text{ in } M. f \ x = \text{RN\_deriv } M \ N \ x$

unfolding eq[symmetric]

by (intro density_unique_iff[THEN iffD1] f borel_measurable_RN_deriv density_RN_deriv_density[symmetric])

lemma *RN_deriv_unique_sigma_finite:*

assumes $f: f \in \text{borel\_measurable } M$

and $eq: \text{density } M f = N$ **and** $fin: \text{sigma\_finite\_measure } N$

shows $\text{AE } x \text{ in } M. f \ x = \text{RN\_deriv } M \ N \ x$

using fin **unfolding** eq[symmetric]

by (intro sigma_finite_density_unique[THEN iffD1] f borel_measurable_RN_deriv density_RN_deriv_density[symmetric])

lemma (in *sigma_finite_measure*) *RN_deriv_distr:*

fixes $T :: 'a \Rightarrow 'b$

assumes $T: T \in \text{measurable } M \ M' \ \text{and } T': T' \in \text{measurable } M' \ M$

and $inv: \forall x \in \text{space } M. T' (T \ x) = x$

and $ac[\text{simp}]: \text{absolutely\_continuous } (\text{distr } M \ M' \ T) (\text{distr } N \ M' \ T)$

and $N: \text{sets } N = \text{sets } M$

shows $\text{AE } x \text{ in } M. \text{RN\_deriv } (\text{distr } M \ M' \ T) (\text{distr } N \ M' \ T) (T \ x) = \text{RN\_deriv } M \ N \ x$

proof (rule RN_deriv_unique)

have [simp]: $\text{sets } N = \text{sets } M$ **by** fact

note sets_eq_imp_space_eq[OF N, simp]

```

have measurable_N[simp]:  $\bigwedge M'. \text{measurable } N \ M' = \text{measurable } M \ M'$  by (auto simp: measurable_def)
{ fix A assume A  $\in$  sets M
  with inv T T' sets.sets_into_space[OF this]
  have T - ' T' - ' A  $\cap$  T - ' space M'  $\cap$  space M = A
    by (auto simp: measurable_def) }
note eq = this[simp]
{ fix A assume A  $\in$  sets M
  with inv T T' sets.sets_into_space[OF this]
  have (T'  $\circ$  T) - ' A  $\cap$  space M = A
    by (auto simp: measurable_def) }
note eq2 = this[simp]
let ?M' = distr M M' T and ?N' = distr N M' T
interpret M': sigma_finite_measure ?M'
proof
  from sigma_finite_countable obtain F
    where F: countable F  $\wedge$  F  $\subseteq$  sets M  $\wedge$   $\bigcup F = \text{space } M \wedge (\forall a \in F. \text{emeasure } M \ a \neq \infty)$  ..
  show  $\exists A. \text{countable } A \wedge A \subseteq \text{sets } (\text{distr } M \ M' \ T) \wedge \bigcup A = \text{space } (\text{distr } M \ M' \ T) \wedge (\forall a \in A. \text{emeasure } (\text{distr } M \ M' \ T) \ a \neq \infty)$ 
  proof (intro exI conjI ballI)
    show *:  $(\lambda A. T' - ' A \cap \text{space } ?M') \text{ ' } F \subseteq \text{sets } ?M'$ 
      using F T' by (auto simp: measurable_def)
    show  $\bigcup ((\lambda A. T' - ' A \cap \text{space } ?M') \text{ ' } F) = \text{space } ?M'$ 
      using F T' [THEN measurable_space] by (auto simp: set_eq_iff)
  next
    fix X assume X  $\in$   $(\lambda A. T' - ' A \cap \text{space } ?M') \text{ ' } F$ 
    then obtain A where [simp]: X = T' - ' A  $\cap$  space ?M' and A  $\in$  F by
auto
    have X  $\in$  sets M' using F T'  $\langle A \in F \rangle$  by auto
    moreover
    have Fi: A  $\in$  sets M using F  $\langle A \in F \rangle$  by auto
    ultimately show  $\text{emeasure } ?M' \ X \neq \infty$ 
      using F T T'  $\langle A \in F \rangle$  by (simp add: emeasure_distr)
  qed (use F in auto)
qed
have (RN_deriv ?M' ?N')  $\circ$  T  $\in$  borel_measurable M
  using T ac by measurable
then show  $(\lambda x. \text{RN\_deriv } ?M' ?N' (T \ x)) \in \text{borel\_measurable } M$ 
  by (simp add: comp_def)

have N = distr N M (T'  $\circ$  T)
  by (subst measure_of_of_measure[of N, symmetric])
  (auto simp add: distr_def sets.sigma_sets_eq intro!: measure_of_eq sets.space_closed)
also have ... = distr (distr N M' T) M T'
  using T T' by (simp add: distr_distr)
also have ... = distr (density (distr M M' T) (RN_deriv (distr M M' T) (distr N M' T))) M T'
  using ac by (simp add: M'.density_RN_deriv)

```

```

also have ... = density M (RN_deriv (distr M M' T) (distr N M' T) o T)
  by (simp add: distr_density_distr[OF T T', OF inv])
finally show density M ( $\lambda x$ . RN_deriv (distr M M' T) (distr N M' T) (T x))
= N
  by (simp add: comp_def)
qed

```

```

lemma (in sigma_finite_measure) RN_deriv_finite:
  assumes N: sigma_finite_measure N and ac: absolutely_continuous M N sets
  N = sets M
  shows AE x in M. RN_deriv M N x  $\neq \infty$ 
proof -
  interpret N: sigma_finite_measure N by fact
  from N show ?thesis
  using sigma_finite_iff_density_finite[OF borel_measurable_RN_deriv, of N]
  density_RN_deriv[OF ac]
  by simp
qed

```

```

lemma (in sigma_finite_measure)
  assumes N: sigma_finite_measure N and ac: absolutely_continuous M N sets
  N = sets M
  and f: f  $\in$  borel_measurable M
  shows RN_deriv_integrable: integrable N f  $\longleftrightarrow$ 
    integrable M ( $\lambda x$ . enn2real (RN_deriv M N x) * f x) (is ?integrable)
  and RN_deriv_integral: integralL N f = ( $\int x$ . enn2real (RN_deriv M N x) *
  f x  $\partial$ M) (is ?integral)
proof -
  note ac(2)[simp] and sets_eq_imp_space_eq[OF ac(2), simp]
  interpret N: sigma_finite_measure N by fact

```

```

  have eq: density M (RN_deriv M N) = density M ( $\lambda x$ . enn2real (RN_deriv M
  N x))
  proof (rule density_cong)
    from RN_deriv_finite[OF assms(1,2,3)]
    show AE x in M. RN_deriv M N x = ennreal (enn2real (RN_deriv M N x))
    by eventually_elim (auto simp: less_top)
  qed (insert ac, auto)

```

```

show ?integrable
  apply (subst density_RN_deriv[OF ac, symmetric])
  unfolding eq
  apply (intro integrable_real_density f AE_I2 enn2real_nonneg)
  apply (insert ac, auto)
  done

```

```

show ?integral
  apply (subst density_RN_deriv[OF ac, symmetric])
  unfolding eq

```

1970

```

    apply (intro integral_real_density f AE_I2 enn2real_nonneg)
    apply (insert ac, auto)
  done
qed

```

proposition (in *sigma_finite_measure*) *real_RN_deriv*:

```

  assumes finite_measure N
  assumes ac: absolutely_continuous M N sets N = sets M
  obtains D where D ∈ borel_measurable M
    and AE x in M. RN_deriv M N x = ennreal (D x)
    and AE x in N. 0 < D x
    and  $\bigwedge x. 0 \leq D x$ 

```

proof

```

  interpret N: finite_measure N by fact

```

```

  note RN = borel_measurable_RN_deriv density_RN_deriv[OF ac]

```

```

  let ?RN =  $\lambda t. \{x \in \text{space } M. \text{RN\_deriv } M \ N \ x = t\}$ 

```

```

  show ( $\lambda x. \text{enn2real } (\text{RN\_deriv } M \ N \ x)$ ) ∈ borel_measurable M
  using RN by auto

```

```

  have N (?RN ∞) = ( $\int^+ x. \text{RN\_deriv } M \ N \ x * \text{indicator } (?RN \ \infty) \ x \ \partial M$ )
    using RN(1) by (subst RN(2)[symmetric]) (auto simp: emeasure_density)
  also have ... = ( $\int^+ x. \infty * \text{indicator } (?RN \ \infty) \ x \ \partial M$ )
    by (intro nn_integral_cong) (auto simp: indicator_def)
  also have ... =  $\infty * \text{emeasure } M \ (?RN \ \infty)$ 
    using RN by (intro nn_integral_cmult_indicator) auto
  finally have eq: N (?RN ∞) =  $\infty * \text{emeasure } M \ (?RN \ \infty)$  .

```

moreover

```

  have emeasure M (?RN ∞) = 0

```

proof (rule ccontr)

```

  assume emeasure M {x ∈ space M. RN_deriv M N x = ∞} ≠ 0
  then have 0 < emeasure M {x ∈ space M. RN_deriv M N x = ∞}
    by (auto simp: zero_less_iff_neq_zero)
  with eq have N (?RN ∞) = ∞ by (simp add: ennreal_mult_eq_top_iff)
  with N.emeasure_finite[of ?RN ∞] RN show False by auto

```

qed

```

  ultimately have AE x in M. RN_deriv M N x < ∞

```

```

  using RN by (intro AE_iff_measurable[THEN iffD2]) (auto simp: less_top[symmetric])

```

```

  then show AE x in M. RN_deriv M N x = ennreal (enn2real (RN_deriv M N
x))

```

```

    by auto

```

```

  then have eq: AE x in N. RN_deriv M N x = ennreal (enn2real (RN_deriv M
N x))

```

```

    using ac absolutely_continuous_AE by auto

```

```

  have N (?RN 0) = ( $\int^+ x. \text{RN\_deriv } M \ N \ x * \text{indicator } (?RN \ 0) \ x \ \partial M$ )

```

```

    by (subst RN(2)[symmetric]) (auto simp: emeasure_density)
  also have ... = ( $\int^+ x. 0 \ \partial M$ )
    by (intro nn_integral_cong) (auto simp: indicator_def)
  finally have AE x in N. RN_deriv M N x  $\neq 0$ 
    using RN by (subst AE_iff_measurable[OF _ refl]) (auto simp: ac cong:
sets_eq_imp_space_eq)
  with eq show AE x in N.  $0 < \text{enn2real} \ (RN\_deriv \ M \ N \ x)$ 
    by (auto simp: enn2real_positive_iff_less_top[symmetric] zero_less_iff_neq_zero)
qed (rule enn2real_nonneg)

```

```

lemma (in sigma_finite_measure) RN_deriv_singleton:
  assumes ac: absolutely_continuous M N sets N = sets M
  and x: {x}  $\in$  sets M
  shows N {x} = RN_deriv M N x * emeasure M {x}
proof -
  from  $\langle \{x\} \in \text{sets } M \rangle$ 
  have density M (RN_deriv M N) {x} = ( $\int^+ w. RN\_deriv \ M \ N \ x \ * \ \text{indicator} \ \{x\} \ w \ \partial M$ )
    by (auto simp: indicator_def emeasure_density intro!: nn_integral_cong)
  with x density_RN_deriv[OF ac] show ?thesis
    by (auto simp: max_def)
qed
end

```

```

theory Set_Integral
  imports Radon_Nikodym
begin

```

definition *set_borel_measurable* $M \ A \ f \equiv (\lambda x. \text{indicator } A \ x *_{\mathbb{R}} f \ x) \in \text{borel\_measurable } M$

definition *set_integrable* $M \ A \ f \equiv \text{integrable } M \ (\lambda x. \text{indicator } A \ x *_{\mathbb{R}} f \ x)$

definition *set_lebesgue_integral* $M \ A \ f \equiv \text{lebesgue\_integral } M \ (\lambda x. \text{indicator } A \ x *_{\mathbb{R}} f \ x)$

```

syntax
  _asciï_set_lebesgue_integral :: pttrn  $\Rightarrow$  'a set  $\Rightarrow$  'a measure  $\Rightarrow$  real  $\Rightarrow$  real
  ((4LINT ( $\_$ ):( $\_$ )/|( $\_$ )./  $\_$ ) [0,60,110,61] 60)

```

translations
 $LINT \ x:A|M. \ f == \text{CONST } \text{set\_lebesgue\_integral } M \ A \ (\lambda x. \ f)$

1972

syntax

$\text{\_lebesgue\_borel\_integral} :: \text{pttrn} \Rightarrow \text{real} \Rightarrow \text{real}$
 $((2LBINT \text{\_./\_}) [0,60] 60)$

syntax

$\text{\_set\_lebesgue\_borel\_integral} :: \text{pttrn} \Rightarrow \text{real set} \Rightarrow \text{real} \Rightarrow \text{real}$
 $((3LBINT \text{\_:\_./\_}) [0,60,61] 60)$

lemma *set_integrable_cong*:

assumes $M = M' \ A = A' \ \wedge x. x \in A \implies f x = f' x$
shows $\text{set\_integrable } M \ A \ f = \text{set\_integrable } M' \ A' \ f'$

proof –

have $(\lambda x. \text{indicator } A \ x *_{\mathbb{R}} f x) = (\lambda x. \text{indicator } A' \ x *_{\mathbb{R}} f' x)$
using *assms* **by** (*auto simp: indicator_def of_bool_def*)
thus *?thesis* **by** (*simp add: set_integrable_def assms*)

qed

lemma *set_borel_measurable_sets*:

fixes $f :: \_ \Rightarrow \_ :: \text{real\_normed\_vector}$
assumes $\text{set\_borel\_measurable } M \ X \ f \ B \in \text{sets borel } X \in \text{sets } M$
shows $f -' B \cap X \in \text{sets } M$

proof –

have $f \in \text{borel\_measurable } (\text{restrict\_space } M \ X)$
using *assms* **unfolding** *set_borel_measurable_def* **by** (*subst borel_measurable_restrict_space_iff*)
auto
then have $f -' B \cap \text{space } (\text{restrict\_space } M \ X) \in \text{sets } (\text{restrict\_space } M \ X)$
by (*rule measurable_sets fact*)
with $\langle X \in \text{sets } M \rangle$ **show** *?thesis*
by (*subst (asm) sets_restrict_space_iff*) (*auto simp: space_restrict_space*)

qed

lemma *set_lebesgue_integral_zero* [*simp*]: $\text{set\_lebesgue\_integral } M \ A \ (\lambda x. 0) = 0$

by (*auto simp: set_lebesgue_integral_def*)

lemma *set_lebesgue_integral_cong*:

assumes $A \in \text{sets } M$ **and** $\forall x. x \in A \implies f x = g x$
shows $(LINT x:A|M. f x) = (LINT x:A|M. g x)$
unfolding *set_lebesgue_integral_def*
using *assms*
by (*metis indicator_simps(2) real_vector.scale_zero_left*)

lemma *set_lebesgue_integral_cong_AE*:

assumes [*measurable*]: $A \in \text{sets } M \ f \in \text{borel\_measurable } M \ g \in \text{borel\_measurable } M$


```

    assumes  $AE\ x \in A\ in\ M. f\ x = g\ x$ 
    shows  $LINT\ x:A|M. f\ x = LINT\ x:A|M. g\ x$ 
  proof-
    have  $AE\ x\ in\ M. indicator\ A\ x *_{\mathbb{R}} f\ x = indicator\ A\ x *_{\mathbb{R}} g\ x$ 
      using assms by auto
    thus ?thesis
      unfolding set_lebesgue_integral_def by (intro integral_cong_AE) auto
  qed

```

```

lemma set_integrable_cong_AE:
   $f \in borel\_measurable\ M \implies g \in borel\_measurable\ M \implies$ 
   $AE\ x \in A\ in\ M. f\ x = g\ x \implies A \in sets\ M \implies$ 
   $set\_integrable\ M\ A\ f = set\_integrable\ M\ A\ g$ 
  unfolding set_integrable_def
  by (rule integrable_cong_AE) auto

```

```

lemma set_integrable_subset:
  fixes  $M\ A\ B$  and  $f :: \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$ 
  assumes  $set\_integrable\ M\ A\ f$  and  $B \in sets\ M\ B \subseteq A$ 
  shows  $set\_integrable\ M\ B\ f$ 
  proof -
    have  $set\_integrable\ M\ B\ (\lambda x. indicator\ A\ x *_{\mathbb{R}} f\ x)$ 
      using assms integrable_mult_indicator set_integrable_def by blast
    with  $\langle B \subseteq A \rangle$  show ?thesis
      unfolding set_integrable_def
      by (simp add: indicator_inter_arith[symmetric] Int_absorb2)
  qed

```

```

lemma set_integrable_restrict_space:
  fixes  $f :: 'a \Rightarrow 'b :: \{banach, second\_countable\_topology\}$ 
  assumes  $f: set\_integrable\ M\ S\ f$  and  $T: T \in sets\ (restrict\_space\ M\ S)$ 
  shows  $set\_integrable\ M\ T\ f$ 
  proof -
    obtain  $T'$  where  $T\_eq: T = S \cap T'$  and  $T' \in sets\ M$ 
      using  $T$  by (auto simp: sets_restrict_space)
    have  $\langle integrable\ M\ (\lambda x. indicator\ T'\ x *_{\mathbb{R}} (indicator\ S\ x *_{\mathbb{R}} f\ x)) \rangle$ 
      using  $\langle T' \in sets\ M \rangle$  f integrable_mult_indicator set_integrable_def by blast
    then show ?thesis
      unfolding set_integrable_def
      unfolding  $T\_eq$  indicator_inter_arith by (simp add: ac_simps)
  qed

```

```

lemma set_integral_scaleR_right [simp]:  $LINT\ t:A|M. a *_{\mathbb{R}} f\ t = a *_{\mathbb{R}} (LINT\ t:A|M. f\ t)$ 
  unfolding set_lebesgue_integral_def
  by (subst integral_scaleR_right[symmetric]) (auto intro!: Bochner_Integration.integral_cong)

```

```

lemma set_integral_mult_right [simp]:
  fixes a :: 'a::{real_normed_field, second_countable_topology}
  shows  $LINT\ t:A|M. a * f\ t = a * (LINT\ t:A|M. f\ t)$ 
  unfolding set_lebesgue_integral_def
  by (subst integral_mult_right_zero[symmetric]) auto

lemma set_integral_mult_left [simp]:
  fixes a :: 'a::{real_normed_field, second_countable_topology}
  shows  $LINT\ t:A|M. f\ t * a = (LINT\ t:A|M. f\ t) * a$ 
  unfolding set_lebesgue_integral_def
  by (subst integral_mult_left_zero[symmetric]) auto

lemma set_integral_divide_zero [simp]:
  fixes a :: 'a::{real_normed_field, field, second_countable_topology}
  shows  $LINT\ t:A|M. f\ t / a = (LINT\ t:A|M. f\ t) / a$ 
  unfolding set_lebesgue_integral_def
  by (subst integral_divide_zero[symmetric], intro Bochner_Integration.integral_cong)
    (auto split: split_indicator)

lemma set_integrable_scaleR_right [simp, intro]:
  shows  $(a \neq 0 \implies set\_integrable\ M\ A\ f) \implies set\_integrable\ M\ A\ (\lambda t. a *_{\mathbb{R}} f\ t)$ 
  unfolding set_integrable_def
  unfolding scaleR_left_commute by (rule integrable_scaleR_right)

lemma set_integrable_scaleR_left [simp, intro]:
  fixes a :: _ :: {banach, second_countable_topology}
  shows  $(a \neq 0 \implies set\_integrable\ M\ A\ f) \implies set\_integrable\ M\ A\ (\lambda t. f\ t *_{\mathbb{R}} a)$ 
  unfolding set_integrable_def
  using integrable_scaleR_left[of a M  $\lambda x. indicator\ A\ x *_{\mathbb{R}} f\ x$ ] by simp

lemma set_integrable_mult_right [simp, intro]:
  fixes a :: 'a::{real_normed_field, second_countable_topology}
  shows  $(a \neq 0 \implies set\_integrable\ M\ A\ f) \implies set\_integrable\ M\ A\ (\lambda t. a * f\ t)$ 
  unfolding set_integrable_def
  using integrable_mult_right[of a M  $\lambda x. indicator\ A\ x *_{\mathbb{R}} f\ x$ ] by simp

lemma set_integrable_mult_right_iff [simp]:
  fixes a :: 'a::{real_normed_field, second_countable_topology}
  assumes  $a \neq 0$ 
  shows  $set\_integrable\ M\ A\ (\lambda t. a * f\ t) \longleftrightarrow set\_integrable\ M\ A\ f$ 
proof
  assume set_integrable M A  $(\lambda t. a * f\ t)$ 
  then have set_integrable M A  $(\lambda t. 1/a * (a * f\ t))$ 
    using set_integrable_mult_right by blast
  then show set_integrable M A f
    using assms by auto
qed auto

```

lemma *set_integrable_mult_left* [*simp*, *intro*]:
fixes $a :: 'a :: \{\text{real\_normed\_field}, \text{second\_countable\_topology}\}$
shows $(a \neq 0 \implies \text{set\_integrable } M \ A \ f) \implies \text{set\_integrable } M \ A \ (\lambda t. f \ t * a)$
unfolding *set_integrable_def*
using *integrable_mult_left*[of $a \ M \ \lambda x. \text{indicator } A \ x *_R f \ x$] **by** *simp*

lemma *set_integrable_mult_left_iff* [*simp*]:
fixes $a :: 'a :: \{\text{real\_normed\_field}, \text{second\_countable\_topology}\}$
assumes $a \neq 0$
shows $\text{set\_integrable } M \ A \ (\lambda t. f \ t * a) \longleftrightarrow \text{set\_integrable } M \ A \ f$
using *assms* **by** (*subst set_integrable_mult_right_iff [symmetric]*) (*auto simp: mult.commute*)

lemma *set_integrable_divide* [*simp*, *intro*]:
fixes $a :: 'a :: \{\text{real\_normed\_field}, \text{field}, \text{second\_countable\_topology}\}$
assumes $a \neq 0 \implies \text{set\_integrable } M \ A \ f$
shows $\text{set\_integrable } M \ A \ (\lambda t. f \ t / a)$
proof –
have $\text{integrable } M \ (\lambda x. \text{indicator } A \ x *_R f \ x / a)$
using *assms* **unfolding** *set_integrable_def* **by** (*rule integrable_divide_zero*)
also have $(\lambda x. \text{indicator } A \ x *_R f \ x / a) = (\lambda x. \text{indicator } A \ x *_R (f \ x / a))$
by (*auto split: split_indicator*)
finally show *?thesis*
unfolding *set_integrable_def* .
qed

lemma *set_integrable_mult_divide_iff* [*simp*]:
fixes $a :: 'a :: \{\text{real\_normed\_field}, \text{second\_countable\_topology}\}$
assumes $a \neq 0$
shows $\text{set\_integrable } M \ A \ (\lambda t. f \ t / a) \longleftrightarrow \text{set\_integrable } M \ A \ f$
by (*simp add: divide_inverse assms*)

lemma *set_integral_add* [*simp*, *intro*]:
fixes $f \ g :: \_ \Rightarrow \_ :: \{\text{banach}, \text{second\_countable\_topology}\}$
assumes $\text{set\_integrable } M \ A \ f \ \text{set\_integrable } M \ A \ g$
shows $\text{set\_integrable } M \ A \ (\lambda x. f \ x + g \ x)$
and $LINT \ x:A | M. f \ x + g \ x = (LINT \ x:A | M. f \ x) + (LINT \ x:A | M. g \ x)$
using *assms* **unfolding** *set_integrable_def set_lebesgue_integral_def* **by** (*simp_all add: scaleR_add_right*)

lemma *set_integral_diff* [*simp*, *intro*]:
assumes $\text{set\_integrable } M \ A \ f \ \text{set\_integrable } M \ A \ g$
shows $\text{set\_integrable } M \ A \ (\lambda x. f \ x - g \ x)$ **and** $LINT \ x:A | M. f \ x - g \ x =$
 $(LINT \ x:A | M. f \ x) - (LINT \ x:A | M. g \ x)$
using *assms* **unfolding** *set_integrable_def set_lebesgue_integral_def* **by** (*simp_all add: scaleR_diff_right*)

lemma *set_integral_uminus*: $\text{set\_integrable } M \ A \ f \implies LINT \ x:A | M. - f \ x = -$
 $(LINT \ x:A | M. f \ x)$

unfolding *set_integrable_def set_lebesgue_integral_def*
by (*subst integral_minus[symmetric]*) *simp_all*

lemma *set_integral_complex_of_real*:
 $LINT\ x:A|M.\ complex\_of\_real\ (f\ x) = of\_real\ (LINT\ x:A|M.\ f\ x)$
unfolding *set_lebesgue_integral_def*
by (*subst integral_complex_of_real[symmetric]*)
(auto intro!: Bochner_Integration.integral_cong split: split_indicator)

lemma *set_integral_mono*:
fixes $f\ g :: \_ \Rightarrow real$
assumes *set_integrable M A f set_integrable M A g*
 $\bigwedge x. x \in A \implies f\ x \leq g\ x$
shows $(LINT\ x:A|M.\ f\ x) \leq (LINT\ x:A|M.\ g\ x)$
using *assms unfolding set_integrable_def set_lebesgue_integral_def*
by (*auto intro: integral_mono split: split_indicator*)

lemma *set_integral_mono_AE*:
fixes $f\ g :: \_ \Rightarrow real$
assumes *set_integrable M A f set_integrable M A g*
 $AE\ x \in A\ in\ M.\ f\ x \leq g\ x$
shows $(LINT\ x:A|M.\ f\ x) \leq (LINT\ x:A|M.\ g\ x)$
using *assms unfolding set_integrable_def set_lebesgue_integral_def*
by (*auto intro: integral_mono_AE split: split_indicator*)

lemma *set_integrable_abs*: $set\_integrable\ M\ A\ f \implies set\_integrable\ M\ A\ (\lambda x. |f\ x| :: real)$
using *integrable_abs[of M $\lambda x. f\ x * indicator\ A\ x$]* **unfolding** *set_integrable_def*
by (*simp add: abs_mult ac_simps*)

lemma *set_integrable_abs_iff*:
fixes $f :: \_ \Rightarrow real$
shows $set\_borel\_measurable\ M\ A\ f \implies set\_integrable\ M\ A\ (\lambda x. |f\ x|) = set\_integrable\ M\ A\ f$
unfolding *set_integrable_def set_borel_measurable_def*
by (*subst (2) integrable_abs_iff[symmetric]*) (*simp_all add: abs_mult ac_simps*)

lemma *set_integrable_abs_iff'*:
fixes $f :: \_ \Rightarrow real$
shows $f \in borel\_measurable\ M \implies A \in sets\ M \implies$
 $set\_integrable\ M\ A\ (\lambda x. |f\ x|) = set\_integrable\ M\ A\ f$
by (*simp add: set_borel_measurable_def set_integrable_abs_iff*)

lemma *set_integrable_discrete_difference*:
fixes $f :: 'a \Rightarrow 'b :: \{banach, second\_countable\_topology\}$
assumes *countable X*
assumes *diff: $(A - B) \cup (B - A) \subseteq X$*
assumes $\bigwedge x. x \in X \implies emeasure\ M\ \{x\} = 0$ $\bigwedge x. x \in X \implies \{x\} \in sets\ M$
shows $set\_integrable\ M\ A\ f \longleftrightarrow set\_integrable\ M\ B\ f$

```

  unfolding set_integrable_def
proof (rule integrable_discrete_difference[where X=X])
  show  $\bigwedge x. x \in \text{space } M \implies x \notin X \implies \text{indicator } A \ x *_R f \ x = \text{indicator } B \ x *_R f \ x$ 
  using diff by (auto split: split_indicator)
qed fact+

```

```

lemma set_integral_discrete_difference:
  fixes f :: 'a  $\Rightarrow$  'b :: {banach, second_countable_topology}
  assumes countable X
  assumes diff:  $(A - B) \cup (B - A) \subseteq X$ 
  assumes  $\bigwedge x. x \in X \implies \text{emeasure } M \ \{x\} = 0$ 
  shows  $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$ 
  shows set_lebesgue_integral M A f = set_lebesgue_integral M B f
  unfolding set_lebesgue_integral_def
proof (rule integral_discrete_difference[where X=X])
  show  $\bigwedge x. x \in \text{space } M \implies x \notin X \implies \text{indicator } A \ x *_R f \ x = \text{indicator } B \ x *_R f \ x$ 
  using diff by (auto split: split_indicator)
qed fact+

```

```

lemma set_integrable_Un:
  fixes f g :: _  $\Rightarrow$  _ :: {banach, second_countable_topology}
  assumes f_A: set_integrable M A f and f_B: set_integrable M B f
  and [measurable]:  $A \in \text{sets } M \ B \in \text{sets } M$ 
  shows set_integrable M  $(A \cup B) \ f$ 
proof -
  have set_integrable M  $(A - B) \ f$ 
  using f_A by (rule set_integrable_subset) auto
  with f_B have integrable M  $(\lambda x. \text{indicator } (A - B) \ x *_R f \ x + \text{indicator } B \ x *_R f \ x)$ 
  unfolding set_integrable_def using integrable_add by blast
  then show ?thesis
  unfolding set_integrable_def
  by (rule integrable_cong[THEN iffD1, rotated 2]) (auto split: split_indicator)
qed

```

```

lemma set_integrable_empty [simp]: set_integrable M {} f
  by (auto simp: set_integrable_def)

```

```

lemma set_integrable_UN:
  fixes f :: _  $\Rightarrow$  _ :: {banach, second_countable_topology}
  assumes finite I  $\bigwedge i. i \in I \implies \text{set_integrable } M \ (A \ i) \ f$ 
   $\bigwedge i. i \in I \implies A \ i \in \text{sets } M$ 
  shows set_integrable M  $(\bigcup i \in I. A \ i) \ f$ 
  using assms
  by (induct I) (auto simp: set_integrable_Un sets.finite_UN)

```

```

lemma set_integral_Un:
  fixes f :: _  $\Rightarrow$  _ :: {banach, second_countable_topology}

```

```

    assumes  $A \cap B = \{\}$ 
    and  $\text{set\_integrable } M \ A \ f$ 
    and  $\text{set\_integrable } M \ B \ f$ 
  shows  $\text{LINT } x:A \cup B | M. f \ x = (\text{LINT } x:A | M. f \ x) + (\text{LINT } x:B | M. f \ x)$ 
    using assms
    unfolding  $\text{set\_integrable\_def}$   $\text{set\_lebesgue\_integral\_def}$ 
    by (auto simp add: indicator\_union\_arith indicator\_inter\_arith[symmetric] scaleR\_add\_left)

```

```

lemma  $\text{set\_integral\_cong\_set}$ :
  fixes  $f :: \_ \Rightarrow \_ :: \{\text{banach, second\_countable\_topology}\}$ 
  assumes  $\text{set\_borel\_measurable } M \ A \ f$   $\text{set\_borel\_measurable } M \ B \ f$ 
    and  $ae: AE \ x \text{ in } M. x \in A \longleftrightarrow x \in B$ 
  shows  $\text{LINT } x:B | M. f \ x = \text{LINT } x:A | M. f \ x$ 
    unfolding  $\text{set\_lebesgue\_integral\_def}$ 
  proof (rule  $\text{integral\_cong\_AE}$ )
    show  $AE \ x \text{ in } M. \text{indicator } B \ x *_R f \ x = \text{indicator } A \ x *_R f \ x$ 
      using ae by (auto simp: subset\_eq split: split\_indicator)
  qed (use assms in  $\langle \text{auto simp: set\_borel\_measurable\_def} \rangle$ )

```

```

proposition  $\text{set\_borel\_measurable\_subset}$ :
  fixes  $f :: \_ \Rightarrow \_ :: \{\text{banach, second\_countable\_topology}\}$ 
  assumes [measurable]:  $\text{set\_borel\_measurable } M \ A \ f$   $B \in \text{sets } M$  and  $B \subseteq A$ 
  shows  $\text{set\_borel\_measurable } M \ B \ f$ 
  proof -
    have  $\text{set\_borel\_measurable } M \ B \ (\lambda x. \text{indicator } A \ x *_R f \ x)$ 
      using assms unfolding  $\text{set\_borel\_measurable\_def}$ 
      using  $\text{borel\_measurable\_indicator borel\_measurable\_scaleR}$  by blast
    moreover have  $(\lambda x. \text{indicator } B \ x *_R \text{indicator } A \ x *_R f \ x) = (\lambda x. \text{indicator } B$ 
 $x *_R f \ x)$ 
      using  $\langle B \subseteq A \rangle$  by (auto simp: fun\_eq\_iff split: split\_indicator)
    ultimately show ?thesis
      unfolding  $\text{set\_borel\_measurable\_def}$  by simp
  qed

```

```

lemma  $\text{set\_integral\_Un\_AE}$ :
  fixes  $f :: \_ \Rightarrow \_ :: \{\text{banach, second\_countable\_topology}\}$ 
  assumes  $ae: AE \ x \text{ in } M. \neg (x \in A \wedge x \in B)$  and [measurable]:  $A \in \text{sets } M \ B \in \text{sets } M$ 
    and  $\text{set\_integrable } M \ A \ f$ 
    and  $\text{set\_integrable } M \ B \ f$ 
  shows  $\text{LINT } x:A \cup B | M. f \ x = (\text{LINT } x:A | M. f \ x) + (\text{LINT } x:B | M. f \ x)$ 
  proof -
    have  $f: \text{set\_integrable } M \ (A \cup B) \ f$ 
      by (intro set\_integrable\_Un assms)
    then have  $f': \text{set\_borel\_measurable } M \ (A \cup B) \ f$ 
      using  $\text{integrable\_iff\_bounded set\_borel\_measurable\_def set\_integrable\_def}$  by blast
    have  $\text{LINT } x:A \cup B | M. f \ x = \text{LINT } x:(A - A \cap B) \cup (B - A \cap B) | M. f \ x$ 
    proof (rule  $\text{set\_integral\_cong\_set}$ )

```

```

  show AE x in M. (x ∈ A - A ∩ B ∪ (B - A ∩ B)) = (x ∈ A ∪ B)
    using ae by auto
  show set_borel_measurable M (A - A ∩ B ∪ (B - A ∩ B)) f
    using f' by (rule set_borel_measurable_subset) auto
qed fact
also have ... = (LINT x:(A - A ∩ B)|M. f x) + (LINT x:(B - A ∩ B)|M. f
x)
  by (auto intro!: set_integral_Un set_integrable_subset[OF f])
also have ... = (LINT x:A|M. f x) + (LINT x:B|M. f x)
  using ae
  by (intro arg_cong2[where f=(+)] set_integral_cong_set)
    (auto intro!: set_borel_measurable_subset[OF f])
finally show ?thesis .
qed

```

```

lemma set_integral_finite_Union:
  fixes f :: _ ⇒ _ :: {banach, second_countable_topology}
  assumes finite I disjoint_family_on A I
    and  $\bigwedge i. i \in I \implies \text{set\_integrable } M (A\ i) f$ 
    and  $\bigwedge i. i \in I \implies A\ i \in \text{sets } M$ 
  shows (LINT x:( $\bigcup i \in I. A\ i$ )|M. f x) = ( $\sum i \in I. \text{LINT } x:A\ i|M. f x$ )
    using assms
proof induction
  case (insert x F)
  then have  $A\ x \cap \bigcup (A\ ' F) = \{\}$ 
    by (meson disjoint_family_on_insert)
  with insert show ?case
    by (simp add: set_integral_Un set_integrable_Un set_integrable_UN disjoint_family_on_insert)
qed (simp add: set_lebesgue_integral_def)

```

```

lemma pos_integrable_to_top:
  fixes l::real
  assumes  $\bigwedge i. A\ i \in \text{sets } M$  mono A
  assumes nneg:  $\bigwedge x\ i. x \in A\ i \implies 0 \leq f\ x$ 
  and intgbl:  $\bigwedge i::\text{nat}. \text{set\_integrable } M (A\ i) f$ 
  and lim:  $(\lambda i::\text{nat}. \text{LINT } x:A\ i|M. f x) \longrightarrow l$ 
  shows set_integrable M ( $\bigcup i. A\ i$ ) f
    unfolding set_integrable_def
  apply (rule integrable_monotone_convergence[where f =  $\lambda i::\text{nat}. \lambda x. \text{indicator } (A\ i)\ x *_R f\ x$  and  $x = l$ ])
  apply (rule intgbl [unfolded set_integrable_def])
  prefer 3 apply (rule lim [unfolded set_lebesgue_integral_def])
  apply (rule AE_I2)
  using  $\langle \text{mono } A \rangle$  apply (auto simp: mono_def nneg split: split_indicator) []
proof (rule AE_I2)
  { fix x assume  $x \in \text{space } M$ 
    show  $(\lambda i. \text{indicator } (A\ i)\ x *_R f\ x) \longrightarrow \text{indicator } (\bigcup i. A\ i)\ x *_R f\ x$ 
    proof cases

```

```

    assume  $\exists i. x \in A\ i$ 
    then obtain  $i$  where  $x \in A\ i$  ..
    then have *: eventually ( $\lambda i. x \in A\ i$ ) sequentially
    using  $\langle x \in A\ i \rangle \langle \text{mono } A \rangle$  by (auto simp: eventually_sequentially_mono_def)
    show ?thesis
    apply (intro tendsto_eventually)
    using *
    apply eventually_elim
    apply (auto split: split_indicator)
    done
  qed auto }
then show ( $\lambda x. \text{indicator } (\bigcup i. A\ i)\ x *_R f\ x$ )  $\in \text{borel\_measurable } M$ 
  apply (rule borel_measurable_LIMSEQ_real)
  apply assumption
  using intgl_set_integrable_def by blast
qed

```

```

lemma lebesgue_integral_countable_add:
  fixes  $f :: \_ \Rightarrow 'a :: \{\text{banach, second\_countable\_topology}\}$ 
  assumes meas[intro]:  $\bigwedge i::\text{nat}. A\ i \in \text{sets } M$ 
    and disj:  $\bigwedge i\ j. i \neq j \implies A\ i \cap A\ j = \{\}$ 
    and intgl: set_integrable  $M\ (\bigcup i. A\ i)\ f$ 
  shows LINT  $x: (\bigcup i. A\ i) | M. f\ x = (\sum i. (\text{LINT } x: (A\ i) | M. f\ x))$ 
    unfolding set_lebesgue_integral_def
proof (subst integral_suminf_symmetric)
  show int_A: integrable  $M\ (\lambda x. \text{indicator } (A\ i)\ x *_R f\ x)$  for  $i$ 
    using intgl unfolding set_integrable_def [symmetric]
    by (rule set_integrable_subset) auto
  { fix  $x$  assume  $x \in \text{space } M$ 
    have ( $\lambda i. \text{indicator } (A\ i)\ x *_R f\ x$ ) sums (indicator  $(\bigcup i. A\ i)\ x *_R f\ x$ )
      by (intro sums_scaleR_left indicator_sums) fact }
  note sums = this

  have norm_f:  $\bigwedge i. \text{set\_integrable } M\ (A\ i)\ (\lambda x. \text{norm } (f\ x))$ 
    using int_A [THEN integrable_norm] unfolding set_integrable_def by auto

  show AE  $x$  in  $M. \text{summable } (\lambda i. \text{norm } (\text{indicator } (A\ i)\ x *_R f\ x))$ 
    using disj by (intro AE_I2) (auto intro!: summable_mult2_sums_summable[OF indicator_sums])

  show summable ( $\lambda i. \text{LINT } x | M. \text{norm } (\text{indicator } (A\ i)\ x *_R f\ x)$ )
proof (rule summableI_nonneg_bounded)
  fix  $n$ 
  show  $0 \leq \text{LINT } x | M. \text{norm } (\text{indicator } (A\ n)\ x *_R f\ x)$ 
    using norm_f by (auto intro!: integral_nonneg_AE)

  have  $(\sum i < n. \text{LINT } x | M. \text{norm } (\text{indicator } (A\ i)\ x *_R f\ x)) = (\sum i < n. \text{LINT } x: A\ i | M. \text{norm } (f\ x))$ 

```



```

    by (simp add: abs_mult set_lebesgue_integral_def)
  also have ... = set_lebesgue_integral M ( $\bigcup_{i < n}. A\ i$ ) ( $\lambda x. \text{norm } (f\ x)$ )
    using norm_f
    by (subst set_integral_finite_Union) (auto simp: disjoint_family_on_def
disj)
  also have ...  $\leq$  set_lebesgue_integral M ( $\bigcup_{i. A\ i}$ ) ( $\lambda x. \text{norm } (f\ x)$ )
    using intgbl[unfolded set_integrable_def, THEN integrable_norm] norm_f
    unfolding set_lebesgue_integral_def set_integrable_def
  apply (intro integral_mono set_integrable_UN[of {.. $n$ }, unfolded set_integrable_def])
    apply (auto split: split_indicator)
  done
  finally show ( $\sum_{i < n}. \text{LINT } x | M. \text{norm } (\text{indicator } (A\ i)\ x *_R f\ x)$ )  $\leq$ 
    set_lebesgue_integral M ( $\bigcup_{i. A\ i}$ ) ( $\lambda x. \text{norm } (f\ x)$ )
    by simp
qed
show LINT x | M. indicator ( $\bigcup (A\ ' \text{UNIV})$ )  $x *_R f\ x = \text{LINT } x | M. (\sum_{i. \text{indicator } (A\ i)\ x *_R f\ x})$ 
  apply (rule Bochner_Integration.integral_cong[OF refl])
  apply (subst suminf_scaleR_left[OF sums_summable[OF indicator_sums, OF
disj], symmetric])
    using sums_unique[OF indicator_sums[OF disj]]
  apply auto
  done
qed

lemma set_integral_cont_up:
  fixes f ::  $\_ \Rightarrow 'a :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
  assumes [measurable]:  $\bigwedge i. A\ i \in \text{sets } M$  and A: incseq A
  and intgbl: set_integrable M ( $\bigcup_{i. A\ i}$ ) f
  shows ( $\lambda i. \text{LINT } x : (A\ i) | M. f\ x$ )  $\longrightarrow \text{LINT } x : (\bigcup_{i. A\ i}) | M. f\ x$ 
    unfolding set_lebesgue_integral_def
  proof (intro integral_dominated_convergence[where w= $\lambda x. \text{indicator } (\bigcup_{i. A\ i})\ x *_R \text{norm } (f\ x)$ ])
    have int_A:  $\bigwedge i. \text{set\_integrable } M\ (A\ i)\ f$ 
      using intgbl by (rule set_integrable_subset) auto
    show  $\bigwedge i. (\lambda x. \text{indicator } (A\ i)\ x *_R f\ x) \in \text{borel\_measurable } M$ 
      using int_A integrable_iff_bounded set_integrable_def by blast
    show ( $\lambda x. \text{indicator } (\bigcup (A\ ' \text{UNIV}))\ x *_R f\ x$ )  $\in \text{borel\_measurable } M$ 
      using integrable_iff_bounded intgbl set_integrable_def by blast
    show integrable M ( $\lambda x. \text{indicator } (\bigcup_{i. A\ i})\ x *_R \text{norm } (f\ x)$ )
      using int_A intgbl integrable_norm unfolding set_integrable_def
      by fastforce
    { fix x i assume  $x \in A\ i$ 
      with A have ( $\lambda xa. \text{indicator } (A\ xa)\ x :: \text{real}$ )  $\longrightarrow 1 \longleftrightarrow (\lambda xa. 1 :: \text{real}) \longrightarrow 1$ 
        by (intro filterlim_cong refl)
        (fastforce simp: eventually_sequentially incseq_def subset_eq intro!: exI[of _ i]) }
    then show AE x in M. ( $\lambda i. \text{indicator } (A\ i)\ x *_R f\ x$ )  $\longrightarrow \text{indicator } (\bigcup_{i. A\ i})\ x *_R f\ x$ 

```

i) $x *_R f x$
 by (intro AE_I2 tendsto_intros) (auto split: split_indicator)
 qed (auto split: split_indicator)

lemma set_integral_cont_down:

fixes $f :: \_ \Rightarrow 'a :: \{\text{banach, second\_countable\_topology}\}$
 assumes [measurable]: $\bigwedge i. A i \in \text{sets } M$ and $A: \text{decseq } A$
 and $\text{int0}: \text{set\_integrable } M (A 0) f$
 shows $(\lambda i::\text{nat}. \text{LINT } x:(A i)|M. f x) \longrightarrow \text{LINT } x:(\bigcap i. A i)|M. f x$
 unfolding set_lebesgue_integral_def
 proof (rule integral_dominated_convergence)
 have $\text{int\_A}: \bigwedge i. \text{set\_integrable } M (A i) f$
 using int0 by (rule set_integrable_subset) (insert A, auto simp: decseq_def)
 have $\text{integrable } M (\lambda c. \text{norm } (\text{indicat\_real } (A 0) c *_R f c))$
 by (metis (no_types) int0 integrable_norm set_integrable_def)
 then show $\text{integrable } M (\lambda x. \text{indicator } (A 0) x *_R \text{norm } (f x))$
 by force
 have $\text{set\_integrable } M (\bigcap i. A i) f$
 using int0 by (rule set_integrable_subset) (insert A, auto simp: decseq_def)
 with int_A show $(\lambda x. \text{indicat\_real } (\bigcap (A ' UNIV)) x *_R f x) \in \text{borel\_measurable } M$
 $\bigwedge i. (\lambda x. \text{indicat\_real } (A i) x *_R f x) \in \text{borel\_measurable } M$
 by (auto simp: set_integrable_def)
 show $\bigwedge i. \text{AE } x \text{ in } M. \text{norm } (\text{indicator } (A i) x *_R f x) \leq \text{indicator } (A 0) x *_R \text{norm } (f x)$
 using A by (auto split: split_indicator simp: decseq_def)
 { fix $x i$ assume $x \in \text{space } M \ x \notin A i$
 with A have $(\lambda i. \text{indicator } (A i) x::\text{real}) \longrightarrow 0 \longleftrightarrow (\lambda i. 0::\text{real}) \longrightarrow 0$
 by (intro filterlim_cong refl)
 (auto split: split_indicator simp: eventually_sequentially decseq_def intro!: exI[of _ i]) }
 then show $\text{AE } x \text{ in } M. (\lambda i. \text{indicator } (A i) x *_R f x) \longrightarrow \text{indicator } (\bigcap i. A i) x *_R f x$
 by (intro AE_I2 tendsto_intros) (auto split: split_indicator)
 qed

lemma set_integral_at_point:

fixes $a :: \text{real}$
 assumes $\text{set\_integrable } M \{a\} f$
 and [simp]: $\{a\} \in \text{sets } M$ and $(\text{emeasure } M) \{a\} \neq \infty$
 shows $(\text{LINT } x:\{a\} | M. f x) = f a * \text{measure } M \{a\}$
 proof -
 have $\text{set\_lebesgue\_integral } M \{a\} f = \text{set\_lebesgue\_integral } M \{a\} (\%x. f a)$
 by (intro set_lebesgue_integral_cong) simp_all
 then show ?thesis using assms
 unfolding set_lebesgue_integral_def by simp
 qed

abbreviation $\text{complex\_integrable} :: 'a \text{ measure} \Rightarrow ('a \Rightarrow \text{complex}) \Rightarrow \text{bool}$ **where**
 $\text{complex\_integrable } M f \equiv \text{integrable } M f$

abbreviation $\text{complex\_lebesgue\_integral} :: 'a \text{ measure} \Rightarrow ('a \Rightarrow \text{complex}) \Rightarrow \text{complex}$ (integral^C) **where**
 $\text{integral}^C M f == \text{integral}^L M f$

syntax

$\text{complex\_lebesgue\_integral} :: \text{pttrn} \Rightarrow \text{complex} \Rightarrow 'a \text{ measure} \Rightarrow \text{complex}$
 $(\int^C \_ . \_ \partial \_ [60,61] 110)$

translations

$\int^C x. f \partial M == \text{CONST complex\_lebesgue\_integral } M (\lambda x. f)$

syntax

$\text{ascii\_complex\_lebesgue\_integral} :: \text{pttrn} \Rightarrow 'a \text{ measure} \Rightarrow \text{real} \Rightarrow \text{real}$
 $((\text{CLINT } \_ | \_ . \_) [0,110,60] 60)$

translations

$\text{CLINT } x | M. f == \text{CONST complex\_lebesgue\_integral } M (\lambda x. f)$

lemma $\text{complex\_integrable\_cnj}$ [simp]:

$\text{complex\_integrable } M (\lambda x. \text{cnj } (f x)) \longleftrightarrow \text{complex\_integrable } M f$

proof

assume $\text{complex\_integrable } M (\lambda x. \text{cnj } (f x))$
then have $\text{complex\_integrable } M (\lambda x. \text{cnj } (\text{cnj } (f x)))$
by ($\text{rule integrable\_cnj}$)
then show $\text{complex\_integrable } M f$
by simp

qed simp

lemma $\text{complex\_of\_real\_integrable\_eq}$:

$\text{complex\_integrable } M (\lambda x. \text{complex\_of\_real } (f x)) \longleftrightarrow \text{integrable } M f$

proof

assume $\text{complex\_integrable } M (\lambda x. \text{complex\_of\_real } (f x))$
then have $\text{integrable } M (\lambda x. \text{Re } (\text{complex\_of\_real } (f x)))$
by ($\text{rule integrable\_Re}$)
then show $\text{integrable } M f$
by simp

qed simp

abbreviation $\text{complex\_set\_integrable} :: 'a \text{ measure} \Rightarrow 'a \text{ set} \Rightarrow ('a \Rightarrow \text{complex}) \Rightarrow \text{bool}$ **where**

$\text{complex\_set\_integrable } M A f \equiv \text{set\_integrable } M A f$

abbreviation $\text{complex\_set\_lebesgue\_integral} :: 'a \text{ measure} \Rightarrow 'a \text{ set} \Rightarrow ('a \Rightarrow \text{complex}) \Rightarrow \text{complex}$ **where**

complex_set_lebesgue_integral $M A f \equiv \text{set\_lebesgue\_integral } M A f$

syntax

_ascii_complex_set_lebesgue_integral :: *pttrn* $\Rightarrow$ 'a set $\Rightarrow$ 'a measure $\Rightarrow$ real $\Rightarrow$ real
 ((*CLINT* _:_|_ . _) [0,60,110,61] 60)

translations

CLINT $x:A \mid M. f == \text{CONST } \text{complex\_set\_lebesgue\_integral } M A (\lambda x. f)$

lemma *set_measurable_continuous_on_ivl*:

assumes *continuous_on* {*a..b*} (*f* :: real $\Rightarrow$ real)

shows *set_borel_measurable* borel {*a..b*} *f*

unfolding *set_borel_measurable_def*

by (rule *borel_measurable_continuous_on_indicator*[*OF* _ *assms*]) *simp*

This notation is from Sébastien Gouëzel: His use is not directly in line with the notations in this file, they are more in line with sum, and more readable he thinks.

abbreviation *set_nn_integral* $M A f \equiv \text{nn\_integral } M (\lambda x. f x * \text{indicator } A x)$

syntax

_set_nn_integral :: *pttrn* $\Rightarrow$ 'a set $\Rightarrow$ 'a measure $\Rightarrow$ ereal $\Rightarrow$ ereal
 ((*f* $^+$ ((_) $\in$ (_). / _) / ∂ _) [0,60,110,61] 60)

_set_lebesgue_integral :: *pttrn* $\Rightarrow$ 'a set $\Rightarrow$ 'a measure $\Rightarrow$ ereal $\Rightarrow$ ereal
 ((*f* ((_) $\in$ (_). / _) / ∂ _) [0,60,110,61] 60)

translations

$\int^+ x \in A. f \partial M == \text{CONST } \text{set\_nn\_integral } M A (\lambda x. f)$

$\int x \in A. f \partial M == \text{CONST } \text{set\_lebesgue\_integral } M A (\lambda x. f)$

lemma *nn_integral_disjoint_pair*:

assumes [*measurable*]: *f* $\in$ *borel_measurable* *M*

B $\in$ *sets* *M* *C* $\in$ *sets* *M*

B $\cap$ *C* = {}

shows ($\int^+ x \in B \cup C. f x \partial M$) = ($\int^+ x \in B. f x \partial M$) + ($\int^+ x \in C. f x \partial M$)

proof –

have *mes*: $\bigwedge D. D \in \text{sets } M \implies (\lambda x. f x * \text{indicator } D x) \in \text{borel\_measurable } M$

by *simp*

have *pos*: $\bigwedge D. \bigwedge E x \text{ in } M. f x * \text{indicator } D x \geq 0$ **using** *assms*(2) **by** *auto*

have $\bigwedge x. f x * \text{indicator } (B \cup C) x = f x * \text{indicator } B x + f x * \text{indicator } C x$

using *assms*(4)

by (*auto split: split_indicator*)

then have ($\int^+ x. f x * \text{indicator } (B \cup C) x \partial M$) = ($\int^+ x. f x * \text{indicator } B x$ + $f x * \text{indicator } C x \partial M$)

by *simp*

also have ... = ($\int^+ x. f x * \text{indicator } B x \partial M$) + ($\int^+ x. f x * \text{indicator } C x \partial M$)

by (rule *nn_integral_add*) (*auto simp add: assms mes pos*)

finally show ?thesis by simp
qed

lemma nn_integral_disjoint_pair_countspace:
assumes $B \cap C = \{\}$
shows $(\int^+ x \in B \cup C. f x \partial \text{count\_space UNIV}) = (\int^+ x \in B. f x \partial \text{count\_space UNIV}) + (\int^+ x \in C. f x \partial \text{count\_space UNIV})$
by (rule nn_integral_disjoint_pair) (simp_all add: assms)

lemma nn_integral_null_delta:
assumes $A \in \text{sets } M$ $B \in \text{sets } M$
 $(A - B) \cup (B - A) \in \text{null\_sets } M$
shows $(\int^+ x \in A. f x \partial M) = (\int^+ x \in B. f x \partial M)$
proof (rule nn_integral_cong_AE)
have *: $A \in \text{sets } M. a \notin (A - B) \cup (B - A)$
using assms(3) AE_not_in by blast
then show $\langle A \in \text{sets } M. f x * \text{indicator } A x = f x * \text{indicator } B x \rangle$
by auto
qed

proposition nn_integral_disjoint_family:
assumes [measurable]: $f \in \text{borel\_measurable } M \wedge (n::\text{nat}). B n \in \text{sets } M$
and disjoint_family B
shows $(\int^+ x \in (\bigcup n. B n). f x \partial M) = (\sum n. (\int^+ x \in B n. f x \partial M))$
proof -
have $(\int^+ x. (\sum n. f x * \text{indicator } (B n) x) \partial M) = (\sum n. (\int^+ x. f x * \text{indicator } (B n) x \partial M))$
by (rule nn_integral_suminf) simp
moreover have $(\sum n. f x * \text{indicator } (B n) x) = f x * \text{indicator } (\bigcup n. B n) x$
for x
proof (cases)
assume $x \in (\bigcup n. B n)$
then obtain n where $x \in B n$ by blast
have a: finite {n} by simp
have $\bigwedge i. i \neq n \implies x \notin B i$ using $\langle x \in B n \rangle$ assms(3) disjoint_family_on_def
by (metis IntI UNIV_I empty_iff)
then have $\bigwedge i. i \notin \{n\} \implies \text{indicator } (B i) x = (0::\text{ennreal})$ using indicator_def
by simp
then have b: $\bigwedge i. i \notin \{n\} \implies f x * \text{indicator } (B i) x = (0::\text{ennreal})$ by simp

define h where $h = (\lambda i. f x * \text{indicator } (B i) x)$
then have $\bigwedge i. i \notin \{n\} \implies h i = 0$ using b by simp
then have $(\sum i. h i) = (\sum i \in \{n\}. h i)$
by (metis sums_unique[OF sums_finite[OF a]])
then have $(\sum i. h i) = h n$ by simp
then have $(\sum n. f x * \text{indicator } (B n) x) = f x * \text{indicator } (B n) x$ using
h_def by simp
then have $(\sum n. f x * \text{indicator } (B n) x) = f x$ using $\langle x \in B n \rangle$ indicator_def
by simp

```

    then show ?thesis using  $\langle x \in (\bigcup n. B\ n) \rangle$  by auto
  next
    assume  $x \notin (\bigcup n. B\ n)$ 
    then have  $\bigwedge n. f\ x * \text{indicator}\ (B\ n)\ x = 0$  by simp
    have  $(\sum n. f\ x * \text{indicator}\ (B\ n)\ x) = 0$ 
      by (simp add:  $\langle \bigwedge n. f\ x * \text{indicator}\ (B\ n)\ x = 0 \rangle$ )
    then show ?thesis using  $\langle x \notin (\bigcup n. B\ n) \rangle$  by auto
  qed
  ultimately show ?thesis by simp
qed

```

```

lemma nn_set_integral_add:
  assumes [measurable]:  $f \in \text{borel\_measurable}\ M\ g \in \text{borel\_measurable}\ M$ 
     $A \in \text{sets}\ M$ 
  shows  $(\int^+ x \in A. (f\ x + g\ x)\ \partial M) = (\int^+ x \in A. f\ x\ \partial M) + (\int^+ x \in A. g\ x\ \partial M)$ 
proof -
  have  $(\int^+ x \in A. (f\ x + g\ x)\ \partial M) = (\int^+ x. (f\ x * \text{indicator}\ A\ x + g\ x * \text{indicator}\ A\ x)\ \partial M)$ 
    by (auto simp add: indicator_def intro!: nn_integral_cong)
  also have  $\dots = (\int^+ x. f\ x * \text{indicator}\ A\ x\ \partial M) + (\int^+ x. g\ x * \text{indicator}\ A\ x\ \partial M)$ 
    apply (rule nn_integral_add) using assms(1) assms(2) by auto
  finally show ?thesis by simp
qed

```

```

lemma nn_set_integral_cong:
  assumes  $AE\ x\ \text{in}\ M. f\ x = g\ x$ 
  shows  $(\int^+ x \in A. f\ x\ \partial M) = (\int^+ x \in A. g\ x\ \partial M)$ 
apply (rule nn_integral_cong_AE) using assms(1) by auto

```

```

lemma nn_set_integral_set_mono:
   $A \subseteq B \implies (\int^+ x \in A. f\ x\ \partial M) \leq (\int^+ x \in B. f\ x\ \partial M)$ 
by (auto intro!: nn_integral_mono_split split_indicator)

```

```

lemma nn_set_integral_mono:
  assumes [measurable]:  $f \in \text{borel\_measurable}\ M\ g \in \text{borel\_measurable}\ M$ 
     $A \in \text{sets}\ M$ 
    and  $AE\ x \in A\ \text{in}\ M. f\ x \leq g\ x$ 
  shows  $(\int^+ x \in A. f\ x\ \partial M) \leq (\int^+ x \in A. g\ x\ \partial M)$ 
by (auto intro!: nn_integral_mono_AE split: split_indicator simp: assms)

```

```

lemma nn_set_integral_space [simp]:
  shows  $(\int^+ x \in \text{space}\ M. f\ x\ \partial M) = (\int^+ x. f\ x\ \partial M)$ 
by (metis (mono_tags, lifting) indicator_simps(1) mult.right_neutral nn_integral_cong)

```

```

lemma nn_integral_count_compose_inj:
  assumes inj_on  $g\ A$ 
  shows  $(\int^+ x \in A. f\ (g\ x)\ \partial \text{count\_space}\ UNIV) = (\int^+ y \in g\ A. f\ y\ \partial \text{count\_space}\ UNIV)$ 
proof -

```

```

have ( $\int^+ x \in A. f (g x) \partial \text{count\_space UNIV}$ ) = ( $\int^+ x. f (g x) \partial \text{count\_space A}$ )
  by (auto simp add: nn_integral_count_space_indicator[symmetric])
also have ... = ( $\int^+ y. f y \partial \text{count\_space } (g'A)$ )
  by (simp add: assms nn_integral_bij_count_space inj_on_imp_bij_betw)
also have ... = ( $\int^+ y \in g'A. f y \partial \text{count\_space UNIV}$ )
  by (auto simp add: nn_integral_count_space_indicator[symmetric])
finally show ?thesis by simp
qed

```

```

lemma nn_integral_count_compose_bij:
  assumes bij_betw g A B
  shows ( $\int^+ x \in A. f (g x) \partial \text{count\_space UNIV}$ ) = ( $\int^+ y \in B. f y \partial \text{count\_space UNIV}$ )
proof -
  have inj_on g A using assms bij_betw_def by auto
  then have ( $\int^+ x \in A. f (g x) \partial \text{count\_space UNIV}$ ) = ( $\int^+ y \in g'A. f y \partial \text{count\_space UNIV}$ )
    by (rule nn_integral_count_compose_inj)
  then show ?thesis using assms by (simp add: bij_betw_def)
qed

```

```

lemma set_integral_null_delta:
  fixes f:: $\_ \Rightarrow \_ :: \{\text{banach, second\_countable\_topology}\}$ 
  assumes [measurable]: integrable M f  $A \in \text{sets } M$   $B \in \text{sets } M$ 
  and null:  $(A - B) \cup (B - A) \in \text{null\_sets } M$ 
  shows ( $\int x \in A. f x \partial M$ ) = ( $\int x \in B. f x \partial M$ )
proof (rule set_integral_cong_set)
  have *:  $AE a \text{ in } M. a \notin (A - B) \cup (B - A)$ 
  using null AE_not_in by blast
  then show  $AE x \text{ in } M. (x \in B) = (x \in A)$ 
  by auto
qed (simp_all add: set_borel_measurable_def)

```

```

lemma set_integral_space:
  assumes integrable M f
  shows ( $\int x \in \text{space } M. f x \partial M$ ) = ( $\int x. f x \partial M$ )
  by (metis (no_types, lifting) indicator_simps(1) integral_cong scaleR_one set_lebesgue_integral_def)

```

```

lemma null_if_pos_func_has_zero_nn_int:
  fixes f:: $'a \Rightarrow \text{ennreal}$ 
  assumes [measurable]:  $f \in \text{borel\_measurable } M$   $A \in \text{sets } M$ 
  and  $AE x \in A \text{ in } M. f x > 0$  ( $\int^+ x \in A. f x \partial M$ ) = 0
  shows  $A \in \text{null\_sets } M$ 
proof -
  have  $AE x \text{ in } M. f x * \text{indicator } A x = 0$ 
  by (subst nn_integral_0_iff_AE[symmetric], auto simp add: assms(4))
  then have  $AE x \in A \text{ in } M. \text{False}$  using assms(3) by auto
  then show  $A \in \text{null\_sets } M$  using assms(2) by (simp add: AE_iff_null_sets)
qed

```

```

lemma null_if_pos_func_has_zero_int:
  assumes [measurable]: integrable  $M$   $f$   $A \in \text{sets } M$ 
    and  $\forall x \in A$  in  $M$ .  $f\ x > 0$   $(\int x \in A. f\ x\ \partial M) = (0::\text{real})$ 
  shows  $A \in \text{null\_sets } M$ 
proof –
  have  $\forall x$  in  $M$ . indicator  $A\ x * f\ x = 0$ 
    apply (subst integral_nonneg_eq_0_iff  $AE[symmetric]$ )
    using assms integrable_mult_indicator [ $OF\ \langle A \in \text{sets } M \rangle\ \text{assms}(1)$ ]
    by (auto simp: set_lebesgue_integral_def)
  then have  $\forall x \in A$  in  $M$ .  $f\ x = 0$  by auto
  then have  $\forall x \in A$  in  $M$ . False using assms(3) by auto
  then show  $A \in \text{null\_sets } M$  using assms(2) by (simp add: AE_iff_null_sets)
qed

```

The next lemma is a variant of *density_unique*. Note that it uses the notation for nonnegative set integrals introduced earlier.

```

lemma (in sigma_finite_measure) density_unique2:
  assumes [measurable]:  $f \in \text{borel\_measurable } M$   $f' \in \text{borel\_measurable } M$ 
  assumes density_eq:  $\bigwedge A. A \in \text{sets } M \implies (\int^+ x \in A. f\ x\ \partial M) = (\int^+ x \in A. f'\ x\ \partial M)$ 
  shows  $\forall x$  in  $M$ .  $f\ x = f'\ x$ 
proof (rule density_unique)
  show density  $M\ f = \text{density } M\ f'$ 
    by (intro measure_eqI) (auto simp: emeasure_density intro!: density_eq)
qed (auto simp add: assms)

```

The next lemma implies the same statement for Banach-space valued functions using Hahn-Banach theorem and linear forms. Since they are not yet easily available, I only formulate it for real-valued functions.

```

lemma density_unique_real:
  fixes  $f\ f': \_ \Rightarrow \text{real}$ 
  assumes  $M[\text{measurable}]$ : integrable  $M\ f$  integrable  $M\ f'$ 
  assumes density_eq:  $\bigwedge A. A \in \text{sets } M \implies (\int x \in A. f\ x\ \partial M) = (\int x \in A. f'\ x\ \partial M)$ 
  shows  $\forall x$  in  $M$ .  $f\ x = f'\ x$ 
proof –
  define  $A$  where  $A = \{x \in \text{space } M. f\ x < f'\ x\}$ 
  then have [measurable]:  $A \in \text{sets } M$  by simp
  have  $(\int x \in A. (f'\ x - f\ x)\ \partial M) = (\int x \in A. f'\ x\ \partial M) - (\int x \in A. f\ x\ \partial M)$ 
    using  $\langle A \in \text{sets } M \rangle\ M\ \text{integrable\_mult\_indicator\_set\_integrable\_def}$  by blast
  then have  $(\int x \in A. (f'\ x - f\ x)\ \partial M) = 0$  using assms(3) by simp
  then have  $A \in \text{null\_sets } M$ 
    using A_def null_if_pos_func_has_zero_int [where  $?f = \lambda x. f'\ x - f\ x$  and  $?A = A$ ] assms by auto
  then have  $\forall x$  in  $M$ .  $x \notin A$  by (simp add: AE_not_in)
  then have  $\forall x$  in  $M$ .  $f'\ x \leq f\ x$  unfolding A_def by auto

  define  $B$  where  $B = \{x \in \text{space } M. f'\ x < f\ x\}$ 

```



```

then have [measurable]: B ∈ sets M by simp
have (∫ x ∈ B. (f x - f' x) ∂M) = (∫ x ∈ B. f x ∂M) - (∫ x ∈ B. f' x ∂M)
  using ‹B ∈ sets M› M integrable_mult_indicator set_integrable_def by blast
then have (∫ x ∈ B. (f x - f' x) ∂M) = 0 using assms(3) by simp
then have B ∈ null_sets M
  using B_def null_if_pos_func_has_zero_int[where ?f = λx. f x - f' x and
?A = B] assms by auto
then have AE x in M. x ∉ B by (simp add: AE_not_in)
then have AE x in M. f' x ≥ f x unfolding B_def by auto
then show ?thesis using * by auto
qed

```

The next lemma shows that L^1 convergence of a sequence of functions follows from almost everywhere convergence and the weaker condition of the convergence of the integrated norms (or even just the nontrivial inequality about them). Useful in a lot of contexts! This statement (or its variations) are known as Scheffe lemma.

The formalization is more painful as one should jump back and forth between reals and ereals and justify all the time positivity or integrability (thankfully, measurability is handled more or less automatically).

proposition *Scheffe_lemma1:*

```

assumes ∧n. integrable M (F n) integrable M f
      AE x in M. (λn. F n x) ⟶ f x
      limsup (λn. ∫+ x. norm(F n x) ∂M) ≤ (∫+ x. norm(f x) ∂M)
shows (λn. ∫+ x. norm(F n x - f x) ∂M) ⟶ 0
proof -
  have [measurable]: ∧n. F n ∈ borel_measurable M f ∈ borel_measurable M
    using assms(1) assms(2) by simp_all
  define G where G = (λn x. norm(f x) + norm(F n x) - norm(F n x - f x))
  have [measurable]: ∧n. G n ∈ borel_measurable M unfolding G_def by simp
  have G_pos[simp]: ∧n x. G n x ≥ 0
    unfolding G_def by (metis ge_iff_diff_ge_0 norm_minus_commute norm_triangle_ineq4)
  have finint: (∫+ x. norm(f x) ∂M) ≠ ∞
    using has_bochner_integral_implies_finite_norm[OF has_bochner_integral_integrable[OF
‹integrable M f›]]
    by simp
  then have fin2: 2 * (∫+ x. norm(f x) ∂M) ≠ ∞
    by (auto simp: ennreal_mult_eq_top_iff)

  {
    fix x assume *: (λn. F n x) ⟶ f x
    then have (λn. norm(F n x)) ⟶ norm(f x) using tendsto_norm by blast
    moreover have (λn. norm(F n x - f x)) ⟶ 0 using * Lim_null tend-
sto_norm_zero_iff by fastforce
    ultimately have a: (λn. norm(F n x) - norm(F n x - f x)) ⟶ norm(f
x) using tendsto_diff by fastforce
    have (λn. norm(f x) + (norm(F n x) - norm(F n x - f x))) ⟶ norm(f
x) + norm(f x)

```

```

    by (rule tendsto_add) (auto simp add: a)
    moreover have  $\bigwedge n. G\ n\ x = \text{norm}(f\ x) + (\text{norm}(F\ n\ x) - \text{norm}(F\ n\ x - f\ x))$ 
  unfolding G_def by simp
  ultimately have  $(\lambda n. G\ n\ x) \longrightarrow 2 * \text{norm}(f\ x)$  by simp
  then have  $(\lambda n. \text{ennreal}(G\ n\ x)) \longrightarrow \text{ennreal}(2 * \text{norm}(f\ x))$  by simp
  then have  $\liminf (\lambda n. \text{ennreal}(G\ n\ x)) = \text{ennreal}(2 * \text{norm}(f\ x))$ 
    using sequentially_bot tendsto_iff_Liminf_eq_Limsup by blast
}
then have  $AE\ x\ in\ M. \liminf (\lambda n. \text{ennreal}(G\ n\ x)) = \text{ennreal}(2 * \text{norm}(f\ x))$ 
using assms(3) by auto
then have  $(\int^+ x. \liminf (\lambda n. \text{ennreal}(G\ n\ x))\ \partial M) = (\int^+ x. 2 * \text{ennreal}(\text{norm}(f\ x))\ \partial M)$ 
  by (simp add: nn_integral_cong_AE ennreal_mult)
also have  $\dots = 2 * (\int^+ x. \text{norm}(f\ x)\ \partial M)$  by (rule nn_integral_cmult) auto
finally have  $\text{int\_liminf}: (\int^+ x. \liminf (\lambda n. \text{ennreal}(G\ n\ x))\ \partial M) = 2 * (\int^+ x. \text{norm}(f\ x)\ \partial M)$ 
  by simp

have  $(\int^+ x. \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))\ \partial M) = (\int^+ x. \text{norm}(f\ x)\ \partial M) + (\int^+ x. \text{norm}(F\ n\ x)\ \partial M)$  for n
  by (rule nn_integral_add) (auto simp add: assms)
then have  $\limsup (\lambda n. (\int^+ x. \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))\ \partial M)) =$ 
 $\limsup (\lambda n. (\int^+ x. \text{norm}(f\ x)\ \partial M) + (\int^+ x. \text{norm}(F\ n\ x)\ \partial M))$ 
  by simp
also have  $\dots = (\int^+ x. \text{norm}(f\ x)\ \partial M) + \limsup (\lambda n. (\int^+ x. \text{norm}(F\ n\ x)\ \partial M))$ 
  by (rule Limsup_const_add, auto simp add: finint)
also have  $\dots \leq (\int^+ x. \text{norm}(f\ x)\ \partial M) + (\int^+ x. \text{norm}(f\ x)\ \partial M)$ 
  using assms(4) by (simp add: add_left_mono)
also have  $\dots = 2 * (\int^+ x. \text{norm}(f\ x)\ \partial M)$ 
  unfolding one_add_one[symmetric] distrib_right by simp
ultimately have  $a: \limsup (\lambda n. (\int^+ x. \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))\ \partial M)) \leq$ 
 $2 * (\int^+ x. \text{norm}(f\ x)\ \partial M)$  by simp

have  $le: \text{ennreal}(\text{norm}(F\ n\ x - f\ x)) \leq \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))$  for n x
  by (simp add: norm_minus_commute norm_triangle_ineq4 ennreal_minus flip: ennreal_plus)
then have  $le2: (\int^+ x. \text{ennreal}(\text{norm}(F\ n\ x - f\ x))\ \partial M) \leq (\int^+ x. \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))\ \partial M)$  for n
  by (rule nn_integral_mono)

have  $2 * (\int^+ x. \text{norm}(f\ x)\ \partial M) = (\int^+ x. \liminf (\lambda n. \text{ennreal}(G\ n\ x))\ \partial M)$ 
  by (simp add: int_liminf)
also have  $\dots \leq \liminf (\lambda n. (\int^+ x. G\ n\ x\ \partial M))$ 
  by (rule nn_integral_liminf) auto
also have  $\liminf (\lambda n. (\int^+ x. G\ n\ x\ \partial M)) =$ 
 $\liminf (\lambda n. (\int^+ x. \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))\ \partial M) - (\int^+ x.$ 

```

```

norm(F n x - f x) ∂M))
  proof (intro arg_cong[where f=liminf] ext)
    fix n
    have  $\bigwedge x. \text{ennreal}(G\ n\ x) = \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x)) - \text{ennreal}(\text{norm}(F\ n\ x - f\ x))$ 
    unfolding G_def by (simp add: ennreal_minus flip: ennreal_plus)
    moreover have  $(\int^+ x. \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x)) - \text{ennreal}(\text{norm}(F\ n\ x - f\ x))\ \partial M)$ 
      =  $(\int^+ x. \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))\ \partial M) - (\int^+ x. \text{norm}(F\ n\ x - f\ x)\ \partial M)$ 
    proof (rule nn_integral_diff)
      from le show  $\text{AE } x \text{ in } M. \text{ennreal}(\text{norm}(F\ n\ x - f\ x)) \leq \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))$ 
      by simp
      from le2 have  $(\int^+ x. \text{ennreal}(\text{norm}(F\ n\ x - f\ x))\ \partial M) < \infty$  using assms(1)
      by (metis has_bochner_integral_implies_finite_norm integrable.simps Bochner_Integration.integrable_diff)
      then show  $(\int^+ x. \text{ennreal}(\text{norm}(F\ n\ x - f\ x))\ \partial M) \neq \infty$  by simp
    qed (auto simp add: assms)
    ultimately show  $(\int^+ x. G\ n\ x\ \partial M) = (\int^+ x. \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))\ \partial M) - (\int^+ x. \text{norm}(F\ n\ x - f\ x)\ \partial M)$ 
    by simp
  qed
  finally have  $2 * (\int^+ x. \text{norm}(f\ x)\ \partial M) + \limsup (\lambda n. (\int^+ x. \text{norm}(F\ n\ x - f\ x)\ \partial M)) \leq$ 
     $\liminf (\lambda n. (\int^+ x. \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))\ \partial M) - (\int^+ x. \text{norm}(F\ n\ x - f\ x)\ \partial M)) +$ 
     $\limsup (\lambda n. (\int^+ x. \text{norm}(F\ n\ x - f\ x)\ \partial M))$ 
    by (intro add_mono) auto
  also have  $\dots \leq (\limsup (\lambda n. \int^+ x. \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))\ \partial M) - \limsup (\lambda n. \int^+ x. \text{norm}(F\ n\ x - f\ x)\ \partial M)) +$ 
     $\limsup (\lambda n. (\int^+ x. \text{norm}(F\ n\ x - f\ x)\ \partial M))$ 
    by (intro add_mono liminf_minus_ennreal le2) auto
  also have  $\dots = \limsup (\lambda n. (\int^+ x. \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))\ \partial M))$ 
    by (intro diff_add_cancel_ennreal Limsup_mono always_eventually_allI le2)
  also have  $\dots \leq 2 * (\int^+ x. \text{norm}(f\ x)\ \partial M)$ 
    by fact
  finally have  $\limsup (\lambda n. (\int^+ x. \text{norm}(F\ n\ x - f\ x)\ \partial M)) = 0$ 
    using fin2 by simp
  then show ?thesis
    by (rule tendsto_0_if_Limsup_eq_0_ennreal)
  qed

```

proposition Scheffe_lemma2:

```

fixes F::nat  $\Rightarrow$  'a  $\Rightarrow$  'b::{banach, second_countable_topology}
assumes  $\bigwedge n::nat. F\ n \in \text{borel\_measurable } M \text{ integrable } M\ f$ 
       $\text{AE } x \text{ in } M. (\lambda n. F\ n\ x) \longrightarrow f\ x$ 

```

```

       $\bigwedge n. (\int^+ x. \text{norm}(F\ n\ x)\ \partial M) \leq (\int^+ x. \text{norm}(f\ x)\ \partial M)$ 
    shows  $(\lambda n. \int^+ x. \text{norm}(F\ n\ x - f\ x)\ \partial M) \longrightarrow 0$ 
  proof (rule Scheffe_lemma1)
    fix n::nat
    have  $(\int^+ x. \text{norm}(f\ x)\ \partial M) < \infty$  using assms(2) by (metis has_bochner_integral_implies_finite_norm
integrable.cases)
    then have  $(\int^+ x. \text{norm}(F\ n\ x)\ \partial M) < \infty$  using assms(4)[of n] by auto
    then show integrable M (F n) by (subst integrable_iff_bounded, simp add:
assms(1)[of n])
  qed (auto simp add: assms Limsup_bounded)

lemma tendsto_set_lebesgue_integral_at_right:
  fixes a b :: real and f :: real  $\Rightarrow$  'a :: {banach,second_countable_topology}
  assumes a < b and sets:  $\bigwedge a'. a' \in \{a<..b\} \implies \{a'..b\} \in \text{sets } M$ 
    and set_integrable M {a<..b} f
  shows  $((\lambda a'. \text{set\_lebesgue\_integral } M\ \{a'..b\}\ f) \longrightarrow$ 
     $\text{set\_lebesgue\_integral } M\ \{a<..b\}\ f) \text{ (at\_right } a)$ 
  proof (rule tendsto_at_right_sequentially[OF assms(1)], goal_cases)
    case (1 S)
    have eq:  $(\bigcup n. \{S\ n..b\}) = \{a<..b\}$ 
    proof safe
      fix x n assume x  $\in \{S\ n..b\}$ 
      with 1(1,2)[of n] show x  $\in \{a<..b\}$  by auto
    next
      fix x assume x  $\in \{a<..b\}$ 
      with order_tendstoD[OF  $\langle S \longrightarrow a \rangle$ , of x] show x  $\in (\bigcup n. \{S\ n..b\})$ 
      by (force simp: eventually_at_top_linorder dest: less_imp_le)
    qed
    have  $(\lambda n. \text{set\_lebesgue\_integral } M\ \{S\ n..b\}\ f) \longrightarrow \text{set\_lebesgue\_integral } M$ 
 $(\bigcup n. \{S\ n..b\})\ f$ 
    by (rule set_integral_cont_up) (insert assms 1, auto simp: eq incseq_def dec-
seq_def less_imp_le)
    with eq show ?case by simp
  qed

```

The next lemmas relate convergence of integrals over an interval to improper integrals.

```

lemma tendsto_set_lebesgue_integral_at_left:
  fixes a b :: real and f :: real  $\Rightarrow$  'a :: {banach,second_countable_topology}
  assumes a < b and sets:  $\bigwedge b'. b' \in \{a..<b\} \implies \{a..b'\} \in \text{sets } M$ 
    and set_integrable M {a..<b} f
  shows  $((\lambda b'. \text{set\_lebesgue\_integral } M\ \{a..b'\}\ f) \longrightarrow$ 
     $\text{set\_lebesgue\_integral } M\ \{a..<b\}\ f) \text{ (at\_left } b)$ 
  proof (rule tendsto_at_left_sequentially[OF assms(1)], goal_cases)
    case (1 S)
    have eq:  $(\bigcup n. \{a..S\ n\}) = \{a..<b\}$ 
    proof safe
      fix x n assume x  $\in \{a..S\ n\}$ 
      with 1(1,2)[of n] show x  $\in \{a..<b\}$  by auto
    next
      fix x assume x  $\in \{a..<b\}$ 
      with order_tendstoD[OF  $\langle S \longrightarrow b \rangle$ , of x] show x  $\in (\bigcup n. \{a..S\ n\})$ 
      by (force simp: eventually_at_top_linorder dest: less_imp_le)
    qed
    have  $(\lambda n. \text{set\_lebesgue\_integral } M\ \{a..S\ n\}\ f) \longrightarrow \text{set\_lebesgue\_integral } M$ 
 $(\bigcup n. \{a..S\ n\})\ f$ 
    by (rule set_integral_cont_down) (insert assms 1, auto simp: eq incseq_def dec-
seq_def less_imp_le)
    with eq show ?case by simp
  qed

```

```

next
  fix x assume x ∈ {a..b}
  with order_tendstoD[OF ‹S ⟶ b›, of x] show x ∈ (⋃ n. {a..S n})
    by (force simp: eventually_at_top_linorder dest: less_imp_le)
qed
have (λn. set_lebesgue_integral M {a..S n} f) ⟶ set_lebesgue_integral M
(⋃ n. {a..S n) f
  by (rule set_integral_cont_up) (insert assms 1, auto simp: eq_incseq_def dec-
seq_def less_imp_le)
  with eq show ?case by simp
qed

proposition tendsto_set_lebesgue_integral_at_top:
  fixes f :: real ⇒ 'a::{banach, second_countable_topology}
  assumes sets: ⋀b. b ≥ a ⟹ {a..b} ∈ sets M
    and int: set_integrable M {a.. } f
  shows ((λb. set_lebesgue_integral M {a..b} f) ⟶ set_lebesgue_integral M
{a.. } f) at_top
proof (rule tendsto_at_topI_sequentially)
  fix X :: nat ⇒ real assume filterlim X at_top sequentially
  show (λn. set_lebesgue_integral M {a..X n} f) ⟶ set_lebesgue_integral M
{a.. } f
    unfolding set_lebesgue_integral_def
  proof (rule integral_dominated_convergence)
    show integrable M (λx. indicat_real {a.. } x *R norm (f x))
      using integrable_norm[OF int[unfolded set_integrable_def]] by simp
    show AE x in M. (λn. indicator {a..X n} x *R f x) ⟶ indicat_real {a.. }
x *R f x
    proof
      fix x
      from ‹filterlim X at_top sequentially›
      have eventually (λn. x ≤ X n) sequentially
        unfolding filterlim_at_top_ge[where c=x] by auto
      then show (λn. indicator {a..X n} x *R f x) ⟶ indicat_real {a.. } x *R
f x
        by (intro tendsto_eventually) (auto split: split_indicator elim!: eventu-
ally_mono)
    qed
    fix n show AE x in M. norm (indicator {a..X n} x *R f x) ≤
      indicator {a.. } x *R norm (f x)
      by (auto split: split_indicator)
  next
    from int show (λx. indicat_real {a.. } x *R f x) ∈ borel_measurable M
      by (simp add: set_integrable_def)
  next
    fix n :: nat
    from sets have {a..X n} ∈ sets M by (cases X n ≥ a) auto
    with int have set_integrable M {a..X n} f
      by (rule set_integrable_subset) auto

```

```

      thus ( $\lambda x. \text{indicat\_real } \{a..X\ n\} \ x *_R f\ x) \in \text{borel\_measurable } M$ 
      by (simp add: set_integrable_def)
    qed
  qed

proposition tendsto_set_lebesgue_integral_at_bot:
  fixes  $f :: \text{real} \Rightarrow 'a::\{\text{banach, second\_countable\_topology}\}$ 
  assumes sets:  $\bigwedge a. a \leq b \implies \{a..b\} \in \text{sets } M$ 
  and int:  $\text{set\_integrable } M \ \{..b\} \ f$ 
  shows  $((\lambda a. \text{set\_lebesgue\_integral } M \ \{a..b\} \ f) \longrightarrow \text{set\_lebesgue\_integral } M \ \{..b\} \ f) \text{ at\_bot}$ 
proof (rule tendsto_at_botI_sequentially)
  fix  $X :: \text{nat} \Rightarrow \text{real}$  assume filterlim  $X \text{ at\_bot sequentially}$ 
  show  $(\lambda n. \text{set\_lebesgue\_integral } M \ \{X\ n..b\} \ f) \longrightarrow \text{set\_lebesgue\_integral } M \ \{..b\} \ f$ 
    unfolding set_lebesgue_integral_def
  proof (rule integral_dominated_convergence)
    show integrable  $M \ (\lambda x. \text{indicat\_real } \{..b\} \ x *_R \text{norm } (f\ x))$ 
      using integrable_norm[OF int[unfolded set_integrable_def]] by simp
    show  $AE\ x \text{ in } M. (\lambda n. \text{indicator } \{X\ n..b\} \ x *_R f\ x) \longrightarrow \text{indicat\_real } \{..b\} \ x *_R f\ x$ 
      proof
        fix  $x$ 
        from  $\langle \text{filterlim } X \text{ at\_bot sequentially} \rangle$ 
        have eventually  $(\lambda n. x \geq X\ n)$  sequentially
          unfolding filterlim_at_bot_le[where  $c=x$ ] by auto
        then show  $(\lambda n. \text{indicator } \{X\ n..b\} \ x *_R f\ x) \longrightarrow \text{indicat\_real } \{..b\} \ x *_R f\ x$ 
          by (intro tendsto_eventually) (auto split: split_indicator elim!: eventually_mono)
      qed
    fix  $n$  show  $AE\ x \text{ in } M. \text{norm } (\text{indicator } \{X\ n..b\} \ x *_R f\ x) \leq \text{indicator } \{..b\} \ x *_R \text{norm } (f\ x)$ 
      by (auto split: split_indicator)
  next
    from int show  $(\lambda x. \text{indicat\_real } \{..b\} \ x *_R f\ x) \in \text{borel\_measurable } M$ 
      by (simp add: set_integrable_def)
  next
    fix  $n :: \text{nat}$ 
    from sets have  $\{X\ n..b\} \in \text{sets } M$  by (cases  $X\ n \leq b$ ) auto
    with int have  $\text{set\_integrable } M \ \{X\ n..b\} \ f$ 
      by (rule set_integrable_subset) auto
    thus  $(\lambda x. \text{indicat\_real } \{X\ n..b\} \ x *_R f\ x) \in \text{borel\_measurable } M$ 
      by (simp add: set_integrable_def)
  qed
qed

```

theorem *integral_Markov_inequality'*:

```

  fixes u :: 'a  $\Rightarrow$  real
  assumes [measurable]: set_integrable M A u and A  $\in$  sets M
  assumes AE x in M.  $x \in A \longrightarrow u\ x \geq 0$  and  $0 < (c::real)$ 
  shows emeasure M  $\{x \in A. u\ x \geq c\} \leq (1/c::real) * (\int x \in A. u\ x\ \partial M)$ 
proof -
  have  $(\lambda x. u\ x * \text{indicator } A\ x) \in \text{borel\_measurable } M$ 
    using assms by (auto simp: set_integrable_def mult_ac)
  hence  $(\lambda x. \text{ennreal } (u\ x * \text{indicator } A\ x)) \in \text{borel\_measurable } M$ 
    by measurable
  also have  $(\lambda x. \text{ennreal } (u\ x * \text{indicator } A\ x)) = (\lambda x. \text{ennreal } (u\ x) * \text{indicator } A\ x)$ 
    by (intro ext) (auto simp: indicator_def)
  finally have meas:  $\dots \in \text{borel\_measurable } M$ .
  from assms(3) have AE: AE x in M.  $0 \leq u\ x * \text{indicator } A\ x$ 
    by eventually_elim (auto simp: indicator_def)
  have nonneg: set_lebesgue_integral M A u  $\geq 0$ 
    unfolding set_lebesgue_integral_def
    by (intro Bochner_Integration.integral_nonneg_AE eventually_mono[OF AE])
    (auto simp: mult_ac)

  have A:  $A \subseteq \text{space } M$ 
    using  $\langle A \in \text{sets } M \rangle$  by (simp add: sets.sets_into_space)
  have  $\{x \in A. u\ x \geq c\} = \{x \in A. \text{ennreal}(1/c) * u\ x \geq 1\}$ 
    using  $\langle c > 0 \rangle$  A by (auto simp: ennreal_mult'[symmetric])
  then have emeasure M  $\{x \in A. u\ x \geq c\} = \text{emeasure } M (\{x \in A. \text{ennreal}(1/c)$ 
     $* u\ x \geq 1\})$ 
    by simp
  also have  $\dots \leq \text{ennreal}(1/c) * (\int^+ x. \text{ennreal}(u\ x) * \text{indicator } A\ x\ \partial M)$ 
    by (intro nn_integral_Markov_inequality meas assms)
  also have  $(\int^+ x. \text{ennreal}(u\ x) * \text{indicator } A\ x\ \partial M) = \text{ennreal } (\text{set\_lebesgue\_integral } M\ A\ u)$ 
    unfolding set_lebesgue_integral_def nn_integral_set_ennreal using assms
  AE
    by (subst nn_integral_eq_integral) (simp_all add: mult_ac set_integrable_def)
  finally show ?thesis
    using  $\langle c > 0 \rangle$  nonneg by (subst ennreal_mult) auto
qed

```

theorem *integral_Markov_inequality'_measure*:

```

  assumes [measurable]: set_integrable M A u and A  $\in$  sets M
  and AE x in M.  $x \in A \longrightarrow 0 \leq u\ x$  and  $0 < (c::real)$ 
  shows measure M  $\{x \in A. u\ x \geq c\} \leq (\int x \in A. u\ x\ \partial M) / c$ 
proof -
  have nonneg: set_lebesgue_integral M A u  $\geq 0$ 
    unfolding set_lebesgue_integral_def
    by (intro Bochner_Integration.integral_nonneg_AE eventually_mono[OF assms(3)])
    (auto simp: mult_ac)
  have le: emeasure M  $\{x \in A. u\ x \geq c\} \leq \text{ennreal } ((1/c) * (\int x \in A. u\ x\ \partial M))$ 

```

```

    by (rule integral_Markov_inequality') (use assms in auto)
  also have ... < top
    by auto
  finally have ennreal (measure M {x∈A. u x ≥ c}) = emeasure M {x∈A. u x ≥
c}
    by (intro emeasure_eq_ennreal_measure [symmetric]) auto
  also note le
  finally show ?thesis using nonneg
    by (subst (asm) ennreal_le_iff)
      (auto intro!: divide_nonneg_pos Bochner_Integration.integral_nonneg_AE
assms)
qed

```

```

theorem (in finite_measure) Chernoff_ineq_ge:
  assumes s: s > 0
  assumes integrable: set_integrable M A (λx. exp (s * f x)) and A ∈ sets M
  shows measure M {x∈A. f x ≥ a} ≤ exp (-s * a) * (∫ x∈A. exp (s * f x) ∂M)
proof -
  have {x∈A. f x ≥ a} = {x∈A. exp (s * f x) ≥ exp (s * a)}
    using s by auto
  also have measure M ... ≤ set_lebesgue_integral M A (λx. exp (s * f x)) / exp
(s * a)
    by (intro integral_Markov_inequality'_measure assms) auto
  finally show ?thesis
    by (simp add: exp_minus_field_simps)
qed

```

```

theorem (in finite_measure) Chernoff_ineq_le:
  assumes s: s > 0
  assumes integrable: set_integrable M A (λx. exp (-s * f x)) and A ∈ sets M
  shows measure M {x∈A. f x ≤ a} ≤ exp (s * a) * (∫ x∈A. exp (-s * f x) ∂M)
proof -
  have {x∈A. f x ≤ a} = {x∈A. exp (-s * f x) ≥ exp (-s * a)}
    using s by auto
  also have measure M ... ≤ set_lebesgue_integral M A (λx. exp (-s * f x)) /
exp (-s * a)
    by (intro integral_Markov_inequality'_measure assms) auto
  finally show ?thesis
    by (simp add: exp_minus_field_simps)
qed

```

end

6.17 Non-Denumerability of the Continuum

```

theory Continuum_Not_Denumerable
imports
  Complex_Main
  HOL-Library.Countable_Set

```


begin

6.17.1 Abstract

The following document presents a proof that the Continuum is uncountable. It is formalised in the Isabelle/Isar theorem proving system.

Theorem: The Continuum $\mathbb{R}$ is not denumerable. In other words, there does not exist a function $f: \mathbb{N} \Rightarrow \mathbb{R}$ such that f is surjective.

Outline: An elegant informal proof of this result uses Cantor's Diagonalisation argument. The proof presented here is not this one.

First we formalise some properties of closed intervals, then we prove the Nested Interval Property. This property relies on the completeness of the Real numbers and is the foundation for our argument. Informally it states that an intersection of countable closed intervals (where each successive interval is a subset of the last) is non-empty. We then assume a surjective function $f: \mathbb{N} \Rightarrow \mathbb{R}$ exists and find a real x such that x is not in the range of f by generating a sequence of closed intervals then using the Nested Interval Property.

theorem *real_non_denum*: $\nexists f :: \text{nat} \Rightarrow \text{real}. \text{surj } f$

proof

assume $\exists f :: \text{nat} \Rightarrow \text{real}. \text{surj } f$

then obtain $f :: \text{nat} \Rightarrow \text{real}$ **where** $\text{surj } f$..

First we construct a sequence of nested intervals, ignoring *range f*.

have $a < b \implies \exists ka kb. ka < kb \wedge \{ka..kb\} \subseteq \{a..b\} \wedge c \notin \{ka..kb\}$ **for** $a b c :: \text{real}$

by (*auto simp add: not_le cong: conj_cong*)

(*metis dense le_less_linear less_linear less_trans order_refl*)

then obtain $i j$ **where** ij :

$a < b \implies i a b c < j a b c$

$a < b \implies \{i a b c .. j a b c\} \subseteq \{a .. b\}$

$a < b \implies c \notin \{i a b c .. j a b c\}$

for $a b c :: \text{real}$

by *metis*

define ivl **where** $ivl =$

$\text{rec\_nat } (f\ 0 + 1, f\ 0 + 2) (\lambda n x. (i\ (fst\ x)\ (snd\ x)\ (f\ n), j\ (fst\ x)\ (snd\ x)\ (f\ n)))$

define I **where** $I\ n = \{fst\ (ivl\ n) .. snd\ (ivl\ n)\}$ **for** n

have ivl [*simp*]:

$ivl\ 0 = (f\ 0 + 1, f\ 0 + 2)$

$\bigwedge n. ivl\ (Suc\ n) = (i\ (fst\ (ivl\ n))\ (snd\ (ivl\ n))\ (f\ n), j\ (fst\ (ivl\ n))\ (snd\ (ivl\ n))\ (f\ n))$

unfolding ivl_def **by** *simp_all*

This is a decreasing sequence of non-empty intervals.

```

have less: fst (ivl n) < snd (ivl n) for n
  by (induct n) (auto intro!: ij)

have decseq I
  unfolding I_def decseq_Suc_iff ivl fst_conv snd_conv
  by (intro ij allI less)

```

Now we apply the finite intersection property of compact sets.

```

have I 0  $\cap$  ( $\bigcap i. I i$ )  $\neq \{\}$ 
proof (rule compact_imp_fip_image)
  fix S :: nat set
  assume fin: finite S
  have  $\{\}$   $\subset I$  (Max (insert 0 S))
    unfolding I_def using less[of Max (insert 0 S)] by auto
  also have  $I$  (Max (insert 0 S))  $\subseteq$  ( $\bigcap i \in \text{insert } 0 S. I i$ )
    using fin decseqD[OF decseq I, of _ Max (insert 0 S)]
    by (auto simp: Max_ge_iff)
  also have ( $\bigcap i \in \text{insert } 0 S. I i$ ) =  $I 0 \cap$  ( $\bigcap i \in S. I i$ )
    by auto
  finally show  $I 0 \cap$  ( $\bigcap i \in S. I i$ )  $\neq \{\}$ 
    by auto
qed (auto simp: I_def)
then obtain x where  $x \in I n$  for n
  by blast
moreover from surj f obtain j where  $x = f j$ 
  by blast
ultimately have  $f j \in I$  (Suc j)
  by blast
with ij(3)[OF less] show False
  unfolding I_def ivl fst_conv snd_conv by auto
qed

```

```

lemma uncountable_UNIV_real: uncountable (UNIV :: real set)
  using real_non_denum unfolding uncountable_def by auto

```

```

lemma bij_betw_open_intervals:
  fixes a b c d :: real
  assumes  $a < b$   $c < d$ 
  shows  $\exists f. \text{bij\_betw } f \{a <..<b\} \{c <..<d\}$ 
proof -
  define f where  $f a b c d x = (d - c) / (b - a) * (x - a) + c$  for a b c d x :: real
  {
    fix a b c d x :: real
    assume *:  $a < b$   $c < d$   $a < x$   $x < b$ 
    moreover from * have  $(d - c) * (x - a) < (d - c) * (b - a)$ 
      by (intro mult_strict_left_mono) simp_all
    moreover from * have  $0 < (d - c) * (x - a) / (b - a)$ 
      by simp
    ultimately have  $f a b c d x < d$   $c < f a b c d x$ 

```

```

    by (simp_all add: f_def field_simps)
  }
  with assms have bij_betw (f a b c d) {a<..b} {c<..d}
    by (intro bij_betw_byWitness[where f'=f c d a b]) (auto simp: f_def)
  then show ?thesis by auto
qed

```

```

lemma bij_betw_tan: bij_betw tan { $-\pi/2 < \cdot < \pi/2$ } UNIV
  using arctan_ubound by (intro bij_betw_byWitness[where f'=arctan]) (auto
simp: arctan arctan_tan)

```

```

lemma uncountable_open_interval: uncountable {a<..b}  $\longleftrightarrow$  a < b for a b ::
real
proof
  show a < b if uncountable {a<..b}
    using uncountable_def that by force
  show uncountable {a<..b} if a < b
  proof -
    obtain f where bij_betw f {a <..b} { $-\pi/2 < \cdot < \pi/2$ }
      using bij_betw_open_intervals[OF a < b, of  $-\pi/2$   $\pi/2$ ] by auto
    then show ?thesis
      by (metis bij_betw_tan uncountable_bij_betw uncountable_UNIV_real)
  qed
qed

```

```

lemma uncountable_half_open_interval_1: uncountable {a..b}  $\longleftrightarrow$  a < b for
a b :: real
  apply auto
  using atLeastLessThan_empty_iff
  apply fastforce
  using uncountable_open_interval [of a b]
  apply (metis countable_Un_iff ivl_disj_un_singleton(3))
  done

```

```

lemma uncountable_half_open_interval_2: uncountable {a<..b}  $\longleftrightarrow$  a < b for
a b :: real
  apply auto
  using atLeastLessThan_empty_iff
  apply fastforce
  using uncountable_open_interval [of a b]
  apply (metis countable_Un_iff ivl_disj_un_singleton(4))
  done

```

```

lemma real_interval_avoid_countable_set:
  fixes a b :: real and A :: real set
  assumes a < b and countable A
  shows  $\exists x \in \{a < \cdot < b\}. x \notin A$ 
proof -
  from countable A have *: countable (A  $\cap$  {a<..b})

```

2000

```

    by auto
  with ⟨a < b⟩ have ¬ countable {a<..b}
    by (simp add: uncountable_open_interval)
  with * have  $A \cap \{a<..b\} \neq \{a<..b\}$ 
    by auto
  then have  $A \cap \{a<..b\} \subset \{a<..b\}$ 
    by (intro psubsetI) auto
  then have  $\exists x. x \in \{a<..b\} - A \cap \{a<..b\}$ 
    by (rule psubset_imp_ex_mem)
  then show ?thesis
    by auto
qed

```

```

lemma uncountable_closed_interval:  $\text{uncountable } \{a..b\} \longleftrightarrow a < b$  for  $a\ b :: \text{real}$ 
  apply (rule iffI)
  apply (metis atLeastAtMost_singleton atLeastatMost_empty countable_finite
    finite.emptyI finite_insert linorder_neqE_linordered_idom)
  using real_interval_avoid_countable_set by fastforce

```

```

lemma open_minus_countable:
  fixes  $S\ A :: \text{real set}$ 
  assumes  $\text{countable } A\ S \neq \{\}$  open  $S$ 
  shows  $\exists x \in S. x \notin A$ 
proof -
  obtain  $x$  where  $x \in S$ 
    using ⟨ $S \neq \{\}$ ⟩ by auto
  then obtain  $e$  where  $0 < e\ \{y. \text{dist } y\ x < e\} \subseteq S$ 
    using ⟨open  $S$ ⟩ by (auto simp: open_dist subset_eq)
  moreover have  $\{y. \text{dist } y\ x < e\} = \{x - e <.. $x + e$ \}$ 
    by (auto simp: dist_real_def)
  ultimately have uncountable  $(S - A)$ 
    using uncountable_open_interval[of  $x - e\ x + e$ ] ⟨countable  $A$ ⟩
    by (intro uncountable_minus_countable) (auto dest: countable_subset)
  then show ?thesis
    unfolding uncountable_def by auto
qed

```

end

6.18 Homotopy of Maps

```

theory Homotopy
  imports Path_Connected Continuum_Not_Denumerable Product_Topology
begin

```

definition *homotopic_with*

where

```

  homotopic_with  $P\ X\ Y\ f\ g \equiv$ 
     $(\exists h. \text{continuous\_map } (\text{prod\_topology } (\text{top\_of\_set } \{0..1::\text{real}\}))\ X)\ Y\ h \wedge$ 

```

$$\begin{aligned}
& (\forall x. h(0, x) = f x) \wedge \\
& (\forall x. h(1, x) = g x) \wedge \\
& (\forall t \in \{0..1\}. P(\lambda x. h(t, x)))
\end{aligned}$$

p, q are functions $X \rightarrow Y$, and the property P restricts all intermediate maps. We often just want to require that P fixes some subset, but to include the case of a loop homotopy, it is convenient to have a general property P .

abbreviation *homotopic_with_canon* ::

$[(\text{'a}::\text{topological\_space} \Rightarrow \text{'b}::\text{topological\_space}) \Rightarrow \text{bool}, \text{'a set}, \text{'b set}, \text{'a} \Rightarrow \text{'b}, \text{'a} \Rightarrow \text{'b}] \Rightarrow \text{bool}$

where

homotopic_with_canon $P S T p q \equiv \text{homotopic\_with } P (\text{top\_of\_set } S) (\text{top\_of\_set } T) p q$

lemma *split_01*: $\{0..1::\text{real}\} = \{0..1/2\} \cup \{1/2..1\}$

by *force*

lemma *split_01_prod*: $\{0..1::\text{real}\} \times X = (\{0..1/2\} \times X) \cup (\{1/2..1\} \times X)$

by *force*

lemma *image_Pair_const*: $(\lambda x. (x, c)) \text{ ' } A = A \times \{c\}$

by *auto*

lemma *fst_o_paired* [*simp*]: $\text{fst} \circ (\lambda(x,y). (f x y, g x y)) = (\lambda(x,y). f x y)$

by *auto*

lemma *snd_o_paired* [*simp*]: $\text{snd} \circ (\lambda(x,y). (f x y, g x y)) = (\lambda(x,y). g x y)$

by *auto*

lemma *continuous_on_o_Pair*: $\llbracket \text{continuous\_on } (T \times X) h; t \in T \rrbracket \Longrightarrow \text{continuous\_on } X (h \circ \text{Pair } t)$

by (*fast intro: continuous_intros elim!: continuous_on_subset*)

lemma *continuous_map_o_Pair*:

assumes $h: \text{continuous\_map } (\text{prod\_topology } X Y) Z h$ **and** $t: t \in \text{topspace } X$

shows $\text{continuous\_map } Y Z (h \circ \text{Pair } t)$

by (*intro continuous_map_compose [OF _ h] continuous_intros; simp add: t*)

6.18.1 Trivial properties

We often want to just localize the ending function equality or whatever.

proposition *homotopic_with*:

assumes $\bigwedge h k. (\bigwedge x. x \in \text{topspace } X \Longrightarrow h x = k x) \Longrightarrow (P h \longleftrightarrow P k)$

shows $\text{homotopic\_with } P X Y p q \longleftrightarrow$

$(\exists h. \text{continuous\_map } (\text{prod\_topology } (\text{subtopology euclideanreal } \{0..1\}) X) Y h \wedge$

$(\forall x \in \text{topspace } X. h(0, x) = p x) \wedge$

$(\forall x \in \text{topspace } X. h(1, x) = q x) \wedge$

```

      (∀ t ∈ {0..1}. P(λx. h(t, x)))
  unfolding homotopic_with_def
  apply (rule iffI, blast, clarify)
  apply (rule_tac x=λ(u,v). if v ∈ topspace X then h(u,v) else if u = 0 then p v
else q v in exI)
  apply auto
  using continuous_map_eq apply fastforce
  apply (drule_tac x=t in bspec, force)
  apply (subst assms; simp)
done

```

```

lemma homotopic_with_mono:
  assumes hom: homotopic_with P X Y f g
  and Q: ∧h. [[continuous_map X Y h; P h]] ⇒ Q h
  shows homotopic_with Q X Y f g
  using hom unfolding homotopic_with_def
  by (force simp: o_def dest: continuous_map_o_Pair intro: Q)

```

```

lemma homotopic_with_imp_continuous_maps:
  assumes homotopic_with P X Y f g
  shows continuous_map X Y f ∧ continuous_map X Y g
proof -
  obtain h :: real × 'a ⇒ 'b
  where conth: continuous_map (prod_topology (top_of_set {0..1}) X) Y h
  and h: ∀ x. h (0, x) = f x ∧ x. h (1, x) = g x
  using assms by (auto simp: homotopic_with_def)
  have *: t ∈ {0..1} ⇒ continuous_map X Y (h ∘ (λx. (t,x))) for t
  by (rule continuous_map_compose [OF _ conth]) (simp add: o_def continu-
ous_map_pairwise)
  show ?thesis
  using h *[of 0] *[of 1] by (simp add: continuous_map_eq)
qed

```

```

lemma homotopic_with_imp_continuous:
  assumes homotopic_with_canon P X Y f g
  shows continuous_on X f ∧ continuous_on X g
  by (meson assms continuous_map_subtopology_eu homotopic_with_imp_continuous_maps)

```

```

lemma homotopic_with_imp_property:
  assumes homotopic_with P X Y f g
  shows P f ∧ P g
proof
  obtain h where h: ∧x. h(0, x) = f x ∧ x. h(1, x) = g x and P: ∧t. t ∈
{0..1::real} ⇒ P(λx. h(t,x))
  using assms by (force simp: homotopic_with_def)
  show P f P g
  using P [of 0] P [of 1] by (force simp: h)+
qed

```

```

lemma homotopic_with_equal:
  assumes  $P\ f\ P\ g$  and contf: continuous_map  $X\ Y\ f$  and fg:  $\bigwedge x. x \in \text{topspace } X \implies f\ x = g\ x$ 
  shows homotopic_with  $P\ X\ Y\ f\ g$ 
  unfolding homotopic_with_def
proof (intro exI conjI allI ballI)
  let  $?h = \lambda(t::\text{real}, x). \text{if } t = 1 \text{ then } g\ x \text{ else } f\ x$ 
  show continuous_map (prod_topology (top_of_set  $\{0..1\}$ )  $X$ )  $Y\ ?h$ 
  proof (rule continuous_map_eq)
    show continuous_map (prod_topology (top_of_set  $\{0..1\}$ )  $X$ )  $Y\ (f \circ \text{snd})$ 
    by (simp add: contf continuous_map_of_snd)
  qed (auto simp: fg)
  show  $P\ (\lambda x. ?h\ (t, x))$  if  $t \in \{0..1\}$  for  $t$ 
  by (cases  $t = 1$ ) (simp_all add: assms)
qed auto

```

```

lemma homotopic_with_imp_subset1:
  homotopic_with_canon  $P\ X\ Y\ f\ g \implies f\ ' X \subseteq Y$ 
  by (simp add: homotopic_with_def image_subset_iff) (metis atLeastAtMost_iff order_refl zero_le_one)

```

```

lemma homotopic_with_imp_subset2:
  homotopic_with_canon  $P\ X\ Y\ f\ g \implies g\ ' X \subseteq Y$ 
  by (simp add: homotopic_with_def image_subset_iff) (metis atLeastAtMost_iff order_refl zero_le_one)

```

```

lemma homotopic_with_subset_left:
   $\llbracket \text{homotopic\_with\_canon } P\ X\ Y\ f\ g; Z \subseteq X \rrbracket \implies \text{homotopic\_with\_canon } P\ Z\ Y\ f\ g$ 
  unfolding homotopic_with_def by (auto elim!: continuous_on_subset ex_forward)

```

```

lemma homotopic_with_subset_right:
   $\llbracket \text{homotopic\_with\_canon } P\ X\ Y\ f\ g; Y \subseteq Z \rrbracket \implies \text{homotopic\_with\_canon } P\ X\ Z\ f\ g$ 
  unfolding homotopic_with_def by (auto elim!: continuous_on_subset ex_forward)

```

6.18.2 Homotopy with P is an equivalence relation

(on continuous functions mapping X into Y that satisfy P , though this only affects reflexivity)

```

lemma homotopic_with_refl [simp]: homotopic_with  $P\ X\ Y\ f\ f \longleftrightarrow \text{continuous\_map } X\ Y\ f \wedge P\ f$ 
  by (auto simp: homotopic_with_imp_continuous_maps intro: homotopic_with_equal dest: homotopic_with_imp_property)

```

```

lemma homotopic_with_symD:
  assumes homotopic_with  $P\ X\ Y\ f\ g$ 
  shows homotopic_with  $P\ X\ Y\ g\ f$ 
proof –

```

```

let ?I01 = subtopology euclideanreal {0..1}
let ?j =  $\lambda y. (1 - \text{fst } y, \text{snd } y)$ 
have 1: continuous_map (prod_topology ?I01 X) (prod_topology euclideanreal
X) ?j
by (intro continuous_intros; simp add: continuous_map_subtopology_fst prod_topology_subtopology)
have *: continuous_map (prod_topology ?I01 X) (prod_topology ?I01 X) ?j
proof -
  have continuous_map (prod_topology ?I01 X) (subtopology (prod_topology
euclideanreal X) ( $\{0..1\} \times \text{topspace } X$ )) ?j
  by (simp add: continuous_map_into_subtopology [OF 1] image_subset_iff)
  then show ?thesis
  by (simp add: prod_topology_subtopology(1))
qed
show ?thesis
using assms
apply (clarsimp simp add: homotopic_with_def)
subgoal for h
  by (rule_tac x=h  $\circ (\lambda y. (1 - \text{fst } y, \text{snd } y))$ ) in exI (simp add: continuous_map_compose [OF *])
  done
qed

lemma homotopic_with_sym:
  homotopic_with P X Y f g  $\longleftrightarrow$  homotopic_with P X Y g f
by (metis homotopic_with_symD)

proposition homotopic_with_trans:
  assumes homotopic_with P X Y f g homotopic_with P X Y g h
  shows homotopic_with P X Y f h
proof -
  let ?X01 = prod_topology (subtopology euclideanreal {0..1}) X
  obtain k1 k2
  where contk1: continuous_map ?X01 Y k1 and contk2: continuous_map ?X01
Y k2
  and k12:  $\forall x. k1 (1, x) = g\ x \ \forall x. k2 (0, x) = g\ x$ 
 $\forall x. k1 (0, x) = f\ x \ \forall x. k2 (1, x) = h\ x$ 
  and P:  $\forall t \in \{0..1\}. P (\lambda x. k1 (t, x)) \ \forall t \in \{0..1\}. P (\lambda x. k2 (t, x))$ 
  using assms by (auto simp: homotopic_with_def)
  define k where k  $\equiv \lambda y. \text{if } \text{fst } y \leq 1/2$ 
 $\text{then } (k1 \circ (\lambda x. (2 *_{\mathbb{R}} \text{fst } x, \text{snd } x)))\ y$ 
 $\text{else } (k2 \circ (\lambda x. (2 *_{\mathbb{R}} \text{fst } x - 1, \text{snd } x)))\ y$ 
  have keq:  $k1 (2 * u, v) = k2 (2 * u - 1, v)$  if  $u = 1/2$  for u v
  by (simp add: k12 that)
  show ?thesis
  unfolding homotopic_with_def
proof (intro exI conjI)
  show continuous_map ?X01 Y k
  unfolding k_def
  proof (rule continuous_map_cases_le)

```



```

show fst: continuous_map ?X01 euclideanreal fst
  using continuous_map_fst continuous_map_in_subtopology by blast
show continuous_map ?X01 euclideanreal ( $\lambda x. 1/2$ )
  by simp
show continuous_map (subtopology ?X01  $\{y \in \text{topspace } ?X01. \text{fst } y \leq 1/2\}$ )
Y
  ( $k1 \circ (\lambda x. (2 *_{\mathbb{R}} \text{fst } x, \text{snd } x))$ )
  apply (intro fst continuous_map_compose [OF _ contk1] continuous_intros
continuous_map_into_subtopology continuous_map_from_subtopology | simp)+
  by (force simp: prod_topology_subtopology)
show continuous_map (subtopology ?X01  $\{y \in \text{topspace } ?X01. 1/2 \leq \text{fst } y\}$ )
Y
  ( $k2 \circ (\lambda x. (2 *_{\mathbb{R}} \text{fst } x - 1, \text{snd } x))$ )
  apply (intro fst continuous_map_compose [OF _ contk2] continuous_intros
continuous_map_into_subtopology continuous_map_from_subtopology | simp)+
  by (force simp: prod_topology_subtopology)
show ( $k1 \circ (\lambda x. (2 *_{\mathbb{R}} \text{fst } x, \text{snd } x))$ )  $y = (k2 \circ (\lambda x. (2 *_{\mathbb{R}} \text{fst } x - 1, \text{snd } x)))$   $y$ 
  if  $y \in \text{topspace } ?X01$  and  $\text{fst } y = 1/2$  for  $y$ 
  using that by (simp add: keq)
qed
show  $\forall x. k(0, x) = f x$ 
  by (simp add: k12 k_def)
show  $\forall x. k(1, x) = h x$ 
  by (simp add: k12 k_def)
show  $\forall t \in \{0..1\}. P(\lambda x. k(t, x))$ 
proof
  fix t show  $t \in \{0..1\} \implies P(\lambda x. k(t, x))$ 
    by (cases  $t \leq 1/2$ ) (auto simp add: k_def P)
qed
qed
qed

```

lemma *homotopic_with_id2*:

```

( $\bigwedge x. x \in \text{topspace } X \implies g(f x) = x$ )  $\implies$  homotopic_with ( $\lambda x. \text{True}$ )  $X X (g \circ f)$  id
  by (metis comp_apply continuous_map_id eq_id_iff homotopic_with_equal homotopic_with_symD)

```

6.18.3 Continuity lemmas

lemma *homotopic_with_compose_continuous_map_left*:

```

[[homotopic_with p  $X1 X2 f g$ ; continuous_map  $X2 X3 h$ ;  $\bigwedge j. p j \implies q(h \circ j)]]$ 
 $\implies$  homotopic_with  $q X1 X3 (h \circ f) (h \circ g)$ 
unfolding homotopic_with_def
apply clarify
subgoal for k
  by (rule_tac  $x = h \circ k$  in exI) (rule conjI continuous_map_compose | simp add:
o_def)+

```

done

lemma *homotopic_with_compose_continuous_map_right*:

assumes *hom*: *homotopic_with* p $X2$ $X3$ f g **and** *conth*: *continuous_map* $X1$ $X2$ h

and $q: \bigwedge j. p\ j \implies q(j \circ h)$

shows *homotopic_with* q $X1$ $X3$ $(f \circ h)$ $(g \circ h)$

proof –

obtain k

where *contk*: *continuous_map* $(\text{prod\_topology } (\text{subtopology euclideanreal } \{0..1\})$ $X2)$ $X3$ k

and $k: \forall x. k\ (0, x) = f\ x \ \forall x. k\ (1, x) = g\ x$ **and** $p: \bigwedge t. t \in \{0..1\} \implies p\ (\lambda x. k\ (t, x))$

using *hom* **unfolding** *homotopic_with_def* **by** *blast*

have *hsnd*: *continuous_map* $(\text{prod\_topology } (\text{subtopology euclideanreal } \{0..1\})$ $X1)$ $X2$ $(h \circ \text{snd})$

by $(\text{rule continuous\_map\_compose } [OF\ \text{continuous\_map\_snd}\ \text{conth}])$

let $?h = k \circ (\lambda(t, x). (t, h\ x))$

show *?thesis*

unfolding *homotopic_with_def*

proof $(\text{intro exI conjI allI ballI})$

have *continuous_map* $(\text{prod\_topology } (\text{top\_of\_set } \{0..1\})$ $X1)$

$(\text{prod\_topology } (\text{top\_of\_set } \{0..1::\text{real}\})$ $X2)$ $(\lambda(t, x). (t, h\ x))$

by $(\text{metis } (\text{mono\_tags}, \text{lifting})\ \text{case\_prod\_beta' comp\_def continuous\_map\_eq continuous\_map\_fst continuous\_map\_pairedI}\ \text{hsnd})$

then show *continuous_map* $(\text{prod\_topology } (\text{subtopology euclideanreal } \{0..1\})$ $X1)$ $X3$ $?h$

by $(\text{intro conjI continuous\_map\_compose } [OF\ \_\ \text{contk}])$

show $q\ (\lambda x. ?h\ (t, x))$ **if** $t \in \{0..1\}$ **for** t

using $q\ [OF\ p\ [OF\ \text{that}]]$ **by** (simp add: o_def)

qed (auto simp: k)

qed

corollary *homotopic_compose*:

assumes *homotopic_with* $(\lambda x. \text{True})$ X Y f f' *homotopic_with* $(\lambda x. \text{True})$ Y Z g g'

shows *homotopic_with* $(\lambda x. \text{True})$ X Z $(g \circ f)$ $(g' \circ f')$

proof $(\text{rule homotopic\_with\_trans } [\text{where } g = g \circ f'])$

show *homotopic_with* $(\lambda x. \text{True})$ X Z $(g \circ f)$ $(g \circ f')$

using *assms* **by** $(\text{simp add: homotopic\_with\_compose\_continuous\_map\_left homotopic\_with\_imp\_continuous\_maps})$

show *homotopic_with* $(\lambda x. \text{True})$ X Z $(g \circ f')$ $(g' \circ f')$

using *assms* **by** $(\text{simp add: homotopic\_with\_compose\_continuous\_map\_right homotopic\_with\_imp\_continuous\_maps})$

qed

proposition *homotopic_with_compose_continuous_right*:

$\llbracket \text{homotopic\_with\_canon } (\lambda f. p\ (f \circ h))\ X\ Y\ f\ g; \text{continuous\_on } W\ h; h\ ' W \subseteq X \rrbracket$

```

     $\implies \text{homotopic\_with\_canon } p \ W \ Y \ (f \circ h) \ (g \circ h)$ 
  apply (clarsimp simp add: homotopic_with_def)
  subgoal for k
    apply (rule_tac x=k  $\circ (\lambda y. (\text{fst } y, h (\text{snd } y)))$  in exI)
    by (intro conjI continuous_intros continuous_on_compose2 [where f=snd
and g=h]; fastforce simp: o_def elim: continuous_on_subset)
  done

```

proposition *homotopic_with_compose_continuous_left:*

$\llbracket \text{homotopic\_with\_canon } (\lambda f. p \ (h \circ f)) \ X \ Y \ f \ g; \text{continuous\_on } Y \ h; h \text{ ' } Y \subseteq Z \rrbracket$

```

     $\implies \text{homotopic\_with\_canon } p \ X \ Z \ (h \circ f) \ (h \circ g)$ 
  apply (clarsimp simp add: homotopic_with_def)
  subgoal for k
    apply (rule_tac x=h  $\circ k$  in exI)
    by (intro conjI continuous_intros continuous_on_compose [where f=snd and
g=h, unfolded o_def]; fastforce simp: o_def elim: continuous_on_subset)
  done

```

lemma *homotopic_from_subtopology:*

```

  homotopic_with P X X' f g  $\implies$  homotopic_with P (subtopology X s) X' f g
  unfolding homotopic_with_def
  by (force simp add: continuous_map_from_subtopology prod_topology_subtopology(2)
elim!: ex_forward)

```

lemma *homotopic_on_emptyI:*

```

  assumes topspace X = {} P f P g
  shows homotopic_with P X X' f g
  unfolding homotopic_with_def
  proof (intro exI conjI ballI)
    show P  $(\lambda x. (\lambda(t,x). \text{if } t = 0 \text{ then } f \ x \text{ else } g \ x) \ (t, x))$  if  $t \in \{0..1\}$  for  $t::\text{real}$ 
    by (cases t = 0, auto simp: assms)
  qed (auto simp: continuous_map_atin assms)

```

lemma *homotopic_on_empty:*

```

  topspace X = {}  $\implies$  (homotopic_with P X X' f g  $\longleftrightarrow$  P f  $\wedge$  P g)
  using homotopic_on_emptyI homotopic_with_imp_property by metis

```

lemma *homotopic_with_canon_on_empty [simp]:* homotopic_with_canon $(\lambda x.$

```

True) {} t f g
  by (auto intro: homotopic_with_equal)

```

lemma *homotopic_constant_maps:*

```

  homotopic_with  $(\lambda x. \text{True}) \ X \ X' \ (\lambda x. a) \ (\lambda x. b) \longleftrightarrow$ 
  topspace X = {}  $\vee$  path_component_of X' a b (is ?lhs = ?rhs)
  proof (cases topspace X = {})
    case False
    then obtain c where c:  $c \in \text{topspace } X$ 
    by blast

```

```

have  $\exists g. \text{continuous\_map } (\text{top\_of\_set } \{0..1::\text{real}\}) \ X' \ g \wedge g \ 0 = a \wedge g \ 1 = b$ 
  if  $x \in \text{topspace } X$  and  $\text{hom: homotopic\_with } (\lambda x. \text{True}) \ X \ X' \ (\lambda x. a) \ (\lambda x. b)$ 
for  $x$ 
proof -
  obtain  $h :: \text{real} \times 'a \Rightarrow 'b$ 
    where  $\text{conth: continuous\_map } (\text{prod\_topology } (\text{top\_of\_set } \{0..1\}) \ X) \ X' \ h$ 
      and  $h: \bigwedge x. h \ (0, x) = a \wedge \bigwedge x. h \ (1, x) = b$ 
    using  $\text{hom}$  by (auto simp: homotopic_with_def)
  have  $\text{cont: continuous\_map } (\text{top\_of\_set } \{0..1\}) \ X' \ (h \circ (\lambda t. (t, c)))$ 
  by (rule continuous_map_compose [OF _ conth] continuous_intros c | simp)+
  then show ?thesis
    by (force simp: h)
qed
moreover have homotopic_with  $(\lambda x. \text{True}) \ X \ X' \ (\lambda x. g \ 0) \ (\lambda x. g \ 1)$ 
  if  $x \in \text{topspace } X \ a = g \ 0 \ b = g \ 1$  continuous_map  $(\text{top\_of\_set } \{0..1\}) \ X' \ g$ 
  for  $x$  and  $g :: \text{real} \Rightarrow 'b$ 
  unfolding homotopic_with_def
  by (force intro!: continuous_map_compose continuous_intros c that)
ultimately show ?thesis
  using False by (auto simp: path_component_of_def pathin_def)
qed (simp add: homotopic_on_empty)

```

```

proposition homotopic_with_eq:
  assumes  $h: \text{homotopic\_with } P \ X \ Y \ f \ g$ 
    and  $f': \bigwedge x. x \in \text{topspace } X \implies f' \ x = f \ x$ 
    and  $g': \bigwedge x. x \in \text{topspace } X \implies g' \ x = g \ x$ 
    and  $P: (\bigwedge h \ k. (\bigwedge x. x \in \text{topspace } X \implies h \ x = k \ x) \implies P \ h \longleftrightarrow P \ k)$ 
  shows homotopic_with  $P \ X \ Y \ f' \ g'$ 
  using  $h$  unfolding homotopic_with_def
  apply clarify
  subgoal for  $h$ 
    apply (rule_tac  $x=\lambda(u,v). \text{if } v \in \text{topspace } X \text{ then } h(u,v) \text{ else if } u = 0 \text{ then } f'$ 
       $v \text{ else } g' \ v$  in exI)
    apply (simp add:  $f' \ g'$ , safe)
    apply (fastforce intro: continuous_map_eq)
    apply (subst  $P$ ; fastforce)
  done
done

```

```

lemma homotopic_with_prod_topology:
  assumes homotopic_with  $p \ X1 \ Y1 \ f \ f'$  and homotopic_with  $q \ X2 \ Y2 \ g \ g'$ 
    and  $r: \bigwedge i \ j. \llbracket p \ i; q \ j \rrbracket \implies r(\lambda(x,y). (i \ x, j \ y))$ 
  shows homotopic_with  $r \ (\text{prod\_topology } X1 \ X2) \ (\text{prod\_topology } Y1 \ Y2)$ 
     $(\lambda z. (f(\text{fst } z), g(\text{snd } z))) \ (\lambda z. (f'(\text{fst } z), g'(\text{snd } z)))$ 
proof -
  obtain  $h$ 
    where  $h: \text{continuous\_map } (\text{prod\_topology } (\text{subtopology euclideanreal } \{0..1\}) \ X1) \ Y1 \ h$ 
    and  $h0: \bigwedge x. h \ (0, x) = f \ x$ 

```

```

    and h1:  $\bigwedge x. h(1, x) = f' x$ 
    and p:  $\bigwedge t. \llbracket 0 \leq t; t \leq 1 \rrbracket \implies p(\lambda x. h(t, x))$ 
    using assms unfolding homotopic_with_def by auto
  obtain k
  where k: continuous_map (prod_topology (subtopology euclideanreal {0..1}))
X2) Y2 k
    and k0:  $\bigwedge x. k(0, x) = g x$ 
    and k1:  $\bigwedge x. k(1, x) = g' x$ 
    and q:  $\bigwedge t. \llbracket 0 \leq t; t \leq 1 \rrbracket \implies q(\lambda x. k(t, x))$ 
    using assms unfolding homotopic_with_def by auto
  let ?hk =  $\lambda(t, x, y). (h(t, x), k(t, y))$ 
  show ?thesis
    unfolding homotopic_with_def
  proof (intro conjI allI exI)
    show continuous_map (prod_topology (subtopology euclideanreal {0..1})) (prod_topology
X1 X2))
      (prod_topology Y1 Y2) ?hk
    unfolding continuous_map_pairwise case_prod_unfold
    by (rule conjI continuous_map_pairedI continuous_intros continuous_map_id
[unfolded id_def]
      continuous_map_fst_of [unfolded o_def] continuous_map_snd_of [unfolded
o_def]
      continuous_map_compose [OF _ h, unfolded o_def]
      continuous_map_compose [OF _ k, unfolded o_def]])+
  next
    fix x
    show ?hk(0, x) = (f(fst x), g(snd x)) ?hk(1, x) = (f'(fst x), g'(snd x))
    by (auto simp: case_prod_beta h0 k0 h1 k1)
  qed (auto simp: p q r)
qed

```

lemma homotopic_with_product_topology:

```

  assumes ht:  $\bigwedge i. i \in I \implies \text{homotopic\_with } (p\ i) (X\ i) (Y\ i) (f\ i) (g\ i)$ 
    and pq:  $\bigwedge h. (\bigwedge i. i \in I \implies p\ i (h\ i)) \implies q(\lambda x. (\lambda i \in I. h\ i\ (x\ i)))$ 
  shows homotopic_with q (product_topology X I) (product_topology Y I)
    ( $\lambda z. (\lambda i \in I. (f\ i)\ (z\ i))$ ) ( $\lambda z. (\lambda i \in I. (g\ i)\ (z\ i))$ )

  proof -
    obtain h
    where h:  $\bigwedge i. i \in I \implies \text{continuous\_map } (\text{prod\_topology } (\text{subtopology euclidean-}
\text{real } \{0..1\}) (X\ i)) (Y\ i) (h\ i)$ 
    and h0:  $\bigwedge i\ x. i \in I \implies h\ i\ (0, x) = f\ i\ x$ 
    and h1:  $\bigwedge i\ x. i \in I \implies h\ i\ (1, x) = g\ i\ x$ 
    and p:  $\bigwedge i\ t. \llbracket i \in I; t \in \{0..1\} \rrbracket \implies p\ i\ (\lambda x. h\ i\ (t, x))$ 
    using ht unfolding homotopic_with_def by metis
  show ?thesis
    unfolding homotopic_with_def
  proof (intro conjI allI exI)
    let ?h =  $\lambda(t, z). \lambda i \in I. h\ i\ (t, z\ i)$ 

```

```

have continuous_map (prod_topology (subtopology euclideanreal {0..1})) (product_topology
X I))
      (Y i) (λx. h i (fst x, snd x i)) if i ∈ I for i
proof –
  have §: continuous_map (prod_topology (top_of_set {0..1})) (product_topology
X I)) (X i) (λx. snd x i)
    using continuous_map_componentwise continuous_map_snd that by fast-
force
  show ?thesis
    unfolding continuous_map_pairwise case_prod_unfold
    by (intro conjI that § continuous_intros continuous_map_compose [OF _
h, unfolded o_def])
  qed
then show continuous_map (prod_topology (subtopology euclideanreal {0..1}))
(product_topology X I))
  (product_topology Y I) ?h
    by (auto simp: continuous_map_componentwise case_prod_beta)
show ?h (0, x) = (λi∈I. f i (x i)) ?h (1, x) = (λi∈I. g i (x i)) for x
    by (auto simp: case_prod_beta h0 h1)
show ∀ t∈{0..1}. q (λx. ?h (t, x))
    by (force intro: p pq)
qed
qed

```

Homotopic triviality implicitly incorporates path-connectedness.

lemma *homotopic_triviality*:

```

shows (∀ f g. continuous_on S f ∧ f ‘ S ⊆ T ∧
      continuous_on S g ∧ g ‘ S ⊆ T
      ⟶ homotopic_with_canon (λx. True) S T f g) ⟷
      (S = {} ∨ path_connected T) ∧
      (∀ f. continuous_on S f ∧ f ‘ S ⊆ T ⟶ (∃ c. homotopic_with_canon (λx.
True) S T f (λx. c)))
      (is ?lhs = ?rhs)
proof (cases S = {} ∨ T = {})
  case True then show ?thesis
    by (auto simp: homotopic_on_emptyI)
next
  case False show ?thesis
    proof
      assume LHS [rule_format]: ?lhs
      have pab: path_component T a b if a ∈ T b ∈ T for a b
      proof –
        have homotopic_with_canon (λx. True) S T (λx. a) (λx. b)
          by (simp add: LHS image_subset_iff that)
        then show ?thesis
          using False homotopic_constant_maps [of top_of_set S top_of_set T a b]
by auto
      qed
    moreover

```

```

  have  $\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S T f (\lambda x. c)$  if  $\text{continuous\_on } S$ 
   $f f ' S \subseteq T$  for  $f$ 
    using False LHS continuous_on_const that by blast
  ultimately show  $?rhs$ 
    by (simp add: path_connected_component)
next
  assume RHS: ?rhs
  with False have  $T: \text{path\_connected } T$ 
    by blast
  show  $?lhs$ 
  proof clarify
    fix  $f g$ 
    assume  $\text{continuous\_on } S f f ' S \subseteq T$   $\text{continuous\_on } S g g ' S \subseteq T$ 
    obtain  $c d$  where  $c: \text{homotopic\_with\_canon } (\lambda x. \text{True}) S T f (\lambda x. c)$  and
 $d: \text{homotopic\_with\_canon } (\lambda x. \text{True}) S T g (\lambda x. d)$ 
      using False  $\langle \text{continuous\_on } S f \rangle \langle f ' S \subseteq T \rangle$  RHS  $\langle \text{continuous\_on } S g \rangle \langle g$ 
 $' S \subseteq T \rangle$  by blast
    then have  $c \in T d \in T$ 
      using False homotopic_with_imp_continuous_maps by fastforce+
    with  $T$  have  $\text{path\_component } T c d$ 
      using path_connected_component by blast
    then have  $\text{homotopic\_with\_canon } (\lambda x. \text{True}) S T (\lambda x. c) (\lambda x. d)$ 
      by (simp add: homotopic_constant_maps)
    with  $c d$  show  $\text{homotopic\_with\_canon } (\lambda x. \text{True}) S T f g$ 
      by (meson homotopic_with_symD homotopic_with_trans)
  qed
qed
qed

```

6.18.4 Homotopy of paths, maintaining the same endpoints

definition $\text{homotopic\_paths} :: ['a \text{ set}, \text{real} \Rightarrow 'a, \text{real} \Rightarrow 'a::\text{topological\_space}] \Rightarrow \text{bool}$

where

$\text{homotopic\_paths } s p q \equiv$
 $\text{homotopic\_with\_canon } (\lambda r. \text{pathstart } r = \text{pathstart } p \wedge \text{pathfinish } r = \text{pathfinish } p) \{0..1\} s p q$

lemma homotopic_paths :

$\text{homotopic\_paths } s p q \longleftrightarrow$
 $(\exists h. \text{continuous\_on } (\{0..1\} \times \{0..1\}) h \wedge$
 $h ' (\{0..1\} \times \{0..1\}) \subseteq s \wedge$
 $(\forall x \in \{0..1\}. h(0, x) = p x) \wedge$
 $(\forall x \in \{0..1\}. h(1, x) = q x) \wedge$
 $(\forall t \in \{0..1::\text{real}\}. \text{pathstart}(h \circ \text{Pair } t) = \text{pathstart } p \wedge$
 $\text{pathfinish}(h \circ \text{Pair } t) = \text{pathfinish } p))$
by (*auto simp: homotopic_paths_def homotopic_with_pathstart_def pathfinish_def*)

proposition $\text{homotopic\_paths\_imp\_pathstart}$:

2012

$\text{homotopic\_paths } s \ p \ q \implies \text{pathstart } p = \text{pathstart } q$
by (metis (mono_tags, lifting) homotopic_paths_def homotopic_with_imp_property)

proposition homotopic_paths_imp_pathfinish:
 $\text{homotopic\_paths } s \ p \ q \implies \text{pathfinish } p = \text{pathfinish } q$
by (metis (mono_tags, lifting) homotopic_paths_def homotopic_with_imp_property)

lemma homotopic_paths_imp_path:
 $\text{homotopic\_paths } s \ p \ q \implies \text{path } p \wedge \text{path } q$
using homotopic_paths_def homotopic_with_imp_continuous_maps_path_def
continuous_map_subtopology_eu **by** blast

lemma homotopic_paths_imp_subset:
 $\text{homotopic\_paths } s \ p \ q \implies \text{path\_image } p \subseteq s \wedge \text{path\_image } q \subseteq s$
by (metis (mono_tags) continuous_map_subtopology_eu homotopic_paths_def
homotopic_with_imp_continuous_maps_path_image_def)

proposition homotopic_paths_refl [simp]: $\text{homotopic\_paths } s \ p \ p \longleftrightarrow \text{path } p \wedge$
 $\text{path\_image } p \subseteq s$
by (simp add: homotopic_paths_def path_def path_image_def)

proposition homotopic_paths_sym: $\text{homotopic\_paths } s \ p \ q \implies \text{homotopic\_paths}$
 $s \ q \ p$
by (metis (mono_tags) homotopic_paths_def homotopic_paths_imp_pathfinish
homotopic_paths_imp_pathstart homotopic_with_symD)

proposition homotopic_paths_sym_eq: $\text{homotopic\_paths } s \ p \ q \longleftrightarrow \text{homotopic\_paths}$
 $s \ q \ p$
by (metis homotopic_paths_sym)

proposition homotopic_paths_trans [trans]:
assumes homotopic_paths $s \ p \ q$ homotopic_paths $s \ q \ r$
shows homotopic_paths $s \ p \ r$
proof –
have pathstart $q = \text{pathstart } p$ pathfinish $q = \text{pathfinish } p$
using assms **by** (simp_all add: homotopic_paths_imp_pathstart homotopic_paths_imp_pathfinish)
then have homotopic_with_canon $(\lambda f. \text{pathstart } f = \text{pathstart } p \wedge \text{pathfinish } f$
 $= \text{pathfinish } p) \ \{0..1\} \ s \ q \ r$
using <homotopic_paths $s \ q \ r$ > homotopic_paths_def **by** force
then show ?thesis
using assms homotopic_paths_def homotopic_with_trans **by** blast
qed

proposition homotopic_paths_eq:
 $\llbracket \text{path } p; \text{path\_image } p \subseteq s; \bigwedge t. t \in \{0..1\} \implies p \ t = q \ t \rrbracket \implies \text{homotopic\_paths}$
 $s \ p \ q$
unfolding homotopic_paths_def
by (rule_eq)
(auto simp: path_def pathstart_def pathfinish_def path_image_def elim: con-

tinuous_on_eq)

proposition *homotopic_paths_reparametrize*:

```

  assumes path p
    and pips: path_image p  $\subseteq$  s
    and conf: continuous_on {0..1} f
    and f01: f ' {0..1}  $\subseteq$  {0..1}
    and [simp]: f(0) = 0 f(1) = 1
    and q:  $\bigwedge t. t \in \{0..1\} \implies q(t) = p(f\ t)$ 
  shows homotopic_paths s p q
proof -
  have contp: continuous_on {0..1} p
  by (metis <path p> path_def)
  then have continuous_on {0..1} (p  $\circ$  f)
  using conf continuous_on_compose continuous_on_subset f01 by blast
  then have path q
  by (simp add: path_def) (metis q continuous_on_cong)
  have pips: path_image q  $\subseteq$  s
  by (metis (no_types, opaque_lifting) pips f01 image_subset_iff path_image_def
q)
  have fb0:  $\bigwedge a\ b. [0 \leq a; a \leq 1; 0 \leq b; b \leq 1] \implies 0 \leq (1 - a) * f\ b + a * b$ 
  using f01 by force
  have fb1:  $[0 \leq a; a \leq 1; 0 \leq b; b \leq 1] \implies (1 - a) * f\ b + a * b \leq 1$  for a b
  using f01 [THEN subsetD, of f b] by (simp add: convex_bound_le)
  have homotopic_paths s q p
proof (rule homotopic_paths_trans)
  show homotopic_paths s q (p  $\circ$  f)
  using q by (force intro: homotopic_paths_eq [OF <path q> pips])
next
  show homotopic_paths s (p  $\circ$  f) p
  using pips [unfolded path_image_def]
  apply (simp add: homotopic_paths_def homotopic_with_def)
  apply (rule_tac x=p  $\circ$  ( $\lambda y. (1 - (fst\ y)) * _R ((f \circ snd)\ y) + (fst\ y) * _R\ snd$ 
y) in exI)
  apply (rule conjI conf continuous_intros continuous_on_subset [OF contp]
| simp)+
  by (auto simp: fb0 fb1 pathstart_def pathfinish_def)
qed
then show ?thesis
by (simp add: homotopic_paths_sym)
qed

```

lemma *homotopic_paths_subset*: $[homotopic\_paths\ s\ p\ q; s \subseteq t] \implies homotopic\_paths\ t\ p\ q$

unfolding *homotopic_paths* by fast

A slightly ad-hoc but useful lemma in constructing homotopies.

lemma *continuous_on_homotopic_join_lemma*:

fixes $q :: [real, real] \Rightarrow 'a::topological\_space$

```

    assumes  $p$ : continuous_on  $(\{0..1\} \times \{0..1\})$   $(\lambda y. p \text{ (fst } y) \text{ (snd } y))$  (is continuous_on ?A ?p)
    and  $q$ : continuous_on  $(\{0..1\} \times \{0..1\})$   $(\lambda y. q \text{ (fst } y) \text{ (snd } y))$  (is continuous_on ?A ?q)
    and  $pf$ :  $\bigwedge t. t \in \{0..1\} \implies \text{pathfinish}(p \text{ } t) = \text{pathstart}(q \text{ } t)$ 
    shows continuous_on  $(\{0..1\} \times \{0..1\})$   $(\lambda y. (p \text{ (fst } y) \text{ } ++ \text{ } q \text{ (fst } y)) \text{ (snd } y))$ 
  proof -
    have  $\S$ :  $(\lambda t. p \text{ (fst } t) \text{ } (2 * \text{snd } t)) = ?p \circ (\lambda y. (\text{fst } y, 2 * \text{snd } y))$ 
       $(\lambda t. q \text{ (fst } t) \text{ } (2 * \text{snd } t - 1)) = ?q \circ (\lambda y. (\text{fst } y, 2 * \text{snd } y - 1))$ 
    by force+
    show ?thesis
      unfolding joinpaths_def
    proof (rule continuous_on_cases_le)
      show continuous_on  $\{y \in ?A. \text{snd } y \leq 1/2\}$   $(\lambda t. p \text{ (fst } t) \text{ } (2 * \text{snd } t))$ 
        continuous_on  $\{y \in ?A. 1/2 \leq \text{snd } y\}$   $(\lambda t. q \text{ (fst } t) \text{ } (2 * \text{snd } t - 1))$ 
        continuous_on ?A snd
      unfolding  $\S$ 
      by (rule continuous_intros continuous_on_subset [OF p] continuous_on_subset [OF q] | force)+
    qed (use pf in <auto simp: mult.commute pathstart_def pathfinish_def>)
  qed

```

Congruence properties of homotopy w.r.t. path-combining operations.

lemma *homotopic_paths_reversepath_D*:

```

    assumes homotopic_paths  $s$   $p$   $q$ 
    shows homotopic_paths  $s$   $(\text{reversepath } p)$   $(\text{reversepath } q)$ 
  using assms
  apply (simp add: homotopic_paths_def homotopic_with_def, clarify)
  apply (rule_tac  $x=h \circ (\lambda x. (\text{fst } x, 1 - \text{snd } x))$  in exI)
  apply (rule conjI continuous_intros)+
  apply (auto simp: reversepath_def pathstart_def pathfinish_def elim!: continuous_on_subset)
  done

```

proposition *homotopic_paths_reversepath*:

```

  homotopic_paths  $s$   $(\text{reversepath } p)$   $(\text{reversepath } q) \longleftrightarrow \text{homotopic\_paths } s \text{ } p \text{ } q$ 
  using homotopic_paths_reversepath_D by force

```

proposition *homotopic_paths_join*:

```

   $\llbracket \text{homotopic\_paths } s \text{ } p \text{ } p'; \text{homotopic\_paths } s \text{ } q \text{ } q'; \text{pathfinish } p = \text{pathstart } q \rrbracket$ 
 $\implies \text{homotopic\_paths } s \text{ } (p \text{ } ++ \text{ } q) \text{ } (p' \text{ } ++ \text{ } q')$ 
  apply (clarsimp simp add: homotopic_paths_def homotopic_with_def)
  apply (rename_tac  $k1$   $k2$ )
  apply (rule_tac  $x=(\lambda y. ((k1 \circ \text{Pair } (\text{fst } y)) \text{ } ++ \text{ } (k2 \circ \text{Pair } (\text{fst } y)))) \text{ (snd } y))$ 
  in exI)
  apply (intro conjI continuous_intros continuous_on_homotopic_join_lemma;
  force simp: joinpaths_def pathstart_def pathfinish_def path_image_def)
  done

```

proposition *homotopic_paths_continuous_image*:

$\llbracket \text{homotopic\_paths } s \text{ f g}; \text{continuous\_on } s \text{ h}; h \text{ ' } s \subseteq t \rrbracket \implies \text{homotopic\_paths } t$
 $(h \circ f) (h \circ g)$

unfolding *homotopic_paths_def*

by (*simp add: homotopic_with_compose_continuous_map_left pathfinish_compose pathstart_compose*)

6.18.5 Group properties for homotopy of paths

So taking equivalence classes under homotopy would give the fundamental group

proposition *homotopic_paths_rid*:

assumes *path p path_image p $\subseteq$ s*

shows *homotopic_paths s (p +++ linepath (pathfinish p) (pathfinish p)) p*

proof –

have $\S$: *continuous_on $\{0..1\}$ ($\lambda t::\text{real. if } t \leq 1/2 \text{ then } 2 *_R t \text{ else } 1$)*

unfolding *split_01*

by (*rule continuous_on_cases continuous_intros | force simp: pathfinish_def joinpaths_def*) +

show *?thesis*

apply (*rule homotopic_paths_sym*)

using *assms unfolding pathfinish_def joinpaths_def*

by (*intro $\S$ continuous_on_cases continuous_intros homotopic_paths_reparametrize*
 $[\text{where } f = \lambda t. \text{ if } t \leq 1/2 \text{ then } 2 *_R t \text{ else } 1]; \text{force}$)

qed

proposition *homotopic_paths_lid*:

$\llbracket \text{path } p; \text{path\_image } p \subseteq s \rrbracket \implies \text{homotopic\_paths } s (\text{linepath } (\text{pathstart } p) (\text{pathstart } p) +++ p) p$

using *homotopic_paths_rid [of reversepath p s]*

by (*metis homotopic_paths_reversepath path_image_reversepath path_reversepath pathfinish_linepath pathfinish_reversepath reversepath_joinpaths reversepath_linepath*)

proposition *homotopic_paths_assoc*:

$\llbracket \text{path } p; \text{path\_image } p \subseteq s; \text{path } q; \text{path\_image } q \subseteq s; \text{path } r; \text{path\_image } r \subseteq s; \text{pathfinish } p = \text{pathstart } q;$

$\text{pathfinish } q = \text{pathstart } r \rrbracket$

$\implies \text{homotopic\_paths } s (p +++ (q +++ r)) ((p +++ q) +++ r)$

apply (*subst homotopic_paths_sym*)

apply (*rule homotopic_paths_reparametrize*

$[\text{where } f = \lambda t. \text{ if } t \leq 1/2 \text{ then } \text{inverse } 2 *_R t$
 $\text{else if } t \leq 3 / 4 \text{ then } t - (1 / 4)$
 $\text{else } 2 *_R t - 1])$

apply (*simp_all del: le_divide_eq_numeral1 add: subset_path_image_join*)

apply (*rule continuous_on_cases_1 continuous_intros | auto simp: joinpaths_def*) +
done

proposition *homotopic_paths_rinv*:
assumes *path p path_image p $\subseteq$ s*
shows *homotopic_paths s (p +++ reversepath p) (linepath (pathstart p) (pathstart p))*
proof –
have *p: continuous_on {0..1} p*
using *assms by (auto simp: path_def)*
let *?A = {0..1} $\times$ {0..1}*
have *continuous_on ?A (λx . (subpath 0 (fst x) p +++ reversepath (subpath 0 (fst x) p)) (snd x))*
unfolding *joinpaths_def subpath_def reversepath_def path_def add_0_right diff_0_right*
proof (*rule continuous_on_cases_le*)
show *continuous_on {x $\in$?A. snd x $\leq$ 1/2} (λt . p (fst t * (2 * snd t)))*
*continuous_on {x $\in$?A. 1/2 $\leq$ snd x} (λt . p (fst t * (1 - (2 * snd t - 1)))))*
continuous_on ?A snd
by (*intro continuous_on_compose2 [OF p] continuous_intros; auto simp add: mult_le_one*)
qed (*auto simp add: algebra_simps*)
then show *?thesis*
using *assms*
apply (*subst homotopic_paths_sym_eq*)
unfolding *homotopic_paths_def homotopic_with_def*
apply (*rule_tac x=(λy . (subpath 0 (fst y) p +++ reversepath (subpath 0 (fst y) p)) (snd y)) in exI*)
apply (*force simp: mult_le_one path_defs joinpaths_def subpath_def reversepath_def*)
done
qed

proposition *homotopic_paths_linv*:
assumes *path p path_image p $\subseteq$ s*
shows *homotopic_paths s (reversepath p +++ p) (linepath (pathfinish p) (pathfinish p))*
using *homotopic_paths_rinv [of reversepath p s] assms by simp*

6.18.6 Homotopy of loops without requiring preservation of endpoints

definition *homotopic_loops* :: *'a::topological_space set $\Rightarrow$ (real $\Rightarrow$ 'a) $\Rightarrow$ (real $\Rightarrow$ 'a) $\Rightarrow$ bool* **where**
homotopic_loops s p q $\equiv$
homotopic_with_canon (λr . pathfinish r = pathstart r) {0..1} s p q

lemma *homotopic_loops*:
homotopic_loops s p q $\longleftrightarrow$
($\exists h$. continuous_on ({0..1::real} $\times$ {0..1}) h $\wedge$
image h ({0..1} $\times$ {0..1}) $\subseteq$ s $\wedge$
($\forall x \in$ {0..1}. h(0,x) = p x) $\wedge$

$(\forall x \in \{0..1\}. h(1, x) = q\ x) \wedge$
 $(\forall t \in \{0..1\}. \text{pathfinish}(h \circ \text{Pair } t) = \text{pathstart}(h \circ \text{Pair } t))$
by (simp add: homotopic_loops_def pathstart_def pathfinish_def homotopic_with)

proposition homotopic_loops_imp_loop:

$\text{homotopic\_loops } s\ p\ q \implies \text{pathfinish } p = \text{pathstart } p \wedge \text{pathfinish } q = \text{pathstart } q$
using homotopic_with_imp_property homotopic_loops_def **by** blast

proposition homotopic_loops_imp_path:

$\text{homotopic\_loops } s\ p\ q \implies \text{path } p \wedge \text{path } q$
unfolding homotopic_loops_def path_def
using homotopic_with_imp_continuous_maps continuous_map_subtopology_eu
by blast

proposition homotopic_loops_imp_subset:

$\text{homotopic\_loops } s\ p\ q \implies \text{path\_image } p \subseteq s \wedge \text{path\_image } q \subseteq s$
unfolding homotopic_loops_def path_image_def
by (meson continuous_map_subtopology_eu homotopic_with_imp_continuous_maps)

proposition homotopic_loops_refl:

$\text{homotopic\_loops } s\ p\ p \longleftrightarrow$
 $\text{path } p \wedge \text{path\_image } p \subseteq s \wedge \text{pathfinish } p = \text{pathstart } p$
by (simp add: homotopic_loops_def path_image_def path_def)

proposition homotopic_loops_sym: $\text{homotopic\_loops } s\ p\ q \implies \text{homotopic\_loops } s\ q\ p$

by (simp add: homotopic_loops_def homotopic_with_sym)

proposition homotopic_loops_sym_eq: $\text{homotopic\_loops } s\ p\ q \longleftrightarrow \text{homotopic\_loops } s\ q\ p$

by (metis homotopic_loops_sym)

proposition homotopic_loops_trans:

$\llbracket \text{homotopic\_loops } s\ p\ q; \text{homotopic\_loops } s\ q\ r \rrbracket \implies \text{homotopic\_loops } s\ p\ r$
unfolding homotopic_loops_def **by** (blast intro: homotopic_with_trans)

proposition homotopic_loops_subset:

$\llbracket \text{homotopic\_loops } s\ p\ q; s \subseteq t \rrbracket \implies \text{homotopic\_loops } t\ p\ q$
by (fastforce simp add: homotopic_loops)

proposition homotopic_loops_eq:

$\llbracket \text{path } p; \text{path\_image } p \subseteq s; \text{pathfinish } p = \text{pathstart } p; \bigwedge t. t \in \{0..1\} \implies p(t) = q(t) \rrbracket$
 $\implies \text{homotopic\_loops } s\ p\ q$

unfolding homotopic_loops_def path_image_def path_def pathstart_def pathfinish_def

by (auto intro: homotopic_with_eq [OF homotopic_with_refl [where f = p, THEN iffD2]])


```

have [simp]:  $\bigwedge t. \llbracket 0 \leq t \wedge t \leq 1 \rrbracket \implies h(t, 1) = h(t, 0)$ 
  using ends by (simp add: pathfinish_def pathstart_def)
have ad hoc_le:  $c * 4 \leq 1 + c * (d * 4)$  if  $\neg d * 4 \leq 3$   $0 \leq c$   $c \leq 1$  for  $c$ 
d::real
proof -
  have  $c * 3 \leq c * (d * 4)$  using that less_eq_real_def by auto
  with  $\langle c \leq 1 \rangle$  show ?thesis by fastforce
qed
have *:  $\bigwedge p x. \llbracket \text{path } p \wedge \text{path}(\text{reversepath } p);$ 
   $\text{path\_image } p \subseteq s \wedge \text{path\_image}(\text{reversepath } p) \subseteq s;$ 
   $\text{pathfinish } p = \text{pathstart}(\text{linepath } a \text{ a} \text{ +++ } \text{reversepath } p) \wedge$ 
   $\text{pathstart}(\text{reversepath } p) = a \wedge \text{pathstart } p = x \rrbracket$ 
   $\implies \text{homotopic\_paths } s (p \text{ +++ } \text{linepath } a \text{ a} \text{ +++ } \text{reversepath } p)$ 
(linepath  $x \ x$ )
  by (metis homotopic_paths_lid homotopic_paths_join
    homotopic_paths_trans homotopic_paths_sym homotopic_paths_rinv)
have 1: homotopic_paths  $s \ p \ (p \text{ +++ } \text{linepath } (\text{pathfinish } p) (\text{pathfinish } p))$ 
  using  $\langle \text{path } p \rangle$  homotopic_paths_rid homotopic_paths_sym pip by blast
moreover have homotopic_paths  $s \ (p \text{ +++ } \text{linepath } (\text{pathfinish } p) (\text{pathfinish } p))$ 
  (linepath ( $\text{pathstart } p$ ) ( $\text{pathstart } p$ ) +++  $p \text{ +++ }$ 
linepath ( $\text{pathfinish } p$ ) ( $\text{pathfinish } p$ ))
  apply (rule homotopic_paths_sym)
  using homotopic_paths_lid [of  $p \text{ +++ } \text{linepath } (\text{pathfinish } p) (\text{pathfinish } p) \ s$ ]
  by (metis 1 homotopic_paths_imp_path homotopic_paths_imp_pathstart homotopic_paths_imp_subset)
moreover
  have homotopic_paths  $s \ (\text{linepath } (\text{pathstart } p) (\text{pathstart } p) \text{ +++ } p \text{ +++ } \text{linepath } (\text{pathfinish } p) (\text{pathfinish } p))$ 
  (( $\lambda u. h(u, 0)$ ) +++ linepath  $a \ a \text{ +++ } \text{reversepath}$ 
( $\lambda u. h(u, 0)$ ))
  unfolding homotopic_paths_def homotopic_with_def
proof (intro exI strip conjI)
  let ?h =  $\lambda y. (\text{subpath } 0 \ (\text{fst } y) \ (\lambda u. h(u, 0)) \text{ +++ } (\lambda u. h(\text{Pair } (\text{fst } y) \ u))$ 
    +++  $\text{subpath } (\text{fst } y) \ 0 \ (\lambda u. h(u, 0))) \ (\text{snd } y)$ )
  have continuous_on ?A ?h
    by (intro continuous_on_homotopic_join_lemma; simp add: path_defs joinpaths_def subpath_def conth c1 c2)
  moreover have ?h ' $?A \subseteq s$ 
    unfolding joinpaths_def subpath_def
    by (force simp: algebra_simps mult_le_one mult_left_le intro: hs [THEN subsetD] ad hoc_le)
  ultimately show continuous_map (prod topology (top_of_set  $\{0..1\}$ ) (top_of_set  $\{0..1\}$ ))
    (top_of_set  $s$ ) ?h
    by (simp add: subpath_reversepath)
qed (use ploop in  $\langle \text{simp\_all add: reversepath\_def path\_defs joinpaths\_def o\_def subpath\_def conth c1 c2} \rangle$ )
moreover have homotopic_paths  $s \ ((\lambda u. h(u, 0)) \text{ +++ } \text{linepath } a \ a \text{ +++ } \text{reversepath } a \ a \text{ +++ } \text{reversepath } a \ a)$ 
  by (metis homotopic_paths_lid homotopic_paths_join homotopic_paths_trans homotopic_paths_sym homotopic_paths_rinv)

```

2020

```

reversepath (λu. h (u, 0)))
      (linepath (pathstart p) (pathstart p))
proof (rule *; simp add: pih0 pathstart_def pathfinish_def conth0)
  show a = (linepath a a +++ reversepath (λu. h (u, 0))) 0 ∧ reversepath (λu.
h (u, 0)) 0 = a
    by (simp_all add: reversepath_def joinpaths_def)
qed
ultimately show ?thesis
  by (blast intro: homotopic_paths_trans)
qed

proposition homotopic_loops_conjugate:
  fixes s :: 'a::real_normed_vector set
  assumes path p path q and pip: path_image p ⊆ s and piq: path_image q ⊆ s
    and pq: pathfinish p = pathstart q and qloop: pathfinish q = pathstart q
    shows homotopic_loops s (p +++ q +++ reversepath p) q
proof -
  have contp: continuous_on {0..1} p using ⟨path p⟩ [unfolded path_def] by blast
  have contq: continuous_on {0..1} q using ⟨path q⟩ [unfolded path_def] by blast
  let ?A = {0..1::real} × {0..1::real}
  have c1: continuous_on ?A (λx. p ((1 - fst x) * snd x + fst x))
  proof (rule continuous_on_compose [of _ _ p, unfolded o_def])
    show continuous_on ((λx. (1 - fst x) * snd x + fst x) ' ?A) p
    by (auto intro: continuous_on_subset [OF contp] simp: algebra_simps add_increasing2
mult_right_le_one_le sum_le_prod1)
  qed (force intro: continuous_intros)
  have c2: continuous_on ?A (λx. p ((fst x - 1) * snd x + 1))
  proof (rule continuous_on_compose [of _ _ p, unfolded o_def])
    show continuous_on ((λx. (fst x - 1) * snd x + 1) ' ?A) p
    by (auto intro: continuous_on_subset [OF contp] simp: algebra_simps add_increasing2
mult_left_le_one_le)
  qed (force intro: continuous_intros)

  have ps1: ∧a b. [b * 2 ≤ 1; 0 ≤ b; 0 ≤ a; a ≤ 1] ⇒ p ((1 - a) * (2 * b) +
a) ∈ s
    using sum_le_prod1
    by (force simp: algebra_simps add_increasing2 mult_left_le intro: pip [unfolded
path_image_def, THEN subsetD])
  have ps2: ∧a b. [¬ 4 * b ≤ 3; b ≤ 1; 0 ≤ a; a ≤ 1] ⇒ p ((a - 1) * (4 * b
- 3) + 1) ∈ s
    apply (rule pip [unfolded path_image_def, THEN subsetD])
    apply (rule image_eqI, blast)
    apply (simp add: algebra_simps)
    by (metis add_mono_thms_linordered_semiring(1) affine_ineq linear mult.commute
mult_left_neutral mult_right_mono
      add.commute zero_le_numeral)
  have qs: ∧a b. [4 * b ≤ 3; ¬ b * 2 ≤ 1] ⇒ q (4 * b - 2) ∈ s
    using path_image_def piq by fastforce
  have homotopic_loops s (p +++ q +++ reversepath p)

```



```

      (linepath (pathstart q) (pathstart q) +++ q +++ linepath
(pathstart q) (pathstart q))
    unfolding homotopic_loops_def homotopic_with_def
  proof (intro exI strip conjI)
    let ?h = (λy. (subpath (fst y) 1 p +++ q +++ subpath 1 (fst y) p) (snd y))
    have continuous_on ?A (λy. q (snd y))
      by (force simp: contq intro: continuous_on_compose [of _ _ q, unfolded
o_def] continuous_on_id continuous_on_snd)
    then have continuous_on ?A ?h
      using pq qloop
      by (intro continuous_on_homotopic_join_lemma) (auto simp: path_defs
joinpaths_def subpath_def c1 c2)
    then show continuous_map (prod_topology (top_of_set {0..1}) (top_of_set
{0..1})) (top_of_set s) ?h
      by (auto simp: joinpaths_def subpath_def ps1 ps2 qs)
    show ?h (1,x) = (linepath (pathstart q) (pathstart q) +++ q +++ linepath
(pathstart q) (pathstart q)) x for x
      using pq by (simp add: pathfinish_def subpath_refl)
    qed (auto simp: subpath_reversepath)
    moreover have homotopic_loops s (linepath (pathstart q) (pathstart q) +++ q
+++ linepath (pathstart q) (pathstart q)) q
      proof -
        have homotopic_paths s (linepath (pathfinish q) (pathfinish q) +++ q) q
          using ⟨path q⟩ homotopic_paths_lid qloop piq by auto
        hence 1: ∧f. homotopic_paths s f q ∨ ¬ homotopic_paths s f (linepath
(pathfinish q) (pathfinish q) +++ q)
          using homotopic_paths_trans by blast
        hence homotopic_paths s (linepath (pathfinish q) (pathfinish q) +++ q +++
linepath (pathfinish q) (pathfinish q)) q
          proof -
            have homotopic_paths s (q +++ linepath (pathfinish q) (pathfinish q)) q
              by (simp add: ⟨path q⟩ homotopic_paths_rid piq)
            thus ?thesis
              by (metis (no_types) 1 ⟨path q⟩ homotopic_paths_join homotopic_paths_rinv
homotopic_paths_sym
                homotopic_paths_trans qloop pathfinish_linepath piq)
          qed
        thus ?thesis
          by (metis (no_types) qloop homotopic_loops_sym homotopic_paths_imp_homotopic_loops
homotopic_paths_imp_pathfinish homotopic_paths_sym)
        qed
      ultimately show ?thesis
        by (blast intro: homotopic_loops_trans)
    qed

lemma homotopic_paths_loop_parts:
  assumes loops: homotopic_loops S (p +++ reversepath q) (linepath a a) and
  path q
  shows homotopic_paths S p q

```

```

proof –
  have paths: homotopic_paths S (p +++ reversepath q) (linepath (pathstart p)
(pathstart p))
    using homotopic_loops_imp_homotopic_paths_null [OF loops] by simp
    then have path p
      using  $\langle \text{path } q \rangle$  homotopic_loops_imp_path loops path_join path_join_path_ends
path_reversepath by blast
    show ?thesis
    proof (cases pathfinish p = pathfinish q)
      case True
        have pipq: path_image p  $\subseteq$  S path_image q  $\subseteq$  S
          by (metis Un_subset_iff paths  $\langle \text{path } p \rangle$   $\langle \text{path } q \rangle$  homotopic_loops_imp_subset
homotopic_paths_imp_path loops
path_image_join path_image_reversepath path_imp_reversepath path_join_eq) +
        have homotopic_paths S p (p +++ (linepath (pathfinish p) (pathfinish p)))
          using  $\langle \text{path } p \rangle$   $\langle \text{path\_image } p \subseteq S \rangle$  homotopic_paths_rid homotopic_paths_sym
by blast
        moreover have homotopic_paths S (p +++ (linepath (pathfinish p) (pathfinish
p))) (p +++ (reversepath q +++ q))
          by (simp add: True  $\langle \text{path } p \rangle$   $\langle \text{path } q \rangle$  pipq homotopic_paths_join homotopic_paths_linv
homotopic_paths_sym)
        moreover have homotopic_paths S (p +++ (reversepath q +++ q)) ((p +++
reversepath q) +++ q)
          by (simp add: True  $\langle \text{path } p \rangle$   $\langle \text{path } q \rangle$  homotopic_paths_assoc pipq)
        moreover have homotopic_paths S ((p +++ reversepath q) +++ q) (linepath
(pathstart p) (pathstart p) +++ q)
          by (simp add:  $\langle \text{path } q \rangle$  homotopic_paths_join paths pipq)
        moreover then have homotopic_paths S (linepath (pathstart p) (pathstart p)
+++ q) q
          by (metis  $\langle \text{path } q \rangle$  homotopic_paths_imp_path homotopic_paths_lid linepath_trivial
path_join_path_ends pathfinish_def pipq(2))
        ultimately show ?thesis
          using homotopic_paths_trans by metis
      next
        case False
        then show ?thesis
          using  $\langle \text{path } q \rangle$  homotopic_loops_imp_path loops path_join_path_ends by
fastforce
    qed
  qed

```

6.18.8 Homotopy of "nearby" function, paths and loops

```

lemma homotopic_with_linear:
  fixes f g ::  $\_ \Rightarrow 'b::\text{real\_normed\_vector}$ 
  assumes contf: continuous_on S f
    and contg: continuous_on S g
    and sub:  $\bigwedge x. x \in S \implies \text{closed\_segment } (f\ x) (g\ x) \subseteq t$ 
  shows homotopic_with_canon ( $\lambda z. \text{True}$ ) S t f g

```

```

unfolding homotopic_with_def
apply (rule_tac x= $\lambda y. ((1 - (fst\ y)) *_{\mathbb{R}} f(snd\ y) + (fst\ y) *_{\mathbb{R}} g(snd\ y))$  in exI)
using sub_closed_segment_def
  by (fastforce intro: continuous_intros continuous_on_subset [OF contf] continuous_on_compose2 [where g=f]
    continuous_on_subset [OF contg] continuous_on_compose2 [where g=g])

```

```

lemma homotopic_paths_linear:
  fixes g h :: real  $\Rightarrow$  'a::real_normed_vector
  assumes path g path h pathstart h = pathstart g pathfinish h = pathfinish g
     $\bigwedge t. t \in \{0..1\} \implies \text{closed\_segment } (g\ t) (h\ t) \subseteq S$ 
  shows homotopic_paths S g h
using assms
unfolding path_def
apply (simp add: closed_segment_def pathstart_def pathfinish_def homotopic_paths_def
  homotopic_with_def)
apply (rule_tac x= $\lambda y. ((1 - (fst\ y)) *_{\mathbb{R}} (g \circ snd)\ y + (fst\ y) *_{\mathbb{R}} (h \circ snd)\ y)$  in exI)
in exI)
apply (intro conjI subsetI continuous_intros; force)
done

```

```

lemma homotopic_loops_linear:
  fixes g h :: real  $\Rightarrow$  'a::real_normed_vector
  assumes path g path h pathfinish g = pathstart g pathfinish h = pathstart h
     $\bigwedge x. t \in \{0..1\} \implies \text{closed\_segment } (g\ t) (h\ t) \subseteq S$ 
  shows homotopic_loops S g h
using assms
unfolding path_defs homotopic_loops_def homotopic_with_def
apply (rule_tac x= $\lambda y. ((1 - (fst\ y)) *_{\mathbb{R}} g(snd\ y) + (fst\ y) *_{\mathbb{R}} h(snd\ y))$  in exI)
  by (force simp: closed_segment_def intro!: continuous_intros intro: continuous_on_compose2 [where g=g] continuous_on_compose2 [where g=h])

```

```

lemma homotopic_paths_nearby_explicit:
  assumes §: path g path h pathstart h = pathstart g pathfinish h = pathfinish g
    and no:  $\bigwedge t x. [t \in \{0..1\}; x \notin S] \implies \text{norm}(h\ t - g\ t) < \text{norm}(g\ t - x)$ 
  shows homotopic_paths S g h
proof (rule homotopic_paths_linear [OF §])
  show  $\bigwedge t. t \in \{0..1\} \implies \text{closed\_segment } (g\ t) (h\ t) \subseteq S$ 
  by (metis no segment_bound(1) subsetI norm_minus_commute not_le)
qed

```

```

lemma homotopic_loops_nearby_explicit:
  assumes §: path g path h pathfinish g = pathstart g pathfinish h = pathstart h
    and no:  $\bigwedge t x. [t \in \{0..1\}; x \notin S] \implies \text{norm}(h\ t - g\ t) < \text{norm}(g\ t - x)$ 
  shows homotopic_loops S g h
proof (rule homotopic_loops_linear [OF §])
  show  $\bigwedge t. t \in \{0..1\} \implies \text{closed\_segment } (g\ t) (h\ t) \subseteq S$ 
  by (metis no segment_bound(1) subsetI norm_minus_commute not_le)

```

2024

qed

lemma *homotopic_nearby_paths*:

fixes $g\ h :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$

assumes $\text{path } g \text{ open } S \text{ path\_image } g \subseteq S$

shows $\exists e. 0 < e \wedge$

$(\forall h. \text{path } h \wedge$

$\text{pathstart } h = \text{pathstart } g \wedge \text{pathfinish } h = \text{pathfinish } g \wedge$

$(\forall t \in \{0..1\}. \text{norm}(h\ t - g\ t) < e) \longrightarrow \text{homotopic\_paths } S\ g\ h)$

proof –

obtain e **where** $e > 0$ **and** $e: \bigwedge x\ y. x \in \text{path\_image } g \implies y \in -S \implies e \leq \text{dist } x\ y$

using *separate_compact_closed* [of $\text{path\_image } g - S$] **assms** **by** *force*

show *?thesis*

using e [unfolded *dist_norm*] $\langle e > 0 \rangle$

by (*fastforce simp: path_image_def intro!: homotopic_paths_nearby_explicit assms exI*)

qed

lemma *homotopic_nearby_loops*:

fixes $g\ h :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$

assumes $\text{path } g \text{ open } S \text{ path\_image } g \subseteq S \text{ pathfinish } g = \text{pathstart } g$

shows $\exists e. 0 < e \wedge$

$(\forall h. \text{path } h \wedge \text{pathfinish } h = \text{pathstart } h \wedge$

$(\forall t \in \{0..1\}. \text{norm}(h\ t - g\ t) < e) \longrightarrow \text{homotopic\_loops } S\ g\ h)$

proof –

obtain e **where** $e > 0$ **and** $e: \bigwedge x\ y. x \in \text{path\_image } g \implies y \in -S \implies e \leq \text{dist } x\ y$

using *separate_compact_closed* [of $\text{path\_image } g - S$] **assms** **by** *force*

show *?thesis*

using e [unfolded *dist_norm*] $\langle e > 0 \rangle$

by (*fastforce simp: path_image_def intro!: homotopic_loops_nearby_explicit assms exI*)

qed

6.18.9 Homotopy and subpaths

lemma *homotopic_join_subpaths1*:

assumes $\text{path } g \text{ and pag: path\_image } g \subseteq s$

and $u: u \in \{0..1\}$ **and** $v: v \in \{0..1\}$ **and** $w: w \in \{0..1\}$ $u \leq v \leq w$

shows $\text{homotopic\_paths } s (\text{subpath } u\ v\ g \text{ +++ subpath } v\ w\ g) (\text{subpath } u\ w\ g)$

proof –

have $1: t * 2 \leq 1 \implies u + t * (v * 2) \leq v + t * (u * 2)$ **for** t

using *affine_ineq* $\langle u \leq v \rangle$ **by** *fastforce*

have $2: t * 2 > 1 \implies u + (2*t - 1) * v \leq v + (2*t - 1) * w$ **for** t

by (*metis add_mono_thms_linordered_semiring(1) diff_gt_0_iff_gt less_eq_real_def mult.commute mult_right_mono* $\langle u \leq v \rangle \langle v \leq w \rangle$)

have $t2: \bigwedge t::\text{real}. t*2 = 1 \implies t = 1/2$ **by** *auto*

have $\text{homotopic\_paths } (\text{path\_image } g) (\text{subpath } u\ v\ g \text{ +++ subpath } v\ w\ g)$

```

(subpath u w g)
proof (cases w = u)
  case True
    then show ?thesis
    by (metis ‹path g› homotopic_paths_rinv path_image_subpath_subset path_subpath
pathstart_subpath reversepath_subpath subpath_refl u v)
  next
    case False
    let ?f =  $\lambda t.$  if  $t \leq 1/2$  then  $\text{inverse}((w - u)) *_R (2 * (v - u)) *_R t$ 
      else  $\text{inverse}((w - u)) *_R ((v - u) + (w - v) *_R (2 *_R t$ 
      - 1))
    show ?thesis
    proof (rule homotopic_paths_sym [OF homotopic_paths_reparametrize [where
f = ?f]])
      show path (subpath u w g)
      using assms(1) path_subpath u w(1) by blast
      show path_image (subpath u w g)  $\subseteq$  path_image g
      by (meson path_image_subpath_subset u w(1))
      show continuous_on {0..1} ?f
      unfolding split_01
      by (rule continuous_on_cases continuous_intros | force simp: pathfinish_def
joinpaths_def dest!: t2)+
      show ?f ‘ {0..1}  $\subseteq$  {0..1}
      using False assms
      by (force simp: field_simps not_le mult_left_mono affine_ineq dest!: 1 2)
      show (subpath u v g +++ subpath v w g) t = subpath u w g (?f t) if t  $\in$ 
{0..1} for t
      using assms
      unfolding joinpaths_def subpath_def by (auto simp add: divide_simps
add.commute mult.commute mult.left_commute)
      qed (use False in auto)
    qed
    then show ?thesis
    by (rule homotopic_paths_subset [OF _ pag])
  qed

```

lemma homotopic_join_subpaths2:

```

assumes homotopic_paths s (subpath u v g +++ subpath v w g) (subpath u w g)
shows homotopic_paths s (subpath w v g +++ subpath v u g) (subpath w u g)
by (metis assms homotopic_paths_reversepath_D pathfinish_subpath pathstart_subpath
reversepath_joinpaths reversepath_subpath)

```

lemma homotopic_join_subpaths3:

```

assumes hom: homotopic_paths s (subpath u v g +++ subpath v w g) (subpath
u w g)
and path g and pag: path_image g  $\subseteq$  s
and u:  $u \in \{0..1\}$  and v:  $v \in \{0..1\}$  and w:  $w \in \{0..1\}$ 
shows homotopic_paths s (subpath v w g +++ subpath w u g) (subpath v u g)
proof -

```

```

have homotopic_paths s (subpath u w g +++ subpath w v g) ((subpath u v g
+++ subpath v w g) +++ subpath w v g)
proof (rule homotopic_paths_join)
  show homotopic_paths s (subpath u w g) (subpath u v g +++ subpath v w g)
  using hom homotopic_paths_sym_eq by blast
  show homotopic_paths s (subpath w v g) (subpath w v g)
  by (metis ⟨path g⟩ homotopic_paths_eq pag path_image_subpath_subset
path_subpath_subset_trans v w)
qed auto
also have homotopic_paths s ((subpath u v g +++ subpath v w g) +++ subpath
w v g) (subpath u v g +++ subpath v w g +++ subpath w v g)
by (rule homotopic_paths_sym [OF homotopic_paths_assoc])
  (use assms in ⟨simp_all add: path_image_subpath_subset [THEN or-
der_trans]⟩)
also have homotopic_paths s (subpath u v g +++ subpath v w g +++ subpath
w v g)
  (subpath u v g +++ linepath (pathfinish (subpath u v g))
(pathfinish (subpath u v g)))
proof (rule homotopic_paths_join; simp)
  show path (subpath u v g) ∧ path_image (subpath u v g) ⊆ s
  by (metis ⟨path g⟩ order.trans pag path_image_subpath_subset path_subpath
u v)
  show homotopic_paths s (subpath v w g +++ subpath w v g) (linepath (g v) (g
v))
  by (metis (no_types, lifting) ⟨path g⟩ homotopic_paths_linv order_trans pag
path_image_subpath_subset path_subpath pathfinish_subpath reversepath_subpath
v w)
qed
also have homotopic_paths s (subpath u v g +++ linepath (pathfinish (subpath
u v g)) (pathfinish (subpath u v g))) (subpath u v g)
proof (rule homotopic_paths_rid)
  show path (subpath u v g)
  using ⟨path g⟩ path_subpath u v by blast
  show path_image (subpath u v g) ⊆ s
  by (meson ⟨path g⟩ order.trans pag path_image_subpath_subset u v)
qed
finally have homotopic_paths s (subpath u w g +++ subpath w v g) (subpath u
v g) .
then show ?thesis
using homotopic_join_subpaths2 by blast
qed

```

proposition homotopic_join_subpaths:

```

⟦path g; path_image g ⊆ s; u ∈ {0..1}; v ∈ {0..1}; w ∈ {0..1}⟧
⇒ homotopic_paths s (subpath u v g +++ subpath v w g) (subpath u w g)
using le_cases3 [of u v w] homotopic_join_subpaths1 homotopic_join_subpaths2
homotopic_join_subpaths3
by metis

```

Relating homotopy of trivial loops to path-connectedness.

```

lemma path_component_imp_homotopic_points:
  assumes path_component  $S$   $a$   $b$ 
  shows homotopic_loops  $S$  (linepath  $a$   $a$ ) (linepath  $b$   $b$ )
proof -
  obtain  $g :: \text{real} \Rightarrow 'a$  where  $g$ : continuous_on  $\{0..1\}$   $g$   $g$  '  $\{0..1\} \subseteq S$   $g$   $0 = a$ 
   $g$   $1 = b$ 
  using assms by (auto simp: path_defs)
  then have continuous_on  $(\{0..1\} \times \{0..1\})$   $(g \circ \text{fst})$ 
  by (fastforce intro!: continuous_intros)+
  with  $g$  show ?thesis
  by (auto simp add: homotopic_loops_def homotopic_with_def path_defs image_subset_iff)
qed

```

```

lemma homotopic_loops_imp_path_component_value:
   $\llbracket \text{homotopic\_loops } S \text{ } p \text{ } q; 0 \leq t; t \leq 1 \rrbracket$ 
   $\implies \text{path\_component } S \text{ } (p \text{ } t) \text{ } (q \text{ } t)$ 
apply (clarsimp simp add: homotopic_loops_def homotopic_with_def path_defs)
apply (rule_tac  $x=h \circ (\lambda u. (u, t))$  in exI)
apply (fastforce elim!: continuous_on_subset intro!: continuous_intros)
done

```

```

lemma homotopic_points_eq_path_component:
  homotopic_loops  $S$  (linepath  $a$   $a$ ) (linepath  $b$   $b$ )  $\longleftrightarrow \text{path\_component } S$   $a$   $b$ 
by (auto simp: path_component_imp_homotopic_points
  dest: homotopic_loops_imp_path_component_value [where t=1])

```

```

lemma path_connected_eq_homotopic_points:
  path_connected  $S \longleftrightarrow$ 
   $(\forall a \text{ } b. a \in S \wedge b \in S \longrightarrow \text{homotopic\_loops } S \text{ } (\text{linepath } a \text{ } a) \text{ } (\text{linepath } b \text{ } b))$ 
by (auto simp: path_connected_def path_component_def homotopic_points_eq_path_component)

```

6.18.10 Simply connected sets

defined as "all loops are homotopic (as loops)"

```

definition simply_connected where
  simply_connected  $S \equiv$ 
   $\forall p \text{ } q. \text{path } p \wedge \text{pathfinish } p = \text{pathstart } p \wedge \text{path\_image } p \subseteq S \wedge$ 
   $\text{path } q \wedge \text{pathfinish } q = \text{pathstart } q \wedge \text{path\_image } q \subseteq S$ 
   $\longrightarrow \text{homotopic\_loops } S \text{ } p \text{ } q$ 

```

```

lemma simply_connected_empty [iff]: simply_connected  $\{\}$ 
  by (simp add: simply_connected_def)

```

```

lemma simply_connected_imp_path_connected:
  fixes  $S :: \_ :: \text{real\_normed\_vector\_set}$ 
  shows simply_connected  $S \implies \text{path\_connected } S$ 
by (simp add: simply_connected_def path_connected_eq_homotopic_points)

```

```

lemma simply_connected_imp_connected:
  fixes  $S :: \_ :: \text{real\_normed\_vector\_set}$ 
  shows simply_connected  $S \implies \text{connected } S$ 
by (simp add: path_connected_imp_connected simply_connected_imp_path_connected)

```

```

lemma simply_connected_eq_contractible_loop_any:
  fixes  $S :: \_ :: \text{real\_normed\_vector\_set}$ 
  shows simply_connected  $S \longleftrightarrow$ 
     $(\forall p \ a. \text{path } p \wedge \text{path\_image } p \subseteq S \wedge \text{pathfinish } p = \text{pathstart } p \wedge a \in S$ 
       $\longrightarrow \text{homotopic\_loops } S \ p \ (\text{linepath } a \ a))$ 
    (is  $?lhs = ?rhs$ )

```

```

proof
  assume  $?lhs$  then show  $?rhs$ 
    unfolding simply_connected_def by force
next
  assume  $?rhs$  then show  $?lhs$ 
    unfolding simply_connected_def
    by (metis pathfinish_in_path_image subsetD homotopic_loops_trans homotopic_loops_sym)
qed

```

```

lemma simply_connected_eq_contractible_loop_some:
  fixes  $S :: \_ :: \text{real\_normed\_vector\_set}$ 
  shows simply_connected  $S \longleftrightarrow$ 
    path_connected  $S \wedge$ 
     $(\forall p. \text{path } p \wedge \text{path\_image } p \subseteq S \wedge \text{pathfinish } p = \text{pathstart } p$ 
       $\longrightarrow (\exists a. a \in S \wedge \text{homotopic\_loops } S \ p \ (\text{linepath } a \ a)))$ 
    (is  $?lhs = ?rhs$ )

```

```

proof
  assume  $?lhs$ 
  then show  $?rhs$ 
    using simply_connected_eq_contractible_loop_any by (blast intro: simply_connected_imp_path_connected)
next
  assume  $r: ?rhs$ 
  note  $pa = r$  [THEN conjunct2, rule_format]
  show  $?lhs$ 
    proof (clarsimp simp add: simply_connected_eq_contractible_loop_any)
      fix  $p \ a$ 
      assume path  $p$  and path_image  $p \subseteq S$  pathfinish  $p = \text{pathstart } p$ 
      and  $a \in S$ 
      with  $pa$  [of p] show homotopic_loops  $S \ p \ (\text{linepath } a \ a)$ 
      using homotopic_loops_trans path_connected_eq_homotopic_points  $r$  by
blast
    qed
  qed

```

```

lemma simply_connected_eq_contractible_loop_all:
  fixes  $S :: \_ :: \text{real\_normed\_vector\_set}$ 
  shows simply_connected  $S \longleftrightarrow$ 

```



```

     $S = \{\} \vee$ 
     $(\exists a \in S. \forall p. \text{path } p \wedge \text{path\_image } p \subseteq S \wedge \text{pathfinish } p = \text{pathstart } p$ 
       $\longrightarrow \text{homotopic\_loops } S \text{ } p \text{ } (\text{linepath } a \text{ } a))$ 
     $(\text{is } ?lhs = ?rhs)$ 
  proof (cases  $S = \{\}$ )
    case True then show ?thesis by force
  next
    case False
    then obtain  $a$  where  $a \in S$  by blast
    show ?thesis
    proof
      assume simply_connected  $S$ 
      then show ?rhs
      using  $\langle a \in S \rangle \langle \text{simply\_connected } S \rangle \text{simply\_connected\_eq\_contractible\_loop\_any}$ 
      by blast
    next
      assume ?rhs
      then show simply_connected  $S$ 
      unfolding simply_connected_eq_contractible_loop_any
      by (meson False homotopic_loops_refl homotopic_loops_sym homotopic_loops_trans

        path_component_imp_homotopic_points path_component_refl)
    qed
  qed

  lemma simply_connected_eq_contractible_path:
    fixes  $S :: \_::\text{real\_normed\_vector\_set}$ 
    shows simply_connected  $S \longleftrightarrow$ 
       $\text{path\_connected } S \wedge$ 
       $(\forall p. \text{path } p \wedge \text{path\_image } p \subseteq S \wedge \text{pathfinish } p = \text{pathstart } p$ 
         $\longrightarrow \text{homotopic\_paths } S \text{ } p \text{ } (\text{linepath } (\text{pathstart } p) \text{ } (\text{pathstart } p)))$ 
       $(\text{is } ?lhs = ?rhs)$ 
  proof
    assume ?lhs
    then show ?rhs
    unfolding simply_connected_imp_path_connected
    by (metis simply_connected_eq_contractible_loop_some homotopic_loops_imp_homotopic_paths_null)
  next
    assume ?rhs
    then show ?lhs
    using homotopic_paths_imp_homotopic_loops simply_connected_eq_contractible_loop_some
    by fastforce
  qed

  lemma simply_connected_eq_homotopic_paths:
    fixes  $S :: \_::\text{real\_normed\_vector\_set}$ 
    shows simply_connected  $S \longleftrightarrow$ 
       $\text{path\_connected } S \wedge$ 
       $(\forall p \ q. \text{path } p \wedge \text{path\_image } p \subseteq S \wedge$ 

```

```

      path q ∧ path_image q ⊆ S ∧
      pathstart q = pathstart p ∧ pathfinish q = pathfinish p
      → homotopic_paths S p q)
    (is ?lhs = ?rhs)
  proof
    assume ?lhs
    then have pc: path_connected S
      and *: ⋀ p. [path p; path_image p ⊆ S;
        pathfinish p = pathstart p]
        ⇒ homotopic_paths S p (linepath (pathstart p) (pathstart p))
      by (auto simp: simply_connected_eq_contractible_path)
    have homotopic_paths S p q
      if path p path_image p ⊆ S path q
      path_image q ⊆ S pathstart q = pathstart p
      pathfinish q = pathfinish p for p q
    proof -
      have homotopic_paths S p (p +++ linepath (pathfinish p) (pathfinish p))
        by (simp add: homotopic_paths_rid homotopic_paths_sym that)
      also have homotopic_paths S (p +++ linepath (pathfinish p) (pathfinish p))
        (p +++ reversepath q +++ q)
        using that
        by (metis homotopic_paths_join homotopic_paths_linv homotopic_paths_refl
          homotopic_paths_sym_eq pathstart_linepath)
      also have homotopic_paths S (p +++ reversepath q +++ q)
        ((p +++ reversepath q) +++ q)
        by (simp add: that homotopic_paths_assoc)
      also have homotopic_paths S ((p +++ reversepath q) +++ q)
        (linepath (pathstart q) (pathstart q) +++ q)
        using * [of p +++ reversepath q] that
        by (simp add: homotopic_paths_join path_image_join)
      also have homotopic_paths S (linepath (pathstart q) (pathstart q) +++ q) q
        using that homotopic_paths_lid by blast
      finally show ?thesis .
    qed
    then show ?rhs
      by (blast intro: pc *)
  next
    assume ?rhs
    then show ?lhs
      by (force simp: simply_connected_eq_contractible_path)
    qed

  proposition simply_connected_Times:
    fixes S :: 'a::real_normed_vector set and T :: 'b::real_normed_vector set
    assumes S: simply_connected S and T: simply_connected T
    shows simply_connected(S × T)
  proof -
    have homotopic_loops (S × T) p (linepath (a, b) (a, b))
      if path p path_image p ⊆ S × T p 1 = p 0 a ∈ S b ∈ T

```

```

    for p a b
  proof -
    have path (fst ∘ p)
    by (simp add: continuous_on_fst Path_Connected.path_continuous_image
[OF ⟨path p⟩])
    moreover have path_image (fst ∘ p) ⊆ S
    using that by (force simp add: path_image_def)
    ultimately have p1: homotopic_loops S (fst ∘ p) (linepath a a)
    using S that
    by (simp add: simply_connected_eq_contractible_loop_any pathfinish_def
pathstart_def)
    have path (snd ∘ p)
    by (simp add: continuous_on_snd Path_Connected.path_continuous_image
[OF ⟨path p⟩])
    moreover have path_image (snd ∘ p) ⊆ T
    using that by (force simp: path_image_def)
    ultimately have p2: homotopic_loops T (snd ∘ p) (linepath b b)
    using T that
    by (simp add: simply_connected_eq_contractible_loop_any pathfinish_def
pathstart_def)
    show ?thesis
    using p1 p2 unfolding homotopic_loops
    apply clarify
    subgoal for h k
    by (rule_tac x=λz. (h z, k z) in exI) (force intro: continuous_intros simp:
path_defs)
    done
  qed
  with assms show ?thesis
  by (simp add: simply_connected_eq_contractible_loop_any pathfinish_def path-
start_def)
qed

```

6.18.11 Contractible sets

definition *contractible* where

contractible $S \equiv \exists a. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S S \text{id } (\lambda x. a)$

proposition *contractible_imp_simply_connected*:

fixes $S :: \_ :: \text{real\_normed\_vector set}$

assumes *contractible* S **shows** *simply_connected* S

proof (*cases* $S = \{\}$)

case *True* **then show** ?thesis **by** *force*

next

case *False*

obtain *a* **where** *a*: *homotopic_with_canon* $(\lambda x. \text{True}) S S \text{id } (\lambda x. a)$

using *assms* **by** (*force simp: contractible_def*)

then have $a \in S$

by (*metis* *False* *homotopic_constant_maps* *homotopic_with_symD* *homotopic_with_trans*)

```

path_component_in_topospace topspace_euclidean_subtopology)
  have  $\forall p. \text{path } p \wedge$ 
    path_image  $p \subseteq S \wedge \text{pathfinish } p = \text{pathstart } p \longrightarrow$ 
    homotopic_loops  $S \ p \ (\text{linepath } a \ a)$ 
  using a apply (clarsimp simp add: homotopic_loops_def homotopic_with_def
path_defs)
  apply (rule_tac  $x = (h \circ (\lambda y. (\text{fst } y, (p \circ \text{snd } y)))$ ) in exI)
  apply (intro conjI continuous_on_compose continuous_intros; force elim: con-
tinuous_on_subset)
  done
  with  $\langle a \in S \rangle$  show ?thesis
  by (auto simp add: simply_connected_eq_contractible_loop_all False)
qed

```

```

corollary contractible_imp_connected:
  fixes  $S :: \_::\text{real\_normed\_vector\_set}$ 
  shows  $\text{contractible } S \implies \text{connected } S$ 
by (simp add: contractible_imp_simply_connected simply_connected_imp_connected)

```

```

lemma contractible_imp_path_connected:
  fixes  $S :: \_::\text{real\_normed\_vector\_set}$ 
  shows  $\text{contractible } S \implies \text{path\_connected } S$ 
by (simp add: contractible_imp_simply_connected simply_connected_imp_path_connected)

```

```

lemma nullhomotopic_through_contractible:
  fixes  $S :: \_::\text{topological\_space}$  set
  assumes  $f: \text{continuous\_on } S \ f \ f' \ S \subseteq T$ 
    and  $g: \text{continuous\_on } T \ g \ g' \ T \subseteq U$ 
    and  $T: \text{contractible } T$ 
  obtains  $c$  where homotopic_with_canon  $(\lambda h. \text{True}) \ S \ U \ (g \circ f) \ (\lambda x. c)$ 
proof -
  obtain  $b$  where  $b: \text{homotopic\_with\_canon } (\lambda x. \text{True}) \ T \ T \ \text{id} \ (\lambda x. b)$ 
  using assms by (force simp: contractible_def)
  have homotopic_with_canon  $(\lambda f. \text{True}) \ T \ U \ (g \circ \text{id}) \ (g \circ (\lambda x. b))$ 
  by (metis Abstract_Topology.continuous_map_subtopology_eu b g homotopic_with_compose_continu
then have homotopic_with_canon  $(\lambda f. \text{True}) \ S \ U \ (g \circ \text{id} \circ f) \ (g \circ (\lambda x. b) \circ f)$ 
  by (simp add: f homotopic_with_compose_continuous_map_right)
  then show ?thesis
  by (simp add: comp_def that)
qed

```

```

lemma nullhomotopic_into_contractible:
  assumes  $f: \text{continuous\_on } S \ f \ f' \ S \subseteq T$ 
    and  $T: \text{contractible } T$ 
  obtains  $c$  where homotopic_with_canon  $(\lambda h. \text{True}) \ S \ T \ f \ (\lambda x. c)$ 
by (rule nullhomotopic_through_contractible [OF f, of id T]) (use assms in auto)

```

```

lemma nullhomotopic_from_contractible:
  assumes  $f: \text{continuous\_on } S \ f \ f' \ S \subseteq T$ 

```

and S : contractible S
 obtains c where homotopic_with_canon $(\lambda h. \text{True}) S T f (\lambda x. c)$
 by (auto simp: comp_def intro: nullhomotopic_through_contractible [OF continuous_on_id _ $f S$])

lemma homotopic_through_contractible:

fixes $S :: \_ :: \text{real\_normed\_vector\_set}$
 assumes continuous_on $S f1 f1 ' S \subseteq T$
 continuous_on $T g1 g1 ' T \subseteq U$
 continuous_on $S f2 f2 ' S \subseteq T$
 continuous_on $T g2 g2 ' T \subseteq U$
 contractible T path_connected U
 shows homotopic_with_canon $(\lambda h. \text{True}) S U (g1 \circ f1) (g2 \circ f2)$
 proof -
 obtain $c1$ where $c1$: homotopic_with_canon $(\lambda h. \text{True}) S U (g1 \circ f1) (\lambda x. c1)$
 by (rule nullhomotopic_through_contractible [of $S f1 T g1 U$]) (use assms in auto)
 obtain $c2$ where $c2$: homotopic_with_canon $(\lambda h. \text{True}) S U (g2 \circ f2) (\lambda x. c2)$
 by (rule nullhomotopic_through_contractible [of $S f2 T g2 U$]) (use assms in auto)
 have $S = \{\}$ $\vee (\exists t. \text{path\_connected } t \wedge t \subseteq U \wedge c2 \in t \wedge c1 \in t)$
 proof (cases $S = \{\}$)
 case True then show ?thesis by force
 next
 case False
 with $c1 c2$ have $c1 \in U c2 \in U$
 using homotopic_with_imp_continuous_maps by fastforce+
 with $\langle \text{path\_connected } U \rangle$ show ?thesis by blast
 qed
 then have homotopic_with_canon $(\lambda h. \text{True}) S U (\lambda x. c2) (\lambda x. c1)$
 by (simp add: path_component homotopic_constant_maps)
 then show ?thesis
 using $c1 c2$ homotopic_with_symD homotopic_with_trans by blast
 qed

lemma homotopic_into_contractible:

fixes $S :: 'a :: \text{real\_normed\_vector\_set}$ and $T :: 'b :: \text{real\_normed\_vector\_set}$
 assumes f : continuous_on $S f f ' S \subseteq T$
 and g : continuous_on $S g g ' S \subseteq T$
 and T : contractible T
 shows homotopic_with_canon $(\lambda h. \text{True}) S T f g$
 using homotopic_through_contractible [of $S f T \text{id } T g \text{id}$]
 by (simp add: assms contractible_imp_path_connected)

lemma homotopic_from_contractible:

fixes $S :: 'a :: \text{real\_normed\_vector\_set}$ and $T :: 'b :: \text{real\_normed\_vector\_set}$
 assumes f : continuous_on $S f f ' S \subseteq T$
 and g : continuous_on $S g g ' S \subseteq T$
 and contractible S path_connected T

shows *homotopic_with_canon* ($\lambda h. \text{True}$) $S \ T \ f \ g$
 using *homotopic_through_contractible* [of $S \ \text{id} \ S \ f \ T \ \text{id} \ g$]
 by (simp add: *assms contractible_imp_path_connected*)

6.18.12 Starlike sets

definition *starlike* $S \longleftrightarrow (\exists a \in S. \forall x \in S. \text{closed\_segment } a \ x \subseteq S)$

lemma *starlike_UNIV* [simp]: *starlike UNIV*
 by (simp add: *starlike_def*)

lemma *convex_imp_starlike*:
 $\text{convex } S \implies S \neq \{\} \implies \text{starlike } S$
 unfolding *convex_contains_segment starlike_def* by auto

lemma *starlike_convex_tweak_boundary_points*:
 fixes $S :: 'a::\text{euclidean\_space}$ set
 assumes $\text{convex } S \ S \neq \{\}$ and $ST: \text{rel\_interior } S \subseteq T$ and $TS: T \subseteq \text{closure } S$
 shows *starlike* T
proof –
 have $\text{rel\_interior } S \neq \{\}$
 by (simp add: *assms rel_interior_eq_empty*)
 then obtain a where $a \in \text{rel\_interior } S$ by blast
 with ST have $a \in T$ by blast
 have $\bigwedge x. x \in T \implies \text{open\_segment } a \ x \subseteq \text{rel\_interior } S$
 by (rule *rel_interior_closure_convex_segment* [OF $\langle \text{convex } S \rangle a$]) (use *assms* in auto)
 then have $\forall x \in T. a \in T \wedge \text{open\_segment } a \ x \subseteq T$
 using ST by (blast intro: $a \in T$) *rel_interior_closure_convex_segment* [OF $\langle \text{convex } S \rangle a$]
 then show ?thesis
 unfolding *starlike_def* using *beI* [OF $\_ \langle a \in T \rangle$]
 by (simp add: *closed_segment_eq_open*)
qed

lemma *starlike_imp_contractible_gen*:
 fixes $S :: 'a::\text{real\_normed\_vector}$ set
 assumes $S: \text{starlike } S$
 and $P: \bigwedge a \ T. \llbracket a \in S; 0 \leq T; T \leq 1 \rrbracket \implies P(\lambda x. (1 - T) *_{\mathbb{R}} x + T *_{\mathbb{R}} a)$
 obtains a where *homotopic_with_canon* $P \ S \ S \ (\lambda x. x) \ (\lambda x. a)$
proof –
 obtain a where $a \in S$ and $a: \bigwedge x. x \in S \implies \text{closed\_segment } a \ x \subseteq S$
 using S by (auto simp: *starlike_def*)
 have $\bigwedge t \ b. 0 \leq t \wedge t \leq 1 \implies$
 $\exists u. (1 - t) *_{\mathbb{R}} b + t *_{\mathbb{R}} a = (1 - u) *_{\mathbb{R}} a + u *_{\mathbb{R}} b \wedge 0 \leq u \wedge u \leq 1$
 by (metis *add_diff_cancel_right' diff_ge_0_iff_ge le_add_diff_inverse pth_c(1)*)
 then have $(\lambda y. (1 - \text{fst } y) *_{\mathbb{R}} \text{snd } y + \text{fst } y *_{\mathbb{R}} a) ' (\{0..1\} \times S) \subseteq S$
 using a [unfolded *closed_segment_def*] by force
 then have *homotopic_with_canon* $P \ S \ S \ (\lambda x. x) \ (\lambda x. a)$

```

    using ‹ $a \in S$ ›
    unfolding homotopic_with_def
    apply (rule_tac x= $\lambda y. (1 - (\text{fst } y)) *_R \text{snd } y + (\text{fst } y) *_R a$  in exI)
    apply (force simp add: P intro: continuous_intros)
    done
  then show ?thesis
    using that by blast
qed

lemma starlike_imp_contractible:
  fixes S :: 'a::real_normed_vector set
  shows starlike S  $\implies$  contractible S
using starlike_imp_contractible_gen contractible_def by (fastforce simp: id_def)

lemma contractible_UNIV [simp]: contractible (UNIV :: 'a::real_normed_vector set)
  by (simp add: starlike_imp_contractible)

lemma starlike_imp_simply_connected:
  fixes S :: 'a::real_normed_vector set
  shows starlike S  $\implies$  simply_connected S
by (simp add: contractible_imp_simply_connected starlike_imp_contractible)

lemma convex_imp_simply_connected:
  fixes S :: 'a::real_normed_vector set
  shows convex S  $\implies$  simply_connected S
using convex_imp_starlike starlike_imp_simply_connected by blast

lemma starlike_imp_path_connected:
  fixes S :: 'a::real_normed_vector set
  shows starlike S  $\implies$  path_connected S
by (simp add: simply_connected_imp_path_connected starlike_imp_simply_connected)

lemma starlike_imp_connected:
  fixes S :: 'a::real_normed_vector set
  shows starlike S  $\implies$  connected S
by (simp add: path_connected_imp_connected starlike_imp_path_connected)

lemma is_interval_simply_connected_1:
  fixes S :: real set
  shows is_interval S  $\longleftrightarrow$  simply_connected S
using convex_imp_simply_connected is_interval_convex_1 is_interval_path_connected_1
simply_connected_imp_path_connected by auto

lemma contractible_empty [simp]: contractible {}
  by (simp add: contractible_def homotopic_on_emptyI)

lemma contractible_convex_tweak_boundary_points:
  fixes S :: 'a::euclidean_space set

```

```

    assumes convex S and TS: rel_interior S ⊆ T T ⊆ closure S
    shows contractible T
  proof (cases S = {})
    case True
    with assms show ?thesis
    by (simp add: subsetCE)
  next
    case False
    show ?thesis
    by (meson False assms starlike_convex_tweak_boundary_points starlike_imp_contractible)
  qed

```

```

lemma convex_imp_contractible:
  fixes S :: 'a::real_normed_vector set
  shows convex S ⟹ contractible S
  using contractible_empty convex_imp_starlike starlike_imp_contractible by blast

```

```

lemma contractible_sing [simp]:
  fixes a :: 'a::real_normed_vector
  shows contractible {a}
  by (rule convex_imp_contractible [OF convex_singleton])

```

```

lemma is_interval_contractible_1:
  fixes S :: real set
  shows is_interval S ⟷ contractible S
  using contractible_imp_simply_connected convex_imp_contractible is_interval_convex_1
    is_interval_simply_connected_1 by auto

```

```

lemma contractible_Times:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S: contractible S and T: contractible T
  shows contractible (S × T)
  proof -
    obtain a h where conth: continuous_on ({0..1} × S) h
      and hsub: h ` ({0..1} × S) ⊆ S
      and [simp]:  $\bigwedge x. x \in S \implies h(0, x) = x$ 
      and [simp]:  $\bigwedge x. x \in S \implies h(1::real, x) = a$ 
    using S by (auto simp: contractible_def homotopic_with)
    obtain b k where contk: continuous_on ({0..1} × T) k
      and ksub: k ` ({0..1} × T) ⊆ T
      and [simp]:  $\bigwedge x. x \in T \implies k(0, x) = x$ 
      and [simp]:  $\bigwedge x. x \in T \implies k(1::real, x) = b$ 
    using T by (auto simp: contractible_def homotopic_with)
    show ?thesis
    apply (simp add: contractible_def homotopic_with)
    apply (rule exI [where x=a])
    apply (rule exI [where x=b])
    apply (rule exI [where x = λz. (h (fst z, fst(snd z)), k (fst z, snd(snd z))])
    using hsub ksub

```



```

apply (fastforce intro!: continuous_intros continuous_on_compose2 [OF conth]
continuous_on_compose2 [OF contk])
done
qed

```

6.18.13 Local versions of topological properties in general

definition *locally* :: ('a::topological_space set $\Rightarrow$ bool) $\Rightarrow$ 'a set $\Rightarrow$ bool

where

```

locally P S  $\equiv$ 
   $\forall w x. \text{openin} (\text{top\_of\_set } S) w \wedge x \in w$ 
   $\longrightarrow (\exists u v. \text{openin} (\text{top\_of\_set } S) u \wedge P v \wedge x \in u \wedge u \subseteq v \wedge v \subseteq w)$ 

```

lemma *locallyI*:

```

assumes  $\bigwedge w x. \llbracket \text{openin} (\text{top\_of\_set } S) w; x \in w \rrbracket$ 
 $\implies \exists u v. \text{openin} (\text{top\_of\_set } S) u \wedge P v \wedge x \in u \wedge u \subseteq v \wedge v \subseteq w$ 
shows locally P S

```

using *assms* **by** (force simp: *locally_def*)

lemma *locallyE*:

```

assumes locally P S  $\text{openin} (\text{top\_of\_set } S) w \wedge x \in w$ 
obtains u v where  $\text{openin} (\text{top\_of\_set } S) u$ 
 $P v \wedge x \in u \wedge u \subseteq v \wedge v \subseteq w$ 

```

using *assms* **unfolding** *locally_def* **by** meson

lemma *locally_mono*:

```

assumes locally P S  $\bigwedge T. P T \implies Q T$ 
shows locally Q S

```

by (metis *assms* *locally_def*)

lemma *locally_open_subset*:

```

assumes locally P S  $\text{openin} (\text{top\_of\_set } S) t$ 
shows locally P t

```

using *assms*

unfolding *locally_def*

by (elim all_forward) (meson dual_order.trans *openin_imp_subset* *openin_subset_trans* *openin_trans*)

lemma *locally_diff_closed*:

```

 $\llbracket \text{locally } P S; \text{closedin} (\text{top\_of\_set } S) t \rrbracket \implies \text{locally } P (S - t)$ 
using locally_open_subset closedin_def by fastforce

```

lemma *locally_empty* [iff]: *locally* P {}

by (simp add: *locally_def* *openin_subtopology*)

lemma *locally_singleton* [iff]:

```

fixes a :: 'a::metric_space
shows locally P {a}  $\longleftrightarrow P \{a\}$ 

```

proof –

```

have  $\forall x::real. \neg 0 < x \implies P \{a\}$ 
  using zero_less_one by blast
then show ?thesis
  unfolding locally_def
  by (auto simp add: openin_euclidean_subtopology_iff subset_singleton_iff
conj_disj_distribR)
qed

```

```

lemma locally_iff:
  locally  $P \ S \longleftrightarrow$ 
    ( $\forall T \ x. \text{open } T \wedge x \in S \cap T \longrightarrow (\exists U. \text{open } U \wedge (\exists V. P \ V \wedge x \in S \cap U \wedge$ 
 $S \cap U \subseteq V \wedge V \subseteq S \cap T)))$ )
  apply (simp add: le_inf_iff locally_def openin_open, safe)
  apply (metis IntE IntI le_inf_iff)
  apply (metis IntI Int_subset_iff)
done

```

```

lemma locally_Int:
  assumes  $S: \text{locally } P \ S$  and  $T: \text{locally } P \ T$ 
  and  $P: \bigwedge S \ T. P \ S \wedge P \ T \implies P(S \cap T)$ 
  shows locally  $P \ (S \cap T)$ 
  unfolding locally_iff
proof clarify
  fix  $A \ x$ 
  assume open  $A \ x \in A \ x \in S \ x \in T$ 
  then obtain  $U1 \ V1 \ U2 \ V2$ 
    where open  $U1 \ P \ V1 \ x \in S \cap U1 \ S \cap U1 \subseteq V1 \wedge V1 \subseteq S \cap A$ 
      open  $U2 \ P \ V2 \ x \in T \cap U2 \ T \cap U2 \subseteq V2 \wedge V2 \subseteq T \cap A$ 
    using  $S \ T$  unfolding locally_iff by (meson IntI)
  then have  $S \cap T \cap (U1 \cap U2) \subseteq V1 \cap V2 \ V1 \cap V2 \subseteq S \cap T \cap A \ x \in S \cap$ 
 $T \cap (U1 \cap U2)$ 
    by blast+
  moreover have  $P \ (V1 \cap V2)$ 
    by (simp add:  $P \ \langle P \ V1 \rangle \ \langle P \ V2 \rangle$ )
  ultimately show  $\exists U. \text{open } U \wedge (\exists V. P \ V \wedge x \in S \cap T \cap U \wedge S \cap T \cap U \subseteq$ 
 $V \wedge V \subseteq S \cap T \cap A)$ 
    using  $\langle \text{open } U1 \rangle \ \langle \text{open } U2 \rangle$  by blast
qed

```

```

lemma locally_Times:
  fixes  $S :: ('a::metric\_space) \text{ set}$  and  $T :: ('b::metric\_space) \text{ set}$ 
  assumes  $PS: \text{locally } P \ S$  and  $QT: \text{locally } Q \ T$  and  $R: \bigwedge S \ T. P \ S \wedge Q \ T \implies$ 
 $R(S \times T)$ 
  shows locally  $R \ (S \times T)$ 
  unfolding locally_def
proof (clarify)
  fix  $W \ x \ y$ 
  assume  $W: \text{openin } (\text{top\_of\_set } (S \times T)) \ W$  and  $xy: (x, y) \in W$ 

```

```

then obtain U V where openin (top_of_set S) U x ∈ U
                        openin (top_of_set T) V y ∈ V U × V ⊆ W
  using Times_in_interior_subtopology by metis
then obtain U1 U2 V1 V2
  where opeS: openin (top_of_set S) U1 ∧ P U2 ∧ x ∈ U1 ∧ U1 ⊆ U2 ∧
U2 ⊆ U
  and opeT: openin (top_of_set T) V1 ∧ Q V2 ∧ y ∈ V1 ∧ V1 ⊆ V2 ∧
V2 ⊆ V
  by (meson PS QT locallyE)
then have openin (top_of_set (S × T)) (U1 × V1)
  by (simp add: openin_Times)
moreover have R (U2 × V2)
  by (simp add: R opeS opeT)
moreover have U1 × V1 ⊆ U2 × V2 ∧ U2 × V2 ⊆ W
  using opeS opeT ⟨U × V ⊆ W⟩ by auto
ultimately show ∃ U V. openin (top_of_set (S × T)) U ∧ R V ∧ (x,y) ∈ U
  ∧ U ⊆ V ∧ V ⊆ W
  using opeS opeT by auto
qed

```

proposition *homeomorphism_locally_imp*:

```

fixes S :: 'a::metric_space set and T :: 'b::t2_space set
assumes S: locally P S and hom: homeomorphism S T f g
  and Q: ∧ S S'. [P S; homeomorphism S S' f g] ⇒ Q S'
  shows locally Q T
proof (clarsimp simp: locally_def)
  fix W y
  assume y ∈ W and openin (top_of_set T) W
  then obtain A where T: open A W = T ∩ A
  by (force simp: openin_open)
  then have W ⊆ T by auto
  have f: ∧ x. x ∈ S ⇒ g(f x) = x f ' S = T continuous_on S f
  and g: ∧ y. y ∈ T ⇒ f(g y) = y g ' T = S continuous_on T g
  using hom by (auto simp: homeomorphism_def)
  have gw: g ' W = S ∩ f - ' W
  using ⟨W ⊆ T⟩ g by force
  have o: openin (top_of_set S) (g ' W)
  proof -
    have continuous_on S f
    using f(3) by blast
    then show openin (top_of_set S) (g ' W)
    by (simp add: gw Collect_conj_eq ⟨openin (top_of_set T) W⟩ continu-
ous_on_open f(2))
  qed
  then obtain U V
  where osu: openin (top_of_set S) U and uv: P V g y ∈ U U ⊆ V V ⊆ g ' W
  using S [unfolded locally_def, rule_format, of g ' W g y] ⟨y ∈ W⟩ by force
  have V ⊆ S using uv by (simp add: gw)

```

```

have fv:  $f' V = T \cap \{x. g x \in V\}$ 
using  $\langle f' S = T \rangle f \langle V \subseteq S \rangle$  by auto
have  $f' V \subseteq W$ 
using uv using Int_lower2 gw image_subsetI mem_Collect_eq subset_iff by
auto
have contvf: continuous_on V f
using  $\langle V \subseteq S \rangle$  continuous_on_subset f(3) by blast
have contvg: continuous_on (f' V) g
using  $\langle f' V \subseteq W \rangle \langle W \subseteq T \rangle$  continuous_on_subset [OF g(3)] by blast
have  $V \subseteq g' f' V$ 
by (metis  $\langle V \subseteq S \rangle$  hom homeomorphism_def homeomorphism_of_subsets
order_refl)
then have homv: homeomorphism V (f' V) f g
using  $\langle V \subseteq S \rangle f$  by (auto simp add: homeomorphism_def contvf contvg)
have openin (top_of_set (g' T)) U
using  $\langle g' T = S \rangle$  by (simp add: osu)
then have 1: openin (top_of_set T) (T  $\cap$  g -' U)
using  $\langle$ continuous_on T g $\rangle$  continuous_on_open [THEN iffD1] by blast
have 2:  $\exists V. Q V \wedge y \in (T \cap g -' U) \wedge (T \cap g -' U) \subseteq V \wedge V \subseteq W$ 
proof (intro exI conjI)
show Q (f' V)
using Q homv  $\langle P V \rangle$  by blast
show  $y \in T \cap g -' U$ 
using T(2)  $\langle y \in W \rangle \langle g y \in U \rangle$  by blast
show  $T \cap g -' U \subseteq f' V$ 
using g(1) image_iff uv(3) by fastforce
show  $f' V \subseteq W$ 
using  $\langle f' V \subseteq W \rangle$  by blast
qed
show  $\exists U. openin (top_of_set T) U \wedge (\exists v. Q v \wedge y \in U \wedge U \subseteq v \wedge v \subseteq W)$ 
by (meson 1 2)
qed

lemma homeomorphism_locally:
fixes f:: 'a::metric_space  $\Rightarrow$  'b::metric_space
assumes hom: homeomorphism S T f g
and eq:  $\bigwedge S T. \text{homeomorphism } S T f g \implies (P S \longleftrightarrow Q T)$ 
shows locally P S  $\longleftrightarrow$  locally Q T
(is ?lhs = ?rhs)
proof
assume ?lhs
then show ?rhs
using eq hom homeomorphism_locally_imp by blast
next
assume ?rhs
then show ?lhs
using eq homeomorphism_sym homeomorphism_symD [OF hom]
by (blast intro: homeomorphism_locally_imp)
qed

```

```

lemma homeomorphic_locally:
  fixes S:: 'a::metric_space set and T:: 'b::metric_space set
  assumes hom: S homeomorphic T
    and iff:  $\bigwedge X Y. X \text{ homeomorphic } Y \implies (P X \longleftrightarrow Q Y)$ 
  shows locally P S  $\longleftrightarrow$  locally Q T
proof -
  obtain f g where hom: homeomorphism S T f g
    using assms by (force simp: homeomorphic_def)
  then show ?thesis
    using homeomorphic_def local.iff
    by (blast intro!: homeomorphism_locally)
qed

lemma homeomorphic_local_compactness:
  fixes S:: 'a::metric_space set and T:: 'b::metric_space set
  shows S homeomorphic T  $\implies$  locally compact S  $\longleftrightarrow$  locally compact T
by (simp add: homeomorphic_compactness homeomorphism_locally)

lemma locally_translation:
  fixes P :: 'a :: real_normed_vector set  $\Rightarrow$  bool
  shows ( $\bigwedge S. P ((+) a \text{ ` } S) = P S$ )  $\implies$  locally P ((+) a \text{ ` } S) = locally P S
  using homeomorphism_locally [OF homeomorphism_translation]
  by (metis (full_types) homeomorphism_image2)

lemma locally_injective_linear_image:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes f: linear f inj f and iff:  $\bigwedge S. P (f \text{ ` } S) \longleftrightarrow Q S$ 
  shows locally P (f \text{ ` } S)  $\longleftrightarrow$  locally Q S
  using homeomorphism_locally [of f \text{ ` } S _ f] linear_homeomorphism_image [OF f]
  by (metis (no_types, lifting) homeomorphism_image2 iff)

lemma locally_open_map_image:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes P: locally P S
    and f: continuous_on S f
    and oo:  $\bigwedge T. \text{openin } (\text{top\_of\_set } S) T \implies \text{openin } (\text{top\_of\_set } (f \text{ ` } S)) (f \text{ ` } T)$ 
    and Q:  $\bigwedge T. \llbracket T \subseteq S; P T \rrbracket \implies Q(f \text{ ` } T)$ 
  shows locally Q (f \text{ ` } S)
proof (clarsimp simp add: locally_def)
  fix W y
  assume oiw: openin (top_of_set (f \text{ ` } S)) W and y  $\in$  W
  then have W  $\subseteq$  f \text{ ` } S by (simp add: openin_euclidean_subtopology_iff)
  have oiwf: openin (top_of_set S) (S  $\cap$  f \text{ ` } W)
    by (rule continuous_on_open [THEN iffD1, rule_format, OF f oiw])
  then obtain x where x  $\in$  S f x = y
    using  $\langle W \subseteq f \text{ ` } S \rangle \langle y \in W \rangle$  by blast

```

```

then obtain  $U\ V$ 
  where  $\text{openin } (\text{top\_of\_set } S)\ U\ P\ V\ x \in U\ U \subseteq V\ V \subseteq S \cap f - ' W$ 
  using  $P\ [\text{unfolded\_locally\_def}, \text{rule\_format}, \text{of } (S \cap f - ' W)\ x]\ \text{oivf } \langle y \in W \rangle$ 
  by auto
then have  $\text{openin } (\text{top\_of\_set } (f - ' S))\ (f - ' U)$ 
  by (simp add: oo)
then show  $\exists X. \text{openin } (\text{top\_of\_set } (f - ' S))\ X \wedge (\exists Y. Q\ Y \wedge y \in X \wedge X \subseteq Y$ 
 $\wedge Y \subseteq W)$ 
  using  $Q\ \langle P\ V \rangle\ \langle U \subseteq V \rangle\ \langle V \subseteq S \cap f - ' W \rangle\ \langle f\ x = y \rangle\ \langle x \in U \rangle$  by blast
qed

```

6.18.14 An induction principle for connected sets

proposition *connected_induction:*

```

assumes connected S
  and opD:  $\bigwedge T\ a. \llbracket \text{openin } (\text{top\_of\_set } S)\ T; a \in T \rrbracket \implies \exists z. z \in T \wedge P\ z$ 
  and opI:  $\bigwedge a. a \in S$ 
            $\implies \exists T. \text{openin } (\text{top\_of\_set } S)\ T \wedge a \in T \wedge$ 
            $(\forall x \in T. \forall y \in T. P\ x \wedge P\ y \wedge Q\ x \longrightarrow Q\ y)$ 
  and etc:  $a \in S\ b \in S\ P\ a\ P\ b\ Q\ a$ 
shows  $Q\ b$ 

```

proof –

```

let  $?A = \{b. \exists T. \text{openin } (\text{top\_of\_set } S)\ T \wedge b \in T \wedge (\forall x \in T. P\ x \longrightarrow Q\ x)\}$ 
let  $?B = \{b. \exists T. \text{openin } (\text{top\_of\_set } S)\ T \wedge b \in T \wedge (\forall x \in T. P\ x \longrightarrow \neg Q\ x)\}$ 
have 1:  $\text{openin } (\text{top\_of\_set } S)\ ?A$ 
  by (subst openin_subopen, blast)
have 2:  $\text{openin } (\text{top\_of\_set } S)\ ?B$ 
  by (subst openin_subopen, blast)
have §:  $?A \cap ?B = \{\}$ 
  by (clarsimp simp: set_eq_iff) (metis (no_types, opaque_lifting) Int_iff opD openin_Int)
have *:  $S \subseteq ?A \cup ?B$ 
  by clarsimp (meson opI)
have  $?A = \{\} \vee ?B = \{\}$ 
  using  $\langle \text{connected } S \rangle\ [\text{unfolded connected\_openin, simplified, rule\_format, OF$ 
  1 § * 2]
  by blast
then show ?thesis
  by clarsimp (meson opI etc)
qed

```

lemma *connected_equivalence_relation_gen:*

```

assumes connected S
  and etc:  $a \in S\ b \in S\ P\ a\ P\ b$ 
  and trans:  $\bigwedge x\ y\ z. \llbracket R\ x\ y; R\ y\ z \rrbracket \implies R\ x\ z$ 
  and opD:  $\bigwedge T\ a. \llbracket \text{openin } (\text{top\_of\_set } S)\ T; a \in T \rrbracket \implies \exists z. z \in T \wedge P\ z$ 
  and opI:  $\bigwedge a. a \in S$ 
            $\implies \exists T. \text{openin } (\text{top\_of\_set } S)\ T \wedge a \in T \wedge$ 
            $(\forall x \in T. \forall y \in T. P\ x \wedge P\ y \longrightarrow R\ x\ y)$ 

```

```

    shows  $R\ a\ b$ 
  proof -
    have  $\bigwedge a\ b\ c. \llbracket a \in S; P\ a; b \in S; c \in S; P\ b; P\ c; R\ a\ b \rrbracket \implies R\ a\ c$ 
      apply (rule connected_induction [OF ‹connected S› opD], simp_all)
      by (meson trans opI)
    then show ?thesis by (metis etc opI)
  qed

lemma connected_induction_simple:
  assumes connected S
    and etc:  $a \in S\ b \in S\ P\ a$ 
    and opI:  $\bigwedge a. a \in S \implies \exists T. \text{openin } (\text{top\_of\_set } S)\ T \wedge a \in T \wedge$ 
       $(\forall x \in T. \forall y \in T. P\ x \longrightarrow P\ y)$ 
  shows  $P\ b$ 
  by (rule connected_induction [OF ‹connected S› __, where  $P = \lambda x. \text{True}$ ])
    (use opI etc in auto)

lemma connected_equivalence_relation:
  assumes connected S
    and etc:  $a \in S\ b \in S$ 
    and sym:  $\bigwedge x\ y. \llbracket R\ x\ y; x \in S; y \in S \rrbracket \implies R\ y\ x$ 
    and trans:  $\bigwedge x\ y\ z. \llbracket R\ x\ y; R\ y\ z; x \in S; y \in S; z \in S \rrbracket \implies R\ x\ z$ 
    and opI:  $\bigwedge a. a \in S \implies \exists T. \text{openin } (\text{top\_of\_set } S)\ T \wedge a \in T \wedge (\forall x \in T.$ 
 $R\ a\ x)$ 
  shows  $R\ a\ b$ 
  proof -
    have  $\bigwedge a\ b\ c. \llbracket a \in S; b \in S; c \in S; R\ a\ b \rrbracket \implies R\ a\ c$ 
      apply (rule connected_induction_simple [OF ‹connected S›], simp_all)
      by (meson local.sym local.trans opI openin_imp_subset subsetCE)
    then show ?thesis by (metis etc opI)
  qed

lemma locally_constant_imp_constant:
  assumes connected S
    and opI:  $\bigwedge a. a \in S \implies \exists T. \text{openin } (\text{top\_of\_set } S)\ T \wedge a \in T \wedge (\forall x \in T. f\ x = f\ a)$ 
  shows  $f\ \text{constant\_on } S$ 
  proof -
    have  $\bigwedge x\ y. x \in S \implies y \in S \implies f\ x = f\ y$ 
      apply (rule connected_equivalence_relation [OF ‹connected S›], simp_all)
      by (metis opI)
    then show ?thesis
      by (metis constant_on_def)
  qed

lemma locally_constant:
  assumes connected S
  shows  $\text{locally } (\lambda U. f\ \text{constant\_on } U)\ S \longleftrightarrow f\ \text{constant\_on } S$  (is ?lhs = ?rhs)

```

```

proof
  assume ?lhs
  then have  $\bigwedge a. a \in S \implies \exists T. \text{openin } (\text{top\_of\_set } S) \ T \wedge a \in T \wedge (\forall x \in T. f$ 
 $x = f a)$ 
    unfolding locally_def
    by (metis (mono_tags, opaque_lifting) constant_on_def constant_on_subset
openin_subtopology_self)
  then show ?rhs
    using assms
    by (simp add: locally_constant_imp_constant)
next
  assume ?rhs then show ?lhs
    using assms by (metis constant_on_subset locallyI openin_imp_subset or-
der_refl)
qed

```

6.18.15 Basic properties of local compactness

proposition *locally_compact*:

fixes $s :: 'a :: \text{metric\_space}$ *set*

shows

$\text{locally\_compact } s \longleftrightarrow$
 $(\forall x \in s. \exists u \ v. x \in u \wedge u \subseteq v \wedge v \subseteq s \wedge$
 $\text{openin } (\text{top\_of\_set } s) \ u \wedge \text{compact } v)$
 (is ?lhs = ?rhs)

proof

assume ?lhs

then show ?rhs

by (meson locallyE openin_subtopology_self)

next

assume r [rule_format]: ?rhs

have *: $\exists u \ v.$

$\text{openin } (\text{top\_of\_set } s) \ u \wedge$
 $\text{compact } v \wedge x \in u \wedge u \subseteq v \wedge v \subseteq s \cap T$

if $\text{open } T \ x \in s \ x \in T$ **for** $x \ T$

proof –

obtain $u \ v$ **where** $uv: x \in u \ u \subseteq v \ v \subseteq s \ \text{compact } v \ \text{openin } (\text{top\_of\_set } s) \ u$

using r [OF $\langle x \in s \rangle$] **by** auto

obtain e **where** $e > 0$ **and** $e: \text{cball } x \ e \subseteq T$

using open_contains_cball $\langle \text{open } T \rangle \langle x \in T \rangle$ **by** blast

show ?thesis

apply (rule_tac $x = (s \cap \text{ball } x \ e) \cap u$ **in** exI)

apply (rule_tac $x = \text{cball } x \ e \cap v$ **in** exI)

using that $\langle e > 0 \rangle \ e \ uv$

apply auto

done

qed

show ?lhs

by (rule locallyI) (metis * Int_iff openin_open)

qed

lemma *locally_compactE*:

fixes $S :: 'a :: \text{metric\_space set}$

assumes *locally_compact S*

obtains $u\ v$ **where** $\bigwedge x. x \in S \implies x \in u\ x \wedge u\ x \subseteq v\ x \wedge v\ x \subseteq S \wedge$
 $\text{openin } (\text{top\_of\_set } S) (u\ x) \wedge \text{compact } (v\ x)$

using *assms unfolding locally_compact by metis*

lemma *locally_compact_alt*:

fixes $S :: 'a :: \text{heine\_borel set}$

shows *locally_compact S* $\longleftrightarrow$

$(\forall x \in S. \exists U. x \in U \wedge$

$\text{openin } (\text{top\_of\_set } S) U \wedge \text{compact}(\text{closure } U) \wedge \text{closure } U \subseteq S)$

(**is** *?lhs = ?rhs*)

proof

assume *?lhs*

then show *?rhs*

by (*meson bounded_subset closure_minimal compact_closure compact_imp_bounded*

compact_imp_closed dual_order.trans locally_compactE)

next

assume *?rhs* **then show** *?lhs*

by (*meson closure_subset locally_compact*)

qed

lemma *locally_compact_Int_cball*:

fixes $S :: 'a :: \text{heine\_borel set}$

shows *locally_compact S* $\longleftrightarrow (\forall x \in S. \exists e. 0 < e \wedge \text{closed}(\text{cball } x\ e \cap S))$

(**is** *?lhs = ?rhs*)

proof

assume *L: ?lhs*

then have $\bigwedge x\ U\ V\ e. [U \subseteq V; V \subseteq S; \text{compact } V; 0 < e; \text{cball } x\ e \cap S \subseteq U]$
 $\implies \text{closed } (\text{cball } x\ e \cap S)$

by (*metis compact_Int compact_cball compact_imp_closed inf.absorb_iff2*
inf.assoc inf.orderE)

with *L* **show** *?rhs*

by (*meson locally_compactE openin_contains_cball*)

next

assume *R: ?rhs*

show *?lhs* **unfolding** *locally_compact*

proof

fix x

assume $x \in S$

then obtain e **where** $e > 0$ **and** $e: \text{closed } (\text{cball } x\ e \cap S)$

using *R* **by** *blast*

then have *compact (cball x e ∩ S)*

by (*simp add: bounded_Int compact_eq_bounded_closed*)

moreover have $\forall y \in \text{ball } x\ e \cap S. \exists \varepsilon > 0. \text{cball } y\ \varepsilon \cap S \subseteq \text{ball } x\ e$

```

    by (meson Elementary_Metric_Spaces.open_ball IntD1 le_infI1 open_contains_cball_eq)
  moreover have openin (top_of_set S) (ball x e ∩ S)
    by (simp add: inf_commute openin_open_Int)
  ultimately show ∃ U V. x ∈ U ∧ U ⊆ V ∧ V ⊆ S ∧ openin (top_of_set S)
    U ∧ compact V
    by (metis Int_iff ⟨0 < e⟩ ⟨x ∈ S⟩ ball_subset_cball centre_in_ball inf_commute
inf_le1 inf_mono order_refl)
qed
qed

```

lemma *locally_compact_compact*:

fixes $S :: 'a :: \text{heine\_borel\_set}$

shows $\text{locally\_compact } S \longleftrightarrow$

$(\forall K. K \subseteq S \wedge \text{compact } K$
 $\longrightarrow (\exists U V. K \subseteq U \wedge U \subseteq V \wedge V \subseteq S \wedge$
 $\text{openin } (\text{top\_of\_set } S) U \wedge \text{compact } V))$
 $(\text{is } ?lhs = ?rhs)$

proof

assume $?lhs$

then obtain $u v$ **where**

$uv: \bigwedge x. x \in S \implies x \in u \wedge u \subseteq v \wedge v \subseteq S \wedge$
 $\text{openin } (\text{top\_of\_set } S) (u \cap v) \wedge \text{compact } (v \cap S)$

by (metis *locally_compactE*)

have $*$: $\exists U V. K \subseteq U \wedge U \subseteq V \wedge V \subseteq S \wedge \text{openin } (\text{top\_of\_set } S) U \wedge$
 $\text{compact } V$

if $K \subseteq S$ **compact** K **for** K

proof –

have $\bigwedge C. (\forall c \in C. \text{openin } (\text{top\_of\_set } K) c) \wedge K \subseteq \bigcup C \implies$
 $\exists D \subseteq C. \text{finite } D \wedge K \subseteq \bigcup D$

using that **by** (simp add: *compact_eq_openin_cover*)

moreover have $\forall c \in (\lambda x. K \cap u \cap x) \text{ ‘ } K. \text{openin } (\text{top\_of\_set } K) c$

using that **by** *clarify* (metis *subsetD inf.absorb_iff2 openin_subset openin_subtopology_Int_subset*
topspace_euclidean_subtopology uv)

moreover have $K \subseteq \bigcup ((\lambda x. K \cap u \cap x) \text{ ‘ } K)$

using that **by** *clarsimp* (meson *subsetCE uv*)

ultimately obtain D **where** $D \subseteq (\lambda x. K \cap u \cap x) \text{ ‘ } K$ **finite** D $K \subseteq \bigcup D$

by *metis*

then obtain T **where** $T: T \subseteq K$ **finite** T $K \subseteq \bigcup ((\lambda x. K \cap u \cap x) \text{ ‘ } T)$

by (metis *finite_subset_image*)

have $Tuv: \bigcup (u \text{ ‘ } T) \subseteq \bigcup (v \text{ ‘ } T)$

using T **that** **by** (force *dest!*: uv)

moreover

have $\text{openin } (\text{top\_of\_set } S) (\bigcup (u \text{ ‘ } T))$

using T **that** uv **by** *fastforce*

moreover

have $\text{compact } (\bigcup (v \text{ ‘ } T))$

by (meson T *compact_UN_subset_eq that*(1) uv)

moreover have $\bigcup (v \text{ ‘ } T) \subseteq S$

by (metis *SUP_least T*(1) *subset_eq that*(1) uv)

```

      ultimately show ?thesis
      using T by auto
    qed
  show ?rhs
  by (blast intro: *)
next
  assume ?rhs
  then show ?lhs
  apply (clarsimp simp add: locally_compact)
  apply (drule_tac x={x} in spec, simp)
  done
qed

lemma open_imp_locally_compact:
  fixes S :: 'a :: heine_borel set
  assumes open S
  shows locally_compact S
proof -
  have *:  $\exists U V. x \in U \wedge U \subseteq V \wedge V \subseteq S \wedge \text{openin } (\text{top\_of\_set } S) U \wedge \text{compact } V$ 
  if  $x \in S$  for  $x$ 
proof -
  obtain e where  $e > 0$  and  $e: \text{cball } x e \subseteq S$ 
  using open_contains_cball assms  $\langle x \in S \rangle$  by blast
  have ope:  $\text{openin } (\text{top\_of\_set } S) (\text{ball } x e)$ 
  by (meson e open_ball ball_subset_cball dual_order.trans open_subset)
  show ?thesis
  proof (intro exI conjI)
    let ?U = ball x e
    let ?V = cball x e
    show  $x \in ?U$   $?U \subseteq ?V$   $?V \subseteq S$  compact ?V
    using  $\langle e > 0 \rangle$  e by auto
    show  $\text{openin } (\text{top\_of\_set } S) ?U$ 
    using ope by blast
  qed
qed
show ?thesis
unfolding locally_compact by (blast intro: *)
qed

lemma closed_imp_locally_compact:
  fixes S :: 'a :: heine_borel set
  assumes closed S
  shows locally_compact S
proof -
  have *:  $\exists U V. x \in U \wedge U \subseteq V \wedge V \subseteq S \wedge \text{openin } (\text{top\_of\_set } S) U \wedge \text{compact } V$ 
  if  $x \in S$  for  $x$ 
  apply (rule_tac  $x = S \cap \text{ball } x 1$  in exI, rule_tac  $x = S \cap \text{cball } x 1$  in exI)

```

```

    using ⟨ $x \in S$ ⟩ assms by auto
  show ?thesis
    unfolding locally_compact by (blast intro: *)
qed

lemma locally_compact_UNIV: locally_compact (UNIV :: 'a :: heine_borel set)
  by (simp add: closed_imp_locally_compact)

lemma locally_compact_Int:
  fixes  $S :: 'a :: t2\_space$  set
  shows  $\llbracket \text{locally\_compact } S; \text{locally\_compact } t \rrbracket \implies \text{locally\_compact } (S \cap t)$ 
  by (simp add: compact_Int locally_Int)

lemma locally_compact_closedin:
  fixes  $S :: 'a :: \text{heine\_borel}$  set
  shows  $\llbracket \text{closedin } (\text{top\_of\_set } S) \ t; \text{locally\_compact } S \rrbracket$ 
     $\implies \text{locally\_compact } t$ 
  unfolding closedin_closed
  using closed_imp_locally_compact locally_compact_Int by blast

lemma locally_compact_delete:
  fixes  $S :: 'a :: t1\_space$  set
  shows  $\text{locally\_compact } S \implies \text{locally\_compact } (S - \{a\})$ 
  by (auto simp: openin_delete locally_open_subset)

lemma locally_closed:
  fixes  $S :: 'a :: \text{heine\_borel}$  set
  shows  $\text{locally\_closed } S \longleftrightarrow \text{locally\_compact } S$ 
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    unfolding locally_def
    apply (elim_all_forward_imp_forward asm_rl exE)
    apply (rule_tac  $x = u \cap \text{ball } x \ 1$  in exI)
    apply (rule_tac  $x = v \cap \text{cball } x \ 1$  in exI)
    apply (force intro: openin_trans)
  done
next
  assume ?rhs then show ?lhs
    using compact_eq_bounded_closed locally_mono by blast
qed

lemma locally_compact_openin_Un:
  fixes  $S :: 'a :: \text{euclidean\_space}$  set
  assumes LCS: locally compact  $S$  and LCT: locally compact  $T$ 
    and opS: openin (top_of_set ( $S \cup T$ ))  $S$ 
    and opT: openin (top_of_set ( $S \cup T$ ))  $T$ 
  shows locally compact ( $S \cup T$ )

```

```

proof -
  have  $\exists e > 0. \text{closed } (\text{cball } x \ e \cap (S \cup T))$  if  $x \in S$  for  $x$ 
  proof -
    obtain  $e1$  where  $e1 > 0$  and  $e1: \text{closed } (\text{cball } x \ e1 \cap S)$ 
      using  $LCS \langle x \in S \rangle$  unfolding  $\text{locally\_compact\_Int\_cball}$  by blast
    moreover obtain  $e2$  where  $e2 > 0$  and  $e2: \text{cball } x \ e2 \cap (S \cup T) \subseteq S$ 
      by ( $\text{meson } \langle x \in S \rangle \text{ opS openin\_contains\_cball}$ )
    then have  $\text{cball } x \ e2 \cap (S \cup T) = \text{cball } x \ e2 \cap S$ 
      by force
    ultimately have  $\text{closed } (\text{cball } x \ (\min \ e1 \ e2) \cap (S \cup T))$ 
      by ( $\text{metis } (\text{no\_types}, \text{lifting}) \text{ cball\_min\_Int closed\_Int closed\_cball inf\_assoc}$ 
 $\text{inf\_commute}$ )
    then show ?thesis
      by ( $\text{metis } \langle 0 < e1 \rangle \langle 0 < e2 \rangle \text{ min\_def}$ )
  qed
  moreover have  $\exists e > 0. \text{closed } (\text{cball } x \ e \cap (S \cup T))$  if  $x \in T$  for  $x$ 
  proof -
    obtain  $e1$  where  $e1 > 0$  and  $e1: \text{closed } (\text{cball } x \ e1 \cap T)$ 
      using  $LCT \langle x \in T \rangle$  unfolding  $\text{locally\_compact\_Int\_cball}$  by blast
    moreover obtain  $e2$  where  $e2 > 0$  and  $e2: \text{cball } x \ e2 \cap (S \cup T) \subseteq T$ 
      by ( $\text{meson } \langle x \in T \rangle \text{ opT openin\_contains\_cball}$ )
    then have  $\text{cball } x \ e2 \cap (S \cup T) = \text{cball } x \ e2 \cap T$ 
      by force
    moreover have  $\text{closed } (\text{cball } x \ e1 \cap (\text{cball } x \ e2 \cap T))$ 
      by ( $\text{metis closed\_Int closed\_cball } e1 \text{ inf\_left\_commute}$ )
    ultimately show ?thesis
      by ( $\text{rule\_tac } x = \min \ e1 \ e2 \text{ in exI}$ ) ( $\text{simp add: } \langle 0 < e2 \rangle \text{ cball\_min\_Int}$ 
 $\text{inf\_assoc}$ )
  qed
  ultimately show ?thesis
    by ( $\text{force simp: locally\_compact\_Int\_cball}$ )
  qed

lemma  $\text{locally\_compact\_closedin\_Un}$ :
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes  $LCS: \text{locally compact } S$  and  $LCT: \text{locally compact } T$ 
    and  $clS: \text{closedin } (\text{top\_of\_set } (S \cup T)) \ S$ 
    and  $clT: \text{closedin } (\text{top\_of\_set } (S \cup T)) \ T$ 
  shows  $\text{locally compact } (S \cup T)$ 
proof -
  have  $\exists e > 0. \text{closed } (\text{cball } x \ e \cap (S \cup T))$  if  $x \in S$   $x \in T$  for  $x$ 
  proof -
    obtain  $e1$  where  $e1 > 0$  and  $e1: \text{closed } (\text{cball } x \ e1 \cap S)$ 
      using  $LCS \langle x \in S \rangle$  unfolding  $\text{locally\_compact\_Int\_cball}$  by blast
    moreover
    obtain  $e2$  where  $e2 > 0$  and  $e2: \text{closed } (\text{cball } x \ e2 \cap T)$ 
      using  $LCT \langle x \in T \rangle$  unfolding  $\text{locally\_compact\_Int\_cball}$  by blast
    moreover have  $\text{closed } (\text{cball } x \ (\min \ e1 \ e2) \cap (S \cup T))$ 
      proof -

```

```

    have closed (cball x e1  $\cap$  (cball x e2  $\cap$  S))
      by (metis closed_Int closed_cball e1 inf_left_commute)
    then show ?thesis
      by (simp add: Int_Un_distrib cball_min_Int closed_Int closed_Un e2
inf_assoc)
  qed
  ultimately show ?thesis
    by (rule_tac x=min e1 e2 in exI) linarith
  qed
  moreover
  have  $\exists e > 0. \text{closed } (\text{cball } x \ e \cap (S \cup T))$  if  $x: x \in S \ x \notin T$  for  $x$ 
  proof -
    obtain e1 where e1 > 0 and e1: closed (cball x e1  $\cap$  S)
      using LCS  $\langle x \in S \rangle$  unfolding locally_compact_Int_cball by blast
    moreover
    obtain e2 where e2 > 0 and cball x e2  $\cap$  (S  $\cup$  T)  $\subseteq$  S - T
      using clT x by (fastforce simp: openin_contains_cball closedin_def)
    then have closed (cball x e2  $\cap$  T)
    proof -
      have {} = T - (T - cball x e2)
        using Diff_subset Int_Diff  $\langle \text{cball } x \ e2 \cap (S \cup T) \subseteq S - T \rangle$  by auto
      then show ?thesis
        by (simp add: Diff_Diff_Int inf_commute)
    qed
    with e1 have closed ((cball x e1  $\cap$  cball x e2)  $\cap$  (S  $\cup$  T))
      apply (simp add: inf_commute inf_sup_distrib2)
      by (metis closed_Int closed_Un closed_cball inf_assoc inf_left_commute)
    then have closed (cball x (min e1 e2)  $\cap$  (S  $\cup$  T))
      by (simp add: cball_min_Int inf_commute)
    ultimately show ?thesis
      using  $\langle 0 < e2 \rangle$  by (rule_tac x=min e1 e2 in exI) linarith
  qed
  moreover
  have  $\exists e > 0. \text{closed } (\text{cball } x \ e \cap (S \cup T))$  if  $x: x \notin S \ x \in T$  for  $x$ 
  proof -
    obtain e1 where e1 > 0 and e1: closed (cball x e1  $\cap$  T)
      using LCT  $\langle x \in T \rangle$  unfolding locally_compact_Int_cball by blast
    moreover
    obtain e2 where e2 > 0 and cball x e2  $\cap$  (S  $\cup$  T)  $\subseteq$  S  $\cup$  T - S
      using clS x by (fastforce simp: openin_contains_cball closedin_def)
    then have closed (cball x e2  $\cap$  S)
      by (metis Diff_disjoint Int_empty_right closed_empty inf.left_commute
inf.orderE inf_sup_absorb)
    with e1 have closed ((cball x e1  $\cap$  cball x e2)  $\cap$  (S  $\cup$  T))
      apply (simp add: inf_commute inf_sup_distrib2)
      by (metis closed_Int closed_Un closed_cball inf_assoc inf_left_commute)
    then have closed (cball x (min e1 e2)  $\cap$  (S  $\cup$  T))
      by (auto simp: cball_min_Int)
    ultimately show ?thesis

```

```

    using ⟨0 < e2⟩ by (rule_tac x=min e1 e2 in exI) linarith
  qed
  ultimately show ?thesis
  by (auto simp: locally_compact_Int_cball)
qed

lemma locally_compact_Times:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  shows ⟨locally_compact S; locally_compact T⟩ ⟹ locally_compact (S × T)
  by (auto simp: compact_Times locally_Times)

lemma locally_compact_compact_subopen:
  fixes S :: 'a :: heine_borel set
  shows
    locally_compact S ⟷
      (∀ K T. K ⊆ S ∧ compact K ∧ open T ∧ K ⊆ T
        ⟶ (∃ U V. K ⊆ U ∧ U ⊆ V ∧ U ⊆ T ∧ V ⊆ S ∧
          openin (top_of_set S) U ∧ compact V))
    (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof clarify
    fix K :: 'a set and T :: 'a set
    assume K ⊆ S and compact K and open T and K ⊆ T
    obtain U V where K ⊆ U ∧ U ⊆ V ∧ V ⊆ S and compact V
      and ope: openin (top_of_set S) U
    using L unfolding locally_compact_compact by (meson ⟨K ⊆ S⟩ ⟨compact
K⟩)
    show ∃ U V. K ⊆ U ∧ U ⊆ V ∧ U ⊆ T ∧ V ⊆ S ∧
      openin (top_of_set S) U ∧ compact V
  proof (intro exI conjI)
    show K ⊆ U ∩ T
    by (simp add: ⟨K ⊆ T⟩ ⟨K ⊆ U⟩)
    show U ∩ T ⊆ closure(U ∩ T)
    by (rule closure_subset)
    show closure(U ∩ T) ⊆ S
    by (metis ⟨U ⊆ V⟩ ⟨V ⊆ S⟩ ⟨compact V⟩ closure_closed closure_mono
compact_imp_closed inf.cobounded1 subset_trans)
    show openin (top_of_set S) (U ∩ T)
    by (simp add: ⟨open T⟩ ope openin_Int_open)
    show compact (closure(U ∩ T))
    by (meson Int_lower1 ⟨U ⊆ V⟩ ⟨compact V⟩ bounded_subset com-
pact_closure compact_eq_bounded_closed)
  qed auto
  qed
next
  assume ?rhs then show ?lhs
  unfolding locally_compact_compact

```

```

proposition Sura_Bura_compact:
  fixes  $S :: 'a::euclidean\_space\ set$ 
  assumes compact S and C:  $C \in components\ S$ 
  shows  $C = \bigcap \{T. C \subseteq T \wedge openin\ (top\_of\_set\ S)\ T \wedge$ 
     $closedin\ (top\_of\_set\ S)\ T\}$ 
    (is  $C = \bigcap\ ?\mathcal{T}$ )

proof
  obtain  $x$  where  $x: C = connected\_component\_set\ S\ x$  and  $x \in S$ 
  using C by (auto simp: components_def)
  have  $C \subseteq S$ 
  by (simp add: C in_components_subset)
  have  $\bigcap\ ?\mathcal{T} \subseteq connected\_component\_set\ S\ x$ 
  proof (rule connected_component_maximal)
  have  $x \in C$ 
  by (simp add:  $\langle x \in S \rangle x$ )
  then show  $x \in \bigcap\ ?\mathcal{T}$ 
  by blast
  have clo: closed  $(\bigcap\ ?\mathcal{T})$ 
  by (simp add:  $\langle compact\ S \rangle closed\_Inter\ closedin\_compact\_eq\ compact\_imp\_closed$ )
  have False
  if K1: closedin  $(top\_of\_set\ (\bigcap\ ?\mathcal{T}))\ K1$  and
    K2: closedin  $(top\_of\_set\ (\bigcap\ ?\mathcal{T}))\ K2$  and
    K12_Int:  $K1 \cap K2 = \{\}$  and K12_Un:  $K1 \cup K2 = \bigcap\ ?\mathcal{T}$  and  $K1 \neq \{\}$ 
    K2  $\neq \{\}$ 
  for  $K1\ K2$ 
  proof -
  have closed K1 closed K2
  using closedin_closed_trans clo K1 K2 by blast+
  then obtain  $V1\ V2$  where open V1 open V2  $K1 \subseteq V1\ K2 \subseteq V2$  and V12:
     $V1 \cap V2 = \{\}$ 
  using separation_normal  $\langle K1 \cap K2 = \{\} \rangle$  bymetis
  have SV12_ne:  $(S - (V1 \cup V2)) \cap (\bigcap\ ?\mathcal{T}) \neq \{\}$ 
  proof (rule compact_imp_fip)
  show compact  $(S - (V1 \cup V2))$ 
  by (simp add:  $\langle open\ V1 \rangle \langle open\ V2 \rangle \langle compact\ S \rangle compact\_diff\ open\_Un$ )
  show clo $\mathcal{T}$ : closed T if  $T \in ?\mathcal{T}$  for T
  using that  $\langle compact\ S \rangle$ 
  by (force intro: closedin_closed_trans simp add: compact_imp_closed)
  show  $(S - (V1 \cup V2)) \cap \bigcap\ \mathcal{F} \neq \{\}$  if finite  $\mathcal{F}$  and  $\mathcal{F}$ :  $\mathcal{F} \subseteq ?\mathcal{T}$  for  $\mathcal{F}$ 
  proof
  assume djo:  $(S - (V1 \cup V2)) \cap \bigcap\ \mathcal{F} = \{\}$ 
  obtain  $D$  where opeD: openin  $(top\_of\_set\ S)\ D$ 

```



```

        and cloD: closedin (top_of_set S) D
        and C ⊆ D and DV12: D ⊆ V1 ∪ V2
    proof (cases  $\mathcal{F} = \{\}$ )
    case True
    with  $\langle C \subseteq S \rangle$  djo that show ?thesis
    by force
    next
    case False show ?thesis
    proof
    show ope: openin (top_of_set S) ( $\bigcap \mathcal{F}$ )
    using openin_Inter  $\langle \text{finite } \mathcal{F} \rangle$  False  $\mathcal{F}$  by blast
    then show closedin (top_of_set S) ( $\bigcap \mathcal{F}$ )
    by (meson cloT  $\mathcal{F}$  closed_Inter closed_subset openin_imp_subset
subset_eq)
    show  $C \subseteq \bigcap \mathcal{F}$ 
    using  $\mathcal{F}$  by auto
    show  $\bigcap \mathcal{F} \subseteq V1 \cup V2$ 
    using ope djo openin_imp_subset by fastforce
    qed
    qed
    have connected C
    by (simp add: x)
    have closed D
    using  $\langle \text{compact } S \rangle$  cloD closedin_closed_trans compact_imp_closed by
blast
    have cloV1: closedin (top_of_set D) (D ∩ closure V1)
    and cloV2: closedin (top_of_set D) (D ∩ closure V2)
    by (simp_all add: closedin_closed_Int)
    moreover have D ∩ closure V1 = D ∩ V1 D ∩ closure V2 = D ∩ V2
    using  $\langle D \subseteq V1 \cup V2 \rangle$   $\langle \text{open } V1 \rangle$   $\langle \text{open } V2 \rangle$  V12
    by (auto simp add: closure_subset [THEN subsetD] closure_iff_nhds_not_empty,
blast+)
    ultimately have cloDV1: closedin (top_of_set D) (D ∩ V1)
    and cloDV2: closedin (top_of_set D) (D ∩ V2)
    by metis+
    then obtain U1 U2 where closed U1 closed U2
    and D1: D ∩ V1 = D ∩ U1 and D2: D ∩ V2 = D ∩ U2
    by (auto simp: closedin_closed)
    have D ∩ U1 ∩ C ≠ {}
    proof
    assume D ∩ U1 ∩ C = {}
    then have *: C ⊆ D ∩ V2
    using D1 DV12  $\langle C \subseteq D \rangle$  by auto
    have 1: openin (top_of_set S) (D ∩ V2)
    by (simp add:  $\langle \text{open } V2 \rangle$  opeD openin_Int_open)
    have 2: closedin (top_of_set S) (D ∩ V2)
    using cloD cloDV2 closedin_trans by blast
    have  $\bigcap ?\mathcal{T} \subseteq D \cap V2$ 
    by (rule Inter_lower) (use * 1 2 in simp)

```

```

    then show False
      using K1 V12  $\langle K1 \neq \{\} \rangle \langle K1 \subseteq V1 \rangle$  closedin_imp_subset by blast
    qed
  moreover have  $D \cap U2 \cap C \neq \{\}$ 
  proof
    assume  $D \cap U2 \cap C = \{\}$ 
    then have *:  $C \subseteq D \cap V1$ 
      using D2 DV12  $\langle C \subseteq D \rangle$  by auto
    have 1: openin (top_of_set S) ( $D \cap V1$ )
      by (simp add:  $\langle \text{open } V1 \rangle$  opeD openin_Int_open)
    have 2: closedin (top_of_set S) ( $D \cap V1$ )
      using cloD cloDV1 closedin_trans by blast
    have  $\bigcap ?\mathcal{T} \subseteq D \cap V1$ 
      by (rule Inter_lower) (use * 1 2 in simp)
    then show False
      using K2 V12  $\langle K2 \neq \{\} \rangle \langle K2 \subseteq V2 \rangle$  closedin_imp_subset by blast
    qed
  ultimately show False
    using  $\langle \text{connected } C \rangle$  [unfolded connected_closed, simplified, rule_format,
of concl:  $D \cap U1 \ D \cap U2$ ]
    using  $\langle C \subseteq D \rangle$  D1 D2 V12 DV12  $\langle \text{closed } U1 \rangle \langle \text{closed } U2 \rangle \langle \text{closed } D \rangle$ 
    by blast
  qed
qed
show False
  by (metis (full_types) DiffE UnE Un_upper2 SV12_ne  $\langle K1 \subseteq V1 \rangle \langle K2 \subseteq V2 \rangle$ 
disjoint_iff_not_equal subsetCE sup_ge1 K12_Un)
qed
then show connected ( $\bigcap ?\mathcal{T}$ )
  by (auto simp: connected_closedin_eq)
show  $\bigcap ?\mathcal{T} \subseteq S$ 
  by (fastforce simp: C in_components_subset)
qed
with x show  $\bigcap ?\mathcal{T} \subseteq C$  by simp
qed auto

```

corollary *Sura_Bura_clopen_subset*:

```

  fixes S :: 'a::euclidean_space set
  assumes S: locally compact S and C:  $C \in \text{components } S$  and compact C
    and U: open U  $C \subseteq U$ 
  obtains K where openin (top_of_set S) K compact K  $C \subseteq K$   $K \subseteq U$ 
proof (rule ccontr)
  assume  $\neg$  thesis
  with that have neg:  $\nexists K. \text{openin } (\text{top\_of\_set } S) K \wedge \text{compact } K \wedge C \subseteq K \wedge K \subseteq U$ 
  by metis
  obtain V K where  $C \subseteq V$   $V \subseteq U$   $V \subseteq K$   $K \subseteq S$  compact K
    and opeSV: openin (top_of_set S) V

```

```

    using  $S \cup \langle \text{compact } C \rangle$  by (meson  $C$  in  $\text{components\_subset locally\_compact\_compact\_subopen}$ )
  let  $?T = \{T. C \subseteq T \wedge \text{openin } (\text{top\_of\_set } K) \ T \wedge \text{compact } T \wedge T \subseteq K\}$ 
  have  $CK: C \in \text{components } K$ 
    by (meson  $C \ \langle C \subseteq V \rangle \ \langle K \subseteq S \rangle \ \langle V \subseteq K \rangle \ \text{components\_intermediate\_subset\_trans}$ )
  with  $\langle \text{compact } K \rangle$ 
  have  $C = \bigcap \{T. C \subseteq T \wedge \text{openin } (\text{top\_of\_set } K) \ T \wedge \text{closedin } (\text{top\_of\_set } K) \ T\}$ 
    by (simp add:  $\text{Sura\_Bura\_compact}$ )
  then have  $Ceq: C = \bigcap ?T$ 
    by (simp add:  $\text{closedin\_compact\_eq } \langle \text{compact } K \rangle$ )
  obtain  $W$  where  $\text{open } W$  and  $W: V = S \cap W$ 
    using  $\text{opeSV}$  by (auto simp:  $\text{openin\_open}$ )
  have  $-(U \cap W) \cap \bigcap ?T \neq \{\}$ 
  proof (rule  $\text{closed\_imp\_fip\_compact}$ )
    show  $-(U \cap W) \cap \bigcap \mathcal{F} \neq \{\}$ 
      if  $\text{finite } \mathcal{F}$  and  $\mathcal{F}: \mathcal{F} \subseteq ?T$  for  $\mathcal{F}$ 
    proof (cases  $\mathcal{F} = \{\}$ )
      case True
      have False if  $U = \text{UNIV } W = \text{UNIV}$ 
      proof -
        have  $V = S$ 
          by (simp add:  $W \ \langle W = \text{UNIV} \rangle$ )
        with  $\text{neg}$  show False
      proof
        using  $\langle C \subseteq V \rangle \ \langle K \subseteq S \rangle \ \langle V \subseteq K \rangle \ \langle V \subseteq U \rangle \ \langle \text{compact } K \rangle$  by auto
      qed
    with True show  $?thesis$ 
      by auto
    next
      case False
      show  $?thesis$ 
      proof
        assume  $-(U \cap W) \cap \bigcap \mathcal{F} = \{\}$ 
        then have  $FUW: \bigcap \mathcal{F} \subseteq U \cap W$ 
          by blast
        have  $C \subseteq \bigcap \mathcal{F}$ 
          using  $\mathcal{F}$  by auto
        moreover have  $\text{compact } (\bigcap \mathcal{F})$ 
          by (metis (no_types, lifting)  $\text{compact\_Inter False mem\_Collect\_eq subsetCE } \mathcal{F}$ )
        moreover have  $\bigcap \mathcal{F} \subseteq K$ 
          using  $\text{False that}(2)$  by fastforce
        moreover have  $\text{opeKF}: \text{openin } (\text{top\_of\_set } K) \ (\bigcap \mathcal{F})$ 
          using  $\text{False } \mathcal{F} \ \langle \text{finite } \mathcal{F} \rangle$  by blast
        then have  $\text{opeVF}: \text{openin } (\text{top\_of\_set } V) \ (\bigcap \mathcal{F})$ 
          using  $W \ \langle K \subseteq S \rangle \ \langle V \subseteq K \rangle \ \text{opeKF} \ \langle \bigcap \mathcal{F} \subseteq K \rangle \ FUW \ \text{openin\_subset\_trans}$ 
        by fastforce
        then have  $\text{openin } (\text{top\_of\_set } S) \ (\bigcap \mathcal{F})$ 
          by (metis  $\text{opeSV openin\_trans}$ )
      proof

```

```

    moreover have  $\bigcap \mathcal{F} \subseteq U$ 
      by (meson  $\langle V \subseteq U \rangle$  opeVF dual_order.trans openin_imp_subset)
    ultimately show False
      using neg by blast
  qed
qed
qed (use  $\langle open\ W \rangle \langle open\ U \rangle$  in auto)
with W Ceq  $\langle C \subseteq V \rangle \langle C \subseteq U \rangle$  show False
  by auto
qed

corollary Sura_Bura_clopen_subset_alt:
  fixes S :: 'a::euclidean_space set
  assumes S: locally compact S and C: C  $\in$  components S and compact C
    and opeSU: openin (top_of_set S) U and C  $\subseteq$  U
  obtains K where openin (top_of_set S) K compact K C  $\subseteq$  K K  $\subseteq$  U
proof -
  obtain V where open V U = S  $\cap$  V
    using opeSU by (auto simp: openin_open)
  with  $\langle C \subseteq U \rangle$  have C  $\subseteq$  V
    by auto
  then show ?thesis
    using Sura_Bura_clopen_subset [OF S C  $\langle compact\ C \rangle \langle open\ V \rangle$ ]
    by (metis  $\langle U = S \cap V \rangle$  inf.bounded_iff openin_imp_subset that)
qed

corollary Sura_Bura:
  fixes S :: 'a::euclidean_space set
  assumes locally compact S C  $\in$  components S compact C
  shows C =  $\bigcap \{K. C \subseteq K \wedge compact\ K \wedge openin\ (top\_of\_set\ S)\ K\}$ 
    (is C = ?rhs)
proof
  show ?rhs  $\subseteq$  C
  proof (clarsimp, rule ccontr)
    fix x
    assume *:  $\forall X. C \subseteq X \wedge compact\ X \wedge openin\ (top\_of\_set\ S)\ X \longrightarrow x \in X$ 
      and x  $\notin$  C
    obtain U V where open U open V  $\{x\} \subseteq U$  C  $\subseteq V$  U  $\cap$  V =  $\{x\}$ 
      using separation_normal [of  $\{x\}$  C]
      by (metis Int_empty_left  $\langle x \notin C \rangle \langle compact\ C \rangle$  closed_empty closed_insert
compact_imp_closed insert_disjoint(1))
    have x  $\notin$  V
      using  $\langle U \cap V = \{x\} \rangle \langle \{x\} \subseteq U \rangle$  by blast
    then show False
      by (meson * Sura_Bura_clopen_subset  $\langle C \subseteq V \rangle \langle open\ V \rangle$  assms(1) assms(2)
assms(3) subsetCE)
  qed
qed blast

```

6.18.17 Special cases of local connectedness and path connectedness

lemma *locally_connected_1*:

assumes

$\bigwedge V x. \llbracket \text{openin } (\text{top\_of\_set } S) \ V; x \in V \rrbracket \implies \exists U. \text{openin } (\text{top\_of\_set } S) \ U$
 $\wedge \text{connected } U \wedge x \in U \wedge U \subseteq V$

shows *locally connected S*

by (*metis assms locally_def*)

lemma *locally_connected_2*:

assumes *locally connected S*

openin (top_of_set S) t

x ∈ t

shows *openin (top_of_set S) (connected_component_set t x)*

proof –

{ **fix** *y* :: 'a

let *?SS* = *top_of_set S*

assume *1*: *openin ?SS t*

$\forall w x. \text{openin } ?SS \ w \wedge x \in w \longrightarrow (\exists u. \text{openin } ?SS \ u \wedge (\exists v. \text{connected}$
 $v \wedge x \in u \wedge u \subseteq v \wedge v \subseteq w))$

and *connected_component t x y*

then have *y ∈ t* **and** *y: y ∈ connected_component_set t x*

using *connected_component_subset* **by** *blast+*

obtain *F* **where**

$\forall x y. (\exists w. \text{openin } ?SS \ w \wedge (\exists u. \text{connected } u \wedge x \in w \wedge w \subseteq u \wedge u \subseteq y)) =$
 $(\text{openin } ?SS \ (F \ a \ y) \wedge (\exists u. \text{connected } u \wedge x \in F \ x \ y \wedge F \ x \ y \subseteq u \wedge u \subseteq y))$

by *moura*

then obtain *G* **where**

$\forall a A. (\exists U. \text{openin } ?SS \ U \wedge (\exists V. \text{connected } V \wedge a \in U \wedge U \subseteq V \wedge V \subseteq$
 $A)) = (\text{openin } ?SS \ (F \ a \ A) \wedge \text{connected } (G \ a \ A) \wedge a \in F \ a \ A \wedge F \ a \ A \subseteq G \ a \ A$
 $\wedge G \ a \ A \subseteq A)$

by *moura*

then have ***: *openin ?SS (F y t) ∧ connected (G y t) ∧ y ∈ F y t ∧ F y t ⊆*
G y t ∧ G y t ⊆ t

using *1 <y ∈ t>* **by** *presburger*

have *G y t ⊆ connected_component_set t y*

by (*metis (no_types) * connected_component_eq_self connected_component_mono*
contra_subsetD)

then have $\exists A. \text{openin } ?SS \ A \wedge y \in A \wedge A \subseteq \text{connected\_component\_set } t \ x$

by (*metis (no_types) * connected_component_eq dual_order.trans y*)

}

then show *?thesis*

using *assms openin_subopen* **by** (*force simp: locally_def*)

qed

lemma *locally_connected_3*:

assumes $\bigwedge t x. \llbracket \text{openin } (\text{top\_of\_set } S) \ t; x \in t \rrbracket$

$\implies \text{openin } (\text{top\_of\_set } S)$

$(\text{connected\_component\_set } t \ x)$

$$\text{openin } (\text{top\_of\_set } S) \ v \ x \in v$$
shows $\exists u. \text{openin } (\text{top\_of\_set } S) \ u \wedge \text{connected } u \wedge x \in u \wedge u \subseteq v$
using *assms connected_component_subset* **by** *fastforce*

lemma *locally_connected*:

$$\text{locally\_connected } S \longleftrightarrow$$

$$(\forall v \ x. \text{openin } (\text{top\_of\_set } S) \ v \wedge x \in v$$

$$\longrightarrow (\exists u. \text{openin } (\text{top\_of\_set } S) \ u \wedge \text{connected } u \wedge x \in u \wedge u \subseteq v))$$

by (*metis locally_connected_1 locally_connected_2 locally_connected_3*)

lemma *locally_connected_open_connected_component*:

$$\text{locally\_connected } S \longleftrightarrow$$

$$(\forall t \ x. \text{openin } (\text{top\_of\_set } S) \ t \wedge x \in t$$

$$\longrightarrow \text{openin } (\text{top\_of\_set } S) \ (\text{connected\_component\_set } t \ x))$$

by (*metis locally_connected_1 locally_connected_2 locally_connected_3*)

lemma *locally_path_connected_1*:
assumes

$$\bigwedge v \ x. \llbracket \text{openin } (\text{top\_of\_set } S) \ v; x \in v \rrbracket$$

$$\implies \exists u. \text{openin } (\text{top\_of\_set } S) \ u \wedge \text{path\_connected } u \wedge x \in u \wedge u \subseteq v$$

shows *locally_path_connected S*
by (*force simp add: locally_def dest: assms*)

lemma *locally_path_connected_2*:
assumes *locally_path_connected S*

$$\text{openin } (\text{top\_of\_set } S) \ t$$

$$x \in t$$

shows $\text{openin } (\text{top\_of\_set } S) \ (\text{path\_component\_set } t \ x)$
proof –
{ *fix* $y :: 'a$
let $?SS = \text{top\_of\_set } S$
assume $1: \text{openin } ?SS \ t$

$$\forall w \ x. \text{openin } ?SS \ w \wedge x \in w \longrightarrow (\exists u. \text{openin } ?SS \ u \wedge (\exists v. \text{path\_connected}$$

$$v \wedge x \in u \wedge u \subseteq v \wedge v \subseteq w))$$

and $\text{path\_component } t \ x \ y$
then have $y \in t$ **and** $y: y \in \text{path\_component\_set } t \ x$
using *path_component_mem(2)* **by** *blast+*
obtain F **where**

$$\forall x \ y. (\exists w. \text{openin } ?SS \ w \wedge (\exists u. \text{path\_connected } u \wedge x \in w \wedge w \subseteq u \wedge u \subseteq$$

$$y)) = (\text{openin } ?SS \ (F \ x \ y) \wedge (\exists u. \text{path\_connected } u \wedge x \in F \ x \ y \wedge F \ x \ y \subseteq u \wedge$$

$$u \subseteq y))$$

by *moura*
then obtain G **where**

$$\forall a \ A. (\exists U. \text{openin } ?SS \ U \wedge (\exists V. \text{path\_connected } V \wedge a \in U \wedge U \subseteq V \wedge$$

$$V \subseteq A)) = (\text{openin } ?SS \ (F \ a \ A) \wedge \text{path\_connected } (G \ a \ A) \wedge a \in F \ a \ A \wedge F \ a \ A$$

$$\subseteq G \ a \ A \wedge G \ a \ A \subseteq A)$$

by *moura*
then have $*$: $\text{openin } ?SS \ (F \ y \ t) \wedge \text{path\_connected } (G \ y \ t) \wedge y \in F \ y \ t \wedge F \ y$

$$t \subseteq G \ y \ t \wedge G \ y \ t \subseteq t$$

```

    using 1 ⟨y ∈ t⟩ by presburger
  have G y t ⊆ path_component_set t y
    using * path_component_maximal rev_subsetD by blast
  then have ∃ A. openin ?SS A ∧ y ∈ A ∧ A ⊆ path_component_set t x
    by (metis * ⟨G y t ⊆ path_component_set t y⟩ dual_order.trans path_component_eq
y)
  }
  then show ?thesis
    using assms openin_subopen by (force simp: locally_def)
qed

lemma locally_path_connected_3:
  assumes ∧ t x. [[openin (top_of_set S) t; x ∈ t]]
    ⇒ openin (top_of_set S) (path_component_set t x)
    openin (top_of_set S) v x ∈ v
  shows ∃ u. openin (top_of_set S) u ∧ path_connected u ∧ x ∈ u ∧ u ⊆ v
proof -
  have path_component v x x
    by (meson assms(3) path_component_refl)
  then show ?thesis
    by (metis assms mem_Collect_eq path_component_subset path_connected_path_component)
qed

proposition locally_path_connected:
  locally_path_connected S ⟷
    (∀ V x. openin (top_of_set S) V ∧ x ∈ V
    → (∃ U. openin (top_of_set S) U ∧ path_connected U ∧ x ∈ U ∧ U ⊆
V))
  by (metis locally_path_connected_1 locally_path_connected_2 locally_path_connected_3)

proposition locally_path_connected_open_path_component:
  locally_path_connected S ⟷
    (∀ t x. openin (top_of_set S) t ∧ x ∈ t
    → openin (top_of_set S) (path_component_set t x))
  by (metis locally_path_connected_1 locally_path_connected_2 locally_path_connected_3)

lemma locally_connected_open_component:
  locally_connected S ⟷
    (∀ t c. openin (top_of_set S) t ∧ c ∈ components t
    → openin (top_of_set S) c)
  by (metis components_iff locally_connected_open_connected_component)

proposition locally_connected_im_kleinen:
  locally_connected S ⟷
    (∀ v x. openin (top_of_set S) v ∧ x ∈ v
    → (∃ u. openin (top_of_set S) u ∧
      x ∈ u ∧ u ⊆ v ∧
      (∀ y. y ∈ u → (∃ c. connected c ∧ c ⊆ v ∧ x ∈ c ∧ y ∈ c))))
  (is ?lhs = ?rhs)

```

2060

```

proof
  assume ?lhs
  then show ?rhs
    by (fastforce simp add: locally_connected)
next
  assume ?rhs
  have *:  $\exists T. \text{openin } (\text{top\_of\_set } S) \ T \wedge x \in T \wedge T \subseteq c$ 
    if  $\text{openin } (\text{top\_of\_set } S) \ t$  and  $c: c \in \text{components } t$  and  $x \in c$  for  $t \ c \ x$ 
  proof -
    from that <?rhs> [rule_format, of t x]
    obtain u where u:
       $\text{openin } (\text{top\_of\_set } S) \ u \wedge x \in u \wedge u \subseteq t \wedge$ 
       $(\forall y. y \in u \longrightarrow (\exists c. \text{connected } c \wedge c \subseteq t \wedge x \in c \wedge y \in c))$ 
    using in_components_subset by auto
    obtain F :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a where
       $\forall x \ y. (\exists z. z \in x \wedge y = \text{connected\_component\_set } x \ z) = (F \ x \ y \in x \wedge y =$ 
       $\text{connected\_component\_set } x \ (F \ x \ y))$ 
    by moura
    then have F:  $F \ t \ c \in t \wedge c = \text{connected\_component\_set } t \ (F \ t \ c)$ 
    by (meson components_iff c)
    obtain G :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a where
       $G: \forall x \ y. (\exists z. z \in y \wedge z \notin x) = (G \ x \ y \in y \wedge G \ x \ y \notin x)$ 
    by moura
    have  $G \ c \ u \notin u \vee G \ c \ u \in c$ 
    using F by (metis (full_types) u connected_componentI connected_component_eq
      mem_Collect_eq that(3))
    then show ?thesis
      using G u by auto
  qed
show ?lhs
  unfolding locally_connected_open_component by (meson * openin_subopen)
qed

proposition locally_path_connected_in_kleinen:
   $\text{locally\_path\_connected } S \longleftrightarrow$ 
   $(\forall v \ x. \text{openin } (\text{top\_of\_set } S) \ v \wedge x \in v$ 
     $\longrightarrow (\exists u. \text{openin } (\text{top\_of\_set } S) \ u \wedge$ 
       $x \in u \wedge u \subseteq v \wedge$ 
       $(\forall y. y \in u \longrightarrow (\exists p. \text{path } p \wedge \text{path\_image } p \subseteq v \wedge$ 
         $\text{pathstart } p = x \wedge \text{pathfinish } p = y))))$ 
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    apply (simp add: locally_path_connected path_connected_def)
    apply (erule all_forward ex_forward imp_forward conjE | simp)+
    by (meson dual_order.trans)
next
  assume ?rhs

```



```

have *:  $\exists T. \text{openin } (\text{top\_of\_set } S) \ T \wedge$ 
       $x \in T \wedge T \subseteq \text{path\_component\_set } u \ z$ 
if  $\text{openin } (\text{top\_of\_set } S) \ u$  and  $z \in u$  and  $c: \text{path\_component } u \ z \ x$  for  $u \ z$ 
 $x$ 
proof –
  have  $x \in u$ 
    by (meson c path_component_mem(2))
  with that ‹?rhs› [rule_format, of u x]
  obtain  $U$  where  $U$ :
     $\text{openin } (\text{top\_of\_set } S) \ U \wedge x \in U \wedge U \subseteq u \wedge$ 
     $(\forall y. y \in U \longrightarrow (\exists p. \text{path } p \wedge \text{path\_image } p \subseteq u \wedge \text{pathstart } p = x \wedge$ 
     $\text{pathfinish } p = y))$ 
    by blast
  show ?thesis
    by (metis U c mem_Collect_eq path_component_def path_component_eq
    subsetI)
  qed
  show ?lhs
    unfolding locally_path_connected_open_path_component
    using * openin_subopen by fastforce
qed

```

lemma *locally_path_connected_imp_locally_connected*:
 $\text{locally path\_connected } S \implies \text{locally connected } S$
using locally_mono path_connected_imp_connected **by** blast

lemma *locally_connected_components*:
 $\llbracket \text{locally connected } S; c \in \text{components } S \rrbracket \implies \text{locally connected } c$
by (meson locally_connected_open_component locally_open_subset openin_subtopology_self)

lemma *locally_path_connected_components*:
 $\llbracket \text{locally path\_connected } S; c \in \text{components } S \rrbracket \implies \text{locally path\_connected } c$
by (meson locally_connected_open_component locally_open_subset locally_path_connected_imp_locally_connected openin_subtopology_self)

lemma *locally_path_connected_connected_component*:
 $\text{locally path\_connected } S \implies \text{locally path\_connected } (\text{connected\_component\_set } S \ x)$
by (metis components_iff_connected_component_eq_empty locally_empty locally_path_connected_components)

lemma *open_imp_locally_path_connected*:
fixes $S :: 'a :: \text{real\_normed\_vector set}$
assumes $\text{open } S$
shows $\text{locally path\_connected } S$
proof (rule locally_mono)
show $\text{locally convex } S$
using assms **unfolding** locally_def
by (meson open_ball centre_in_ball convex_ball openE open_subset openin_imp_subset openin_open_trans subset_trans)

show $\bigwedge T :: 'a \text{ set. } \text{convex } T \implies \text{path\_connected } T$
using *convex_imp_path_connected* **by** *blast*
qed

lemma *open_imp_locally_connected*:
fixes $S :: 'a :: \text{real\_normed\_vector set}$
shows $\text{open } S \implies \text{locally\_connected } S$
by (*simp add: locally_path_connected_imp_locally_connected open_imp_locally_path_connected*)

lemma *locally_path_connected_UNIV*: $\text{locally\_path\_connected } (\text{UNIV} :: 'a :: \text{real\_normed\_vector set})$
by (*simp add: open_imp_locally_path_connected*)

lemma *locally_connected_UNIV*: $\text{locally\_connected } (\text{UNIV} :: 'a :: \text{real\_normed\_vector set})$
by (*simp add: open_imp_locally_connected*)

lemma *openin_connected_component_locally_connected*:
 $\text{locally\_connected } S$
 $\implies \text{openin } (\text{top\_of\_set } S) (\text{connected\_component\_set } S \ x)$
by (*metis connected_component_eq_empty locally_connected_2 openin_empty openin_subtopology_self*)

lemma *openin_components_locally_connected*:
 $\llbracket \text{locally\_connected } S; c \in \text{components } S \rrbracket \implies \text{openin } (\text{top\_of\_set } S) \ c$
using *locally_connected_open_component openin_subtopology_self* **by** *blast*

lemma *openin_path_component_locally_path_connected*:
 $\text{locally\_path\_connected } S$
 $\implies \text{openin } (\text{top\_of\_set } S) (\text{path\_component\_set } S \ x)$
by (*metis (no_types) empty_iff locally_path_connected_2 openin_subopen openin_subtopology_self path_component_eq_empty*)

lemma *closedin_path_component_locally_path_connected*:
assumes $\text{locally\_path\_connected } S$
shows $\text{closedin } (\text{top\_of\_set } S) (\text{path\_component\_set } S \ x)$
proof –
have $\text{openin } (\text{top\_of\_set } S) (\bigcup (\{\text{path\_component\_set } S \ y \mid y. y \in S\} - \{\text{path\_component\_set } S \ x\}))$
using *locally_path_connected_2 assms* **by** *fastforce*
then show *?thesis*
by (*simp add: closedin_def path_component_subset complement_path_component_Union*)
qed

lemma *convex_imp_locally_path_connected*:
fixes $S :: 'a :: \text{real\_normed\_vector set}$
assumes $\text{convex } S$
shows $\text{locally\_path\_connected } S$
proof (*clarsimp simp add: locally_path_connected*)

```

fix V x
assume openin (top_of_set S) V and x ∈ V
then obtain T e where V = S ∩ T x ∈ S 0 < e ball x e ⊆ T
  by (metis Int_iff openE openin_open)
then have openin (top_of_set S) (S ∩ ball x e) path_connected (S ∩ ball x e)
  by (simp_all add: assms convex_Int convex_imp_path_connected openin_open_Int)
then show ∃ U. openin (top_of_set S) U ∧ path_connected U ∧ x ∈ U ∧ U ⊆
V
  using ⟨0 < e⟩ ⟨V = S ∩ T⟩ ⟨ball x e ⊆ T⟩ ⟨x ∈ S⟩ by auto
qed

lemma convex_imp_locally_connected:
  fixes S :: 'a:: real_normed_vector set
  shows convex S ⟹ locally_connected S
  by (simp add: locally_path_connected_imp_locally_connected convex_imp_locally_path_connected)

```

6.18.18 Relations between components and path components

```

lemma path_component_eq_connected_component:
  assumes locally_path_connected S
  shows (path_component S x = connected_component S x)
proof (cases x ∈ S)
  case True
  have openin (top_of_set (connected_component_set S x)) (path_component_set
S x)
  proof (rule openin_subset_trans)
    show openin (top_of_set S) (path_component_set S x)
      by (simp add: True assms locally_path_connected_2)
    show connected_component_set S x ⊆ S
      by (simp add: connected_component_subset)
  qed (simp add: path_component_subset_connected_component)
  moreover have closedin (top_of_set (connected_component_set S x)) (path_component_set
S x)
  proof (rule closedin_subset_trans [of S])
    show closedin (top_of_set S) (path_component_set S x)
      by (simp add: assms closedin_path_component_locally_path_connected)
    show connected_component_set S x ⊆ S
      by (simp add: connected_component_subset)
  qed (simp add: path_component_subset_connected_component)
  ultimately have *: path_component_set S x = connected_component_set S x
  by (metis connected_component connected_clopen True path_component_eq_empty)
  then show ?thesis
    by blast
next
  case False then show ?thesis
    by (metis Collect_empty_eq_bot connected_component_eq_empty path_component_eq_empty)
qed

```

```

lemma path_component_eq_connected_component_set:

```

locally path_connected $S \implies (\text{path\_component\_set } S \ x = \text{connected\_component\_set } S \ x)$

by (*simp add: path_component_eq_connected_component*)

lemma *locally_path_connected_path_component*:

locally path_connected $S \implies \text{locally path\_connected } (\text{path\_component\_set } S \ x)$

using *locally_path_connected_connected_component path_component_eq_connected_component*
by *fastforce*

lemma *open_path_connected_component*:

fixes $S :: 'a :: \text{real\_normed\_vector\_set}$

shows $\text{open } S \implies \text{path\_component } S \ x = \text{connected\_component } S \ x$

by (*simp add: path_component_eq_connected_component open_imp_locally_path_connected*)

lemma *open_path_connected_component_set*:

fixes $S :: 'a :: \text{real\_normed\_vector\_set}$

shows $\text{open } S \implies \text{path\_component\_set } S \ x = \text{connected\_component\_set } S \ x$

by (*simp add: open_path_connected_component*)

proposition *locally_connected_quotient_image*:

assumes $\text{lcS: locally\_connected } S$

and $\text{oo: } \bigwedge T. T \subseteq f \text{ ` } S$

$\implies \text{openin } (\text{top\_of\_set } S) (S \cap f \text{ ` } T) \longleftrightarrow$
 $\text{openin } (\text{top\_of\_set } (f \text{ ` } S)) T$

shows $\text{locally\_connected } (f \text{ ` } S)$

proof (*clarsimp simp: locally_connected_open_component*)

fix $U \ C$

assume $\text{opefSU: openin } (\text{top\_of\_set } (f \text{ ` } S)) \ U$ **and** $C \in \text{components } U$

then have $C \subseteq U \ U \subseteq f \text{ ` } S$

by (*meson in_components_subset openin_imp_subset*)

then have $\text{openin } (\text{top\_of\_set } (f \text{ ` } S)) \ C \longleftrightarrow$

$\text{openin } (\text{top\_of\_set } S) (S \cap f \text{ ` } C)$

by (*auto simp: oo*)

moreover have $\text{openin } (\text{top\_of\_set } S) (S \cap f \text{ ` } C)$

proof (*subst openin_subopen, clarify*)

fix x

assume $x \in S \ f \ x \in C$

show $\exists T. \text{openin } (\text{top\_of\_set } S) \ T \wedge x \in T \wedge T \subseteq (S \cap f \text{ ` } C)$

proof (*intro conjI exI*)

show $\text{openin } (\text{top\_of\_set } S) (\text{connected\_component\_set } (S \cap f \text{ ` } U) \ x)$

proof (*rule ccontr*)

assume $** : \neg \text{openin } (\text{top\_of\_set } S) (\text{connected\_component\_set } (S \cap f \text{ ` } U) \ x)$

then have $x \notin (S \cap f \text{ ` } U)$

using $\langle U \subseteq f \text{ ` } S \rangle \text{opefSU lcS locally\_connected\_2 oo}$ **by** *blast*

with $**$ **show** *False*

by (*metis (no_types) connected_component_eq_empty empty_iff openin_subopen*)

qed

```

next
  show  $x \in \text{connected\_component\_set } (S \cap f - ' U) x$ 
    using  $\langle C \subseteq U \rangle \langle f x \in C \rangle \langle x \in S \rangle$  by auto
next
  have contf: continuous_on S f
    by (simp add: continuous_on_open oo openin_imp_subset)
  then have continuous_on (connected_component_set (S  $\cap$  f - ' U) x) f
  by (meson connected_component_subset continuous_on_subset inf.boundedE)
  then have connected (f ' connected_component_set (S  $\cap$  f - ' U) x)
  by (rule connected_continuous_image [OF _ connected_connected_component])
  moreover have f ' connected_component_set (S  $\cap$  f - ' U) x  $\subseteq$  U
    using connected_component_in by blast
  moreover have C  $\cap$  f ' connected_component_set (S  $\cap$  f - ' U) x  $\neq$  {}
    using  $\langle C \subseteq U \rangle \langle f x \in C \rangle \langle x \in S \rangle$  by fastforce
  ultimately have fC: f ' (connected_component_set (S  $\cap$  f - ' U) x)  $\subseteq$  C
    by (rule components_maximal [OF  $\langle C \in \text{components } U \rangle$ ])
  have cUC: connected_component_set (S  $\cap$  f - ' U) x  $\subseteq$  (S  $\cap$  f - ' C)
    using connected_component_subset fC by blast
  have connected_component_set (S  $\cap$  f - ' U) x  $\subseteq$  connected_component_set
(S  $\cap$  f - ' C) x
  proof -
    { assume  $x \in \text{connected\_component\_set } (S \cap f - ' U) x$ 
      then have ?thesis
        using cUC connected_component_idemp connected_component_mono
      }
  by blast }
  then show ?thesis
    using connected_component_eq_empty by auto
  qed
  also have ...  $\subseteq$  (S  $\cap$  f - ' C)
    by (rule connected_component_subset)
  finally show connected_component_set (S  $\cap$  f - ' U) x  $\subseteq$  (S  $\cap$  f - ' C) .
  qed
  qed
  ultimately show openin (top_of_set (f ' S)) C
    by metis
  qed

```

The proof resembles that above but is not identical!

proposition *locally_path_connected_quotient_image*:

assumes *lcS*: *locally_path_connected S*

and oo: $\bigwedge T. T \subseteq f ' S$

$\implies \text{openin } (\text{top\_of\_set } S) (S \cap f - ' T) \longleftrightarrow \text{openin } (\text{top\_of\_set } (f ' S)) T$

shows *locally_path_connected* (f ' S)

proof (*clarsimp simp: locally_path_connected_open_path_component*)

fix U y

assume *opefSU*: *openin* (top_of_set (f ' S)) U and $y \in U$

then have *path_component_set* U y $\subseteq$ U U $\subseteq$ f ' S

by (*meson path_component_subset openin_imp_subset*)+

```

then have openin (top_of_set (f ' S)) (path_component_set U y)  $\longleftrightarrow$ 
      openin (top_of_set S) (S  $\cap$  f - ' path_component_set U y)
proof -
  have path_component_set U y  $\subseteq$  f ' S
  using  $\langle U \subseteq f ' S \rangle \langle \text{path\_component\_set } U y \subseteq U \rangle$  by blast
  then show ?thesis
    using oo by blast
qed
moreover have openin (top_of_set S) (S  $\cap$  f - ' path_component_set U y)
proof (subst openin_subopen, clarify)
  fix x
  assume x  $\in$  S and Uyfx: path_component U y (f x)
  then have f x  $\in$  U
  using path_component_mem by blast
  show  $\exists T. \text{openin (top\_of\_set } S) \ T \wedge x \in T \wedge T \subseteq (S \cap f - ' \text{path\_component\_set } U y)$ 
  proof (intro conjI exI)
    show openin (top_of_set S) (path_component_set (S  $\cap$  f - ' U) x)
    proof (rule ccontr)
      assume **:  $\neg \text{openin (top\_of\_set } S) \ (\text{path\_component\_set } (S \cap f - ' U) x)$ 
    x)
      then have x  $\notin (S \cap f - ' U)$ 
      by (metis (no_types, lifting)  $\langle U \subseteq f ' S \rangle \text{opefSU lcS oo locally\_path\_connected\_open\_path\_compon}$ )
      then show False
      using **  $\langle \text{path\_component\_set } U y \subseteq U \rangle \langle x \in S \rangle \langle \text{path\_component } U y$ 
      (f x)  $\rangle$  by blast
    qed
  next
    show x  $\in$  path_component_set (S  $\cap$  f - ' U) x
    by (simp add:  $\langle f x \in U \rangle \langle x \in S \rangle \text{path\_component\_refl}$ )
  next
    have contf: continuous_on S f
    by (simp add: continuous_on_open oo openin_imp_subset)
    then have continuous_on (path_component_set (S  $\cap$  f - ' U) x) f
    by (meson Int_lower1 continuous_on_subset path_component_subset)
    then have path_connected (f ' path_component_set (S  $\cap$  f - ' U) x)
    by (simp add: path_connected_continuous_image)
    moreover have f ' path_component_set (S  $\cap$  f - ' U) x  $\subseteq$  U
    using path_component_mem by fastforce
    moreover have f x  $\in$  f ' path_component_set (S  $\cap$  f - ' U) x
    by (force simp:  $\langle x \in S \rangle \langle f x \in U \rangle \text{path\_component\_refl\_eq}$ )
    ultimately have f ' (path_component_set (S  $\cap$  f - ' U) x)  $\subseteq$  path_component_set
    U (f x)
    by (meson path_component_maximal)
    also have ...  $\subseteq$  path_component_set U y
    by (simp add: Uyfx path_component_maximal path_component_subset
    path_component_sym)
    finally have fC: f ' (path_component_set (S  $\cap$  f - ' U) x)  $\subseteq$  path_component_set
    U y .

```

```

    have cUC: path_component_set (S ∩ f - ' U) x ⊆ (S ∩ f - ' path_component_set
    U y)
    using path_component_subset fC by blast
    have path_component_set (S ∩ f - ' U) x ⊆ path_component_set (S ∩ f - '
    path_component_set U y) x
    proof -
      have ∧a. path_component_set (path_component_set (S ∩ f - ' U) x) a ⊆
    path_component_set (S ∩ f - ' path_component_set U y) a
      using cUC path_component_mono by blast
      then show ?thesis
      using path_component_path_component by blast
    qed
    also have ... ⊆ (S ∩ f - ' path_component_set U y)
    by (rule path_component_subset)
    finally show path_component_set (S ∩ f - ' U) x ⊆ (S ∩ f - ' path_component_set
    U y) .
  qed
  qed
  ultimately show openin (top_of_set (f ' S)) (path_component_set U y)
  by metis
qed

```

6.18.19 Components, continuity, openin, closedin

```

lemma continuous_on_components_gen:
  fixes f :: 'a::topological_space ⇒ 'b::topological_space
  assumes ∧C. C ∈ components S ⇒
    openin (top_of_set S) C ∧ continuous_on C f
  shows continuous_on S f
proof (clarsimp simp: continuous_openin_preimage_eq)
  fix t :: 'b set
  assume open t
  have *: S ∩ f - ' t = (⋃ c ∈ components S. c ∩ f - ' t)
  by auto
  show openin (top_of_set S) (S ∩ f - ' t)
  unfolding * using ⟨open t⟩ assms continuous_openin_preimage_gen openin_trans
  openin_Union by blast
qed

```

```

lemma continuous_on_components:
  fixes f :: 'a::topological_space ⇒ 'b::topological_space
  assumes locally_connected S ∧ C. C ∈ components S ⇒ continuous_on C f
  shows continuous_on S f
proof (rule continuous_on_components_gen)
  fix C
  assume C ∈ components S
  then show openin (top_of_set S) C ∧ continuous_on C f
  by (simp add: assms openin_components_locally_connected)
qed

```

```

lemma continuous_on_components_eq:
  locally_connected S
   $\implies (\text{continuous\_on } S f \longleftrightarrow (\forall c \in \text{components } S. \text{continuous\_on } c f))$ 
by (meson continuous_on_components continuous_on_subset in_components_subset)

lemma continuous_on_components_open:
  fixes S :: 'a::real_normed_vector set
  assumes open S
   $\bigwedge c. c \in \text{components } S \implies \text{continuous\_on } c f$ 
  shows continuous_on S f
using continuous_on_components open_imp_locally_connected assms by blast

lemma continuous_on_components_open_eq:
  fixes S :: 'a::real_normed_vector set
  shows open S  $\implies (\text{continuous\_on } S f \longleftrightarrow (\forall c \in \text{components } S. \text{continuous\_on } c f))$ 
using continuous_on_subset in_components_subset
by (blast intro: continuous_on_components_open)

lemma closedin_union_complement_components:
  assumes U: locally_connected U
  and S: closedin (top_of_set U) S
  and cuS:  $c \subseteq \text{components}(U - S)$ 
  shows closedin (top_of_set U)  $(S \cup \bigcup c)$ 
proof -
  have di:  $(\bigwedge S T. S \in c \wedge T \in c' \implies \text{disjnt } S T) \implies \text{disjnt } (\bigcup c) (\bigcup c')$  for  $c'$ 
  by (simp add: disjnt_def) blast
  have S  $\subseteq U$ 
  using S closedin_imp_subset by blast
  moreover have  $U - S = \bigcup c \cup \bigcup (\text{components } (U - S) - c)$ 
  by (metis Diff_partition Union_components Union_Un_distrib assms(3))
  moreover have  $\text{disjnt } (\bigcup c) (\bigcup (\text{components } (U - S) - c))$ 
  apply (rule di)
  by (metis di DiffD1 DiffD2 assms(3) components_nonoverlap disjnt_def subsetCE)
  ultimately have  $eq: S \cup \bigcup c = U - (\bigcup (\text{components}(U - S) - c))$ 
  by (auto simp: disjnt_def)
  have *:  $\text{openin } (\text{top\_of\_set } U) (\bigcup (\text{components } (U - S) - c))$ 
  proof (rule openin_Union [OF openin_trans [of U - S]])
    show  $\text{openin } (\text{top\_of\_set } (U - S)) T$  if  $T \in \text{components } (U - S) - c$  for  $T$ 
    using that by (simp add: U S locally_diff_closed openin_components_locally_connected)
    show  $\text{openin } (\text{top\_of\_set } U) (U - S)$  if  $T \in \text{components } (U - S) - c$  for  $T$ 
    using that by (simp add: openin_diff S)
  qed
  have  $\text{closedin } (\text{top\_of\_set } U) (U - \bigcup (\text{components } (U - S) - c))$ 
  by (metis closedin_diff closedin_topospace topspace_euclidean_subtopology *)
  then have  $\text{openin } (\text{top\_of\_set } U) (U - (U - \bigcup (\text{components } (U - S) - c)))$ 
  by (simp add: openin_diff)

```


6.18.20 Existence of isometry between subspaces of same dimension

```

lemma isometry_subset_subspace:
  fixes  $S :: 'a::euclidean\_space\ set$ 
    and  $T :: 'b::euclidean\_space\ set$ 
  assumes  $S: subspace\ S$ 
    and  $T: subspace\ T$ 
    and  $d: dim\ S \leq dim\ T$ 
  obtains  $f$  where  $linear\ f \wedge S \subseteq T \wedge x. x \in S \implies norm(f\ x) = norm\ x$ 
proof –
  obtain  $B$  where  $B \subseteq S$  and  $Borth: pairwise\ orthogonal\ B$ 
    and  $B1: \bigwedge x. x \in B \implies norm\ x = 1$ 
    and  $independent\ B\ finite\ B\ card\ B = dim\ S\ span\ B = S$ 

```

```

    by (metis orthonormal_basis_subspace [OF S] independent_finite)
  obtain C where C ⊆ T and Corth: pairwise_orthogonal C
    and C1:  $\bigwedge x. x \in C \implies \text{norm } x = 1$ 
    and independent C finite C card C = dim T span C = T
    by (metis orthonormal_basis_subspace [OF T] independent_finite)
  obtain fb where fb : B ⊆ C inj_on fb B
  by (metis ⟨card B = dim S⟩ ⟨card C = dim T⟩ ⟨finite B⟩ ⟨finite C⟩ card_le_inj
d)
  then have pairwise_orth_fb: pairwise (λv j. orthogonal (fb v) (fb j)) B
    using Corth unfolding pairwise_def inj_on_def
    by (blast intro: orthogonal_clauses)
  obtain f where linear f and ffb:  $\bigwedge x. x \in B \implies f x = \text{fb } x$ 
    using linear_independent_extend ⟨independent B⟩ by fastforce
  have span (f ` B) ⊆ span C
    by (metis fb ` B ⊆ C ffb image_cong span_mono)
  then have f ` S ⊆ T
    unfolding ⟨span B = S⟩ ⟨span C = T⟩ span_linear_image[OF ⟨linear f⟩] .
  have [simp]:  $\bigwedge x. x \in B \implies \text{norm } (\text{fb } x) = \text{norm } x$ 
    using B1 C1 ⟨fb ` B ⊆ C⟩ by auto
  have norm (f x) = norm x if x ∈ S for x
  proof -
    interpret linear f by fact
    obtain a where x:  $x = (\sum v \in B. a v *_R v)$ 
      using ⟨finite B⟩ ⟨span B = S⟩ ⟨x ∈ S⟩ span_finite by fastforce
    have norm (f x) ^ 2 = norm  $(\sum v \in B. a v *_R \text{fb } v)$  ^ 2 by (simp add: sum_scale
fffb x)
    also have ... =  $(\sum v \in B. \text{norm } ((a v *_R \text{fb } v)) ^ 2)$ 
    proof (rule norm_sum_Pythagorean [OF ⟨finite B⟩])
      show pairwise (λv j. orthogonal (a v *_R fb v) (a j *_R fb j)) B
        by (rule pairwise_ortho_scaleR [OF pairwise_orth_fb])
    qed
    also have ... = norm x ^ 2
      by (simp add: x pairwise_ortho_scaleR Borth norm_sum_Pythagorean [OF
⟨finite B⟩])
    finally show ?thesis
      by (simp add: norm_eq_sqrt_inner)
  qed
  then show ?thesis
    by (rule that [OF ⟨linear f⟩ ⟨f ` S ⊆ T⟩])
qed

```

proposition *isometries_subspaces:*

```

fixes S :: 'a::euclidean_space set
and T :: 'b::euclidean_space set
assumes S: subspace S
and T: subspace T
and d: dim S = dim T

```

```

obtains f g where linear f linear g f ` S = T g ` T = S
 $\bigwedge x. x \in S \implies \text{norm}(f x) = \text{norm } x$ 

```

$$\begin{aligned} \bigwedge x. x \in T &\implies \text{norm}(g\ x) = \text{norm}\ x \\ \bigwedge x. x \in S &\implies g(f\ x) = x \\ \bigwedge x. x \in T &\implies f(g\ x) = x \end{aligned}$$

proof –

```

obtain B where  $B \subseteq S$  and Borth: pairwise orthogonal B
  and B1:  $\bigwedge x. x \in B \implies \text{norm}\ x = 1$ 
  and independent B finite B card B = dim S span B = S
  by (metis orthonormal_basis_subspace [OF S] independent_finite)
obtain C where  $C \subseteq T$  and Corth: pairwise orthogonal C
  and C1:  $\bigwedge x. x \in C \implies \text{norm}\ x = 1$ 
  and independent C finite C card C = dim T span C = T
  by (metis orthonormal_basis_subspace [OF T] independent_finite)
obtain fb where bij_betw fb B C
  by (metis ‹finite B› ‹finite C› bij_betw_iff_card ‹card B = dim S› ‹card C =
dim T› d)
then have pairwise_orth_fb: pairwise ( $\lambda v\ j. \text{orthogonal}\ (fb\ v)\ (fb\ j)$ ) B
  using Corth unfolding pairwise_def inj_on_def bij_betw_def
  by (blast intro: orthogonal_clauses)
obtain f where linear f and ffb:  $\bigwedge x. x \in B \implies f\ x = fb\ x$ 
  using linear_independent_extend ‹independent B› by fastforce
interpret f: linear f by fact
define gb where gb  $\equiv \text{inv\_into}\ B\ fb$ 
then have pairwise_orth_gb: pairwise ( $\lambda v\ j. \text{orthogonal}\ (gb\ v)\ (gb\ j)$ ) C
  using Borth ‹bij_betw fb B C› unfolding pairwise_def bij_betw_def by force
obtain g where linear g and ggb:  $\bigwedge x. x \in C \implies g\ x = gb\ x$ 
  using linear_independent_extend ‹independent C› by fastforce
interpret g: linear g by fact
have span (f ‹ B)  $\subseteq \text{span}\ C$ 
  by (metis ‹bij_betw fb B C› bij_betw_imp_surj_on eq_iff ffb image_cong)
then have f ‹ S  $\subseteq T$ 
  unfolding ‹span B = S› ‹span C = T› span_linear_image [OF ‹linear f›] .
have [simp]:  $\bigwedge x. x \in B \implies \text{norm}\ (fb\ x) = \text{norm}\ x$ 
  using B1 C1 ‹bij_betw fb B C› bij_betw_imp_surj_on by fastforce
have f [simp]:  $\text{norm}\ (f\ x) = \text{norm}\ x\ g\ (f\ x) = x$  if  $x \in S$  for x
proof –
  obtain a where x:  $x = (\sum v \in B. a\ v *_{\mathbb{R}} v)$ 
  using ‹finite B› ‹span B = S› ‹x ∈ S› span_finite by fastforce
  have f x =  $(\sum v \in B. f\ (a\ v *_{\mathbb{R}} v))$ 
  using linear_sum [OF ‹linear f›] x by auto
  also have ... =  $(\sum v \in B. a\ v *_{\mathbb{R}} f\ v)$ 
  by (simp add: f.sum f.scale)
  also have ... =  $(\sum v \in B. a\ v *_{\mathbb{R}} fb\ v)$ 
  by (simp add: ffb cong: sum.cong)
  finally have *:  $f\ x = (\sum v \in B. a\ v *_{\mathbb{R}} fb\ v)$  .
  then have  $(\text{norm}\ (f\ x))^2 = (\text{norm}\ (\sum v \in B. a\ v *_{\mathbb{R}} fb\ v))^2$  by simp
  also have ... =  $(\sum v \in B. \text{norm}\ ((a\ v *_{\mathbb{R}} fb\ v))^2)$ 
proof (rule norm_sum_Pythagorean [OF ‹finite B›])
  show pairwise ( $\lambda v\ j. \text{orthogonal}\ (a\ v *_{\mathbb{R}} fb\ v)\ (a\ j *_{\mathbb{R}} fb\ j)$ ) B
  by (rule pairwise_ortho_scaleR [OF pairwise_orth_fb])

```

```

qed
also have ... = (norm x)2
  by (simp add: x pairwise_ortho_scaleR Borth norm_sum_Pythagorean [OF
    ⟨finite B⟩])
finally show norm (f x) = norm x
  by (simp add: norm_eq_sqrt_inner)
have g (f x) = g (∑ v∈B. a v *R fb v) by (simp add: *)
also have ... = (∑ v∈B. g (a v *R fb v))
  by (simp add: g.sum g.scale)
also have ... = (∑ v∈B. a v *R g (fb v))
  by (simp add: g.scale)
also have ... = (∑ v∈B. a v *R v)
proof (rule sum.cong [OF refl])
  show a x *R g (fb x) = a x *R x if x ∈ B for x
  using that ⟨bij_betw fb B C⟩ bij_betwE bij_betw_inv_into_left gb_def ggb
by fastforce
qed
also have ... = x
  using x by blast
finally show g (f x) = x .
qed
have [simp]: ∧x. x ∈ C ⟹ norm (gb x) = norm x
  by (metis B1 C1 ⟨bij_betw fb B C⟩ bij_betw_imp_surj_on gb_def inv_into_into)
have g [simp]: f (g x) = x if x ∈ T for x
proof -
  obtain a where x: x = (∑ v ∈ C. a v *R v)
  using ⟨finite C⟩ ⟨span C = T⟩ ⟨x ∈ T⟩ span_finite by fastforce
  have g x = (∑ v ∈ C. g (a v *R v))
  by (simp add: x.g.sum)
  also have ... = (∑ v ∈ C. a v *R g v)
  by (simp add: g.scale)
  also have ... = (∑ v ∈ C. a v *R gb v)
  by (simp add: ggb cong: sum.cong)
  finally have f (g x) = f (∑ v∈C. a v *R gb v) by simp
  also have ... = (∑ v∈C. f (a v *R gb v))
  by (simp add: f.scale f.sum)
  also have ... = (∑ v∈C. a v *R f (gb v))
  by (simp add: f.scale f.sum)
  also have ... = (∑ v∈C. a v *R v)
  using ⟨bij_betw fb B C⟩
  by (simp add: bij_betw_def gb_def bij_betw_inv_into_right ffb inv_into_into)
  also have ... = x
  using x by blast
  finally show f (g x) = x .
qed
have gim: g ‘ T = S
  by (metis (full_types) S T ⟨f ‘ S ⊆ T⟩ d_dim_eq_span dim_image_le f(2)
    g.linear_axioms
    image_iff linear_subspace_image span_eq_iff subset_iff)

```

```

have fim:  $f \text{ ' } S = T$ 
  using  $\langle g \text{ ' } T = S \rangle$  image_iff by fastforce
have [simp]:  $\text{norm } (g \ x) = \text{norm } x$  if  $x \in T$  for  $x$ 
  using fim that by auto
show ?thesis
  by (rule that [OF  $\langle \text{linear } f \rangle \langle \text{linear } g \rangle$ ]) (simp_all add: fim gim)
qed

```

```

corollary isometry_subspaces:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  and  $T :: 'b::\text{euclidean\_space}$  set
  assumes  $S$ :  $\text{subspace } S$ 
  and  $T$ :  $\text{subspace } T$ 
  and  $d$ :  $\text{dim } S = \text{dim } T$ 
  obtains  $f$  where  $\text{linear } f$   $f \text{ ' } S = T$   $\bigwedge x. x \in S \implies \text{norm}(f \ x) = \text{norm } x$ 
using isometries_subspaces [OF assms]
by metis

```

```

corollary isomorphisms_UNIV_UNIV:
  assumes  $\text{DIM}('M) = \text{DIM}('N)$ 
  obtains  $f :: 'M::\text{euclidean\_space} \Rightarrow 'N::\text{euclidean\_space}$  and  $g$ 
  where  $\text{linear } f$   $\text{linear } g$ 
     $\bigwedge x. \text{norm}(f \ x) = \text{norm } x \bigwedge y. \text{norm}(g \ y) = \text{norm } y$ 
     $\bigwedge x. g \ (f \ x) = x \bigwedge y. f(g \ y) = y$ 
  using assms by (auto intro: isometries_subspaces [of UNIV::'M set UNIV::'N set])

```

```

lemma homeomorphic_subspaces:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  and  $T :: 'b::\text{euclidean\_space}$  set
  assumes  $S$ :  $\text{subspace } S$ 
  and  $T$ :  $\text{subspace } T$ 
  and  $d$ :  $\text{dim } S = \text{dim } T$ 
  shows  $S$  homeomorphic  $T$ 
proof -
  obtain  $f \ g$  where  $\text{linear } f$   $\text{linear } g$   $f \text{ ' } S = T$   $g \text{ ' } T = S$ 
     $\bigwedge x. x \in S \implies g(f \ x) = x \bigwedge x. x \in T \implies f(g \ x) = x$ 
  by (blast intro: isometries_subspaces [OF assms])
  then show ?thesis
    unfolding homeomorphic_def homeomorphism_def
    apply (rule_tac  $x=f$  in exI, rule_tac  $x=g$  in exI)
    apply (auto simp: linear_continuous_on linear_conv_bounded_linear)
    done
qed

```

```

lemma homeomorphic_affine_sets:
  assumes  $\text{affine } S$   $\text{affine } T$   $\text{aff\_dim } S = \text{aff\_dim } T$ 
  shows  $S$  homeomorphic  $T$ 
proof (cases  $S = \{\}$   $\vee T = \{\}$ )

```

```

    case True with assms aff_dim_empty homeomorphic_empty show ?thesis
      by metis
next
case False
then obtain a b where ab: a ∈ S b ∈ T by auto
then have ss: subspace ((+) (- a) ' S) subspace ((+) (- b) ' T)
  using affine_diffs_subspace assms by blast+
have dd: dim ((+) (- a) ' S) = dim ((+) (- b) ' T)
  using assms ab by (simp add: aff_dim_eq_dim [OF hull_inc] image_def)
have S homeomorphic ((+) (- a) ' S)
  by (fact homeomorphic_translation)
also have ... homeomorphic ((+) (- b) ' T)
  by (rule homeomorphic_subspaces [OF ss dd])
also have ... homeomorphic T
  using homeomorphic_translation [of T - b] by (simp add: homeomorphic_sym
[of T])
  finally show ?thesis .
qed

```

6.18.21 Retracts, in a general sense, preserve (co)homotopic triviality)

```

locale Retracts =
  fixes s h t k
  assumes conth: continuous_on s h
    and imh: h ' s = t
    and contk: continuous_on t k
    and imk: k ' t ⊆ s
    and idhk:  $\bigwedge y. y \in t \implies h(k y) = y$ 

```

begin

```

lemma homotopically_trivial_retraction_gen:
  assumes P:  $\bigwedge f. \llbracket \text{continuous\_on } U f; f ' U \subseteq t; Q f \rrbracket \implies P(k \circ f)$ 
    and Q:  $\bigwedge f. \llbracket \text{continuous\_on } U f; f ' U \subseteq s; P f \rrbracket \implies Q(h \circ f)$ 
    and Qeq:  $\bigwedge h k. (\bigwedge x. x \in U \implies h x = k x) \implies Q h = Q k$ 
    and hom:  $\bigwedge f g. \llbracket \text{continuous\_on } U f; f ' U \subseteq s; P f; \text{continuous\_on } U g; g ' U \subseteq s; P g \rrbracket$ 
       $\implies \text{homotopic\_with\_canon } P U s f g$ 
    and contf: continuous_on U f and imf: f ' U ⊆ t and Qf: Q f
    and contg: continuous_on U g and img: g ' U ⊆ t and Qg: Q g
  shows homotopic_with_canon Q U t f g

```

proof –

```

  have continuous_on U (k ∘ f)
    using contf continuous_on_compose continuous_on_subset contk imf by blast
  moreover have (k ∘ f) ' U ⊆ s
    using imf imk by fastforce
  moreover have P (k ∘ f)
    by (simp add: P Qf contf imf)

```

```

moreover have continuous_on U (k ∘ g)
  using contg continuous_on compose continuous_on_subset contk img by blast
moreover have (k ∘ g) ' U ⊆ s
  using img imk by fastforce
moreover have P (k ∘ g)
  by (simp add: P Qg contg img)
ultimately have homotopic_with_canon P U s (k ∘ f) (k ∘ g)
  by (rule hom)
then have homotopic_with_canon Q U t (h ∘ (k ∘ f)) (h ∘ (k ∘ g))
apply (rule homotopic_with_compose_continuous_left [OF homotopic_with_mono])
  using Q by (auto simp: conth imh)
then show ?thesis
proof (rule homotopic_with_eq; simp)
  show  $\bigwedge h k. (\bigwedge x. x \in U \implies h x = k x) \implies Q h = Q k$ 
    using Qeq topspace_euclidean_subtopology by blast
  show  $\bigwedge x. x \in U \implies f x = h (k (f x)) \wedge x. x \in U \implies g x = h (k (g x))$ 
    using idhk imf img by auto
qed
qed

lemma homotopically_trivial_retraction_null_gen:
  assumes P:  $\bigwedge f. \llbracket \text{continuous\_on } U f; f ' U \subseteq t; Q f \rrbracket \implies P(k \circ f)$ 
    and Q:  $\bigwedge f. \llbracket \text{continuous\_on } U f; f ' U \subseteq s; P f \rrbracket \implies Q(h \circ f)$ 
    and Qeq:  $\bigwedge h k. (\bigwedge x. x \in U \implies h x = k x) \implies Q h = Q k$ 
    and hom:  $\bigwedge f. \llbracket \text{continuous\_on } U f; f ' U \subseteq s; P f \rrbracket$ 
       $\implies \exists c. \text{homotopic\_with\_canon } P U s f (\lambda x. c)$ 
    and conf: continuous_on U f and imf: f ' U ⊆ t and Qf: Q f
  obtains c where homotopic_with_canon Q U t f (λx. c)
proof -
  have feq:  $\bigwedge x. x \in U \implies (h \circ (k \circ f)) x = f x$  using idhk imf by auto
  have continuous_on U (k ∘ f)
    using conf continuous_on compose continuous_on_subset contk imf by blast
  moreover have (k ∘ f) ' U ⊆ s
    using imf imk by fastforce
  moreover have P (k ∘ f)
    by (simp add: P Qf conf imf)
  ultimately obtain c where homotopic_with_canon P U s (k ∘ f) (λx. c)
    by (metis hom)
  then have homotopic_with_canon Q U t (h ∘ (k ∘ f)) (h ∘ (λx. c))
  apply (rule homotopic_with_compose_continuous_left [OF homotopic_with_mono])
    using Q by (auto simp: conth imh)
  then have homotopic_with_canon Q U t f (λx. h c)
proof (rule homotopic_with_eq)
  show  $\bigwedge x. x \in \text{topspace } (\text{top\_of\_set } U) \implies f x = (h \circ (k \circ f)) x$ 
    using feq by auto
  show  $\bigwedge h k. (\bigwedge x. x \in \text{topspace } (\text{top\_of\_set } U) \implies h x = k x) \implies Q h = Q k$ 
    using Qeq topspace_euclidean_subtopology by blast
qed auto
then show ?thesis

```

using *that by blast*
qed

lemma *cohomotopically_trivial_retraction_gen*:
assumes $P: \bigwedge f. \llbracket \text{continuous\_on } t \ f; f' \ t \subseteq U; Q \ f \rrbracket \implies P(f \circ h)$
and $Q: \bigwedge f. \llbracket \text{continuous\_on } s \ f; f' \ s \subseteq U; P \ f \rrbracket \implies Q(f \circ k)$
and $Qeq: \bigwedge h \ k. (\bigwedge x. x \in t \implies h \ x = k \ x) \implies Q \ h = Q \ k$
and $hom: \bigwedge f \ g. \llbracket \text{continuous\_on } s \ f; f' \ s \subseteq U; P \ f; \text{continuous\_on } s \ g; g' \ s \subseteq U; P \ g \rrbracket \implies \text{homotopic\_with\_canon } P \ s \ U \ f \ g$
and $\text{contf: continuous\_on } t \ f$ **and** $\text{imf: } f' \ t \subseteq U$ **and** $Qf: Q \ f$
and $\text{contg: continuous\_on } t \ g$ **and** $\text{img: } g' \ t \subseteq U$ **and** $Qg: Q \ g$
shows $\text{homotopic\_with\_canon } Q \ t \ U \ f \ g$
proof –
have $\text{feq: } \bigwedge x. x \in t \implies (f \circ h \circ k) \ x = f \ x$ **using** *idhk imf by auto*
have $\text{geq: } \bigwedge x. x \in t \implies (g \circ h \circ k) \ x = g \ x$ **using** *idhk img by auto*
have $\text{continuous\_on } s \ (f \circ h)$
using $\text{contf conth continuous\_on\_compose imh by blast}$
moreover **have** $(f \circ h)' \ s \subseteq U$
using *imf imh by fastforce*
moreover **have** $P \ (f \circ h)$
by (*simp add: P Qf contf imf*)
moreover **have** $\text{continuous\_on } s \ (g \circ h)$
using $\text{contg continuous\_on\_compose continuous\_on\_subset conth imh by blast}$
moreover **have** $(g \circ h)' \ s \subseteq U$
using *img imh by fastforce*
moreover **have** $P \ (g \circ h)$
by (*simp add: P Qg contg img*)
ultimately **have** $\text{homotopic\_with\_canon } P \ s \ U \ (f \circ h) \ (g \circ h)$
by (*rule hom*)
then **have** $\text{homotopic\_with\_canon } Q \ t \ U \ (f \circ h \circ k) \ (g \circ h \circ k)$
apply (*rule homotopic_with_compose_continuous_right [OF homotopic_with_mono]*)
using Q **by** (*auto simp: contk imk*)
then **show** *?thesis*
proof (*rule homotopic_with_eq*)
show $f \ x = (f \circ h \circ k) \ x \ g \ x = (g \circ h \circ k) \ x$
if $x \in \text{topspace } (\text{top\_of\_set } t)$ **for** x
using $\text{feq geq that by force+}$
qed (*use Qeq topspace_euclidean_subtopology in blast*)
qed

lemma *cohomotopically_trivial_retraction_null_gen*:
assumes $P: \bigwedge f. \llbracket \text{continuous\_on } t \ f; f' \ t \subseteq U; Q \ f \rrbracket \implies P(f \circ h)$
and $Q: \bigwedge f. \llbracket \text{continuous\_on } s \ f; f' \ s \subseteq U; P \ f \rrbracket \implies Q(f \circ k)$
and $Qeq: \bigwedge h \ k. (\bigwedge x. x \in t \implies h \ x = k \ x) \implies Q \ h = Q \ k$
and $hom: \bigwedge f \ g. \llbracket \text{continuous\_on } s \ f; f' \ s \subseteq U; P \ f \rrbracket \implies \exists c. \text{homotopic\_with\_canon } P \ s \ U \ f \ (\lambda x. c)$
and $\text{contf: continuous\_on } t \ f$ **and** $\text{imf: } f' \ t \subseteq U$ **and** $Qf: Q \ f$
obtains c **where** $\text{homotopic\_with\_canon } Q \ t \ U \ f \ (\lambda x. c)$


```

proof –
  have feq:  $\bigwedge x. x \in t \implies (f \circ h \circ k) x = f x$  using idhk imf by auto
  have continuous_on s (f ∘ h)
    using conf conth continuous_on_compose imh by blast
  moreover have (f ∘ h) ‘ s ⊆ U
    using imf imh by fastforce
  moreover have P (f ∘ h)
    by (simp add: P Qf conf imf)
  ultimately obtain c where homotopic_with_canon P s U (f ∘ h) (λx. c)
    by (metis hom)
  then have §: homotopic_with_canon Q t U (f ∘ h ∘ k) ((λx. c) ∘ k)
  proof (rule homotopic_with_compose_continuous_right [OF homotopic_with_mono])
    show  $\bigwedge h. \llbracket \text{continuous\_map } (\text{top\_of\_set } s) (\text{top\_of\_set } U) h; P h \rrbracket \implies Q (h$ 
    ∘ k)
    using Q by auto
  qed (auto simp: contk imk)
  moreover have homotopic_with_canon Q t U f (λx. c)
    using homotopic_with_eq [OF §] feq Qeq by fastforce
  ultimately show ?thesis
    using that by blast
qed

end

```

```

lemma simply_connected_retraction_gen:
  shows  $\llbracket \text{simply\_connected } S; \text{continuous\_on } S h; h ‘ S = T; \text{continuous\_on } T k; k ‘ T \subseteq S; \bigwedge y. y \in T \implies h(k y) = y \rrbracket$ 
     $\implies \text{simply\_connected } T$ 
apply (simp add: simply_connected_def path_def path_image_def homotopic_loops_def,
  clarify)
apply (rule Retracts.homotopically_trivial_retraction_gen
  [of S h _ k _ λp. pathfinish p = pathstart p λp. pathfinish p = pathstart p])
apply (simp_all add: Retracts_def pathfinish_def pathstart_def)
done

```

```

lemma homeomorphic_simply_connected:
   $\llbracket S \text{ homeomorphic } T; \text{simply\_connected } S \rrbracket \implies \text{simply\_connected } T$ 
by (auto simp: homeomorphic_def homeomorphism_def intro: simply_connected_retraction_gen)

```

```

lemma homeomorphic_simply_connected_eq:
   $S \text{ homeomorphic } T \implies (\text{simply\_connected } S \longleftrightarrow \text{simply\_connected } T)$ 
by (metis homeomorphic_simply_connected homeomorphic_sym)

```

6.18.22 Homotopy equivalence

6.18.23 Homotopy equivalence of topological spaces.

```

definition homotopy_equivalent_space
  (infix homotopy'_equivalent'_space 50)
  where X homotopy_equivalent_space Y ≡

```

```

(∃ f g. continuous_map X Y f ∧
  continuous_map Y X g ∧
  homotopic_with (λx. True) X X (g ∘ f) id ∧
  homotopic_with (λx. True) Y Y (f ∘ g) id)

```

```

lemma homeomorphic_imp_homotopy_equivalent_space:
  X homeomorphic_space Y ⇒ X homotopy_equivalent_space Y
unfolding homeomorphic_space_def homotopy_equivalent_space_def
apply (erule ex_forward)+
by (simp add: homotopic_with_equal homotopic_with_sym homeomorphic_maps_def)

```

```

lemma homotopy_equivalent_space_refl:
  X homotopy_equivalent_space X
by (simp add: homeomorphic_imp_homotopy_equivalent_space homeomorphic_space_refl)

```

```

lemma homotopy_equivalent_space_sym:
  X homotopy_equivalent_space Y ⇔ Y homotopy_equivalent_space X
by (meson homotopy_equivalent_space_def)

```

```

lemma homotopy_eqv_trans [trans]:
assumes 1: X homotopy_equivalent_space Y and 2: Y homotopy_equivalent_space
U

```

shows X homotopy_equivalent_space U

proof –

```

obtain f1 g1 where f1: continuous_map X Y f1
and g1: continuous_map Y X g1
and hom1: homotopic_with (λx. True) X X (g1 ∘ f1) id
      homotopic_with (λx. True) Y Y (f1 ∘ g1) id
using 1 by (auto simp: homotopy_equivalent_space_def)
obtain f2 g2 where f2: continuous_map Y U f2
and g2: continuous_map U Y g2
and hom2: homotopic_with (λx. True) Y Y (g2 ∘ f2) id
      homotopic_with (λx. True) U U (f2 ∘ g2) id
using 2 by (auto simp: homotopy_equivalent_space_def)
have homotopic_with (λf. True) X Y (g2 ∘ f2 ∘ f1) (id ∘ f1)
using f1 hom2(1) homotopic_with_compose_continuous_map_right by metis
then have homotopic_with (λf. True) X Y (g2 ∘ (f2 ∘ f1)) (id ∘ f1)
by (simp add: o_assoc)
then have homotopic_with (λx. True) X X
  (g1 ∘ (g2 ∘ (f2 ∘ f1))) (g1 ∘ (id ∘ f1))
by (simp add: g1 homotopic_with_compose_continuous_map_left)
moreover have homotopic_with (λx. True) X X (g1 ∘ id ∘ f1) id
using hom1 by simp
ultimately have SS: homotopic_with (λx. True) X X (g1 ∘ g2 ∘ (f2 ∘ f1)) id
by (metis comp_assoc homotopic_with_trans id_comp)
have homotopic_with (λf. True) U Y (f1 ∘ g1 ∘ g2) (id ∘ g2)
using g2 hom1(2) homotopic_with_compose_continuous_map_right by fast-
force
then have homotopic_with (λf. True) U Y (f1 ∘ (g1 ∘ g2)) (id ∘ g2)

```

```

    by (simp add: o_assoc)
  then have homotopic_with ( $\lambda x. \text{True}$ )  $U\ U$ 
    (f2  $\circ$  (f1  $\circ$  (g1  $\circ$  g2))) (f2  $\circ$  (id  $\circ$  g2))
    by (simp add: f2 homotopic_with_compose_continuous_map_left)
  moreover have homotopic_with ( $\lambda x. \text{True}$ )  $U\ U$  (f2  $\circ$  id  $\circ$  g2) id
    using hom2 by simp
  ultimately have UU: homotopic_with ( $\lambda x. \text{True}$ )  $U\ U$  (f2  $\circ$  f1  $\circ$  (g1  $\circ$  g2)) id
    by (simp add: fun.map_comp hom2(2) homotopic_with_trans)
  show ?thesis
    unfolding homotopy_equivalent_space_def
    by (blast intro: f1 f2 g1 g2 continuous_map_compose SS UU)
qed

```

```

lemma deformation_retraction_imp_homotopy_equivalent_space:
   $\llbracket \text{homotopic\_with } (\lambda x. \text{True})\ X\ X\ (s \circ r)\ \text{id};\ \text{retraction\_maps}\ X\ Y\ r\ s \rrbracket$ 
 $\implies X\ \text{homotopy\_equivalent\_space}\ Y$ 
  unfolding homotopy_equivalent_space_def retraction_maps_def
  using homotopic_with_id2 by fastforce

```

```

lemma deformation_retract_imp_homotopy_equivalent_space:
   $\llbracket \text{homotopic\_with } (\lambda x. \text{True})\ X\ X\ r\ \text{id};\ \text{retraction\_maps}\ X\ Y\ r\ \text{id} \rrbracket$ 
 $\implies X\ \text{homotopy\_equivalent\_space}\ Y$ 
  using deformation_retraction_imp_homotopy_equivalent_space by force

```

```

lemma deformation_retract_of_space:
   $S \subseteq \text{topspace}\ X \wedge$ 
   $(\exists r. \text{homotopic\_with } (\lambda x. \text{True})\ X\ X\ \text{id}\ r \wedge \text{retraction\_maps}\ X\ (\text{subtopology}\ X\ S)\ r\ \text{id}) \iff$ 
   $S\ \text{retract\_of\_space}\ X \wedge (\exists f. \text{homotopic\_with } (\lambda x. \text{True})\ X\ X\ \text{id}\ f \wedge f\ '(\text{topspace}\ X) \subseteq S)$ 
  proof (cases  $S \subseteq \text{topspace}\ X$ )
    case True
      moreover have  $(\exists r. \text{homotopic\_with } (\lambda x. \text{True})\ X\ X\ \text{id}\ r \wedge \text{retraction\_maps}\ X\ (\text{subtopology}\ X\ S)\ r\ \text{id})$ 
         $\iff (S\ \text{retract\_of\_space}\ X \wedge (\exists f. \text{homotopic\_with } (\lambda x. \text{True})\ X\ X\ \text{id}\ f \wedge f\ '(\text{topspace}\ X) \subseteq S))$ 
        unfolding retract_of_space_def
      proof safe
        fix f r
        assume f: homotopic_with ( $\lambda x. \text{True}$ )  $X\ X\ \text{id}\ f$ 
        and fS:  $f\ '(\text{topspace}\ X) \subseteq S$ 
        and r: continuous_map  $X\ (\text{subtopology}\ X\ S)\ r$ 
        and req:  $\forall x \in S. r\ x = x$ 
        show  $\exists r. \text{homotopic\_with } (\lambda x. \text{True})\ X\ X\ \text{id}\ r \wedge \text{retraction\_maps}\ X\ (\text{subtopology}\ X\ S)\ r\ \text{id}$ 
          proof (intro exI conjI)
            have homotopic_with ( $\lambda x. \text{True}$ )  $X\ X\ f\ r$ 
              proof (rule homotopic_with_eq)
                show homotopic_with ( $\lambda x. \text{True}$ )  $X\ X\ (r \circ f)\ (r \circ \text{id})$ 

```

```

      by (metis continuous_map_into_fulltopology f homotopic_with_compose_continuous_map_left
homotopic_with_symD r)
      show  $f x = (r \circ f) x$  if  $x \in \text{topspace } X$  for  $x$ 
      using that fS req by auto
    qed auto
  then show homotopic_with ( $\lambda x. \text{True}$ )  $X X$  id  $r$ 
  by (rule homotopic_with_trans [OF f])
next
  show retraction_maps  $X (\text{subtopology } X S)$   $r$  id
  by (simp add: r req retraction_maps_def)
qed
qed (use True in ⟨auto simp: retraction_maps_def topspace_subtopology_subset
continuous_map_in_subtopology⟩)
ultimately show ?thesis by simp
qed (auto simp: retract_of_space_def retraction_maps_def)

```

6.18.24 Contractible spaces

The definition (which agrees with "contractible" on subsets of Euclidean space) is a little cryptic because we don't in fact assume that the constant "a" is in the space. This forces the convention that the empty space / set is contractible, avoiding some special cases.

definition *contractible_space* where

contractible_space $X \equiv \exists a. \text{homotopic\_with } (\lambda x. \text{True}) X X \text{ id } (\lambda x. a)$

lemma *contractible_space_top_of_set* [simp]: *contractible_space* (*top_of_set* S)
 $\longleftrightarrow$ *contractible* S

by (auto simp: *contractible_space_def contractible_def*)

lemma *contractible_space_empty*:

topspace $X = \{\}$ $\implies$ *contractible_space* X

unfolding *contractible_space_def homotopic_with_def*

apply (rule_tac $x = \text{undefined}$ in *exI*)

apply (rule_tac $x = \lambda(t, x). \text{if } t = 0 \text{ then } x \text{ else undefined}$ in *exI*)

apply (auto simp: *continuous_map_on_empty*)

done

lemma *contractible_space_singleton*:

topspace $X = \{a\}$ $\implies$ *contractible_space* X

unfolding *contractible_space_def homotopic_with_def*

apply (rule_tac $x = a$ in *exI*)

apply (rule_tac $x = \lambda(t, x). \text{if } t = 0 \text{ then } x \text{ else } a$ in *exI*)

apply (auto intro: *continuous_map_eq* [where $f = \lambda z. a$])

done

lemma *contractible_space_subset_singleton*:

topspace $X \subseteq \{a\}$ $\implies$ *contractible_space* X

by (meson *contractible_space_empty contractible_space_singleton subset_singletonD*)

```

lemma contractible_space_subtopology_singleton:
  contractible_space(subtopology X {a})
  by (meson contractible_space_subset_singleton insert_subset path_connectedin_singleton
  path_connectedin_subtopology subsetI)

lemma contractible_space:
  contractible_space X  $\longleftrightarrow$ 
    topspace X = {}  $\vee$ 
    ( $\exists a \in \text{topspace } X. \text{homotopic\_with } (\lambda x. \text{True}) X X \text{id } (\lambda x. a)$ )
proof (cases topspace X = {})
  case False
  then show ?thesis
    using homotopic_with_imp_continuous_maps by (fastforce simp: contractible_space_def)
qed (simp add: contractible_space_empty)

lemma contractible_imp_path_connected_space:
  assumes contractible_space X shows path_connected_space X
proof (cases topspace X = {})
  case False
  have *: path_connected_space X
    if  $a \in \text{topspace } X$  and  $\text{conth}: \text{continuous\_map } (\text{prod\_topology } (\text{top\_of\_set } \{0..1\}) X) X h$ 
    and  $h: \forall x. h(0, x) = x \ \forall x. h(1, x) = a$ 
    for  $a$  and  $h :: \text{real} \times 'a \Rightarrow 'a$ 
  proof -
    have path_component_of X b a if  $b \in \text{topspace } X$  for  $b$ 
    unfolding path_component_of_def
    proof (intro exI conjI)
      let ?g =  $h \circ (\lambda x. (x, b))$ 
      show pathin X ?g
      unfolding pathin_def
    proof (rule continuous_map_compose [OF _ conth])
      show continuous_map (top_of_set {0..1}) (prod_topology (top_of_set {0..1}) X) ( $\lambda x. (x, b)$ )
      using that by (auto intro!: continuous_intros)
    qed
    qed (use h in auto)
  then show ?thesis
    by (metis path_component_of_equiv path_connected_space_iff_path_component)
  qed
show ?thesis
    using assms False by (auto simp: contractible_space homotopic_with_def *)
qed (simp add: path_connected_space_topspace_empty)

lemma contractible_imp_connected_space:
  contractible_space X  $\implies$  connected_space X
  by (simp add: contractible_imp_path_connected_space path_connected_imp_connected_space)

```

lemma *contractible_space_alt*:

contractible_space $X \iff (\forall a \in \text{topspace } X. \text{homotopic\_with } (\lambda x. \text{True}) X X \text{ id } (\lambda x. a))$ (**is** ?lhs = ?rhs)

proof

assume X : ?lhs

then obtain a **where** a : *homotopic_with* $(\lambda x. \text{True}) X X \text{ id } (\lambda x. a)$

by (*auto simp: contractible_space_def*)

show ?rhs

proof

show *homotopic_with* $(\lambda x. \text{True}) X X \text{ id } (\lambda x. b)$ **if** $b \in \text{topspace } X$ **for** b

proof (*rule homotopic_with_trans [OF a]*)

show *homotopic_with* $(\lambda x. \text{True}) X X (\lambda x. a) (\lambda x. b)$

using *homotopic_constant_maps path_connected_space_imp_path_component_of*

by (*metis (full_types) X a continuous_map_const contractible_imp_path_connected_space homotopic_with_imp_continuous_maps that*)

qed

qed

next

assume R : ?rhs

then show ?lhs

unfolding *contractible_space_def* **by** (*metis equals0I homotopic_on_emptyI*)

qed

lemma *compose_const* [*simp*]: $f \circ (\lambda x. a) = (\lambda x. f a) (\lambda x. a) \circ g = (\lambda x. a)$

by (*simp_all add: o_def*)

lemma *nullhomotopic_through_contractible_space*:

assumes f : *continuous_map* $X Y f$ **and** g : *continuous_map* $Y Z g$ **and** Y : *contractible_space* Y

obtains c **where** *homotopic_with* $(\lambda h. \text{True}) X Z (g \circ f) (\lambda x. c)$

proof –

obtain b **where** b : *homotopic_with* $(\lambda x. \text{True}) Y Y \text{ id } (\lambda x. b)$

using Y **by** (*auto simp: contractible_space_def*)

show *thesis*

using *homotopic_with_compose_continuous_map_right*

[*OF homotopic_with_compose_continuous_map_left [OF b g] f*]

by (*force simp add: that*)

qed

lemma *nullhomotopic_into_contractible_space*:

assumes f : *continuous_map* $X Y f$ **and** Y : *contractible_space* Y

obtains c **where** *homotopic_with* $(\lambda h. \text{True}) X Y f (\lambda x. c)$

using *nullhomotopic_through_contractible_space [OF f _ Y]*

by (*metis continuous_map_id id_comp*)

lemma *nullhomotopic_from_contractible_space*:

assumes f : *continuous_map* $X Y f$ **and** X : *contractible_space* X

obtains c **where** *homotopic_with* $(\lambda h. \text{True}) X Y f (\lambda x. c)$

```

using nullhomotopic_through_contractible_space [OF _ f X]
by (metis comp_id continuous_map_id)

lemma homotopy_dominated_contractibility:
  assumes f: continuous_map X Y f and g: continuous_map Y X g
  and hom: homotopic_with ( $\lambda x. \text{True}$ ) Y Y (f  $\circ$  g) id and X: contractible_space X
  shows contractible_space Y
proof -
  obtain c where c: homotopic_with ( $\lambda h. \text{True}$ ) X Y f ( $\lambda x. c$ )
  using nullhomotopic_from_contractible_space [OF f X] .
  have homotopic_with ( $\lambda x. \text{True}$ ) Y Y (f  $\circ$  g) ( $\lambda x. c$ )
  using homotopic_with_compose_continuous_map_right [OF c g] by fastforce
  then have homotopic_with ( $\lambda x. \text{True}$ ) Y Y id ( $\lambda x. c$ )
  using homotopic_with_trans [OF _ hom] homotopic_with_symD by blast
  then show ?thesis
  unfolding contractible_space_def ..
qed

lemma homotopy_equivalent_space_contractibility:
  X homotopy_equivalent_space Y  $\implies$  (contractible_space X  $\longleftrightarrow$  contractible_space Y)
  unfolding homotopy_equivalent_space_def
  by (blast intro: homotopy_dominated_contractibility)

lemma homeomorphic_space_contractibility:
  X homeomorphic_space Y
   $\implies$  (contractible_space X  $\longleftrightarrow$  contractible_space Y)
  by (simp add: homeomorphic_imp_homotopy_equivalent_space homotopy_equivalent_space_contractibility)

lemma contractible_eq_homotopy_equivalent_singleton_subtopology:
  contractible_space X  $\longleftrightarrow$ 
  (topspace X = {}  $\vee$  ( $\exists a \in \text{topspace } X. X \text{ homotopy\_equivalent\_space}$ 
  (subtopology X {a}))(is ?lhs = ?rhs))
proof (cases topspace X = {})
  case False
  show ?thesis
  proof
    assume ?lhs
    then obtain a where a: homotopic_with ( $\lambda x. \text{True}$ ) X X id ( $\lambda x. a$ )
    by (auto simp: contractible_space_def)
    then have a  $\in$  topspace X
    by (metis False continuous_map_const homotopic_with_imp_continuous_maps)
    then have homotopic_with ( $\lambda x. \text{True}$ ) (subtopology X {a}) (subtopology X {a}) id ( $\lambda x. a$ )
    using connectedin_absolute connectedin_sing contractible_space_alt contractible_space_subtopology_singleton by fastforce
    then have X homotopy_equivalent_space subtopology X {a}
    unfolding homotopy_equivalent_space_def using  $\langle a \in \text{topspace } X \rangle$ 
  qed

```

```

    by (metis (full_types) a comp_id continuous_map_const continuous_map_id_subt
empty_subsetI homotopic_with_symD
    id_comp insertI1 insert_subset topspace_subtopology_subset)
  with ⟨a ∈ topspace X⟩ show ?rhs
    by blast
next
  assume ?rhs
  then show ?lhs
    by (meson False contractible_space_subtopology_singleton homotopy_equivalent_space_contractibili
qed
qed (simp add: contractible_space_empty)

```

```

lemma contractible_space_retraction_map_image:
  assumes retraction_map X Y f and X: contractible_space X
  shows contractible_space Y
proof -
  obtain g where f: continuous_map X Y f and g: continuous_map Y X g and
fg:  $\forall y \in \text{topspace } Y. f(g\ y) = y$ 
    using assms by (auto simp: retraction_map_def retraction_maps_def)
  obtain a where a: homotopic_with ( $\lambda x. \text{True}$ ) X X id ( $\lambda x. a$ )
    using X by (auto simp: contractible_space_def)
  have homotopic_with ( $\lambda x. \text{True}$ ) Y Y id ( $\lambda x. f\ a$ )
  proof (rule homotopic_with_eq)
    show homotopic_with ( $\lambda x. \text{True}$ ) Y Y (f ∘ id ∘ g) (f ∘ ( $\lambda x. a$ ) ∘ g)
    using fg a homotopic_with_compose_continuous_map_left homotopic_with_compose_continuous
  by metis
qed (use fg in auto)
then show ?thesis
  unfolding contractible_space_def by blast
qed

```

```

lemma contractible_space_prod_topology:
  contractible_space(prod_topology X Y)  $\longleftrightarrow$ 
  topspace X = {}  $\vee$  topspace Y = {}  $\vee$  contractible_space X  $\wedge$  contractible_space
Y
proof (cases topspace X = {}  $\vee$  topspace Y = {})
case True
  then have topspace (prod_topology X Y) = {}
    by simp
  then show ?thesis
    by (auto simp: contractible_space_empty)
next
case False
  have contractible_space(prod_topology X Y)  $\longleftrightarrow$  contractible_space X  $\wedge$  con-
tractible_space Y
  proof safe
    assume XY: contractible_space (prod_topology X Y)
    with False have retraction_map (prod_topology X Y) X fst
      by (auto simp: contractible_space False retraction_map_fst)

```



```

then show contractible_space X
  by (rule contractible_space_retraction_map_image [OF _ XY])
have retraction_map (prod_topology X Y) Y snd
  using False XY by (auto simp: contractible_space False retraction_map_snd)
then show contractible_space Y
  by (rule contractible_space_retraction_map_image [OF _ XY])
next
assume contractible_space X and contractible_space Y
with False obtain a b
  where a ∈ topspace X and a: homotopic_with (λx. True) X X id (λx. a)
    and b ∈ topspace Y and b: homotopic_with (λx. True) Y Y id (λx. b)
  by (auto simp: contractible_space)
with False show contractible_space (prod_topology X Y)
  apply (simp add: contractible_space)
  apply (rule_tac x=a in bexI)
  apply (rule_tac x=b in bexI)
  using homotopic_with_prod_topology [OF a b]
  apply (metis (no_types, lifting) case_prod_Pair case_prod_beta' eq_id_iff)
  apply auto
  done
qed
with False show ?thesis
  by auto
qed

```

```

lemma contractible_space_product_topology:
  contractible_space(product_topology X I) ⟷
    topspace (product_topology X I) = {} ∨ (∀ i ∈ I. contractible_space(X i))
proof (cases topspace (product_topology X I) = {})
case False
  have 1: contractible_space (X i)
    if XI: contractible_space (product_topology X I) and i ∈ I
    for i
  proof (rule contractible_space_retraction_map_image [OF _ XI])
    show retraction_map (product_topology X I) (X i) (λx. x i)
      using False by (simp add: retraction_map_product_projection ⟨i ∈ I⟩)
  qed
  have 2: contractible_space (product_topology X I)
    if x ∈ topspace (product_topology X I) and cs: ∀ i ∈ I. contractible_space (X i)
    for x :: 'a ⇒ 'b
  proof -
    obtain f where f: ∧ i. i ∈ I ⟹ homotopic_with (λx. True) (X i) (X i) id (λx.
f i)
    using cs unfolding contractible_space_def by metis
    have homotopic_with (λx. True)
      (product_topology X I) (product_topology X I) id (λx. restrict
f I)
    by (rule homotopic_with_eq [OF homotopic_with_product_topology [OF f]])

```

```

(auto)
  then show ?thesis
    by (auto simp: contractible_space_def)
  qed
  show ?thesis
    using False 1 2 by blast
qed (simp add: contractible_space_empty)

lemma contractible_space_subtopology_euclideanreal [simp]:
  contractible_space(subtopology euclideanreal S)  $\longleftrightarrow$  is_interval S
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then have path_connectedin (subtopology euclideanreal S) S
    using contractible_imp_path_connected_space path_connectedin_topspace path_connectedin_absolut
    by (simp add: contractible_imp_path_connected)
  then show ?rhs
    by (simp add: is_interval_path_connected_1)
next
  assume ?rhs
  then have convex S
    by (simp add: is_interval_convex_1)
  show ?lhs
  proof (cases S = {})
    case False
    then obtain z where z  $\in$  S
      by blast
    show ?thesis
      unfolding contractible_space_def homotopic_with_def
      proof (intro exI conjI allI)
        note § = convexD [OF <convex S>, simplified]
        show continuous_map (prod_topology (top_of_set {0..1}) (top_of_set S))
          (top_of_set S)
          ( $\lambda(t,x). (1 - t) * x + t * z$ )
          using <z  $\in$  S>
          by (auto simp add: case_prod_unfold intro!: continuous_intros §)
      qed auto
    qed (simp add: contractible_space_empty)
  qed

corollary contractible_space_euclideanreal: contractible_space euclideanreal
proof -
  have contractible_space (subtopology euclideanreal UNIV)
    using contractible_space_subtopology_euclideanreal by blast
  then show ?thesis
    by simp
qed

```

abbreviation $\text{homotopy\_eqv} :: 'a::\text{topological\_space set} \Rightarrow 'b::\text{topological\_space set} \Rightarrow \text{bool}$
 $\quad (\text{infix homotopy\_eqv } 50)$
where $S \text{ homotopy\_eqv } T \equiv \text{top\_of\_set } S \text{ homotopy\_equivalent\_space top\_of\_set } T$

lemma $\text{homeomorphic\_imp\_homotopy\_eqv}: S \text{ homeomorphic } T \Longrightarrow S \text{ homotopy\_eqv } T$
unfolding $\text{homeomorphic\_def homeomorphism\_def homotopy\_equivalent\_space\_def}$
by $(\text{metis continuous\_map\_subtopology\_eu homotopic\_with\_id2 openin\_imp\_subset openin\_subtopology\_self topspace\_euclidean\_subtopology})$

lemma $\text{homotopy\_eqv\_inj\_linear\_image}$:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes $\text{linear } f \text{ inj } f$
shows $(f \text{ ' } S) \text{ homotopy\_eqv } S$
by $(\text{metis assms homeomorphic\_sym homeomorphic\_imp\_homotopy\_eqv linear\_homeomorphic\_image})$

lemma $\text{homotopy\_eqv\_translation}$:
fixes $S :: 'a::\text{real\_normed\_vector set}$
shows $(+) a \text{ ' } S \text{ homotopy\_eqv } S$
using $\text{homeomorphic\_imp\_homotopy\_eqv homeomorphic\_translation homeomorphic\_sym}$ **by** blast

lemma $\text{homotopy\_eqv\_homotopic\_triviality\_imp}$:
fixes $S :: 'a::\text{real\_normed\_vector set}$
and $T :: 'b::\text{real\_normed\_vector set}$
and $U :: 'c::\text{real\_normed\_vector set}$
assumes $S \text{ homotopy\_eqv } T$
and $f: \text{continuous\_on } U \text{ f f ' } U \subseteq T$
and $g: \text{continuous\_on } U \text{ g g ' } U \subseteq T$
and $\text{homUS}: \bigwedge g. [\text{continuous\_on } U \text{ f; f ' } U \subseteq S; \text{continuous\_on } U \text{ g; g ' } U \subseteq S] \Longrightarrow \text{homotopic\_with\_canon } (\lambda x. \text{True}) \text{ U S f g}$
shows $\text{homotopic\_with\_canon } (\lambda x. \text{True}) \text{ U T f g}$

proof –

obtain $h \text{ k where } h: \text{continuous\_on } S \text{ h h ' } S \subseteq T$
and $k: \text{continuous\_on } T \text{ k k ' } T \subseteq S$
and $\text{hom}: \text{homotopic\_with\_canon } (\lambda x. \text{True}) \text{ S S (k } \circ \text{ h) id}$
 $\text{homotopic\_with\_canon } (\lambda x. \text{True}) \text{ T T (h } \circ \text{ k) id}$
using $\text{assms by (auto simp: homotopy\_equivalent\_space\_def)}$

```

have homotopic_with_canon ( $\lambda f. \text{True}$ )  $U S (k \circ f) (k \circ g)$ 
proof (rule homUS)
  show continuous_on  $U (k \circ f)$  continuous_on  $U (k \circ g)$ 
  using continuous_on_compose continuous_on_subset  $f g k$  by blast+
qed (use  $f g k$  in  $\langle (\text{force simp: } o\_def) + \rangle$ )
then have homotopic_with_canon ( $\lambda x. \text{True}$ )  $U T (h \circ (k \circ f)) (h \circ (k \circ g))$ 
  by (rule homotopic_with_compose_continuous_left; simp add:  $h$ )
moreover have homotopic_with_canon ( $\lambda x. \text{True}$ )  $U T (h \circ k \circ f) (id \circ f)$ 
  by (rule homotopic_with_compose_continuous_right [where  $X=T$  and  $Y=T$ ];
simp add: hom  $f$ )
moreover have homotopic_with_canon ( $\lambda x. \text{True}$ )  $U T (h \circ k \circ g) (id \circ g)$ 
  by (rule homotopic_with_compose_continuous_right [where  $X=T$  and  $Y=T$ ];
simp add: hom  $g$ )
ultimately show homotopic_with_canon ( $\lambda x. \text{True}$ )  $U T f g$ 
  unfolding  $o\_assoc$ 
  by (metis homotopic_with_trans homotopic_with_sym  $id\_comp$ )
qed

```

```

lemma homotopy_eqv_homotopic_triviality:
  fixes  $S :: 'a::real\_normed\_vector\_set$ 
  and  $T :: 'b::real\_normed\_vector\_set$ 
  and  $U :: 'c::real\_normed\_vector\_set$ 
  assumes  $S \text{ homotopy\_eqv } T$ 
  shows  $(\forall f g. \text{continuous\_on } U f \wedge f ' U \subseteq S \wedge$ 
     $\text{continuous\_on } U g \wedge g ' U \subseteq S$ 
     $\longrightarrow \text{homotopic\_with\_canon } (\lambda x. \text{True}) U S f g) \longleftrightarrow$ 
 $(\forall f g. \text{continuous\_on } U f \wedge f ' U \subseteq T \wedge$ 
     $\text{continuous\_on } U g \wedge g ' U \subseteq T$ 
     $\longrightarrow \text{homotopic\_with\_canon } (\lambda x. \text{True}) U T f g)$ 
    (is  $?lhs = ?rhs$ )
proof
  assume  $?lhs$ 
  then show  $?rhs$ 
  by (metis assms homotopy_eqv_homotopic_triviality_imp)
next
  assume  $?rhs$ 
  moreover
  have  $T \text{ homotopy\_eqv } S$ 
  using assms homotopy_equivalent_space_sym by blast
  ultimately show  $?lhs$ 
  by (blast intro: homotopy_eqv_homotopic_triviality_imp)
qed

```

```

lemma homotopy_eqv_cohomotopic_triviality_null_imp:
  fixes  $S :: 'a::real\_normed\_vector\_set$ 
  and  $T :: 'b::real\_normed\_vector\_set$ 
  and  $U :: 'c::real\_normed\_vector\_set$ 
  assumes  $S \text{ homotopy\_eqv } T$ 

```

```

    and f: continuous_on T f f ' T  $\subseteq$  U
    and homSU:  $\bigwedge f. \llbracket \text{continuous\_on } S f; f ' S \subseteq U \rrbracket$ 
                 $\implies \exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S U f (\lambda x. c)$ 
    obtains c where homotopic_with_canon ( $\lambda x. \text{True}$ ) T U f ( $\lambda x. c$ )
  proof -
    obtain h k where h: continuous_on S h h ' S  $\subseteq$  T
      and k: continuous_on T k k ' T  $\subseteq$  S
      and hom: homotopic_with_canon ( $\lambda x. \text{True}$ ) S S (k  $\circ$  h) id
                homotopic_with_canon ( $\lambda x. \text{True}$ ) T T (h  $\circ$  k) id
    using assms by (auto simp: homotopy_equivalent_space_def)
    obtain c where homotopic_with_canon ( $\lambda x. \text{True}$ ) S U (f  $\circ$  h) ( $\lambda x. c$ )
    proof (rule exE [OF homSU])
      show continuous_on S (f  $\circ$  h)
        using continuous_on_compose continuous_on_subset f h by blast
    qed (use f h in force)
    then have homotopic_with_canon ( $\lambda x. \text{True}$ ) T U ((f  $\circ$  h)  $\circ$  k) (( $\lambda x. c$ )  $\circ$  k)
      by (rule homotopic_with_compose_continuous_right [where X=S]) (use k in
    auto)
    moreover have homotopic_with_canon ( $\lambda x. \text{True}$ ) T U (f  $\circ$  id) (f  $\circ$  (h  $\circ$  k))
      by (rule homotopic_with_compose_continuous_left [where Y=T])
        (use f in  $\langle$ auto simp add: hom homotopic_with_symD $\rangle$ )
    ultimately show ?thesis
      using that homotopic_with_trans by (fastforce simp add: o_def)
  qed

```

```

lemma homotopy_eqv_cohomotopic_triviality_null:
  fixes S :: 'a::real_normed_vector set
  and T :: 'b::real_normed_vector set
  and U :: 'c::real_normed_vector set
  assumes S homotopy_eqv T
  shows ( $\forall f. \text{continuous\_on } S f \wedge f ' S \subseteq U$ 
     $\longrightarrow (\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S U f (\lambda x. c))$ )  $\longleftrightarrow$ 
    ( $\forall f. \text{continuous\_on } T f \wedge f ' T \subseteq U$ 
     $\longrightarrow (\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) T U f (\lambda x. c))$ )
  by (rule iffI; metis assms homotopy_eqv_cohomotopic_triviality_null_imp homotopy_equivalent_space_sym)

```

Similar to the proof above

```

lemma homotopy_eqv_homotopic_triviality_null_imp:
  fixes S :: 'a::real_normed_vector set
  and T :: 'b::real_normed_vector set
  and U :: 'c::real_normed_vector set
  assumes S homotopy_eqv T
  and f: continuous_on U f f ' U  $\subseteq$  T
  and homSU:  $\bigwedge f. \llbracket \text{continuous\_on } U f; f ' U \subseteq S \rrbracket$ 
                 $\implies \exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) U S f (\lambda x. c)$ 
  shows  $\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) U T f (\lambda x. c)$ 
  proof -
    obtain h k where h: continuous_on S h h ' S  $\subseteq$  T

```

```

    and k: continuous_on T k k ' T ⊆ S
    and hom: homotopic_with_canon (λx. True) S S (k ∘ h) id
              homotopic_with_canon (λx. True) T T (h ∘ k) id
    using assms by (auto simp: homotopy_equivalent_space_def)
  obtain c::'a where homotopic_with_canon (λx. True) U S (k ∘ f) (λx. c)
  proof (rule exE [OF homSU [of k ∘ f]])
    show continuous_on U (k ∘ f)
      using continuous_on_compose continuous_on_subset f k by blast
  qed (use f k in force)
  then have homotopic_with_canon (λx. True) U T (h ∘ (k ∘ f)) (h ∘ (λx. c))
    by (rule homotopic_with_compose_continuous_left [where Y=S]) (use h in
  auto)
  moreover have homotopic_with_canon (λx. True) U T (id ∘ f) ((h ∘ k) ∘ f)
    by (rule homotopic_with_compose_continuous_right [where X=T])
      (use f in ⟨auto simp add: hom homotopic_with_symD⟩)
  ultimately show ?thesis
    using homotopic_with_trans by (fastforce simp add: o_def)
  qed

```

```

lemma homotopy_eqv_homotopic_triviality_null:
  fixes S :: 'a::real_normed_vector set
    and T :: 'b::real_normed_vector set
    and U :: 'c::real_normed_vector set
  assumes S homotopy_eqv T
  shows (∀ f. continuous_on U f ∧ f ' U ⊆ S
    → (∃ c. homotopic_with_canon (λx. True) U S f (λx. c))) ↔
    (∀ f. continuous_on U f ∧ f ' U ⊆ T
    → (∃ c. homotopic_with_canon (λx. True) U T f (λx. c)))
  by (rule iffI; metis assms homotopy_eqv_homotopic_triviality_null_imp homo-
    topy_equivalent_space_sym)

```

```

lemma homotopy_eqv_contractible_sets:
  fixes S :: 'a::real_normed_vector set
    and T :: 'b::real_normed_vector set
  assumes contractible S contractible T S = {} ↔ T = {}
  shows S homotopy_eqv T
  proof (cases S = {})
    case True with assms show ?thesis
      by (simp add: homeomorphic_imp_homotopy_eqv)
    next
      case False
    with assms obtain a b where a ∈ S b ∈ T
      by auto
    then show ?thesis
      unfolding homotopy_equivalent_space_def
      apply (rule_tac x=λx. b in exI, rule_tac x=λx. a in exI)
      apply (intro assms conjI continuous_on_id' homotopic_into_contractible;
      force)
      done
  qed

```

qed

```
lemma homotopy_eqv_empty1 [simp]:
  fixes S :: 'a::real_normed_vector set
  shows S homotopy_eqv ({ }::'b::real_normed_vector set)  $\longleftrightarrow$  S = { } (is ?lhs =
    ?rhs)
proof
  assume ?lhs then show ?rhs
  by (metis continuous_map_subtopology_eu_empty_iff_equalityI homotopy_equivalent_space_def
    image_subset_iff subsetI)
qed (simp add: homotopy_eqv_contractible_sets)
```

```
lemma homotopy_eqv_empty2 [simp]:
  fixes S :: 'a::real_normed_vector set
  shows ({ }::'b::real_normed_vector set) homotopy_eqv S  $\longleftrightarrow$  S = { }
  using homotopy_equivalent_space_sym homotopy_eqv_empty1 by blast
```

```
lemma homotopy_eqv_contractibility:
  fixes S :: 'a::real_normed_vector set and T :: 'b::real_normed_vector set
  shows S homotopy_eqv T  $\implies$  (contractible S  $\longleftrightarrow$  contractible T)
  by (meson contractible_space_top_of_set homotopy_equivalent_space_contractibility)
```

```
lemma homotopy_eqv_sing:
  fixes S :: 'a::real_normed_vector set and a :: 'b::real_normed_vector
  shows S homotopy_eqv {a}  $\longleftrightarrow$  S  $\neq$  { }  $\wedge$  contractible S
proof (cases S = { })
  case False then show ?thesis
  by (metis contractible_sing_empty_not_insert homotopy_eqv_contractibility
    homotopy_eqv_contractible_sets)
qed simp
```

```
lemma homeomorphic_contractible_eq:
  fixes S :: 'a::real_normed_vector set and T :: 'b::real_normed_vector set
  shows S homeomorphic T  $\implies$  (contractible S  $\longleftrightarrow$  contractible T)
  by (simp add: homeomorphic_imp_homotopy_eqv homotopy_eqv_contractibility)
```

```
lemma homeomorphic_contractible:
  fixes S :: 'a::real_normed_vector set and T :: 'b::real_normed_vector set
  shows  $\llbracket$ contractible S; S homeomorphic T $\rrbracket \implies$  contractible T
  by (metis homeomorphic_contractible_eq)
```

6.18.25 Misc other results

```
lemma bounded_connected_Compl_real:
  fixes S :: real set
  assumes bounded S and conn: connected(– S)
  shows S = { }
proof –
  obtain a b where S  $\subseteq$  box a b
```

```

    by (meson assms bounded_subset_box_symmetric)
  then have  $a \notin S \wedge b \notin S$ 
    by auto
  then have  $\forall x. a \leq x \wedge x \leq b \longrightarrow x \in -S$ 
    by (meson Compl_iff conn_connected_iff_interval)
  then show ?thesis
    using  $\langle S \subseteq \text{box } a \ b \rangle$  by auto
qed

```

corollary *bounded_path_connected_Compl_real*:

```

  fixes  $S :: \text{real set}$ 
  assumes bounded  $S$  path_connected( $-S$ ) shows  $S = \{\}$ 
  by (simp add: assms bounded_connected_Compl_real path_connected_imp_connected)

```

lemma *bounded_connected_Compl_1*:

```

  fixes  $S :: 'a::\{\text{euclidean\_space}\} \text{ set}$ 
  assumes bounded  $S$  and conn: connected( $-S$ ) and  $1: \text{DIM}('a) = 1$ 
  shows  $S = \{\}$ 
proof -
  have  $\text{DIM}('a) = \text{DIM}(\text{real})$ 
    by (simp add: 1)
  then obtain  $f::'a \Rightarrow \text{real}$  and  $g$ 
  where linear  $f$   $\bigwedge x. \text{norm}(f\ x) = \text{norm } x$  and  $fg: \bigwedge x. g(f\ x) = x \bigwedge y. f(g\ y) = y$ 
    by (rule isomorphisms_UNIV_UNIV) blast
  with  $\langle \text{bounded } S \rangle$  have bounded  $(f^{-1} S)$ 
    using bounded_linear_image_linear_linear by blast
  have bij  $f$  by (metis  $fg$  bijI')
  have connected  $(f^{-1}(-S))$ 
    using connected_linear_image assms  $\langle \text{linear } f \rangle$  by blast
  moreover have  $f^{-1}(-S) = -(f^{-1} S)$ 
    by (simp add:  $\langle \text{bij } f \rangle$  bij_image_Compl_eq)
  finally have connected  $(-(f^{-1} S))$ 
    by simp
  then have  $f^{-1} S = \{\}$ 
    using  $\langle \text{bounded } (f^{-1} S) \rangle$  bounded_connected_Compl_real by blast
  then show ?thesis
    by blast
qed

```

6.18.26 Some Uncountable Sets

lemma *uncountable_closed_segment*:

```

  fixes  $a :: 'a::\text{real\_normed\_vector}$ 
  assumes  $a \neq b$  shows uncountable (closed_segment  $a\ b$ )
unfolding path_image_linepath [symmetric] path_image_def
  using inj_on_linepath [OF assms] uncountable_closed_interval [of 0 1]
    countable_image_inj_on by auto

```

lemma *uncountable_open_segment*:


```

fixes  $a :: 'a::\text{real\_normed\_vector}$ 
assumes  $a \neq b$  shows  $\text{uncountable } (\text{open\_segment } a \ b)$ 
by (simp add: assms open_segment_def uncountable_closed_segment uncountable_minus_countable)

```

```

lemma  $\text{uncountable\_convex}$ :
fixes  $a :: 'a::\text{real\_normed\_vector}$ 
assumes  $\text{convex } S \ a \in S \ b \in S \ a \neq b$ 
shows  $\text{uncountable } S$ 
proof –
have  $\text{uncountable } (\text{closed\_segment } a \ b)$ 
by (simp add: uncountable_closed_segment assms)
then show ?thesis
by (meson assms convex_contains_segment countable_subset)
qed

```

```

lemma  $\text{uncountable\_ball}$ :
fixes  $a :: 'a::\text{euclidean\_space}$ 
assumes  $r > 0$ 
shows  $\text{uncountable } (\text{ball } a \ r)$ 
proof –
have  $\text{uncountable } (\text{open\_segment } a \ (a + r *_{\mathbb{R}} (\text{SOME } i. i \in \text{Basis})))$ 
by (metis Basis_zero SOME_Basis add_cancel_right_right assms less_le scale_eq_0_iff uncountable_open_segment)
moreover have  $\text{open\_segment } a \ (a + r *_{\mathbb{R}} (\text{SOME } i. i \in \text{Basis})) \subseteq \text{ball } a \ r$ 
using assms by (auto simp: in_segment algebra_simps dist_norm SOME_Basis)
ultimately show ?thesis
by (metis countable_subset)
qed

```

```

lemma  $\text{ball\_minus\_countable\_nonempty}$ :
assumes  $\text{countable } (A :: 'a :: \text{euclidean\_space set}) \ r > 0$ 
shows  $\text{ball } z \ r - A \neq \{\}$ 
proof
assume *:  $\text{ball } z \ r - A = \{\}$ 
have  $\text{uncountable } (\text{ball } z \ r - A)$ 
by (intro uncountable_minus_countable assms uncountable_ball)
thus False by (subst (asm) *) auto
qed

```

```

lemma  $\text{uncountable\_cball}$ :
fixes  $a :: 'a::\text{euclidean\_space}$ 
assumes  $r > 0$ 
shows  $\text{uncountable } (\text{cball } a \ r)$ 
using assms countable_subset uncountable_ball by auto

```

```

lemma  $\text{pairwise\_disjnt\_countable}$ :
fixes  $\mathcal{N} :: \text{nat set set}$ 
assumes  $\text{pairwise disjnt } \mathcal{N}$ 

```

```

    shows countable  $\mathcal{N}$ 
  proof -
    have inj_on  $(\lambda X. \text{SOME } n. n \in X) (\mathcal{N} - \{\{\}\})$ 
      by (clarsimp simp: inj_on_def) (metis assms disjoint_iff pairwiseD some_in_eq)
    then show ?thesis
      by (metis countable_Diff_eq countable_def)
  qed

```

```

lemma pairwise_disjnt_countable_Union:
  assumes countable  $(\bigcup \mathcal{N})$  and pwd: pairwise disjnt  $\mathcal{N}$ 
  shows countable  $\mathcal{N}$ 
proof -
  obtain  $f :: \_ \Rightarrow \text{nat}$  where  $f: \text{inj\_on } f (\bigcup \mathcal{N})$ 
    using assms by blast
  then have pairwise_disjnt  $(\bigcup X \in \mathcal{N}. \{f \text{ ` } X\})$ 
    using assms by (force simp: pairwise_def disjoint_inj_on_iff [OF f])
  then have countable  $(\bigcup X \in \mathcal{N}. \{f \text{ ` } X\})$ 
    using pairwise_disjnt_countable by blast
  then show ?thesis
    by (meson pwd countable_image_inj_on disjoint_image f inj_on_image pairwise_disjnt_countable)
  qed

```

```

lemma connected_uncountable:
  fixes  $S :: 'a::\text{metric\_space}$  set
  assumes connected  $S$   $a \in S$   $b \in S$   $a \neq b$  shows uncountable  $S$ 
proof -
  have continuous_on  $S$  (dist  $a$ )
    by (intro continuous_intros)
  then have connected (dist  $a$  `  $S$ )
    by (metis connected_continuous_image <connected  $S$ >)
  then have closed_segment 0 (dist  $a$   $b$ )  $\subseteq$  (dist  $a$  `  $S$ )
    by (simp add: assms closed_segment_subset is_interval_connected_1 is_interval_convex)
  then have uncountable (dist  $a$  `  $S$ )
    by (metis < $a \neq b$ > countable_subset dist_eq_0_iff uncountable_closed_segment)
  then show ?thesis
    by blast
  qed

```

```

lemma path_connected_uncountable:
  fixes  $S :: 'a::\text{metric\_space}$  set
  assumes path_connected  $S$   $a \in S$   $b \in S$   $a \neq b$  shows uncountable  $S$ 
  using path_connected_imp_connected assms connected_uncountable by metis

```

```

lemma connected_finite_iff_singleton:
  fixes  $S :: 'a::\text{metric\_space}$  set
  assumes connected  $S$ 
  shows  $\text{finite } S \longleftrightarrow S = \{\}$   $\vee$   $(\exists a. S = \{a\})$  (is  $\_ = ?rhs$ )
proof -

```

```

have uncountable S if  $\neg ?rhs$ 
  using connected_uncountable assms that by blast
then show ?thesis
  using uncountable_infinite by auto
qed

```

```

lemma connected_card_eq_iff_nontrivial:
  fixes S :: 'a::metric_space set
  shows connected S  $\implies$  uncountable S  $\longleftrightarrow$   $\neg(\exists a. S \subseteq \{a\})$ 
  by (metis connected_uncountable finite.emptyI finite.insertI rev_finite_subset
  singleton_iff subsetI uncountable_infinite)

```

```

lemma simple_path_image_uncountable:
  fixes g :: real  $\Rightarrow$  'a::metric_space
  assumes simple_path g
  shows uncountable (path_image g)
proof -
  have g 0  $\in$  path_image g g (1/2)  $\in$  path_image g
    by (simp_all add: path_defs)
  moreover have g 0  $\neq$  g (1/2)
    using assms by (fastforce simp add: simple_path_def)
  ultimately have  $\forall a. \neg \text{path\_image } g \subseteq \{a\}$ 
    by blast
  then show ?thesis
    using assms connected_simple_path_image connected_uncountable by blast
qed

```

```

lemma arc_image_uncountable:
  fixes g :: real  $\Rightarrow$  'a::metric_space
  assumes arc g
  shows uncountable (path_image g)
  by (simp add: arc_imp_simple_path assms simple_path_image_uncountable)

```

6.18.27 Some simple positive connection theorems

```

proposition path_connected_convex_diff_countable:
  fixes U :: 'a::euclidean_space set
  assumes convex U  $\neg$  collinear U countable S
  shows path_connected(U - S)
proof (clarsimp simp add: path_connected_def)
  fix a b
  assume a  $\in$  U a  $\notin$  S b  $\in$  U b  $\notin$  S
  let ?m = midpoint a b
  show  $\exists g. \text{path } g \wedge \text{path\_image } g \subseteq U - S \wedge \text{pathstart } g = a \wedge \text{pathfinish } g = b$ 
  proof (cases a = b)
    case True
    then show ?thesis
      by (metis DiffI  $\langle a \in U \rangle \langle a \notin S \rangle$  path_component_def path_component_refl)
  next

```

```

case False
then have  $a \neq ?m \ b \neq ?m$ 
  using midpoint_eq_endpoint by fastforce+
have  $?m \in U$ 
  using  $\langle a \in U \rangle \langle b \in U \rangle \langle \text{convex } U \rangle \text{convex\_contains\_segment}$  by force
obtain  $c$  where  $c \in U$  and  $nc\_abc: \neg \text{collinear } \{a, b, c\}$ 
  by (metis False  $\langle a \in U \rangle \langle b \in U \rangle \langle \neg \text{collinear } U \rangle \text{collinear\_triples insert\_absorb}$ )
  have  $ncoll\_mca: \neg \text{collinear } \{?m, c, a\}$ 
    by (metis (full_types)  $\langle a \neq ?m \rangle \text{collinear\_3\_trans collinear\_midpoint insert\_commute } nc\_abc$ )
  have  $ncoll\_mcb: \neg \text{collinear } \{?m, c, b\}$ 
    by (metis (full_types)  $\langle b \neq ?m \rangle \text{collinear\_3\_trans collinear\_midpoint insert\_commute } nc\_abc$ )
  have  $c \neq ?m$ 
    by (metis collinear_midpoint insert\_commute  $nc\_abc$ )
then have  $\text{closed\_segment } ?m \ c \subseteq U$ 
  by (simp add:  $\langle c \in U \rangle \langle ?m \in U \rangle \langle \text{convex } U \rangle \text{closed\_segment\_subset}$ )
then obtain  $z$  where  $z: z \in \text{closed\_segment } ?m \ c$ 
  and  $disjS: (\text{closed\_segment } a \ z \cup \text{closed\_segment } z \ b) \cap S = \{\}$ 
proof –
  have False if  $\text{closed\_segment } ?m \ c \subseteq \{z. (\text{closed\_segment } a \ z \cup \text{closed\_segment } z \ b) \cap S \neq \{\}\}$ 
  proof –
    have  $closb: \text{closed\_segment } ?m \ c \subseteq \{z \in \text{closed\_segment } ?m \ c. \text{closed\_segment } a \ z \cap S \neq \{\}\} \cup \{z \in \text{closed\_segment } ?m \ c. \text{closed\_segment } z \ b \cap S \neq \{\}\}$ 
    using that by blast
    have *: countable  $\{z \in \text{closed\_segment } ?m \ c. \text{closed\_segment } z \ u \cap S \neq \{\}\}$ 
    if  $u \in U \ u \notin S$  and  $ncoll: \neg \text{collinear } \{?m, c, u\}$  for  $u$ 
    proof –
      have **: False if  $x1: x1 \in \text{closed\_segment } ?m \ c$  and  $x2: x2 \in \text{closed\_segment } ?m \ c$ 
        and  $x1 \neq x2 \ x1 \neq u$ 
        and  $w: w \in \text{closed\_segment } x1 \ u \ w \in \text{closed\_segment } x2 \ u$ 
        and  $w \in S$  for  $x1 \ x2 \ w$ 
      proof –
        have  $x1 \in \text{affine hull } \{?m, c\} \ x2 \in \text{affine hull } \{?m, c\}$ 
        using segment_as_ball  $x1 \ x2$  by auto
        then have  $coll\_x1: \text{collinear } \{x1, ?m, c\}$  and  $coll\_x2: \text{collinear } \{?m, c, x2\}$ 
        by (simp_all add: affine_hull_3_imp_collinear) (metis affine_hull_3_imp_collinear insert\_commute)
        have  $\neg \text{collinear } \{x1, u, x2\}$ 
        proof
          assume  $\text{collinear } \{x1, u, x2\}$ 
          then have  $\text{collinear } \{?m, c, u\}$ 
          by (metis (full_types)  $\langle c \neq ?m \rangle coll\_x1 \ coll\_x2 \text{collinear\_3\_trans insert\_commute } ncoll \ \langle x1 \neq x2 \rangle$ )

```

```

      with ncoll show False ..
    qed
  then have closed_segment x1 u  $\cap$  closed_segment u x2 = {u}
    by (blast intro!: Int_closed_segment)
  then have w = u
    using closed_segment_commute w by auto
  show ?thesis
    using  $\langle u \notin S \rangle \langle w = u \rangle$  that( $\gamma$ ) by auto
  qed
  then have disj: disjoint (( $\bigcup z \in \text{closed\_segment } ?m \ c. \{\text{closed\_segment } z$ 
 $u \cap S\}$ ))
    by (fastforce simp: pairwise_def disjnt_def)
  have cou: countable (( $\bigcup z \in \text{closed\_segment } ?m \ c. \{\text{closed\_segment } z \cap$ 
 $S\} - \{\{\}\}$ )
    apply (rule pairwise_disjnt_countable_Union [OF __ pairwise_subset
[OF disj]])
    apply (rule countable_subset [OF __  $\langle \text{countable } S \rangle$ ], auto)
    done
  define f where  $f \equiv \lambda X. (\text{THE } z. z \in \text{closed\_segment } ?m \ c \wedge X =$ 
 $\text{closed\_segment } z \cap S)$ 
  show ?thesis
    proof (rule countable_subset [OF __ countable_image [OF cou, where
 $f=f$ ]], clarify)
      fix x
      assume  $x: x \in \text{closed\_segment } ?m \ c \wedge \text{closed\_segment } x \cap S \neq \{\}$ 
      show  $x \in f \text{ ` } ((\bigcup z \in \text{closed\_segment } ?m \ c. \{\text{closed\_segment } z \cap S\} -$ 
 $\{\{\}\})$ 
        proof (rule_tac  $x = \text{closed\_segment } x \cap S$  in image_eqI)
          show  $x = f (\text{closed\_segment } x \cap S)$ 
            unfolding f_def
            by (rule the_equality [symmetric]) (use x in  $\langle \text{auto dest: **} \rangle$ )
        qed (use x in auto)
      qed
    qed
  have uncountable (closed_segment ?m c)
    by (metis  $\langle c \neq ?m \rangle$  uncountable_closed_segment)
  then show False
    using closb * [OF  $\langle a \in U \rangle \langle a \notin S \rangle$  ncoll_mca] * [OF  $\langle b \in U \rangle \langle b \notin S \rangle$ 
ncoll_mcb]
    by (simp add: closed_segment_commute countable_subset)
  qed
  then show ?thesis
    by (force intro: that)
  qed
  show ?thesis
    proof (intro exI conjI)
      have path_image (linepath a z +++ linepath z b)  $\subseteq U$ 
        by (metis  $\langle a \in U \rangle \langle b \in U \rangle \langle \text{closed\_segment } ?m \ c \subseteq U \rangle z \in \text{convex } U$ 
closed_segment_subset contra_subsetD path_image_linepath subset_path_image_join)

```

```

    with disjS show path_image (linepath a z +++ linepath z b)  $\subseteq$  U - S
    by (force simp: path_image_join)
  qed auto
qed
qed

```

```

corollary connected_convex_diff_countable:
  fixes U :: 'a::euclidean_space set
  assumes convex U  $\neg$  collinear U countable S
  shows connected(U - S)
  by (simp add: asms path_connected_convex_diff_countable path_connected_imp_connected)

```

```

lemma path_connected_punctured_convex:
  assumes convex S and aff: aff_dim S  $\neq$  1
  shows path_connected(S - {a})
proof -
  consider aff_dim S = -1 | aff_dim S = 0 | aff_dim S  $\geq$  2
  using asms aff_dim_geq [of S] by linarith
  then show ?thesis
  proof cases
    assume aff_dim S = -1
    then show ?thesis
      by (metis aff_dim_empty empty_Diff path_connected_empty)
  next
    assume aff_dim S = 0
    then show ?thesis
      by (metis aff_dim_eq_0 Diff_cancel Diff_empty Diff_insert0 convex_empty
convex_imp_path_connected path_connected_singleton singletonD)
  next
    assume ge2: aff_dim S  $\geq$  2
    then have  $\neg$  collinear S
    proof (clarsimp simp add: collinear_affine_hull)
      fix u v
      assume S  $\subseteq$  affine_hull {u, v}
      then have aff_dim S  $\leq$  aff_dim {u, v}
        by (metis (no_types) aff_dim_affine_hull aff_dim_subset)
      with ge2 show False
      by (metis (no_types) aff_dim_2 antisym aff_not_numeral_le_zero one_le_numerical
order_trans)
    qed
    moreover have countable {a}
      by simp
    ultimately show ?thesis
      by (metis path_connected_convex_diff_countable [OF <convex S>])
  qed
qed

```

```

lemma connected_punctured_convex:

```

shows $\llbracket \text{convex } S; \text{aff\_dim } S \neq 1 \rrbracket \implies \text{connected}(S - \{a\})$
using *path_connected_imp_connected path_connected_punctured_convex* **by** *blast*

lemma *path_connected_complement_countable:*

fixes $S :: 'a::\text{euclidean\_space set}$
assumes $2 \leq \text{DIM}('a)$ *countable* S
shows *path_connected*($- S$)
proof $-$
have $\neg \text{collinear } (\text{UNIV} :: 'a \text{ set})$
using *assms* **by** (*auto simp: collinear_aff_dim [of UNIV :: 'a set]*)
then have *path_connected*($\text{UNIV} - S$)
by (*simp add: countable S path_connected_convex_diff_countable*)
then show *?thesis*
by (*simp add: Compl_eq_Diff_UNIV*)

qed

proposition *path_connected_openin_diff_countable:*

fixes $S :: 'a::\text{euclidean\_space set}$
assumes *connected* S **and** *ope: openin* (*top_of_set* (*affine hull* S)) S
and $\neg \text{collinear } S$ *countable* T
shows *path_connected*($S - T$)
proof (*clarsimp simp add: path_connected_component*)
fix $x y$
assume $xy: x \in S \ x \notin T \ y \in S \ y \notin T$
show *path_component* ($S - T$) $x y$
proof (*rule connected_equivalence_relation_gen [OF connected S]*, **where** $P = \lambda x. x \notin T$)
show $\exists z. z \in U \wedge z \notin T$ **if** *opeU: openin* (*top_of_set* S) U **and** $x \in U$ **for** $U x$

proof $-$

have *openin* (*top_of_set* (*affine hull* S)) U
using *opeU* *ope openin_trans* **by** *blast*
with $\langle x \in U \rangle$ **obtain** r **where** $U_{\text{sub}}: U \subseteq \text{affine hull } S$ **and** $r > 0$
and $\text{sub}U: \text{ball } x \ r \cap \text{affine hull } S \subseteq U$
by (*auto simp: openin_contains_ball*)
with $\langle x \in U \rangle$ **have** $x: x \in \text{ball } x \ r \cap \text{affine hull } S$
by *auto*
have $\neg S \subseteq \{x\}$
using $\langle \neg \text{collinear } S \rangle$ *collinear_subset* **by** *blast*
then obtain x' **where** $x' \neq x \ x' \in S$
by *blast*

obtain y **where** $y: y \neq x \ y \in \text{ball } x \ r \cap \text{affine hull } S$

proof

show $x + (r / 2 / \text{norm}(x' - x)) *_{\mathbb{R}} (x' - x) \neq x$
using $\langle x' \neq x \rangle \langle r > 0 \rangle$ **by** *auto*
show $x + (r / 2 / \text{norm}(x' - x)) *_{\mathbb{R}} (x' - x) \in \text{ball } x \ r \cap \text{affine hull } S$
using $\langle x' \neq x \rangle \langle r > 0 \rangle \langle x' \in S \rangle$ **by** *blast*
by (*simp add: dist_norm mem_affine_3_minus_hull_inc*)

qed

```

have convex (ball x r ∩ affine hull S)
  by (simp add: affine_imp_convex convex_Int)
with x y subU have uncountable U
  by (meson countable_subset uncountable_convex)
then have ¬ U ⊆ T
  using ⟨countable T⟩ countable_subset by blast
then show ?thesis by blast
qed
show ∃ U. openin (top_of_set S) U ∧ x ∈ U ∧
  (∀ x ∈ U. ∀ y ∈ U. x ∉ T ∧ y ∉ T ⟶ path_component (S - T) x y)
  if x ∈ S for x
proof -
  obtain r where Ssub: S ⊆ affine hull S and r > 0
    and subS: ball x r ∩ affine hull S ⊆ S
  using ope ⟨x ∈ S⟩ by (auto simp: openin_contains_ball)
  then have conv: convex (ball x r ∩ affine hull S)
    by (simp add: affine_imp_convex convex_Int)
  have ¬ aff_dim (affine hull S) ≤ 1
    using ⟨¬ collinear S⟩ collinear_aff_dim by auto
  then have ¬ aff_dim (ball x r ∩ affine hull S) ≤ 1
    by (metis (no_types, opaque_lifting) aff_dim_convex_Int_open IntI open_ball
  ⟨0 < r⟩ aff_dim_affine_hull affine_affine_hull affine_imp_convex centre_in_ball
  empty_iff hull_subset inf_commute subsetCE that)
  then have ¬ collinear (ball x r ∩ affine hull S)
    by (simp add: collinear_aff_dim)
  then have *: path_connected ((ball x r ∩ affine hull S) - T)
    by (rule path_connected_convex_diff_countable [OF conv _ ⟨countable T⟩])
  have ST: ball x r ∩ affine hull S - T ⊆ S - T
    using subS by auto
  show ?thesis
proof (intro exI conjI)
  show x ∈ ball x r ∩ affine hull S
    using ⟨x ∈ S⟩ ⟨r > 0⟩ by (simp add: hull_inc)
  have openin (top_of_set (affine hull S)) (ball x r ∩ affine hull S)
    by (subst inf_commute) (simp add: openin_Int_open)
  then show openin (top_of_set S) (ball x r ∩ affine hull S)
    by (rule openin_subset_trans [OF _ subS Ssub])
  qed (use * path_component_trans in ⟨auto simp: path_connected_component
  path_component_of_subset [OF ST]⟩)
  qed
  qed (use xy path_component_trans in auto)
qed

corollary connected_openin_diff_countable:
  fixes S :: 'a::euclidean_space set
  assumes connected S and ope: openin (top_of_set (affine hull S)) S
    and ¬ collinear S countable T
  shows connected(S - T)
  by (metis path_connected_imp_connected path_connected_openin_diff_countable

```


[OF assms])

corollary *path_connected_open_diff_countable:*

```

  fixes  $S :: 'a::euclidean\_space$  set
  assumes  $2 \leq DIM('a)$  open S connected S countable T
  shows path_connected(S - T)
proof (cases  $S = \{\}$ )
  case True
  then show ?thesis
    by (simp)
next
  case False
  show ?thesis
  proof (rule path_connected_openin_diff_countable)
    show openin (top_of_set (affine hull S)) S
      by (simp add: assms hull_subset open_subset)
    show  $\neg$  collinear S
      using assms False by (simp add: collinear_aff_dim aff_dim_open)
    qed (simp_all add: assms)
qed

```

corollary *connected_open_diff_countable:*

```

  fixes  $S :: 'a::euclidean\_space$  set
  assumes  $2 \leq DIM('a)$  open S connected S countable T
  shows connected(S - T)
by (simp add: assms path_connected_imp_connected path_connected_open_diff_countable)

```

6.18.28 Self-homeomorphisms shuffling points about

The theorem *homeomorphism_moving_points_exists*

lemma *homeomorphism_moving_point_1:*

```

  fixes  $a :: 'a::euclidean\_space$ 
  assumes affine T  $a \in T$  and  $u: u \in \text{ball } a \ r \cap T$ 
  obtains  $f \ g$  where homeomorphism (cball a r  $\cap$  T) (cball a r  $\cap$  T) f g
     $f \ a = u \wedge x. x \in \text{sphere } a \ r \implies f \ x = x$ 
proof -
  have now: norm (u - a) < r and  $u \in T$ 
    using  $u$  by (auto simp: dist_norm norm_minus_commute)
  then have  $0 < r$ 
    by (metis DiffD1 Diff_Diff_Int ball_eq_empty centre_in_ball not_le u)
  define  $f$  where  $f \equiv \lambda x. (1 - \text{norm}(x - a) / r) *_R (u - a) + x$ 
  have *: False if  $\text{eq: } x + (\text{norm } y / r) *_R u = y + (\text{norm } x / r) *_R u$ 
    and now: norm u < r and  $y x: \text{norm } y < \text{norm } x$  for  $x \ y$  and  $u::'a$ 
proof -
  have  $x = y + (\text{norm } x / r - (\text{norm } y / r)) *_R u$ 
    using eq by (simp add: algebra_simps)
  then have  $\text{norm } x = \text{norm } (y + ((\text{norm } x - \text{norm } y) / r) *_R u)$ 
    by (metis diff_divide_distrib)
  also have  $\dots \leq \text{norm } y + \text{norm}(((\text{norm } x - \text{norm } y) / r) *_R u)$ 

```

```

    using norm_triangle_ineq by blast
  also have ... = norm y + (norm x - norm y) * (norm u / r)
    using yx ⟨r > 0⟩
    by (simp add: field_split_simps)
  also have ... < norm y + (norm x - norm y) * 1
  proof (subst add_less_cancel_left)
    show (norm x - norm y) * (norm u / r) < (norm x - norm y) * 1
    proof (rule mult_strict_left_mono)
      show norm u / r < 1
      using ⟨0 < r⟩ divide_less_eq_1_pos nou by blast
    qed (simp add: yx)
  qed
  also have ... = norm x
    by simp
  finally show False by simp
qed
have inj f
  unfolding f_def
proof (clarify simp: inj_on_def)
  fix x y
  assume (1 - norm (x - a) / r) *R (u - a) + x =
    (1 - norm (y - a) / r) *R (u - a) + y
  then have eq: (x - a) + (norm (y - a) / r) *R (u - a) = (y - a) + (norm
(x - a) / r) *R (u - a)
    by (auto simp: algebra_simps)
  show x=y
  proof (cases norm (x - a) = norm (y - a))
    case True
    then show ?thesis
      using eq by auto
  next
    case False
    then consider norm (x - a) < norm (y - a) | norm (x - a) > norm (y -
a)
      by linarith
    then have False
  proof cases
    case 1 show False
      using * [OF _ nou 1] eq by simp
  next
    case 2 with * [OF eq nou] show False
      by auto
  qed
  then show x=y ..
qed
qed
then have inj_onf: inj_on f (cball a r ∩ T)
  using inj_on_Int by fastforce
have conf: continuous_on (cball a r ∩ T) f

```

```

    unfolding f_def using ‹0 < r› by (intro continuous_intros) blast
  have fim: f ‘ (cball a r ∩ T) = cball a r ∩ T
  proof
    have *: norm (y + (1 - norm y / r) *R u) ≤ r if norm y ≤ r norm u < r
  for y u::'a
    proof -
      have norm (y + (1 - norm y / r) *R u) ≤ norm y + norm((1 - norm y /
r) *R u)
        using norm_triangle_ineq by blast
      also have ... = norm y + abs(1 - norm y / r) * norm u
        by simp
      also have ... ≤ r
    proof -
      have (r - norm u) * (r - norm y) ≥ 0
        using that by auto
      then have r * norm u + r * norm y ≤ r * r + norm u * norm y
        by (simp add: algebra_simps)
      then show ?thesis
        using that ‹0 < r› by (simp add: abs_if field_simps)
    qed
    finally show ?thesis .
  qed
  have f ‘ (cball a r) ⊆ cball a r
    using * nou
    apply (clarsimp simp: dist_norm norm_minus_commute f_def)
    by (metis diff_add_eq diff_diff_add diff_diff_eq2 norm_minus_commute)
  moreover have f ‘ T ⊆ T
    unfolding f_def using ‹affine T› ‹a ∈ T› ‹u ∈ T›
    by (force simp: add_commute mem_affine_3_minus)
  ultimately show f ‘ (cball a r ∩ T) ⊆ cball a r ∩ T
    by blast
next
  show cball a r ∩ T ⊆ f ‘ (cball a r ∩ T)
  proof (clarsimp simp add: dist_norm norm_minus_commute)
    fix x
    assume x: norm (x - a) ≤ r and x ∈ T
    have ∃ v ∈ {0..1}. ((1 - v) * r - norm ((x - a) - v *R (u - a))) • 1 = 0
      by (rule ivt_decreasing_component_on_1) (auto simp: x continuous_intros)
    then obtain v where 0 ≤ v v ≤ 1
      and v: (1 - v) * r = norm ((x - a) - v *R (u - a))
      by auto
    then have n: norm (a - (x - v *R (u - a))) = r - r * v
      by (simp add: field_simps norm_minus_commute)
    show x ∈ f ‘ (cball a r ∩ T)
    proof (rule image_eqI)
      show x = f (x - v *R (u - a))
        using ‹r > 0› v by (simp add: f_def) (simp add: field_simps)
      have x - v *R (u - a) ∈ cball a r
        using ‹r > 0› ‹0 ≤ v›

```

```

      by (simp add: dist_norm n)
    moreover have  $x - v *_R (u - a) \in T$ 
      by (simp add: f_def ‹ $u \in T$ › ‹ $x \in T$ › assms mem_affine_3_minus2)
    ultimately show  $x - v *_R (u - a) \in cball\ a\ r \cap T$ 
      by blast
  qed
qed
qed
have compact (cball a r  $\cap$  T)
  by (simp add: affine_closed compact_Int_closed ‹affine T›)
then obtain g where homeomorphism (cball a r  $\cap$  T) (cball a r  $\cap$  T) f g
  by (metis homeomorphism_compact [OF _ contf_fim_inj_onf])
then show thesis
  apply (rule_tac f=f in that)
  using ‹r > 0› by (simp_all add: f_def dist_norm norm_minus_commute)
qed

corollary homeomorphism_moving_point_2:
  fixes a :: 'a::euclidean_space
  assumes affine T a  $\in$  T and u: u  $\in$  ball a r  $\cap$  T and v: v  $\in$  ball a r  $\cap$  T
  obtains f g where homeomorphism (cball a r  $\cap$  T) (cball a r  $\cap$  T) f g
     $f\ u = v \wedge x. \llbracket x \in sphere\ a\ r; x \in T \rrbracket \implies f\ x = x$ 

proof -
  have 0 < r
    by (metis DiffD1 Diff_Diff_Int ball_eq_empty centre_in_ball not_le u)
  obtain f1 g1 where hom1: homeomorphism (cball a r  $\cap$  T) (cball a r  $\cap$  T) f1
    g1
    and f1 a = u and f1:  $\wedge x. x \in sphere\ a\ r \implies f1\ x = x$ 
    using homeomorphism_moving_point_1 [OF ‹affine T› ‹a  $\in$  T› u] by blast
  obtain f2 g2 where hom2: homeomorphism (cball a r  $\cap$  T) (cball a r  $\cap$  T) f2
    g2
    and f2 a = v and f2:  $\wedge x. x \in sphere\ a\ r \implies f2\ x = x$ 
    using homeomorphism_moving_point_1 [OF ‹affine T› ‹a  $\in$  T› v] by blast
  show ?thesis
  proof
    show homeomorphism (cball a r  $\cap$  T) (cball a r  $\cap$  T) (f2  $\circ$  g1) (f1  $\circ$  g2)
      by (metis homeomorphism_compose homeomorphism_symD hom1 hom2)
    have g1 u = a
      using ‹0 < r› ‹f1 a = u› assms hom1 homeomorphism_apply1 by fastforce
    then show (f2  $\circ$  g1) u = v
      by (simp add: ‹f2 a = v›)
    show  $\wedge x. \llbracket x \in sphere\ a\ r; x \in T \rrbracket \implies (f2 \circ g1)\ x = x$ 
      using f1 f2 hom1 homeomorphism_apply1 by fastforce
  qed
qed

```

```

corollary homeomorphism_moving_point_3:
  fixes a :: 'a::euclidean_space

```

```

assumes affine  $T$   $a \in T$  and  $ST$ :  $\text{ball } a \ r \cap T \subseteq S \ S \subseteq T$ 
and  $u$ :  $u \in \text{ball } a \ r \cap T$  and  $v$ :  $v \in \text{ball } a \ r \cap T$ 
obtains  $f \ g$  where homeomorphism  $S \ S \ f \ g$ 
 $f \ u = v \ \{x. \neg (f \ x = x \wedge g \ x = x)\} \subseteq \text{ball } a \ r \cap T$ 
proof –
obtain  $f \ g$  where hom: homeomorphism  $(\text{cball } a \ r \cap T) (\text{cball } a \ r \cap T) f \ g$ 
and  $f \ u = v$  and fid:  $\bigwedge x. \llbracket x \in \text{sphere } a \ r; x \in T \rrbracket \implies f \ x = x$ 
using homeomorphism_moving_point_2 [OF  $\langle \text{affine } T \rangle \langle a \in T \rangle u \ v$ ] by blast
have gid:  $\bigwedge x. \llbracket x \in \text{sphere } a \ r; x \in T \rrbracket \implies g \ x = x$ 
using fid hom homeomorphism_apply1 by fastforce
define ff where  $\text{ff} \equiv \lambda x. \text{if } x \in \text{ball } a \ r \cap T \text{ then } f \ x \text{ else } x$ 
define gg where  $\text{gg} \equiv \lambda x. \text{if } x \in \text{ball } a \ r \cap T \text{ then } g \ x \text{ else } x$ 
show ?thesis
proof
show homeomorphism  $S \ S \ \text{ff} \ \text{gg}$ 
proof (rule homeomorphismI)
have continuous_on  $((\text{cball } a \ r \cap T) \cup (T - \text{ball } a \ r)) \ \text{ff}$ 
unfolding ff_def
using homeomorphism_cont1 [OF hom]
by (intro continuous_on_cases) (auto simp: affine_closed  $\langle \text{affine } T \rangle \text{fid}$ )
then show continuous_on  $S \ \text{ff}$ 
by (rule continuous_on_subset) (use  $ST$  in auto)
have continuous_on  $((\text{cball } a \ r \cap T) \cup (T - \text{ball } a \ r)) \ \text{gg}$ 
unfolding gg_def
using homeomorphism_cont2 [OF hom]
by (intro continuous_on_cases) (auto simp: affine_closed  $\langle \text{affine } T \rangle \text{gid}$ )
then show continuous_on  $S \ \text{gg}$ 
by (rule continuous_on_subset) (use  $ST$  in auto)
show  $\text{ff} \ ' S \subseteq S$ 
proof (clarsimp simp add: ff_def)
fix  $x$ 
assume  $x \in S$  and  $x$ :  $\text{dist } a \ x < r$  and  $x \in T$ 
then have  $f \ x \in \text{cball } a \ r \cap T$ 
using homeomorphism_image1 [OF hom] by force
then show  $f \ x \in S$ 
using  $ST(1) \ \langle x \in T \rangle \text{gid}$  hom homeomorphism_def  $x$  by fastforce
qed
show  $\text{gg} \ ' S \subseteq S$ 
proof (clarsimp simp add: gg_def)
fix  $x$ 
assume  $x \in S$  and  $x$ :  $\text{dist } a \ x < r$  and  $x \in T$ 
then have  $g \ x \in \text{cball } a \ r \cap T$ 
using homeomorphism_image2 [OF hom] by force
then have  $g \ x \in \text{ball } a \ r$ 
using homeomorphism_apply2 [OF hom]
by (metis Diff_Diff_Int Diff_iff  $\langle x \in T \rangle \text{cball\_def}$  fid le_less
mem_Collect_eq mem_ball mem_sphere  $x$ )
then show  $g \ x \in S$ 
using  $ST(1) \ \langle g \ x \in \text{cball } a \ r \cap T \rangle$  by force

```

```

    qed
  show  $\bigwedge x. x \in S \implies gg \text{ (ff } x) = x$ 
    unfolding ff_def gg_def
    using homeomorphism_apply1 [OF hom] homeomorphism_image1 [OF
hom]
    by simp (metis Int_iff homeomorphism_apply1 [OF hom] fid image_eqI
less_eq_real_def mem_cball mem_sphere)
  show  $\bigwedge x. x \in S \implies ff \text{ (gg } x) = x$ 
    unfolding ff_def gg_def
    using homeomorphism_apply2 [OF hom] homeomorphism_image2 [OF
hom]
    by simp (metis Int_iff fid image_eqI less_eq_real_def mem_cball mem_sphere)
  qed
  show ff u = v
    using u by (auto simp: ff_def  $\langle f u = v \rangle$ )
  show  $\{x. \neg (ff \text{ } x = x \wedge gg \text{ } x = x)\} \subseteq ball \text{ } a \text{ } r \cap T$ 
    by (auto simp: ff_def gg_def)
  qed
qed

```

proposition *homeomorphism_moving_point:*

```

  fixes a :: 'a::euclidean_space
  assumes ope: openin (top_of_set (affine hull S)) S
    and S  $\subseteq T$ 
    and TS:  $T \subseteq affine \text{ hull } S$ 
    and S: connected  $S \text{ } a \in S \text{ } b \in S$ 
  obtains f g where homeomorphism T T f g f a = b
     $\{x. \neg (f \text{ } x = x \wedge g \text{ } x = x)\} \subseteq S$ 
    bounded  $\{x. \neg (f \text{ } x = x \wedge g \text{ } x = x)\}$ 

```

proof –

```

  have 1:  $\exists h k. homeomorphism \text{ } T \text{ } T \text{ } h \text{ } k \wedge h \text{ (f } d) = d \wedge$ 
     $\{x. \neg (h \text{ } x = x \wedge k \text{ } x = x)\} \subseteq S \wedge bounded \{x. \neg (h \text{ } x = x \wedge k \text{ } x = x)\}$ 
    if  $d \in S \text{ } f \text{ } d \in S$  and homfg: homeomorphism T T f g
    and S:  $\{x. \neg (f \text{ } x = x \wedge g \text{ } x = x)\} \subseteq S$ 
    and bo: bounded  $\{x. \neg (f \text{ } x = x \wedge g \text{ } x = x)\}$  for d f g

```

proof (intro exI conjI)

```

  show homfg: homeomorphism T T g f
    by (metis homeomorphism_symD homfg)
  then show g (f d) = d
    by (meson  $\langle S \subseteq T \rangle homeomorphism\_def subsetD \langle d \in S \rangle$ )
  show  $\{x. \neg (g \text{ } x = x \wedge f \text{ } x = x)\} \subseteq S$ 
    using S by blast
  show bounded  $\{x. \neg (g \text{ } x = x \wedge f \text{ } x = x)\}$ 
    using bo by (simp add: conj_commute)

```

qed

```

  have 2:  $\exists f g. homeomorphism \text{ } T \text{ } T \text{ } f \text{ } g \wedge f \text{ } x = f2 \text{ (f1 } x) \wedge$ 
     $\{x. \neg (f \text{ } x = x \wedge g \text{ } x = x)\} \subseteq S \wedge bounded \{x. \neg (f \text{ } x = x \wedge g \text{ } x =$ 

```

$x)\}$

```

    if  $x \in S$   $f1\ x \in S$   $f2\ (f1\ x) \in S$ 
      and  $hom: \text{homeomorphism } T\ T\ f1\ g1\ \text{homeomorphism } T\ T\ f2\ g2$ 
      and  $sub: \{x. \neg (f1\ x = x \wedge g1\ x = x)\} \subseteq S \quad \{x. \neg (f2\ x = x \wedge g2\ x = x)\} \subseteq S$ 
    and  $bo: \text{bounded } \{x. \neg (f1\ x = x \wedge g1\ x = x)\} \quad \text{bounded } \{x. \neg (f2\ x = x \wedge g2\ x = x)\}$ 
  for  $x\ f1\ f2\ g1\ g2$ 
proof (intro exI conjI)
  show  $homgf: \text{homeomorphism } T\ T\ (f2 \circ f1)\ (g1 \circ g2)$ 
    by (metis homeomorphism_compose hom)
  then show  $(f2 \circ f1)\ x = f2\ (f1\ x)$ 
    by force
  show  $\{x. \neg ((f2 \circ f1)\ x = x \wedge (g1 \circ g2)\ x = x)\} \subseteq S$ 
    using sub by force
  have bounded  $(\{x. \neg (f1\ x = x \wedge g1\ x = x)\} \cup \{x. \neg (f2\ x = x \wedge g2\ x = x)\})$ 
    using bo by simp
  then show bounded  $\{x. \neg ((f2 \circ f1)\ x = x \wedge (g1 \circ g2)\ x = x)\}$ 
    by (rule bounded_subset) auto
qed
have  $\exists: \exists U. \text{openin } (\text{top\_of\_set } S)\ U \wedge$ 
   $d \in U \wedge$ 
   $(\forall x \in U.$ 
     $\exists f\ g. \text{homeomorphism } T\ T\ f\ g \wedge f\ d = x \wedge$ 
     $\{x. \neg (f\ x = x \wedge g\ x = x)\} \subseteq S \wedge$ 
     $\text{bounded } \{x. \neg (f\ x = x \wedge g\ x = x)\})$ 
  if  $d \in S$  for  $d$ 
proof -
  obtain  $r$  where  $r > 0$  and  $r: \text{ball } d\ r \cap \text{affine hull } S \subseteq S$ 
    by (metis  $\langle d \in S \rangle$  ope openin_contains_ball)
  have  $*$ :  $\exists f\ g. \text{homeomorphism } T\ T\ f\ g \wedge f\ d = e \wedge$ 
     $\{x. \neg (f\ x = x \wedge g\ x = x)\} \subseteq S \wedge$ 
     $\text{bounded } \{x. \neg (f\ x = x \wedge g\ x = x)\}$  if  $e \in S\ e \in \text{ball } d\ r$  for  $e$ 
  apply (rule homeomorphism_moving_point_3 [of affine hull  $S\ d\ r\ T\ d\ e$ ])
  using  $r\ \langle S \subseteq T \rangle\ TS$  that
  apply (auto simp:  $\langle d \in S \rangle\ \langle 0 < r \rangle\ \text{hull\_inc}$ )
  using bounded_subset by blast
  show ?thesis
    by (rule_tac  $x=S \cap \text{ball } d\ r$  in exI) (fastforce simp: openin_open_Int  $\langle 0 < r \rangle$  that intro: *)
qed
have  $\exists f\ g. \text{homeomorphism } T\ T\ f\ g \wedge f\ a = b \wedge$ 
   $\{x. \neg (f\ x = x \wedge g\ x = x)\} \subseteq S \wedge \text{bounded } \{x. \neg (f\ x = x \wedge g\ x = x)\}$ 
  by (rule connected_equivalence_relation [OF  $S$ ]; blast intro: 1 2 3)
then show ?thesis
  using that by auto
qed

```

lemma *homeomorphism_moving_points_exists_gen*:

```

assumes  $K$ : finite  $K \wedge i. i \in K \implies x\ i \in S \wedge y\ i \in S$ 
           pairwise  $(\lambda i\ j. (x\ i \neq x\ j) \wedge (y\ i \neq y\ j))\ K$ 
and  $2 \leq \text{aff\_dim}\ S$ 
and ope: openin (top_of_set (affine hull  $S$ ))  $S$ 
and  $S \subseteq T\ T \subseteq \text{affine hull}\ S$  connected  $S$ 
shows  $\exists f\ g. \text{homeomorphism}\ T\ T\ f\ g \wedge (\forall i \in K. f(x\ i) = y\ i) \wedge$ 
         $\{x. \neg (f\ x = x \wedge g\ x = x)\} \subseteq S \wedge \text{bounded}\ \{x. \neg (f\ x = x \wedge g\ x = x)\}$ 
using assms
proof (induction  $K$ )
  case empty
  then show ?case
    by (force simp: homeomorphism_ident)
  next
    case (insert  $i\ K$ )
    then have xney:  $\bigwedge j. [j \in K; j \neq i] \implies x\ i \neq x\ j \wedge y\ i \neq y\ j$ 
      and pw: pairwise  $(\lambda i\ j. x\ i \neq x\ j \wedge y\ i \neq y\ j)\ K$ 
      and  $x\ i \in S\ y\ i \in S$ 
      and xyS:  $\bigwedge i. i \in K \implies x\ i \in S \wedge y\ i \in S$ 
      by (simp_all add: pairwise_insert)
    obtain  $f\ g$  where homfg: homeomorphism  $T\ T\ f\ g$  and feq:  $\bigwedge i. i \in K \implies f(x\ i) = y\ i$ 
      and fg_sub:  $\{x. \neg (f\ x = x \wedge g\ x = x)\} \subseteq S$ 
      and bo_fg: bounded  $\{x. \neg (f\ x = x \wedge g\ x = x)\}$ 
      using insert.IH [OF xyS pw] insert.prems by (blast intro: that)
    then have  $\exists f\ g. \text{homeomorphism}\ T\ T\ f\ g \wedge (\forall i \in K. f(x\ i) = y\ i) \wedge$ 
       $\{x. \neg (f\ x = x \wedge g\ x = x)\} \subseteq S \wedge \text{bounded}\ \{x. \neg (f\ x = x \wedge g\ x = x)\}$ 
    using insert by blast
  have aff_eq: affine hull  $(S - y\ 'K) = \text{affine hull}\ S$ 
proof (rule affine_hull_Diff [OF ope])
  show finite  $(y\ 'K)$ 
    by (simp add: insert.hyps(1))
  show  $y\ 'K \subseteq S$ 
    using  $\langle y\ i \in S \rangle$  insert.hyps(2) xney xyS by fastforce
qed
have f_in_S:  $f\ x \in S$  if  $x \in S$  for  $x$ 
  using homfg fg_sub homeomorphism_apply1  $\langle S \subseteq T \rangle$ 
proof –
  have  $(f\ (f\ x) \neq f\ x \vee g\ (f\ x) \neq f\ x) \vee f\ x \in S$ 
    by (metis  $\langle S \subseteq T \rangle$  homfg subsetD homeomorphism_apply1 that)
  then show ?thesis
    using fg_sub by force
qed
obtain  $h\ k$  where homhk: homeomorphism  $T\ T\ h\ k$  and heq:  $h\ (f\ (x\ i)) = y\ i$ 
      and hk_sub:  $\{x. \neg (h\ x = x \wedge k\ x = x)\} \subseteq S - y\ 'K$ 
      and bo_hk: bounded  $\{x. \neg (h\ x = x \wedge k\ x = x)\}$ 
proof (rule homeomorphism_moving_point [of  $S - y\ 'K\ T\ f(x\ i)\ y\ i$ ])
  show openin (top_of_set (affine hull  $(S - y\ 'K)$ ))  $(S - y\ 'K)$ 
    by (simp add: aff_eq openin_diff finite_imp_closedin image_subset_iff)

```



```

hull_inc insert xyS)
  show  $S - y \text{ ' } K \subseteq T$ 
    using  $\langle S \subseteq T \rangle$  by auto
  show  $T \subseteq \text{affine hull } (S - y \text{ ' } K)$ 
    using insert by (simp add: aff_eq)
  show connected  $(S - y \text{ ' } K)$ 
  proof (rule connected_openin_diff_countable [OF  $\langle \text{connected } S \rangle$  ope])
    show  $\neg \text{collinear } S$ 
      using collinear_aff_dim  $\langle 2 \leq \text{aff\_dim } S \rangle$  by force
    show countable  $(y \text{ ' } K)$ 
      using countable_finite insert.hyps(1) by blast
  qed
  have  $\bigwedge k. \llbracket f(x\ i) = y\ k; k \in K \rrbracket \implies \text{False}$ 
    by (metis freq homfg  $\langle x\ i \in S \rangle$  homeomorphism_def  $\langle S \subseteq T \rangle \langle i \notin K \rangle$ )
subsetCE xney xyS)
  then show  $f(x\ i) \in S - y \text{ ' } K$ 
    by (auto simp: f_in_S  $\langle x\ i \in S \rangle$ )
  show  $y\ i \in S - y \text{ ' } K$ 
    using insert.hyps xney by (auto simp:  $\langle y\ i \in S \rangle$ )
  qed blast
show ?case
proof (intro exI conjI)
  show homeomorphism  $T\ T\ (h \circ f)\ (g \circ k)$ 
    using homfg homhk homeomorphism_compose by blast
  show  $\forall i \in \text{insert } i\ K. (h \circ f)(x\ i) = y\ i$ 
    using freq hk_sub by (auto simp: heq)
  show  $\{x. \neg ((h \circ f)\ x = x \wedge (g \circ k)\ x = x)\} \subseteq S$ 
    using fg_sub hk_sub by force
  have bounded  $(\{x. \neg (f\ x = x \wedge g\ x = x)\} \cup \{x. \neg (h\ x = x \wedge k\ x = x)\})$ 
    using bo_fg bo_hk bounded_Un by blast
  then show bounded  $\{x. \neg ((h \circ f)\ x = x \wedge (g \circ k)\ x = x)\}$ 
    by (rule bounded_subset) auto
  qed
qed

proposition homeomorphism_moving_points_exists:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes 2:  $2 \leq \text{DIM}('a)$  open  $S$  connected  $S \subseteq T$  finite  $K$ 
    and KS:  $\bigwedge i. i \in K \implies x\ i \in S \wedge y\ i \in S$ 
    and pw: pairwise  $(\lambda i\ j. (x\ i \neq x\ j) \wedge (y\ i \neq y\ j))\ K$ 
    and  $S: S \subseteq T \subseteq \text{affine hull } S$  connected  $S$ 
  obtains  $f\ g$  where homeomorphism  $T\ T\ f\ g \bigwedge i. i \in K \implies f(x\ i) = y\ i$ 
     $\{x. \neg (f\ x = x \wedge g\ x = x)\} \subseteq S$  bounded  $\{x. \neg (f\ x = x \wedge g\ x =$ 
 $x)\}$ 
  proof (cases  $S = \{\}$ )
    case True
      then show ?thesis
        using KS homeomorphism_ident that by fastforce
  next

```

```

case False
then have affS: affine hull S = UNIV
  by (simp add: affine_hull_open ⟨open S⟩)
then have ope: openin (top_of_set (affine hull S)) S
  using ⟨open S⟩ open_openin by auto
have 2 ≤ DIM('a) by (rule 2)
also have ... = aff_dim (UNIV :: 'a set)
  by simp
also have ... ≤ aff_dim S
  by (metis aff_dim_UNIV aff_dim_affine_hull aff_dim_le_DIM affS)
finally have 2 ≤ aff_dim S
  by linarith
then show ?thesis
  using homeomorphism_moving_points_exists_gen [OF ⟨finite K⟩ KS pw _
ope S] that by fastforce
qed

```

The theorem *homeomorphism_grouping_points_exists*

lemma *homeomorphism_grouping_point_1*:

fixes *a::real and c::real*

assumes $a < b < d$

obtains *f g* **where** *homeomorphism (cbox a b) (cbox c d) f g f a = c f b = d*

proof –

define *f* **where** $f \equiv \lambda x. ((d - c) / (b - a)) * x + (c - a * ((d - c) / (b - a)))$

show $\exists g. \text{homeomorphism } (cbox\ a\ b)\ (cbox\ c\ d)\ f\ g$

proof (rule *homeomorphism_compact*)

show *continuous_on (cbox a b) f*

unfolding *f_def* **by** (intro *continuous_intros*)

have $f \text{ ' } \{a..b\} = \{c..d\}$

unfolding *f_def image_affinity_atLeastAtMost*

using *assms sum_sqs_eq* **by** (auto simp: *field_split_simps*)

then show $f \text{ ' } cbox\ a\ b = cbox\ c\ d$

by *auto*

show *inj_on f (cbox a b)*

unfolding *f_def inj_on_def* **using** *assms* **by** *auto*

qed *auto*

then obtain *g* **where** *homeomorphism (cbox a b) (cbox c d) f g ..*

then show *?thesis*

proof

show $f\ a = c$

by (simp add: *f_def*)

show $f\ b = d$

using *assms sum_sqs_eq [of a b]* **by** (auto simp: *f_def field_split_simps*)

qed

qed

lemma *homeomorphism_grouping_point_2*:

```

fixes a::real and w::real
assumes hom_ab: homeomorphism (cbox a b) (cbox u v) f1 g1
      and hom_bc: homeomorphism (cbox b c) (cbox v w) f2 g2
      and b ∈ cbox a c v ∈ cbox u w
      and eq: f1 a = u f1 b = v f2 b = v f2 c = w
obtains f g where homeomorphism (cbox a c) (cbox u w) f g f a = u f c = w
       $\bigwedge x. x \in \text{cbox } a \ b \implies f x = f1 \ x \bigwedge x. x \in \text{cbox } b \ c \implies f x = f2 \ x$ 
proof -
  have le: a ≤ b b ≤ c u ≤ v v ≤ w
    using assms by simp_all
  then have ac: cbox a c = cbox a b ∪ cbox b c and uw: cbox u w = cbox u v ∪
cbox v w
    by auto
  define f where f ≡ λx. if x ≤ b then f1 x else f2 x
  have ∃g. homeomorphism (cbox a c) (cbox u w) f g
  proof (rule homeomorphism_compact)
    have cf1: continuous_on (cbox a b) f1
      using hom_ab homeomorphism_cont1 by blast
    have cf2: continuous_on (cbox b c) f2
      using hom_bc homeomorphism_cont1 by blast
    show continuous_on (cbox a c) f
      unfolding f_def using le eq
      by (force intro: continuous_on_cases_le [OF continuous_on_subset [OF cf1]
continuous_on_subset [OF cf2]])
    have f' cbox a b = f1 ' cbox a b f' cbox b c = f2 ' cbox b c
      unfolding f_def using eq by force+
    then show f' cbox a c = cbox u w
      unfolding ac uw image_Un by (metis hom_ab hom_bc homeomorphism_def)
    have neq12: f1 x ≠ f2 y if x: a ≤ x x ≤ b and y: b < y y ≤ c for x y
    proof -
      have f1 x ∈ cbox u v
        by (metis hom_ab homeomorphism_def image_eqI mem_box_real(2) x)
      moreover have f2 y ∈ cbox v w
        by (metis (full_types) hom_bc homeomorphism_def image_subset_iff
mem_box_real(2) not_le not_less_iff_gr_or_eq order_refl y)
      moreover have f2 y ≠ f2 b
        by (metis cancel_comm_monoid_add_class.diff_cancel diff_gt_0_iff_gt
hom_bc homeomorphism_def le(2) less_imp_le less_numeral_extra(3) mem_box_real(2)
order_refl y)
      ultimately show ?thesis
        using le eq by simp
    qed
    have inj_on f1 (cbox a b)
      by (metis (full_types) hom_ab homeomorphism_def inj_onI)
    moreover have inj_on f2 (cbox b c)
      by (metis (full_types) hom_bc homeomorphism_def inj_onI)
    ultimately show inj_on f (cbox a c)
      apply (simp (no_asm) add: inj_on_def)
      apply (simp add: f_def inj_on_eq_iff)

```

```

    using neq12 by force
  qed auto
  then obtain g where homeomorphism (cbox a c) (cbox u v) f g ..
  then show ?thesis
    using eq_f_def le that by force
  qed

lemma homeomorphism_grouping_point_3:
  fixes a::real
  assumes cbox_sub: cbox c d  $\subseteq$  cbox a b cbox u v  $\subseteq$  cbox a b
    and box_ne: box c d  $\neq$  {} box u v  $\neq$  {}
  obtains f g where homeomorphism (cbox a b) (cbox a b) f g f a = a f b = b
     $\bigwedge x. x \in \text{cbox } c \ d \implies f x \in \text{cbox } u \ v$ 
proof -
  have less: a < c a < u d < b v < b c < d u < v cbox c d  $\neq$  {}
    using assms
    by (simp_all add: cbox_sub subset_eq)
  obtain f1 g1 where 1: homeomorphism (cbox a c) (cbox a u) f1 g1
    and f1_eq: f1 a = a f1 c = u
    using homeomorphism_grouping_point_1 [OF <a < c> <a < u>] .
  obtain f2 g2 where 2: homeomorphism (cbox c d) (cbox u v) f2 g2
    and f2_eq: f2 c = u f2 d = v
    using homeomorphism_grouping_point_1 [OF <c < d> <u < v>] .
  obtain f3 g3 where 3: homeomorphism (cbox d b) (cbox v b) f3 g3
    and f3_eq: f3 d = v f3 b = b
    using homeomorphism_grouping_point_1 [OF <d < b> <v < b>] .
  obtain f4 g4 where 4: homeomorphism (cbox a d) (cbox a v) f4 g4 and f4 a =
a f4 d = v
    and f4_eq:  $\bigwedge x. x \in \text{cbox } a \ c \implies f4 x = f1 x \bigwedge x. x \in \text{cbox } c \ d \implies$ 
f4 x = f2 x
    using homeomorphism_grouping_point_2 [OF 1 2] less by (auto simp: f1_eq
f2_eq)
  obtain f g where fg: homeomorphism (cbox a b) (cbox a b) f g f a = a f b = b
    and f_eq:  $\bigwedge x. x \in \text{cbox } a \ d \implies f x = f4 x \bigwedge x. x \in \text{cbox } d \ b \implies f x$ 
= f3 x
    using homeomorphism_grouping_point_2 [OF 4 3] less by (auto simp: f4_eq
f3_eq f2_eq f1_eq)
  show ?thesis
  proof (rule that [OF fg])
    show f x  $\in$  cbox u v if x  $\in$  cbox c d for x
      using that f4_eq f_eq homeomorphism_image1 [OF 2]
      by (metis atLeastAtMost_iff box_real(2) image_eqI less(1) less_eq_real_def
order_trans)
  qed
qed

```

```

lemma homeomorphism_grouping_point_4:
  fixes T :: real set

```

```

assumes open U open S connected S U ≠ {} finite K K ⊆ S U ⊆ S S ⊆ T
obtains f g where homeomorphism T T f g
       $\bigwedge x. x \in K \implies f x \in U \{x. (\neg (f x = x \wedge g x = x))\} \subseteq S$ 
      bounded  $\{x. (\neg (f x = x \wedge g x = x))\}$ 
proof -
  obtain c d where box c d ≠ {} cbox c d ⊆ U
  proof -
    obtain u where u ∈ U
    using ⟨U ≠ {}⟩ by blast
    then obtain e where e > 0 cball u e ⊆ U
    using ⟨open U⟩ open_contains_cball by blast
    then show ?thesis
    by (rule_tac c=u and d=u+e in that) (auto simp: dist_norm subset_iff)
  qed
  have compact K
  by (simp add: ⟨finite K⟩ finite_imp_compact)
  obtain a b where box a b ≠ {} K ⊆ cbox a b cbox a b ⊆ S
  proof (cases K = {})
    case True then show ?thesis
    using ⟨box c d ≠ {}⟩ ⟨cbox c d ⊆ U⟩ ⟨U ⊆ S⟩ that by blast
  next
    case False
    then obtain a b where a ∈ K b ∈ K
    and a:  $\bigwedge x. x \in K \implies a \leq x$  and b:  $\bigwedge x. x \in K \implies x \leq b$ 
    using compact_attains_inf compact_attains_sup by (metis ⟨compact K⟩)+
    obtain e where e > 0 cball b e ⊆ S
    using ⟨open S⟩ open_contains_cball
    by (metis ⟨b ∈ K⟩ ⟨K ⊆ S⟩ subsetD)
    show ?thesis
    proof
      show box a (b + e) ≠ {}
      using ⟨0 < e⟩ ⟨b ∈ K⟩ a by force
      show K ⊆ cbox a (b + e)
      using ⟨0 < e⟩ a b by fastforce
      have a ∈ S
      using ⟨a ∈ K⟩ assms(6) by blast
      have b + e ∈ S
      using ⟨0 < e⟩ ⟨cball b e ⊆ S⟩ by (force simp: dist_norm)
      show cbox a (b + e) ⊆ S
      using ⟨a ∈ S⟩ ⟨b + e ∈ S⟩ ⟨connected S⟩ connected_contains_Icc by auto
    qed
  qed
  obtain w z where cbox w z ⊆ S and sub_wz: cbox a b ∪ cbox c d ⊆ box w z
  proof -
    have a ∈ S b ∈ S
    using ⟨box a b ≠ {}⟩ ⟨cbox a b ⊆ S⟩ by auto
    moreover have c ∈ S d ∈ S
    using ⟨box c d ≠ {}⟩ ⟨cbox c d ⊆ U⟩ ⟨U ⊆ S⟩ by force+
    ultimately have min a c ∈ S max b d ∈ S

```

```

    by linarith+
  then obtain e1 e2 where e1 > 0 cball (min a c) e1 ⊆ S e2 > 0 cball (max
b d) e2 ⊆ S
    using ⟨open S⟩ open_contains_cball by metis
  then have *: min a c - e1 ∈ S max b d + e2 ∈ S
    by (auto simp: dist_norm)
  show ?thesis
  proof
    show cbox (min a c - e1) (max b d + e2) ⊆ S
      using * ⟨connected S⟩ connected_contains_Icc by auto
    show cbox a b ∪ cbox c d ⊆ box (min a c - e1) (max b d + e2)
      using ⟨0 < e1⟩ ⟨0 < e2⟩ by auto
  qed
qed
then
obtain f g where hom: homeomorphism (cbox w z) (cbox w z) f g
  and f w = w f z = z
  and fin:  $\bigwedge x. x \in \text{cbox } a \ b \implies f \ x \in \text{cbox } c \ d$ 
  using homeomorphism_grouping_point_3 [of a b w z c d]
  using ⟨box a b ≠ {}⟩ ⟨box c d ≠ {}⟩ by blast
have confg: continuous_on (cbox w z) f continuous_on (cbox w z) g
  using hom homeomorphism_def by blast+
define f' where f' ≡ λx. if x ∈ cbox w z then f x else x
define g' where g' ≡ λx. if x ∈ cbox w z then g x else x
show ?thesis
  proof
    have T: cbox w z ∪ (T - box w z) = T
      using ⟨cbox w z ⊆ S⟩ ⟨S ⊆ T⟩ by auto
    show homeomorphism T T f' g'
    proof
      have clo: closedin (top_of_set (cbox w z ∪ (T - box w z))) (T - box w z)
        by (metis Diff_Diff_Int Diff_subset T closedin_def open_box openin_open_Int
topspace_euclidean_subtopology)
      have  $\bigwedge x. \llbracket w \leq x \wedge x \leq z; w < x \longrightarrow \neg x < z \rrbracket \implies f \ x = x$ 
        using ⟨f w = w⟩ ⟨f z = z⟩ by auto
      moreover have  $\bigwedge x. \llbracket w \leq x \wedge x \leq z; w < x \longrightarrow \neg x < z \rrbracket \implies g \ x = x$ 
        using ⟨f w = w⟩ ⟨f z = z⟩ hom homeomorphism_apply1 by fastforce
      ultimately
      have continuous_on (cbox w z ∪ (T - box w z)) f' continuous_on (cbox w z
∪ (T - box w z)) g'
        unfolding f'_def g'_def
        by (intro continuous_on_cases_local confg continuous_on_id clo; auto
simp: closed_subset)+
      then show continuous_on T f' continuous_on T g'
        by (simp_all only: T)
      show f' ' T ⊆ T
        unfolding f'_def
        by clarsimp (metis ⟨cbox w z ⊆ S⟩ ⟨S ⊆ T⟩ subsetD hom homeomorphism_def
imageI mem_box_real(2))
    end
  end

```

```

  show  $g' \cdot T \subseteq T$ 
    unfolding  $g'_\text{def}$ 
  by clarsimp (metis  $\langle \text{cbox } w \ z \subseteq S \rangle \langle S \subseteq T \rangle \text{subsetD hom homeomorphism\_def}$ 
imageI mem_box_real(2))
  show  $\bigwedge x. x \in T \implies g' (f' x) = x$ 
    unfolding  $f'_\text{def } g'_\text{def}$ 
    using homeomorphism_apply1 [OF hom] homeomorphism_image1 [OF
hom] by fastforce
  show  $\bigwedge y. y \in T \implies f' (g' y) = y$ 
    unfolding  $f'_\text{def } g'_\text{def}$ 
    using homeomorphism_apply2 [OF hom] homeomorphism_image2 [OF
hom] by fastforce
qed
show  $\bigwedge x. x \in K \implies f' x \in U$ 
  using fin_sub_wz  $\langle K \subseteq \text{cbox } a \ b \rangle \langle \text{cbox } c \ d \subseteq U \rangle$  by (force simp:  $f'_\text{def}$ )
show  $\{x. \neg (f' x = x \wedge g' x = x)\} \subseteq S$ 
  using  $\langle \text{cbox } w \ z \subseteq S \rangle$  by (auto simp:  $f'_\text{def } g'_\text{def}$ )
show bounded  $\{x. \neg (f' x = x \wedge g' x = x)\}$ 
proof (rule bounded_subset [of cbox w z])
  show bounded (cbox w z)
    using bounded_cbox by blast
  show  $\{x. \neg (f' x = x \wedge g' x = x)\} \subseteq \text{cbox } w \ z$ 
    by (auto simp:  $f'_\text{def } g'_\text{def}$ )
qed
qed
qed

```

proposition homeomorphism_grouping_points_exists:

```

fixes  $S :: 'a::\text{euclidean\_space}$  set
assumes open U open S connected S  $U \neq \{\}$  finite K  $K \subseteq S$   $U \subseteq S$   $S \subseteq T$ 
obtains  $f \ g$  where homeomorphism T T  $f \ g \{x. (\neg (f x = x \wedge g x = x))\} \subseteq S$ 
      bounded  $\{x. (\neg (f x = x \wedge g x = x))\} \bigwedge x. x \in K \implies f x \in U$ 
proof (cases  $2 \leq \text{DIM}('a)$ )
case True
  have TS:  $T \subseteq \text{affine hull } S$ 
    using affine_hull_open assms by blast
  have infinite U
    using  $\langle \text{open } U \rangle \langle U \neq \{\} \rangle$  finite_imp_not_open by blast
  then obtain P where  $P \subseteq U$  finite P  $\text{card } K = \text{card } P$ 
    using infinite_arbitrarily_large by metis
  then obtain  $\gamma$  where  $\gamma$ :  $\text{bij\_betw } \gamma \ K \ P$ 
    using  $\langle \text{finite } K \rangle$  finite_same_card_bij by blast
  obtain  $f \ g$  where homeomorphism T T  $f \ g \bigwedge i. i \in K \implies f (id \ i) = \gamma \ i \{x. \neg$ 
 $(f x = x \wedge g x = x)\} \subseteq S$  bounded  $\{x. \neg (f x = x \wedge g x = x)\}$ 
    proof (rule homeomorphism_moving_points_exists [OF True  $\langle \text{open } S \rangle \langle \text{con-$ 
connected S  $\rangle \langle S \subseteq T \rangle \langle \text{finite } K \rangle$ ])
      show  $\bigwedge i. i \in K \implies id \ i \in S \wedge \gamma \ i \in S$ 
        using  $\langle P \subseteq U \rangle \langle \text{bij\_betw } \gamma \ K \ P \rangle \langle K \subseteq S \rangle \langle U \subseteq S \rangle \text{bij\_betwE}$  by blast
      show pairwise  $(\lambda i \ j. id \ i \neq id \ j \wedge \gamma \ i \neq \gamma \ j) \ K$ 

```

```

    using  $\gamma$  by (auto simp: pairwise_def bij_betw_def inj_on_def)
  qed (use affine_hull_open assms that in auto)
  then show ?thesis
    using  $\gamma$   $\langle P \subseteq U \rangle$  bij_betwE by (fastforce simp add: intro!: that)
next
case False
with DIM_positive have DIM('a) = 1
  by (simp add: dual_order.antisym)
then obtain h::'a  $\Rightarrow$  real and j
where linear h linear j
  and noh:  $\bigwedge x. \text{norm}(h\ x) = \text{norm}\ x$  and noj:  $\bigwedge y. \text{norm}(j\ y) = \text{norm}\ y$ 
  and hj:  $\bigwedge x. j(h\ x) = x$   $\bigwedge y. h(j\ y) = y$ 
  and ranh: surj h
using isomorphisms_UNIV_UNIV
  by (metis (mono_tags, opaque_lifting) DIM_real UNIV_eq_I range_eqI)
obtain f g where hom: homeomorphism (h ' T) (h ' T) f g
  and f:  $\bigwedge x. x \in h\ ' K \implies f\ x \in h\ ' U$ 
  and sub:  $\{x. \neg (f\ x = x \wedge g\ x = x)\} \subseteq h\ ' S$ 
  and bou: bounded  $\{x. \neg (f\ x = x \wedge g\ x = x)\}$ 
  apply (rule homeomorphism_grouping_point_4 [of h ' U h ' S h ' K h ' T])
  by (simp_all add: assms image_mono  $\langle$ linear h $\rangle$  open_surjective_linear_image
connected_linear_image ranh)
have jf:  $j\ (f\ (h\ x)) = x \iff f\ (h\ x) = h\ x$  for x
  by (metis hj)
have jg:  $j\ (g\ (h\ x)) = x \iff g\ (h\ x) = h\ x$  for x
  by (metis hj)
have cont_hj: continuous_on X h continuous_on Y j for X Y
  by (simp_all add:  $\langle$ linear h $\rangle$   $\langle$ linear j $\rangle$  linear_linear linear_continuous_on)
show ?thesis
proof
  show homeomorphism T T (j  $\circ$  f  $\circ$  h) (j  $\circ$  g  $\circ$  h)
proof
  show continuous_on T (j  $\circ$  f  $\circ$  h) continuous_on T (j  $\circ$  g  $\circ$  h)
  using hom homeomorphism_def
  by (blast intro: continuous_on_compose cont_hj)+
  show (j  $\circ$  f  $\circ$  h) ' T  $\subseteq$  T (j  $\circ$  g  $\circ$  h) ' T  $\subseteq$  T
  by auto (metis (mono_tags, opaque_lifting) hj(1) hom homeomorphism_def
imageE imageI)+
  show  $\bigwedge x. x \in T \implies (j \circ g \circ h)\ ((j \circ f \circ h)\ x) = x$ 
  using hj hom homeomorphism_apply1 by fastforce
  show  $\bigwedge y. y \in T \implies (j \circ f \circ h)\ ((j \circ g \circ h)\ y) = y$ 
  using hj hom homeomorphism_apply2 by fastforce
qed
show  $\{x. \neg ((j \circ f \circ h)\ x = x \wedge (j \circ g \circ h)\ x = x)\} \subseteq S$ 
proof (clarsimp simp: jf jg hj)
  show  $f\ (h\ x) = h\ x \implies g\ (h\ x) \neq h\ x \implies x \in S$  for x
  using sub [THEN subsetD, of h x] hj by simp (metis imageE)
qed
have bounded (j '  $\{x. (\neg (f\ x = x \wedge g\ x = x))\}$ )

```



```

    by (rule bounded_linear_image [OF bou]) (use ⟨linear j⟩ linear_conv_bounded_linear
in auto)
  moreover
  have *: {x. ¬((j ∘ f ∘ h) x = x ∧ (j ∘ g ∘ h) x = x)} = j ' {x. (¬ (f x = x ∧
g x = x))}
    using hj by (auto simp: jf jg image_iff,metis+)
  ultimately show bounded {x. ¬ ((j ∘ f ∘ h) x = x ∧ (j ∘ g ∘ h) x = x)}
    by metis
  show  $\bigwedge x. x \in K \implies (j \circ f \circ h) x \in U$ 
    using f hj by fastforce
qed
qed

```

proposition *homeomorphism_grouping_points_exists_gen:*

```

fixes S :: 'a::euclidean_space set
assumes opeU: openin (top_of_set S) U
    and opeS: openin (top_of_set (affine hull S)) S
    and U ≠ {} finite K K ⊆ S and S: S ⊆ T T ⊆ affine hull S connected S
obtains f g where homeomorphism T T f g {x. (¬ (f x = x ∧ g x = x))} ⊆ S
    bounded {x. (¬ (f x = x ∧ g x = x))}  $\bigwedge x. x \in K \implies f x \in U$ 
proof (cases 2 ≤ aff_dim S)
  case True
  have opeU': openin (top_of_set (affine hull S)) U
    using opeS opeU openin_trans by blast
  obtain u where u ∈ U u ∈ S
    using ⟨U ≠ {}⟩ opeU openin_imp_subset by fastforce+
  have infinite U
  proof (rule infinite_openin [OF opeU ⟨u ∈ U⟩])
    show u islimpt S
      using True ⟨u ∈ S⟩ assms(8) connected_imp_perfect_aff_dim by fastforce
  qed
  then obtain P where P ⊆ U finite P card K = card P
    using infinite_arbitrarily_large by metis
  then obtain γ where γ: bij_betw γ K P
    using ⟨finite K⟩ finite_same_card_bij by blast
  have  $\exists f g. \text{homeomorphism } T T f g \wedge (\forall i \in K. f(id\ i) = \gamma\ i) \wedge$ 
    {x. ¬ (f x = x ∧ g x = x)} ⊆ S ∧ bounded {x. ¬ (f x = x ∧ g x = x)}
  proof (rule homeomorphism_moving_points_exists_gen [OF ⟨finite K⟩ _ _
True opeS S])
    show  $\bigwedge i. i \in K \implies id\ i \in S \wedge \gamma\ i \in S$ 
      by (metis id_apply opeU openin_contains_cball subsetCE ⟨P ⊆ U⟩ ⟨bij_betw
γ K P⟩ ⟨K ⊆ S⟩ bij_betwE)
    show pairwise (λi j. id i ≠ id j ∧ γ i ≠ γ j) K
      using γ by (auto simp: pairwise_def bij_betw_def inj_on_def)
  qed
  then show ?thesis
    using γ ⟨P ⊆ U⟩ bij_betwE by (fastforce simp add: intro!: that)
next

```

```

case False
with aff_dim_geq [of S] consider aff_dim S = -1 | aff_dim S = 0 | aff_dim
S = 1 by linarith
then show ?thesis
proof cases
  assume aff_dim S = -1
  then have S = {}
  using aff_dim_empty by blast
  then have False
  using ⟨U ≠ {}⟩ ⟨K ⊆ S⟩ openin_imp_subset [OF opeU] by blast
  then show ?thesis ..
next
  assume aff_dim S = 0
  then obtain a where S = {a}
  using aff_dim_eq_0 by blast
  then have K ⊆ U
  using ⟨U ≠ {}⟩ ⟨K ⊆ S⟩ openin_imp_subset [OF opeU] by blast
  show ?thesis
  using ⟨K ⊆ U⟩ by (intro that [of id id]) (auto intro: homeomorphismI)
next
  assume aff_dim S = 1
  then have affine_hull_S_homeomorphic (UNIV :: real set)
  by (auto simp: homeomorphic_affine_sets)
  then obtain h: 'a ⇒ real and j where homhj: homeomorphism (affine_hull S)
UNIV h j
  using homeomorphic_def by blast
  then have h: ∧x. x ∈ affine_hull S ⇒ j(h(x)) = x and j: ∧y. j y ∈ affine
hull S ∧ h(j y) = y
  by (auto simp: homeomorphism_def)
  have connh: connected (h ' S)
  by (meson Topological_Spaces.connected_continuous_image ⟨connected S⟩
homeomorphism_cont1 homeomorphism_of_subsets homhj hull_subset_top_greatest)
  have hUS: h ' U ⊆ h ' S
  by (meson homeomorphism_imp_open_map homeomorphism_of_subsets
homhj hull_subset_opeS opeU open_UNIV openin_open_eq)
  have opn: openin (top_of_set (affine_hull S)) U ⇒ open (h ' U) for U
  using homeomorphism_imp_open_map [OF homhj] by simp
  have open (h ' U) open (h ' S)
  by (auto intro: opeS opeU openin_trans opn)
  then obtain f g where hom: homeomorphism (h ' T) (h ' U) f g
  and f: ∧x. x ∈ h ' K ⇒ f x ∈ h ' U
  and sub: {x. ¬ (f x = x ∧ g x = x)} ⊆ h ' S
  and bou: bounded {x. ¬ (f x = x ∧ g x = x)}
  apply (rule homeomorphism_grouping_points_exists [of h ' U h ' S h ' K h
' T])
  using assms by (auto simp: connh hUS)
  have jf: ∧x. x ∈ affine_hull S ⇒ j (f (h x)) = x ⇔ f (h x) = h x
  by (metis h j)
  have jg: ∧x. x ∈ affine_hull S ⇒ j (g (h x)) = x ⇔ g (h x) = h x

```

```

    by (metis h j)
  have cont_hj: continuous_on T h continuous_on Y j for Y
  proof (rule continuous_on_subset [OF _ ⟨T ⊆ affine hull S⟩])
    show continuous_on (affine hull S) h
      using homeomorphism_def homhj by blast
  qed (meson continuous_on_subset homeomorphism_def homhj top_greatest)
  define f' where f' ≡ λx. if x ∈ affine hull S then (j ∘ f ∘ h) x else x
  define g' where g' ≡ λx. if x ∈ affine hull S then (j ∘ g ∘ h) x else x
  show ?thesis
  proof
    show homeomorphism T T f' g'
    proof
      have continuous_on T (j ∘ f ∘ h)
        using hom homeomorphism_def by (intro continuous_on_compose
cont_hj) blast
      then show continuous_on T f'
        apply (rule continuous_on_eq)
        using ⟨T ⊆ affine hull S⟩ f'_def by auto
      have continuous_on T (j ∘ g ∘ h)
        using hom homeomorphism_def by (intro continuous_on_compose
cont_hj) blast
      then show continuous_on T g'
        apply (rule continuous_on_eq)
        using ⟨T ⊆ affine hull S⟩ g'_def by auto
      show f' ' T ⊆ T
      proof (clarsimp simp: f'_def)
        fix x assume x ∈ T
        then have f (h x) ∈ h ' T
          by (metis (no_types) hom homeomorphism_def image_subset_iff sub-
set_refl)
        then show j (f (h x)) ∈ T
          using ⟨T ⊆ affine hull S⟩ h by auto
      qed
      show g' ' T ⊆ T
      proof (clarsimp simp: g'_def)
        fix x assume x ∈ T
        then have g (h x) ∈ h ' T
          by (metis (no_types) hom homeomorphism_def image_subset_iff sub-
set_refl)
        then show j (g (h x)) ∈ T
          using ⟨T ⊆ affine hull S⟩ h by auto
      qed
      show ∧x. x ∈ T ⇒ g' (f' x) = x
        using h j hom homeomorphism_apply1 by (fastforce simp add: f'_def
g'_def)
      show ∧y. y ∈ T ⇒ f' (g' y) = y
        using h j hom homeomorphism_apply2 by (fastforce simp add: f'_def
g'_def)
    qed
  qed

```

```

next
  have §:  $\bigwedge x y. \llbracket x \in \text{affine hull } S; h\ x = h\ y; y \in S \rrbracket \implies x \in S$ 
    by (metis h hull_inc)
  show  $\{x. \neg (f'\ x = x \wedge g'\ x = x)\} \subseteq S$ 
    using sub by (simp add: f'_def g'_def jf jg) (force elim: §)
next
  have compact (j ' closure  $\{x. \neg (f\ x = x \wedge g\ x = x)\}$ )
    using bou by (auto simp: compact_continuous_image cont_hj)
  then have bounded (j '  $\{x. \neg (f\ x = x \wedge g\ x = x)\}$ )
    by (rule bounded_closure_image [OF compact_imp_bounded])
  moreover
  have *:  $\{x \in \text{affine hull } S. j\ (f\ (h\ x)) \neq x \vee j\ (g\ (h\ x)) \neq x\} = j\ ' \{x. (\neg (f\ x = x \wedge g\ x = x))\}$ 
    using h j by (auto simp: image_iff; metis)
  ultimately have bounded  $\{x \in \text{affine hull } S. j\ (f\ (h\ x)) \neq x \vee j\ (g\ (h\ x)) \neq x\}$ 
    by metis
  then show bounded  $\{x. \neg (f'\ x = x \wedge g'\ x = x)\}$ 
    by (simp add: f'_def g'_def Collect_mono bounded_subset)
next
  show  $f'\ x \in U$  if  $x \in K$  for  $x$ 
  proof -
    have  $U \subseteq S$ 
      using opeU openin_imp_subset by blast
    then have  $j\ (f\ (h\ x)) \in U$ 
      using f h hull_subset that by fastforce
    then show  $f'\ x \in U$ 
      using  $\langle K \subseteq S \rangle S\ f'_\text{def}$  that by auto
  qed
qed
qed
qed
qed

```

6.18.29 Nullhomotopic mappings

A mapping out of a sphere is nullhomotopic iff it extends to the ball. This even works out in the degenerate cases when the radius is ≤ 0 , and we also don't need to explicitly assume continuity since it's already implicit in both sides of the equivalence.

```

lemma nullhomotopic_from_lemma:
  assumes contg: continuous_on (cball a r - {a}) g
    and fa:  $\bigwedge e. 0 < e$ 
       $\implies \exists d. 0 < d \wedge (\forall x. x \neq a \wedge \text{norm}(x - a) < d \implies \text{norm}(g\ x - f\ a) < e)$ 
    and r:  $\bigwedge x. x \in \text{cball } a\ r \wedge x \neq a \implies f\ x = g\ x$ 
  shows continuous_on (cball a r) f
proof (clarsimp simp: continuous_on_eq_continuous_within Ball_def)
  fix x

```

```

assume x: dist a x ≤ r
show continuous (at x within cball a r) f
proof (cases x=a)
  case True
  then show ?thesis
    by (metis continuous_within_eps_delta fa dist_norm dist_self r)
next
  case False
  show ?thesis
proof (rule continuous_transform_within [where f=g and d = norm(x-a)])
  have ∃ d>0. ∀ x'∈cball a r.
    dist x' x < d ⟶ dist (g x') (g x) < e if e>0 for e
  proof -
    obtain d where d > 0
    and d: ∧x'. [dist x' a ≤ r; x' ≠ a; dist x' x < d] ⟹
      dist (g x') (g x) < e
    using contg False x ⟨e>0⟩
    unfolding continuous_on_iff by (fastforce simp add: dist_commute intro:
that)
    show ?thesis
    using ⟨d > 0⟩ ⟨x ≠ a⟩
    by (rule_tac x=min d (norm(x - a)) in exI)
      (auto simp: dist_commute dist_norm [symmetric] intro!: d)
  qed
  then show continuous (at x within cball a r) g
    using contg False by (auto simp: continuous_within_eps_delta)
  show 0 < norm (x - a)
    using False by force
  show x ∈ cball a r
    by (simp add: x)
  show ∧x'. [x' ∈ cball a r; dist x' x < norm (x - a)]
    ⟹ g x' = f x'
    by (metis dist_commute dist_norm less_le r)
  qed
qed
qed
qed

proposition nullhomotopic_from_sphere_extension:
fixes f :: 'M::euclidean_space ⇒ 'a::real_normed_vector
shows (∃ c. homotopic_with_canon (λx. True) (sphere a r) S f (λx. c)) ⟷
  (∃ g. continuous_on (cball a r) g ∧ g ' (cball a r) ⊆ S ∧
    (∀ x ∈ sphere a r. g x = f x))
  (is ?lhs = ?rhs)
proof (cases r 0::real rule: linorder_cases)
  case less
  then show ?thesis
    by (simp add: homotopic_on_emptyI)
next
  case equal

```

```

show ?thesis
proof
  assume L: ?lhs
  with equal have [simp]:  $f\ a \in S$ 
    using homotopic_with_imp_subset1 by fastforce
  obtain h::  $\text{real} \times 'M \Rightarrow 'a$ 
    where h: continuous_on  $(\{0..1\} \times \{a\})$  h h ' $(\{0..1\} \times \{a\}) \subseteq S$  h  $(0, a)$ 
= f a
    using L equal by (auto simp: homotopic_with)
  then have continuous_on  $(\text{cball } a\ r)$   $(\lambda x. h\ (0, a))$   $(\lambda x. h\ (0, a))$  ' $\text{cball } a\ r$ 
 $\subseteq S$ 
    by (auto simp: equal)
  then show ?rhs
    using h(3) local.equal by force
next
  assume ?rhs
  then show ?lhs
    using equal continuous_on_const by (force simp add: homotopic_with)
qed
next
  case greater
  let ?P = continuous_on  $\{x. \text{norm}(x - a) = r\}$  f  $\wedge f$  ' $\{x. \text{norm}(x - a) = r\}$ 
 $\subseteq S$ 
  have ?P if ?lhs using that
  proof
    fix c
    assume c: homotopic_with_canon  $(\lambda x. \text{True})$   $(\text{sphere } a\ r)$  S f  $(\lambda x. c)$ 
    then have contf: continuous_on  $(\text{sphere } a\ r)$  f
      by (metis homotopic_with_imp_continuous)
    moreover have fim:  $f$  ' $\text{sphere } a\ r \subseteq S$ 
    by (meson continuous_map_subtopology_eu c homotopic_with_imp_continuous_maps)
    show ?P
      using contf fim by (auto simp: sphere_def dist_norm norm_minus_commute)
  qed
  moreover have ?P if ?rhs using that
  proof
    fix g
    assume g: continuous_on  $(\text{cball } a\ r)$  g  $\wedge g$  ' $\text{cball } a\ r \subseteq S \wedge (\forall xa \in \text{sphere } a\ r. g\ xa = f\ xa)$ 
    then have f ' $\{x. \text{norm}(x - a) = r\} \subseteq S$ 
      using sphere_cball [of a r] unfolding image_subset_iff sphere_def
      by (metis dist_commute dist_norm mem_Collect_eq subset_eq)
    with g show ?P
      by (auto simp: dist_norm norm_minus_commute elim!: continuous_on_eq
[OF continuous_on_subset])
  qed
  moreover have ?thesis if ?P
  proof
    assume ?lhs

```

```

then obtain c where homotopic_with_canon ( $\lambda x. \text{True}$ ) (sphere a r) S ( $\lambda x. c$ ) f
using homotopic_with_sym by blast
then obtain h where conth: continuous_on ( $\{0..1::\text{real}\} \times \text{sphere a r}$ ) h
and him:  $h \restriction (\{0..1\} \times \text{sphere a r}) \subseteq S$ 
and h:  $\bigwedge x. h(0, x) = c \wedge \bigwedge x. h(1, x) = f x$ 
by (auto simp: homotopic_with_def)
obtain b1::'M where b1  $\in$  Basis
using SOME_Basis by auto
have c  $\in$  h  $\restriction (\{0..1\} \times \text{sphere a r})$ 
proof
show c = h (0, a + r *R b1)
by (simp add: h)
show (0, a + r *R b1)  $\in$   $\{0..1::\text{real}\} \times \text{sphere a r}$ 
using greater  $\langle b1 \in \text{Basis} \rangle$  by (auto simp: dist_norm)
qed
then have c  $\in$  S
using him by blast
have uconth: uniformly_continuous_on ( $\{0..1::\text{real}\} \times (\text{sphere a r})$ ) h
by (force intro: compact_Times conth compact_uniformly_continuous)
let ?g =  $\lambda x. h (\text{norm } (x - a) / r,$ 
a + (if x = a then r *R b1 else (r / norm(x - a)) *R (x - a)))
let ?g' =  $\lambda x. h (\text{norm } (x - a) / r, a + (r / \text{norm}(x - a)) *R (x - a))$ 
show ?rhs
proof (intro exI conjI)
have continuous_on (cball a r - {a}) ?g'
using greater
by (force simp: dist_norm norm_minus_commute intro: continuous_on_compose2
[OF conth] continuous_intros)
then show continuous_on (cball a r) ?g
proof (rule nullhomotopic_from_lemma)
show  $\exists d > 0. \forall x. x \neq a \wedge \text{norm } (x - a) < d \longrightarrow \text{norm } (?g' x - ?g a) < e$ 
if 0 < e for e
proof -
obtain d where 0 < d
and d:  $\bigwedge x x'. \llbracket x \in \{0..1\} \times \text{sphere a r}; x' \in \{0..1\} \times \text{sphere a r}; \text{norm } (x' - x) < d \rrbracket$ 
 $\implies \text{norm } (h x' - h x) < e$ 
using uniformly_continuous_onE [OF uconth  $\langle 0 < e \rangle$ ] by (auto simp:
dist_norm)
have *:  $\text{norm } (h (\text{norm } (x - a) / r,$ 
a + (r / norm(x - a)) *R (x - a)) - h (0, a + r *R b1)) <
e (is  $\text{norm } (?ha - ?hb) < e$ )
if x  $\neq$  a  $\text{norm } (x - a) < r$   $\text{norm } (x - a) < d * r$  for x
proof -
have  $\text{norm } (?ha - ?hb) = \text{norm } (?ha - h (0, a + (r / \text{norm } (x - a))$ 
*R (x - a)))
by (simp add: h)
also have ... < e

```

```

      using greater ⟨0 < d⟩ ⟨b1 ∈ Basis⟩ that
    by (intro d) (simp_all add: dist_norm, simp add: field_simps)
  finally show ?thesis .
qed
show ?thesis
  using greater ⟨0 < d⟩
  by (rule_tac x = min r (d * r) in exI) (auto simp: *)
qed
show  $\bigwedge x. x \in \text{cball } a \ r \wedge x \neq a \implies ?g \ x = ?g' \ x$ 
  by auto
qed
next
  show  $?g \text{ ' } \text{cball } a \ r \subseteq S$ 
    using greater him ⟨c ∈ S⟩
    by (force simp: h dist_norm norm_minus_commute)
next
  show  $\forall x \in \text{sphere } a \ r. ?g \ x = f \ x$ 
    using greater by (auto simp: h dist_norm norm_minus_commute)
qed
next
  assume ?rhs
  then obtain g where contg: continuous_on (cball a r) g
    and gim:  $g \text{ ' } \text{cball } a \ r \subseteq S$ 
    and gf:  $\forall x \in \text{sphere } a \ r. g \ x = f \ x$ 
    by auto
  let ?h =  $\lambda y. g \ (a + (\text{fst } y) *_R (\text{snd } y - a))$ 
  have continuous_on ({0..1} × sphere a r) ?h
  proof (rule continuous_on_compose2 [OF contg])
    show continuous_on ({0..1} × sphere a r) ( $\lambda x. a + \text{fst } x *_R (\text{snd } x - a)$ )
      by (intro continuous_intros)
    qed (auto simp: dist_norm norm_minus_commute mult_left_le_one_le)
  moreover
  have  $?h \text{ ' } ({0..1} \times \text{sphere } a \ r) \subseteq S$ 
    by (auto simp: dist_norm norm_minus_commute mult_left_le_one_le gim)
  [THEN subsetD])
  moreover
  have  $\forall x \in \text{sphere } a \ r. ?h \ (0, x) = g \ a \ \forall x \in \text{sphere } a \ r. ?h \ (1, x) = f \ x$ 
    by (auto simp: dist_norm norm_minus_commute mult_left_le_one_le gf)
  ultimately have homotopic_with_canon ( $\lambda x. \text{True}$ ) (sphere a r) S ( $\lambda x. g \ a$ ) f
    by (auto simp: homotopic_with)
  then show ?lhs
    using homotopic_with_symD by blast
qed
ultimately
show ?thesis by meson
qed
end

```


6.19 Homeomorphism Theorems

theory *Homeomorphism*

imports *Homotopy*

begin

lemma *homeomorphic_spheres'*:

fixes $a :: 'a::\text{euclidean\_space}$ **and** $b :: 'b::\text{euclidean\_space}$

assumes $0 < \delta$ **and** *dimeq*: $\text{DIM}('a) = \text{DIM}('b)$

shows (*sphere* a δ) *homeomorphic* (*sphere* b δ)

proof –

obtain $f :: 'a \Rightarrow 'b$ **and** g **where** *linear* f *linear* g

and *fg*: $\bigwedge x. \text{norm}(f\ x) = \text{norm}\ x \bigwedge y. \text{norm}(g\ y) = \text{norm}\ y \bigwedge x. g(f\ x) = x$
 $\bigwedge y. f(g\ y) = y$

by (*blast intro: isomorphisms_UNIV_UNIV [OF dimeq]*)

then have *continuous_on UNIV* f *continuous_on UNIV* g

using *linear_continuous_on linear_linear* **by** *blast+*

then show *?thesis*

unfolding *homeomorphic_minimal*

apply(*rule_tac* $x = \lambda x. b + f(x - a)$ **in** *exI*)

apply(*rule_tac* $x = \lambda x. a + g(x - b)$ **in** *exI*)

using *assms*

apply (*force intro: continuous_intros*

continuous_on_compose2 [of _ f] continuous_on_compose2 [of _
 $g]$ *simp: dist_commute dist_norm fg*)

done

qed

lemma *homeomorphic_spheres_gen*:

fixes $a :: 'a::\text{euclidean\_space}$ **and** $b :: 'b::\text{euclidean\_space}$

assumes $0 < r$ $0 < s$ $\text{DIM}('a::\text{euclidean\_space}) = \text{DIM}('b::\text{euclidean\_space})$

shows (*sphere* a r) *homeomorphic* (*sphere* b s)

using *assms homeomorphic_trans [OF homeomorphic_spheres homeomorphic_spheres']*
by *auto*

6.19.1 Homeomorphism of all convex compact sets with nonempty interior

proposition

fixes $S :: 'a::\text{euclidean\_space}$ *set*

assumes *compact* S **and** 0 : $0 \in \text{rel\_interior}\ S$

and *star*: $\bigwedge x. x \in S \implies \text{open\_segment}\ 0\ x \subseteq \text{rel\_interior}\ S$

shows *starlike_compact_projective1_0*:

$S - \text{rel\_interior}\ S$ *homeomorphic* *sphere* $0\ 1 \cap \text{affine hull}\ S$

(*is ?SMINUS homeomorphic ?SPHER*)

and *starlike_compact_projective2_0*:

S *homeomorphic* *cball* $0\ 1 \cap \text{affine hull}\ S$

(*is* S *homeomorphic* *?CBALL*)

proof –

```

have starI:  $(u *_{\mathbb{R}} x) \in \text{rel\_interior } S$  if  $x \in S$   $0 \leq u$   $u < 1$  for  $x \ u$ 
proof (cases  $x=0 \vee u=0$ )
  case True with 0 show ?thesis by force
next
  case False with that show ?thesis
    by (auto simp: in_segment intro: star [THEN subsetD])
qed
have  $0 \in S$  using assms rel_interior_subset by auto
define proj where  $\text{proj} \equiv \lambda x::'a. x /_{\mathbb{R}} \text{norm } x$ 
have eqI:  $x = y$  if  $\text{proj } x = \text{proj } y$   $\text{norm } x = \text{norm } y$  for  $x \ y$ 
  using that by (force simp: proj_def)
then have iff_eq:  $\bigwedge x \ y. (\text{proj } x = \text{proj } y \wedge \text{norm } x = \text{norm } y) \longleftrightarrow x = y$ 
  by blast
have projI:  $x \in \text{affine hull } S \implies \text{proj } x \in \text{affine hull } S$  for  $x$ 
  by (metis  $\langle 0 \in S \rangle$  affine_hull_span_0 hull_inc span_mul proj_def)
have nproj1 [simp]:  $x \neq 0 \implies \text{norm}(\text{proj } x) = 1$  for  $x$ 
  by (simp add: proj_def)
have proj0_iff [simp]:  $\text{proj } x = 0 \longleftrightarrow x = 0$  for  $x$ 
  by (simp add: proj_def)
have cont_proj: continuous_on  $(\text{UNIV} - \{0\})$  proj
  unfolding proj_def by (rule continuous_intros | force)+
have proj_spherI:  $\bigwedge x. \llbracket x \in \text{affine hull } S; x \neq 0 \rrbracket \implies \text{proj } x \in ?\text{SPHER}$ 
  by (simp add: projI)
have bounded S closed S
  using  $\langle \text{compact } S \rangle$  compact_eq_bounded_closed by blast+
have inj_on_proj: inj_on proj  $(S - \text{rel\_interior } S)$ 
proof
  fix  $x \ y$ 
  assume  $x: x \in S - \text{rel\_interior } S$ 
    and  $y: y \in S - \text{rel\_interior } S$  and eq:  $\text{proj } x = \text{proj } y$ 
  then have xynot:  $x \neq 0 \ y \neq 0 \ x \in S \ y \in S \ x \notin \text{rel\_interior } S \ y \notin \text{rel\_interior } S$ 
S
    using 0 by auto
  consider  $\text{norm } x = \text{norm } y \mid \text{norm } x < \text{norm } y \mid \text{norm } x > \text{norm } y$  by linarith
  then show  $x = y$ 
  proof cases
    assume  $\text{norm } x = \text{norm } y$ 
      with iff_eq eq show  $x = y$  by blast
  next
    assume *:  $\text{norm } x < \text{norm } y$ 
    have  $x /_{\mathbb{R}} \text{norm } x = \text{norm } x *_{\mathbb{R}} (x /_{\mathbb{R}} \text{norm } x) /_{\mathbb{R}} \text{norm } (\text{norm } x *_{\mathbb{R}} (x /_{\mathbb{R}} \text{norm } x))$ 
    norm x))
      by force
    then have proj  $((\text{norm } x / \text{norm } y) *_{\mathbb{R}} y) = \text{proj } x$ 
      by (metis (no_types) divide_inverse local.proj_def eq scaleR_scaleR)
    then have [simp]:  $(\text{norm } x / \text{norm } y) *_{\mathbb{R}} y = x$ 
      by (rule eqI) (simp add:  $\langle y \neq 0 \rangle$ )
    have no:  $0 \leq \text{norm } x / \text{norm } y$   $\text{norm } x / \text{norm } y < 1$ 
      using * by (auto simp: field_split_simps)

```

```

    then show  $x = y$ 
      using starI [OF  $\langle y \in S \rangle$  no] xynot by auto
  next
    assume *:  $\text{norm } x > \text{norm } y$ 
    have  $y /_R \text{norm } y = \text{norm } y *_R (y /_R \text{norm } y) /_R \text{norm } (\text{norm } y *_R (y /_R \text{norm } y))$ 
    by force
    then have  $\text{proj } ((\text{norm } y / \text{norm } x) *_R x) = \text{proj } y$ 
      by (metis (no_types) divide_inverse local.proj_def eq scaleR_scaleR)
    then have [simp]:  $(\text{norm } y / \text{norm } x) *_R x = y$ 
      by (rule eqI) (simp add:  $\langle x \neq 0 \rangle$ )
    have no:  $0 \leq \text{norm } y / \text{norm } x \text{ norm } y / \text{norm } x < 1$ 
      using * by (auto simp: field_split_simps)
    then show  $x = y$ 
      using starI [OF  $\langle x \in S \rangle$  no] xynot by auto
  qed
qed
have  $\exists \text{surf. homeomorphism } (S - \text{rel\_interior } S) \text{ ?SPHER proj surf}$ 
proof (rule homeomorphism_compact)
  show compact  $(S - \text{rel\_interior } S)$ 
    using  $\langle \text{compact } S \rangle$  compact_rel_boundary by blast
  show continuous_on  $(S - \text{rel\_interior } S)$  proj
    using 0 by (blast intro: continuous_on_subset [OF cont_proj])
  show  $\text{proj } ' (S - \text{rel\_interior } S) = \text{?SPHER}$ 
  proof
    show  $\text{proj } ' (S - \text{rel\_interior } S) \subseteq \text{?SPHER}$ 
      using 0 by (force simp: hull_inc projI intro: nproj1)
    show  $\text{?SPHER} \subseteq \text{proj } ' (S - \text{rel\_interior } S)$ 
      proof (clarsimp simp: proj_def)
        fix x
        assume  $x \in \text{affine hull } S$  and nox:  $\text{norm } x = 1$ 
        then have  $x \neq 0$  by auto
        obtain d where  $0 < d$  and dx:  $(d *_R x) \in \text{rel\_frontier } S$ 
          and ri:  $\bigwedge e. [0 \leq e; e < d] \implies (e *_R x) \in \text{rel\_interior } S$ 
          using ray_to_rel_frontier [OF  $\langle \text{bounded } S \rangle$  0]  $\langle x \in \text{affine hull } S \rangle \langle x \neq 0 \rangle$  by auto
        show  $x \in (\lambda x. x /_R \text{norm } x) ' (S - \text{rel\_interior } S)$ 
        proof
          show  $x = d *_R x /_R \text{norm } (d *_R x)$ 
            using  $\langle 0 < d \rangle$  by (auto simp: nox)
          show  $d *_R x \in S - \text{rel\_interior } S$ 
            using dx  $\langle \text{closed } S \rangle$  by (auto simp: rel_frontier_def)
        qed
      qed
  qed
qed
qed
then obtain surf where surf:  $\text{homeomorphism } (S - \text{rel\_interior } S) \text{ ?SPHER proj surf}$ 
by blast

```

```

then have cont_surf: continuous_on (proj ` (S - rel_interior S)) surf
  by (auto simp: homeomorphism_def)
have surf_nz:  $\bigwedge x. x \in ?SPHER \implies surf\ x \neq 0$ 
  by (metis 0 DiffE homeomorphism_def imageI surf)
have cont_nosp: continuous_on (?SPHER) ( $\lambda x. norm\ x *_R ((surf\ o\ proj)\ x)$ )
proof (intro continuous_intros)
  show continuous_on (sphere 0 1  $\cap$  affine hull S) proj
    by (rule continuous_on_subset [OF cont_proj], force)
  show continuous_on (proj ` (sphere 0 1  $\cap$  affine hull S)) surf
    by (intro continuous_on_subset [OF cont_surf]) (force simp: homeomorphism_image1 [OF surf] dest: proj_spherI)
qed
have surfpS:  $\bigwedge x. [norm\ x = 1; x \in \text{affine hull } S] \implies surf\ (proj\ x) \in S$ 
  by (metis (full_types) DiffE  $\langle 0 \in S \rangle$  homeomorphism_def image_eqI norm_zero proj_spherI real_vector.scale_zero_left scaleR_one surf)
have *:  $\exists y. norm\ y = 1 \wedge y \in \text{affine hull } S \wedge x = surf\ (proj\ y)$ 
  if  $x \in S \wedge x \notin \text{rel\_interior } S$  for x
proof -
  have proj_x  $\in ?SPHER$ 
    by (metis (full_types) 0 hull_inc proj_spherI that)
  moreover have surf (proj x) = x
    by (metis Diff_iff homeomorphism_def surf that)
  ultimately show ?thesis
    by (metis  $\langle \bigwedge x. x \in ?SPHER \implies surf\ x \neq 0 \rangle$  hull_inc inverse_1 local.proj_def norm_sgn projI scaleR_one sgn_div_norm that(1))
qed
have surfp_notin:  $\bigwedge x. [norm\ x = 1; x \in \text{affine hull } S] \implies surf\ (proj\ x) \notin \text{rel\_interior } S$ 
  by (metis (full_types) DiffE one_neq_zero homeomorphism_def image_eqI norm_zero proj_spherI surf)
have no_sp_im: ( $\lambda x. norm\ x *_R surf\ (proj\ x)$ ) ` (?SPHER) = S - rel_interior S
  by (auto simp: surfpS image_def Bex_def surfp_notin *)
have inj_spher: inj_on ( $\lambda x. norm\ x *_R surf\ (proj\ x)$ ) ?SPHER
proof
  fix x y
  assume xy:  $x \in ?SPHER \wedge y \in ?SPHER$ 
  and eq:  $norm\ x *_R surf\ (proj\ x) = norm\ y *_R surf\ (proj\ y)$ 
  then have  $norm\ x = 1 \wedge norm\ y = 1 \wedge x \in \text{affine hull } S \wedge y \in \text{affine hull } S$ 
    using 0 by auto
  with eq show x = y
    by (simp add: proj_def) (metis surf xy homeomorphism_def)
qed
have co01: compact ?SPHER
  by (simp add: compact_Int_closed)
show ?SMINUS homeomorphic ?SPHER
  using homeomorphic_def surf by blast
have proj_scaleR:  $\bigwedge a\ x. 0 < a \implies proj\ (a *_R x) = proj\ x$ 
  by (simp add: proj_def)

```

```

have cont_sp0: continuous_on (affine hull S - {0}) (surf o proj)
proof (rule continuous_on_compose [OF continuous_on_subset [OF cont_proj]])
  show continuous_on (proj ' (affine hull S - {0}) ) surf
  using homeomorphism_image1 proj_spherI surf by (intro continuous_on_subset
[OF cont_surf]) fastforce
qed auto
obtain B where B>0 and B:  $\bigwedge x. x \in S \implies \text{norm } x \leq B$ 
by (metis compact_imp_bounded ⟨compact S⟩ bounded_pos less_eq_real_def)
have cont_nosp: continuous (at x within ?CBALL) ( $\lambda x. \text{norm } x *_R \text{surf } (\text{proj } x)$ )
x))
  if norm x ≤ 1 x ∈ affine hull S for x
proof (cases x=0)
case True
  have (norm  $\longrightarrow$  0) (at 0 within cball 0 1  $\cap$  affine hull S)
  by (simp add: tendsto_norm_zero eventually_at)
  with True show ?thesis
  apply (simp add: continuous_within)
  apply (rule lim_null_scaleR_bounded [where B=B])
  using B ⟨0 < B⟩ local.proj_def projI surfpS by (auto simp: eventually_at)
next
case False
  then have  $\forall_F x \text{ in } at x. (x \in \text{affine hull } S - \{0\}) = (x \in \text{affine hull } S)$ 
  by (force simp: False eventually_at)
  moreover
  have continuous (at x within affine hull S - {0}) ( $\lambda x. \text{surf } (\text{proj } x)$ )
  using cont_sp0 False that by (auto simp add: continuous_on_eq_continuous_within)
  ultimately have *: continuous (at x within affine hull S) ( $\lambda x. \text{surf } (\text{proj } x)$ )
  by (simp add: continuous_within Lim_transform_within_set continuous_on_eq_continuous_within)
  show ?thesis
  by (intro continuous_within_subset [where s = affine hull S, OF Int_lower2]
continuous_intros *)
qed
have cont_nosp2: continuous_on ?CBALL ( $\lambda x. \text{norm } x *_R ((\text{surf } o \text{proj}) x)$ )
by (simp add: continuous_on_eq_continuous_within cont_nosp)
have norm y *_R surf (proj y) ∈ S if y ∈ cball 0 1 and yaff: y ∈ affine hull S
for y
proof (cases y=0)
case True then show ?thesis
by (simp add: ⟨0 ∈ S⟩)
next
case False
  then have norm y *_R surf (proj y) = norm y *_R surf (proj (y /R norm y))
  by (simp add: proj_def)
  have norm y ≤ 1 using that by simp
  have surf (proj (y /R norm y)) ∈ S
  using False local.proj_def nproj1 projI surfpS yaff by blast
  then have surf (proj y) ∈ S
  by (simp add: False proj_def)
  then show norm y *_R surf (proj y) ∈ S

```

```

    by (metis dual_order.antisym le_less_linear norm_ge_zero rel_interior_subset
scaleR_one
      starI subset_eq ⟨norm y ≤ 1⟩)
qed
moreover have x ∈ (λx. norm x *R surf (proj x)) ‘ (?CBALL) if x ∈ S for x
proof (cases x=0)
  case True with that hull_inc show ?thesis by fastforce
next
  case False
  then have psp: proj (surf (proj x)) = proj x
    by (metis homeomorphism_def hull_inc proj_spherI surf that)
  have nxx: norm x *R proj x = x
    by (simp add: False local.proj_def)
  have affineI: (1 / norm (surf (proj x))) *R x ∈ affine hull S
    by (metis ⟨0 ∈ S⟩ affine_hull_span_0 hull_inc span_clauses(4) that)
  have sproj_nz: surf (proj x) ≠ 0
    by (metis False proj0_iff psp)
  then have proj x = proj (proj x)
    by (metis False nxx proj_scaleR zero_less_norm_iff)
  moreover have scaleproj: ∧a r. r *R proj a = (r / norm a) *R a
    by (simp add: divide_inverse local.proj_def)
  ultimately have (norm (surf (proj x)) / norm x) *R x ∉ rel_interior S
    by (metis (no_types) sproj_nz divide_self_if hull_inc norm_eq_zero nproj1
projI psp scaleR_one surfp_notin that)
  then have (norm (surf (proj x)) / norm x) ≥ 1
    using starI [OF that] by (meson starI [OF that] le_less_linear norm_ge_zero
zero_le_divide_iff)
  then have nole: norm x ≤ norm (surf (proj x))
    by (simp add: le_divide_eq_1)
  let ?inx = x /R norm (surf (proj x))
  show ?thesis
proof
  show x = norm ?inx *R surf (proj ?inx)
  by (simp add: field_simps) (metis inverse_eq_divide nxx positive_imp_inverse_positive
proj_scaleR psp scaleproj sproj_nz zero_less_norm_iff)
  qed (auto simp: field_split_simps nole affineI)
qed
ultimately have im_cball: (λx. norm x *R surf (proj x)) ‘ ?CBALL = S
  by blast
have inj_cball: inj_on (λx. norm x *R surf (proj x)) ?CBALL
proof
  fix x y
  assume x ∈ ?CBALL y ∈ ?CBALL
  and eq: norm x *R surf (proj x) = norm y *R surf (proj y)
  then have x: x ∈ affine hull S and y: y ∈ affine hull S
    using 0 by auto
  show x = y
proof (cases x=0 ∨ y=0)
  case True then show x = y using eq proj_spherI surf_nz x y by force

```

```

next
  case False
  with x y have speq: surf (proj x) = surf (proj y)
  by (metis eq homeomorphism_apply2 proj_scaleR proj_spherI surf zero_less_norm_iff)
  then have norm x = norm y
  by (metis ‹x ∈ affine hull S› ‹y ∈ affine hull S› eq proj_spherI real_vector.scale_cancel_right
surf_nz)
  moreover have proj x = proj y
  by (metis (no_types) False speq homeomorphism_apply2 proj_spherI surf x
y)
  ultimately show x = y
  using eq eqI by blast
qed
qed
have co01: compact ?CBALL
  by (simp add: compact_Int_closed)
show S homeomorphic ?CBALL
  using homeomorphic_compact [OF co01 cont_nosp2 [unfolded o_def] im_cball
inj_cball] homeomorphic_sym by blast
qed

corollary
  fixes S :: 'a::euclidean_space set
  assumes compact S and a: a ∈ rel_interior S
  and star:  $\bigwedge x. x \in S \implies \text{open\_segment } a \ x \subseteq \text{rel\_interior } S$ 
  shows starlike_compact_projective1:
    S - rel_interior S homeomorphic sphere a 1  $\cap$  affine hull S
  and starlike_compact_projective2:
    S homeomorphic cball a 1  $\cap$  affine hull S
proof -
  have 1: compact ((+) (-a) ' S) by (meson assms compact_translation)
  have 2: 0 ∈ rel_interior ((+) (-a) ' S)
  using a rel_interior_translation [of - a S] by (simp cong: image_cong_simp)
  have 3: open_segment 0 x  $\subseteq$  rel_interior ((+) (-a) ' S) if x ∈ ((+) (-a) ' S)
for x
proof -
  have x+a ∈ S using that by auto
  then have open_segment a (x+a)  $\subseteq$  rel_interior S by (metis star)
  then show ?thesis using open_segment_translation [of a 0 x]
  using rel_interior_translation [of - a S] by (fastforce simp add: ac_simps
image_iff cong: image_cong_simp)
qed
have S - rel_interior S homeomorphic ((+) (-a) ' S) - rel_interior ((+) (-a)
' S)
  by (metis rel_interior_translation translation_diff homeomorphic_translation)
also have ... homeomorphic sphere 0 1  $\cap$  affine hull ((+) (-a) ' S)
  by (rule starlike_compact_projective1_0 [OF 1 2 3])
also have ... = (+) (-a) ' (sphere a 1  $\cap$  affine hull S)
  by (metis affine_hull_translation left_minus sphere_translation translation_Int)

```

also have ... *homeomorphic sphere* $a \cdot 1 \cap \text{affine hull } S$
 using *homeomorphic_translation homeomorphic_sym* by blast
 finally show $S - \text{rel\_interior } S$ *homeomorphic sphere* $a \cdot 1 \cap \text{affine hull } S$.

have S *homeomorphic* $((+) (-a) \cdot S)$
 by (*metis homeomorphic_translation*)
 also have ... *homeomorphic cball* $0 \cdot 1 \cap \text{affine hull } ((+) (-a) \cdot S)$
 by (*rule starlike_compact_projective2_0 [OF 1 2 3]*)
 also have ... $= (+) (-a) \cdot (\text{cball } a \cdot 1 \cap \text{affine hull } S)$
 by (*metis affine_hull_translation left_minus cball_translation translation_Int*)
 also have ... *homeomorphic cball* $a \cdot 1 \cap \text{affine hull } S$
 using *homeomorphic_translation homeomorphic_sym* by blast
 finally show S *homeomorphic cball* $a \cdot 1 \cap \text{affine hull } S$.

qed

corollary *starlike_compact_projective_special*:

assumes *compact* S
 and *cb01*: *cball* $(0::'a::\text{euclidean\_space}) \cdot 1 \subseteq S$
 and *scale*: $\bigwedge x u. \llbracket x \in S; 0 \leq u; u < 1 \rrbracket \implies u *_R x \in S - \text{frontier } S$
 shows S *homeomorphic* $(\text{cball } (0::'a::\text{euclidean\_space}) \cdot 1)$

proof –

have *ball* $0 \cdot 1 \subseteq \text{interior } S$
 using *cb01 interior_cball interior_mono* by blast
 then have *0*: $0 \in \text{rel\_interior } S$
 by (*meson centre_in_ball subsetD interior_subset_rel_interior le_numeral_extra(2)*)
 not_le)

have [*simp*]: *affine hull* $S = \text{UNIV}$
 using $\langle \text{ball } 0 \cdot 1 \subseteq \text{interior } S \rangle$ by (*auto intro!: affine_hull_nonempty_interior*)
 have *star*: *open_segment* $0 \cdot x \subseteq \text{rel\_interior } S$ if $x \in S$ for x
proof
 fix p assume $p \in \text{open\_segment } 0 \cdot x$
 then obtain u where $x \neq 0$ and u : $0 \leq u < 1$ and p : $u *_R x = p$
 by (*auto simp: in_segment*)
 then show $p \in \text{rel\_interior } S$
 using *scale [OF that u] closure_subset frontier_def interior_subset_rel_interior*

by *fastforce*

qed

show ?thesis

using *starlike_compact_projective2_0 [OF $\langle \text{compact } S \rangle 0 \text{ star}$]* by *simp*

qed

lemma *homeomorphic_convex_lemma*:

fixes $S :: 'a::\text{euclidean\_space}$ set and $T :: 'b::\text{euclidean\_space}$ set
 assumes *convex* S *compact* S *convex* T *compact* T
 and *affeq*: *aff_dim* $S = \text{aff\_dim } T$
 shows $(S - \text{rel\_interior } S)$ *homeomorphic* $(T - \text{rel\_interior } T) \wedge$
 S *homeomorphic* T

proof (*cases* $\text{rel\_interior } S = \{\}$ $\vee \text{rel\_interior } T = \{\}$)

case *True*


```

    then show ?thesis
      by (metis Diff_empty affeq ⟨convex S⟩ ⟨convex T⟩ aff_dim_empty homeo-
morphically_empty_rel_interior_eq_empty aff_dim_empty)
next
  case False
  then obtain a b where a:  $a \in \text{rel\_interior } S$  and b:  $b \in \text{rel\_interior } T$  by auto
  have starS:  $\bigwedge x. x \in S \implies \text{open\_segment } a \ x \subseteq \text{rel\_interior } S$ 
    using rel_interior_closure_convex_segment
    a ⟨convex S⟩ closure_subset subsetCE by blast
  have starT:  $\bigwedge x. x \in T \implies \text{open\_segment } b \ x \subseteq \text{rel\_interior } T$ 
    using rel_interior_closure_convex_segment
    b ⟨convex T⟩ closure_subset subsetCE by blast
  let ?aS = (+) (-a) ' S and ?bT = (+) (-b) ' T
  have 0:  $0 \in \text{affine\_hull } ?aS \ 0 \in \text{affine\_hull } ?bT$ 
    by (metis a b subsetD hull_inc image_eqI left_minus rel_interior_subset)+
  have subs:  $\text{subspace } (\text{span } ?aS) \ \text{subspace } (\text{span } ?bT)$ 
    by (rule subspace_span)+
  moreover
  have dim (span ((+) (- a) ' S)) = dim (span ((+) (- b) ' T))
    by (metis 0 aff_dim_translation_eq aff_dim_zero affeq dim_span nat_int)
  ultimately obtain f g where linear f linear g
    and fim:  $f ' \text{span } ?aS = \text{span } ?bT$ 
    and gim:  $g ' \text{span } ?bT = \text{span } ?aS$ 
    and fno:  $\bigwedge x. x \in \text{span } ?aS \implies \text{norm}(f \ x) = \text{norm } x$ 
    and gno:  $\bigwedge x. x \in \text{span } ?bT \implies \text{norm}(g \ x) = \text{norm } x$ 
    and gf:  $\bigwedge x. x \in \text{span } ?aS \implies g(f \ x) = x$ 
    and fg:  $\bigwedge x. x \in \text{span } ?bT \implies f(g \ x) = x$ 
    by (rule isometries_subspaces) blast
  have [simp]: continuous_on A f for A
    using ⟨linear f⟩ linear_conv_bounded_linear linear_continuous_on by blast
  have [simp]: continuous_on B g for B
    using ⟨linear g⟩ linear_conv_bounded_linear linear_continuous_on by blast
  have eqspanS:  $\text{affine\_hull } ?aS = \text{span } ?aS$ 
    by (metis a affine_hull_span_0 subsetD hull_inc image_eqI left_minus rel_interior_subset)
  have eqspanT:  $\text{affine\_hull } ?bT = \text{span } ?bT$ 
    by (metis b affine_hull_span_0 subsetD hull_inc image_eqI left_minus rel_interior_subset)
  have S homeomorphic cball a 1  $\cap$  affine hull S
    by (rule starlike_compact_projective2 [OF ⟨compact S⟩ a starS])
  also have ... homeomorphic (+) (-a) ' (cball a 1  $\cap$  affine hull S)
    by (metis homeomorphic_translation)
  also have ... = cball 0 1  $\cap$  (+) (-a) ' (affine hull S)
    by (auto simp: dist_norm)
  also have ... = cball 0 1  $\cap$  span ?aS
    using eqspanS affine_hull_translation by blast
  also have ... homeomorphic cball 0 1  $\cap$  span ?bT
  proof (rule homeomorphicI)
    show fim1:  $f ' (\text{cball } 0 \ 1 \cap \text{span } ?aS) = \text{cball } 0 \ 1 \cap \text{span } ?bT$ 
  proof
    show  $f ' (\text{cball } 0 \ 1 \cap \text{span } ?aS) \subseteq \text{cball } 0 \ 1 \cap \text{span } ?bT$ 

```

```

    using fim fno by auto
    show cball 0 1  $\cap$  span ?bT  $\subseteq$  f ' (cball 0 1  $\cap$  span ?aS)
    by clarify (metis IntI fg gim gno image_eqI mem_cball_0)
  qed
  show g ' (cball 0 1  $\cap$  span ?bT) = cball 0 1  $\cap$  span ?aS
  proof
    show g ' (cball 0 1  $\cap$  span ?bT)  $\subseteq$  cball 0 1  $\cap$  span ?aS
    using gim gno by auto
    show cball 0 1  $\cap$  span ?aS  $\subseteq$  g ' (cball 0 1  $\cap$  span ?bT)
    by clarify (metis IntI fim1 gf image_eqI)
  qed
  qed (auto simp: fg gf)
  also have ... = cball 0 1  $\cap$  (+) (-b) ' (affine hull T)
    using eqspanT affine_hull_translation by blast
  also have ... = (+) (-b) ' (cball b 1  $\cap$  affine hull T)
    by (auto simp: dist_norm)
  also have ... homeomorphic (cball b 1  $\cap$  affine hull T)
    by (metis homeomorphic_translation homeomorphic_sym)
  also have ... homeomorphic T
    by (metis starlike_compact_projective2 [OF <compact T> b starT] homeomor-
      phic_sym)
  finally have 1: S homeomorphic T .

  have S - rel_interior S homeomorphic sphere a 1  $\cap$  affine hull S
    by (rule starlike_compact_projective1 [OF <compact S> a starS])
  also have ... homeomorphic (+) (-a) ' (sphere a 1  $\cap$  affine hull S)
    by (metis homeomorphic_translation)
  also have ... = sphere 0 1  $\cap$  (+) (-a) ' (affine hull S)
    by (auto simp: dist_norm)
  also have ... = sphere 0 1  $\cap$  span ?aS
    using eqspanS affine_hull_translation by blast
  also have ... homeomorphic sphere 0 1  $\cap$  span ?bT
  proof (rule homeomorphicI)
    show fim1: f ' (sphere 0 1  $\cap$  span ?aS) = sphere 0 1  $\cap$  span ?bT
    proof
      show f ' (sphere 0 1  $\cap$  span ?aS)  $\subseteq$  sphere 0 1  $\cap$  span ?bT
      using fim fno by auto
      show sphere 0 1  $\cap$  span ?bT  $\subseteq$  f ' (sphere 0 1  $\cap$  span ?aS)
      by clarify (metis IntI fg gim gno image_eqI mem_sphere_0)
    qed
    show g ' (sphere 0 1  $\cap$  span ?bT) = sphere 0 1  $\cap$  span ?aS
    proof
      show g ' (sphere 0 1  $\cap$  span ?bT)  $\subseteq$  sphere 0 1  $\cap$  span ?aS
      using gim gno by auto
      show sphere 0 1  $\cap$  span ?aS  $\subseteq$  g ' (sphere 0 1  $\cap$  span ?bT)
      by clarify (metis IntI fim1 gf image_eqI)
    qed
  qed (auto simp: fg gf)
  also have ... = sphere 0 1  $\cap$  (+) (-b) ' (affine hull T)

```

```

    using eqspanT affine_hull_translation by blast
  also have ... = (+) (-b) ' (sphere b 1  $\cap$  affine hull T)
    by (auto simp: dist_norm)
  also have ... homeomorphic (sphere b 1  $\cap$  affine hull T)
    by (metis homeomorphic_translation homeomorphic_sym)
  also have ... homeomorphic T - rel_interior T
    by (metis starlike_compact_projective1 [OF <compact T> b starT] homeomor-
        phic_sym)
  finally have 2: S - rel_interior S homeomorphic T - rel_interior T .
  show ?thesis
    using 1 2 by blast
qed

```

```

lemma homeomorphic_convex_compact_sets:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes convex S compact S convex T compact T
    and affeq: aff_dim S = aff_dim T
  shows S homeomorphic T
using homeomorphic_convex_lemma [OF assms] assms
by (auto simp: rel_frontier_def)

```

```

lemma homeomorphic_rel_frontiers_convex_bounded_sets:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes convex S bounded S convex T bounded T
    and affeq: aff_dim S = aff_dim T
  shows rel_frontier S homeomorphic rel_frontier T
using assms homeomorphic_convex_lemma [of closure S closure T]
by (simp add: rel_frontier_def convex_rel_interior_closure)

```

6.19.2 Homeomorphisms between punctured spheres and affine sets

Including the famous stereoscopic projection of the 3-D sphere to the complex plane

The special case with centre 0 and radius 1

```

lemma homeomorphic_punctured_affine_sphere_affine_01:
  assumes b  $\in$  sphere 0 1 affine T 0  $\in$  T b  $\in$  T affine p
    and affT: aff_dim T = aff_dim p + 1
  shows (sphere 0 1  $\cap$  T) - {b} homeomorphic p
proof -
  have [simp]: norm b = 1 b·b = 1
    using assms by (auto simp: norm_eq_1)
  have [simp]: T  $\cap$  {v. b·v = 0}  $\neq$  {}
    using <0  $\in$  T> by auto
  have [simp]:  $\neg$  T  $\subseteq$  {v. b·v = 0}
    using <norm b = 1> <b  $\in$  T> by auto
  define f where f  $\equiv$   $\lambda x. 2 *_R b + (2 / (1 - b \cdot x)) *_R (x - b)$ 
  define g where g  $\equiv$   $\lambda y. b + (4 / (norm y ^ 2 + 4)) *_R (y - 2 *_R b)$ 

```

```

have fg[simp]:  $\bigwedge x. \llbracket x \in T; b \cdot x = 0 \rrbracket \implies f(g\ x) = x$ 
  unfolding f_def g_def by (simp add: algebra_simps field_split_simps add_nonneg_eq_0_iff)
have no:  $(\text{norm } (f\ x))^2 = 4 * (1 + b \cdot x) / (1 - b \cdot x)$ 
  if norm x = 1 and b · x ≠ 1 for x
  using that sum_sqs_eq [of 1 b · x]
  apply (simp flip: dot_square_norm add: norm_eq_1 nonzero_eq_divide_eq)
  apply (simp add: f_def vector_add_divide_simps inner_simps)
  apply (auto simp add: field_split_simps inner_commute)
done
have [simp]:  $\bigwedge u::\text{real}. 8 + u * (u * 8) = u * 16 \longleftrightarrow u = 1$ 
  by algebra
have gf[simp]:  $\bigwedge x. \llbracket \text{norm } x = 1; b \cdot x \neq 1 \rrbracket \implies g(f\ x) = x$ 
  unfolding g_def no by (auto simp: f_def field_split_simps)
have g1:  $\text{norm } (g\ x) = 1$  if  $x \in T$  and  $b \cdot x = 0$  for x
  using that
  apply (simp only: g_def)
  apply (rule power2_eq_imp_eq)
  apply (simp_all add: dot_square_norm [symmetric] divide_simps vector_add_divide_simps)
  apply (simp add: algebra_simps inner_commute)
done
have ne1:  $b \cdot g\ x \neq 1$  if  $x \in T$  and  $b \cdot x = 0$  for x
  using that unfolding g_def
  apply (simp_all add: dot_square_norm [symmetric] divide_simps vector_add_divide_simps
    add_nonneg_eq_0_iff)
  apply (auto simp: algebra_simps)
done
have subspace T
  by (simp add: assms subspace_affine)
have gT:  $\bigwedge x. \llbracket x \in T; b \cdot x = 0 \rrbracket \implies g\ x \in T$ 
  unfolding g_def
  by (blast intro: ⟨subspace T⟩ ⟨b ∈ T⟩ subspace_add subspace_mul subspace_diff)
have f' :  $\{x. \text{norm } x = 1 \wedge b \cdot x \neq 1\} \subseteq \{x. b \cdot x = 0\}$ 
  unfolding f_def using ⟨norm b = 1⟩ norm_eq_1
  by (force simp: field_simps inner_add_right inner_diff_right)
moreover have f' :  $T \subseteq T$ 
  unfolding f_def using assms ⟨subspace T⟩
  by (auto simp add: inner_add_right inner_diff_right mem_affine_3_minus
    subspace_mul)
moreover have  $\{x. b \cdot x = 0\} \cap T \subseteq f' (\{x. \text{norm } x = 1 \wedge b \cdot x \neq 1\} \cap T)$ 
  by clarify (metis (mono_tags) IntI ne1 fg gT g1 imageI mem_Collect_eq)
ultimately have  $\text{imf}: f' (\{x. \text{norm } x = 1 \wedge b \cdot x \neq 1\} \cap T) = \{x. b \cdot x = 0\} \cap T$ 
  by blast
have no4:  $\bigwedge y. b \cdot y = 0 \implies \text{norm } ((y \cdot y + 4) *_R b + 4 *_R (y - 2 *_R b)) = y \cdot y + 4$ 
  apply (rule power2_eq_imp_eq)
  apply (simp_all flip: dot_square_norm)
  apply (auto simp: power2_eq_square algebra_simps inner_commute)
done

```

```

have [simp]:  $\bigwedge x. \llbracket \text{norm } x = 1; b \cdot x \neq 1 \rrbracket \implies b \cdot f x = 0$ 
  by (simp add: f_def algebra_simps field_split_simps)
have [simp]:  $\bigwedge x. \llbracket x \in T; \text{norm } x = 1; b \cdot x \neq 1 \rrbracket \implies f x \in T$ 
  unfolding f_def
  by (blast intro:  $\langle \text{subspace } T \rangle \langle b \in T \rangle \text{subspace\_add subspace\_mul subspace\_diff}$ )
have g '  $\{x. b \cdot x = 0\} \subseteq \{x. \text{norm } x = 1 \wedge b \cdot x \neq 1\}$ 
  unfolding g_def
  apply (clarsimp simp: no4_vector_add_divide_simps divide_simps add_nonneg_eq_0_iff
dot_square_norm [symmetric])
  apply (auto simp: algebra_simps)
  done
moreover have g '  $T \subseteq T$ 
  unfolding g_def
  by (blast intro:  $\langle \text{subspace } T \rangle \langle b \in T \rangle \text{subspace\_add subspace\_mul subspace\_diff}$ )
moreover have  $\{x. \text{norm } x = 1 \wedge b \cdot x \neq 1\} \cap T \subseteq g ' (\{x. b \cdot x = 0\} \cap T)$ 
  by clarify (metis (mono_tags, lifting) IntI gf_image_iff imf mem_Collect_eq)
ultimately have img:  $g ' (\{x. b \cdot x = 0\} \cap T) = \{x. \text{norm } x = 1 \wedge b \cdot x \neq 1\} \cap T$ 
  by blast
have aff: affine  $(\{x. b \cdot x = 0\} \cap T)$ 
  by (blast intro: affine_hyperplane assms)
have contf: continuous_on  $(\{x. \text{norm } x = 1 \wedge b \cdot x \neq 1\} \cap T)$  f
  unfolding f_def by (rule continuous_intros | force)+
have contg: continuous_on  $(\{x. b \cdot x = 0\} \cap T)$  g
  unfolding g_def by (rule continuous_intros | force simp: add_nonneg_eq_0_iff)+
have  $(\text{sphere } 0 \ 1 \cap T) - \{b\} = \{x. \text{norm } x = 1 \wedge (b \cdot x \neq 1)\} \cap T$ 
  using  $\langle \text{norm } b = 1 \rangle$  by (auto simp: norm_eq_1) (metis vector_eq  $\langle b \cdot b = 1 \rangle$ )
also have ... homeomorphic  $\{x. b \cdot x = 0\} \cap T$ 
  by (rule homeomorphicI [OF imf img contf contg]) auto
also have ... homeomorphic p
proof (rule homeomorphic_affine_sets [OF aff  $\langle \text{affine } p \rangle$ ])
  show aff_dim  $(\{x. b \cdot x = 0\} \cap T) = \text{aff\_dim } p$ 
  by (simp add: Int_commute aff_dim_affine_Int_hyperplane [OF  $\langle \text{affine } T \rangle$ 
affT])
qed
finally show ?thesis .
qed

```

```

theorem homeomorphic_punctured_affine_sphere_affine:
  fixes a :: 'a :: euclidean_space
  assumes  $0 < r$   $b \in \text{sphere } a \ r$   $\text{affine } T$   $a \in T$   $b \in T$   $\text{affine } p$ 
    and aff:  $\text{aff\_dim } T = \text{aff\_dim } p + 1$ 
  shows  $(\text{sphere } a \ r \cap T) - \{b\}$  homeomorphic p
proof -
  have  $a \neq b$  using assms by auto
  then have inj:  $\text{inj } (\lambda x::'a. x /_R \text{norm } (a - b))$ 
    by (simp add: inj_on_def)
  have  $((\text{sphere } a \ r \cap T) - \{b\})$  homeomorphic
     $(+) (-a) ' ((\text{sphere } a \ r \cap T) - \{b\})$ 

```

```

    by (rule homeomorphic_translation)
  also have ... homeomorphic (*R) (inverse r) ‘ (+) (− a) ‘ (sphere a r ∩ T − {b})
    by (metis ‹0 < r› homeomorphic_scaling inverse_inverse_eq inverse_zero less_irrefl)
  also have ... = sphere 0 1 ∩ ((*R) (inverse r) ‘ (+) (− a) ‘ T) − {(b − a) /R r}
    using assms by (auto simp: dist_norm norm_minus_commute divide_simps)
  also have ... homeomorphic p
    using assms affine_translation [symmetric, of − a] aff_dim_translation_eq [of − a]
    by (intro homeomorphic_punctured_affine_sphere_affine_01) (auto simp: dist_norm norm_minus_commute affine_scaling inj)
  finally show ?thesis .
qed

```

corollary *homeomorphic_punctured_sphere_affine:*

```

  fixes a :: 'a :: euclidean_space
  assumes 0 < r and b: b ∈ sphere a r
    and affine T and affS: aff_dim T + 1 = DIM('a)
  shows (sphere a r − {b}) homeomorphic T
    using homeomorphic_punctured_affine_sphere_affine [of r b a UNIV T] assms
  by auto

```

corollary *homeomorphic_punctured_sphere_hyperplane:*

```

  fixes a :: 'a :: euclidean_space
  assumes 0 < r and b: b ∈ sphere a r
    and c ≠ 0
  shows (sphere a r − {b}) homeomorphic {x::'a. c • x = d}
    using assms
  by (intro homeomorphic_punctured_sphere_affine) (auto simp: affine_hyperplane of_nat_diff)

```

proposition *homeomorphic_punctured_sphere_affine_gen:*

```

  fixes a :: 'a :: euclidean_space
  assumes convex S bounded S and a: a ∈ rel_frontier S
    and affine T and affS: aff_dim S = aff_dim T + 1
  shows rel_frontier S − {a} homeomorphic T
proof −
  obtain U :: 'a set where affine U convex U and affdS: aff_dim U = aff_dim S
    using choose_affine_subset [OF affine_UNIV aff_dim_geq]
    by (meson aff_dim_affine_hull affine_affine_hull affine_imp_convex)
  have S ≠ {} using assms by auto
  then obtain z where z ∈ U
    by (metis aff_dim_negative_iff equals0I affdS)
  then have bnc: ball z 1 ∩ U ≠ {} by force
  then have [simp]: aff_dim(ball z 1 ∩ U) = aff_dim U
    using aff_dim_convex_Int_open [OF ‹convex U› open_ball]
    by (fastforce simp add: Int_commute)

```

```

have rel_frontier S homeomorphic rel_frontier (ball z 1  $\cap$  U)
  by (rule homeomorphic_rel_frontiers_convex_bounded_sets)
  (auto simp:  $\langle$ affine U $\rangle$  affine_imp_convex convex_Int affdS assms)
also have ... = sphere z 1  $\cap$  U
  using convex_affine_rel_frontier_Int [of ball z 1 U]
  by (simp add:  $\langle$ affine U $\rangle$  bne)
finally have rel_frontier S homeomorphic sphere z 1  $\cap$  U .
then obtain h k where him: h ' $\langle$  rel_frontier S = sphere z 1  $\cap$  U
  and kim: k ' $\langle$  (sphere z 1  $\cap$  U) = rel_frontier S
  and hcon: continuous_on (rel_frontier S) h
  and kcon: continuous_on (sphere z 1  $\cap$  U) k
  and kh:  $\bigwedge x. x \in \text{rel\_frontier } S \implies k(h(x)) = x$ 
  and hk:  $\bigwedge y. y \in \text{sphere } z \ 1 \cap U \implies h(k(y)) = y$ 
  unfolding homeomorphic_def homeomorphism_def by auto
have rel_frontier S - {a} homeomorphic (sphere z 1  $\cap$  U) - {h a}
proof (rule homeomorphicI)
  show h: h ' $\langle$  (rel_frontier S - {a}) = sphere z 1  $\cap$  U - {h a}
    using him a kh by auto metis
  show k: k ' $\langle$  (sphere z 1  $\cap$  U - {h a}) = rel_frontier S - {a}
    by (force simp: h [symmetric] image_comp o_def kh)
qed (auto intro: continuous_on_subset hcon kcon simp: kh hk)
also have ... homeomorphic T
  by (rule homeomorphic_punctured_affine_sphere_affine)
  (use a him in  $\langle$ auto simp: affS affdS  $\langle$ affine T $\rangle$   $\langle$ affine U $\rangle$   $\langle$ z  $\in$  U $\rangle$ )
finally show ?thesis .
qed

```

When dealing with AR, ANR and ANR later, it's useful to know that every set is homeomorphic to a closed subset of a convex set, and if the set is locally compact we can take the convex set to be the universe.

```

proposition homeomorphic_closedin_convex:
  fixes S :: 'm::euclidean_space set
  assumes aff_dim S < DIM('n)
  obtains U and T :: 'n::euclidean_space set
    where convex U U  $\neq$  {} closedin (top_of_set U) T
      S homeomorphic T
proof (cases S = {})
  case True then show ?thesis
    by (rule_tac U=UNIV and T={} in that) auto
next
  case False
  then obtain a where a  $\in$  S by auto
  obtain i::'n where i: i  $\in$  Basis i  $\neq$  0
    using SOME_Basis Basis_zero by force
  have 0  $\in$  affine hull ((+) (- a) ' $\langle$  S)
    by (simp add:  $\langle$ a  $\in$  S $\rangle$  hull_inc)
  then have dim ((+) (- a) ' $\langle$  S) = aff_dim ((+) (- a) ' $\langle$  S)
    by (simp add: aff_dim_zero)
  also have ... < DIM('n)

```

```

    by (simp add: aff_dim_translation_eq_subtract assms cong: image_cong_simp)
  finally have dd: dim ((+) (- a) ' S) < DIM('n)
    by linarith
  have span: span {x. i • x = 0} = {x. i • x = 0}
    using span_eq_iff [symmetric, of {x. i • x = 0}] subspace_hyperplane [of i]
by simp
  have dim ((+) (- a) ' S) ≤ dim {x. i • x = 0}
    using dd by (simp add: dim_hyperplane [OF ⟨i ≠ 0⟩])
  then obtain T where subspace T and Tsub: T ⊆ {x. i • x = 0}
    and dimT: dim T = dim ((+) (- a) ' S)
    by (rule choose_subspace_of_subspace) (simp add: span)
  have subspace (span ((+) (- a) ' S))
    using subspace_span by blast
  then obtain h k where linear h linear k
    and heq: h ' span ((+) (- a) ' S) = T
    and keq: k ' T = span ((+) (- a) ' S)
    and hinu [simp]:  $\bigwedge x. x \in \text{span } ((+) (- a) ' S) \implies k(h\ x) = x$ 
    and kinu [simp]:  $\bigwedge x. x \in T \implies h(k\ x) = x$ 
    by (auto simp: dimT intro: isometries_subspaces [OF _ ⟨subspace T⟩] dimT)
  have hcont: continuous_on A h and kcont: continuous_on B k for A B
    using ⟨linear h⟩ ⟨linear k⟩ linear_continuous_on linear_conv_bounded_linear
by blast+
  have ihhh[simp]:  $\bigwedge x. x \in S \implies i \cdot h\ (x - a) = 0$ 
    using Tsub [THEN subsetD] heq span_superset by fastforce
  have sphere 0 1 - {i} homeomorphic {x. i • x = 0}
  proof (rule homeomorphic_punctured_sphere_affine)
    show affine {x. i • x = 0}
      by (auto simp: affine_hyperplane)
    show aff_dim {x. i • x = 0} + 1 = int DIM('n)
      using i by clarsimp (metis DIM_positive Suc_pred add.commute of_nat_Suc)
  qed (use i in auto)
  then obtain f g where fg: homeomorphism (sphere 0 1 - {i}) {x. i • x = 0} f
g
    by (force simp: homeomorphic_def)
  show ?thesis
  proof
    have h ' (+) (- a) ' S ⊆ T
      using heq span_superset span_linear_image by blast
    then have g ' h ' (+) (- a) ' S ⊆ g ' {x. i • x = 0}
      using Tsub by (simp add: image_mono)
    also have ... ⊆ sphere 0 1 - {i}
      by (simp add: fg [unfolded homeomorphism_def])
    finally have gh_sub_sph: (g ∘ h) ' (+) (- a) ' S ⊆ sphere 0 1 - {i}
      by (metis image_comp)
    then have gh_sub_cb: (g ∘ h) ' (+) (- a) ' S ⊆ cball 0 1
      by (metis Diff_subset order_trans sphere_cball)
    have [simp]:  $\bigwedge u. u \in S \implies \text{norm } (g\ (h\ (u - a))) = 1$ 
      using gh_sub_sph [THEN subsetD] by (auto simp: o_def)
    show convex (ball 0 1 ∪ (g ∘ h) ' (+) (- a) ' S)

```



```

    by (meson ball_subset_cball convex_intermediate_ball gh_sub_cb sup.bounded_iff
sup.cobounded1)
    show closedin (top_of_set (ball 0 1  $\cup$  ( $g \circ h$ ) ' (+) (- a) ' S)) (( $g \circ h$ ) ' (+)
(- a) ' S)
      unfolding closedin_closed
      by (rule_tac x=sphere 0 1 in exI) auto
    have ghcont: continuous_on (( $\lambda x. x - a$ ) ' S) ( $\lambda x. g (h x)$ )
    by (rule continuous_on_compose2 [OF homeomorphism_cont2 [OF fg] hcont],
force)
    have kfcont: continuous_on (( $\lambda x. g (h (x - a))$ ) ' S) ( $\lambda x. k (f x)$ )
    proof (rule continuous_on_compose2 [OF kcont])
      show continuous_on (( $\lambda x. g (h (x - a))$ ) ' S) f
        using homeomorphism_cont1 [OF fg] gh_sub_sph by (fastforce intro:
continuous_on_subset)
      qed auto
    have S homeomorphic (+) (- a) ' S
      by (fact homeomorphic_translation)
    also have ... homeomorphic ( $g \circ h$ ) ' (+) (- a) ' S
    apply (simp add: homeomorphic_def homeomorphism_def cong: image_cong_simp)
    apply (rule_tac x= $g \circ h$  in exI)
    apply (rule_tac x= $k \circ f$  in exI)
    apply (auto simp: ghcont kfcont span_base homeomorphism_apply2 [OF fg]
image_comp cong: image_cong_simp)
    done
    finally show S homeomorphic ( $g \circ h$ ) ' (+) (- a) ' S .
  qed auto
qed

```

6.19.3 Locally compact sets in an open set

Locally compact sets are closed in an open set and are homeomorphic to an absolutely closed set if we have one more dimension to play with.

lemma *locally_compact_open_Int_closure*:

fixes $S :: 'a :: \text{metric\_space set}$

assumes *locally_compact S*

obtains T **where** *open T S = T $\cap$ closure S*

proof –

have $\forall x \in S. \exists T v u. u = S \cap T \wedge x \in u \wedge u \subseteq v \wedge v \subseteq S \wedge \text{open } T \wedge \text{compact } v$

by (*metis* *assms* *locally_compact* *openin_open*)

then obtain $t v$ **where**

$tv: \bigwedge x. x \in S$

$\implies v x \subseteq S \wedge \text{open } (t x) \wedge \text{compact } (v x) \wedge (\exists u. x \in u \wedge u \subseteq v x \wedge u$

$= S \cap t x)$

by *metis*

then have $o: \text{open } (\bigcup (t ' S))$

by *blast*

have $S = \bigcup (v ' S)$

using tv *by* *blast*

```

also have ... =  $\bigcup (t \text{ ' } S) \cap \text{closure } S$ 
proof
  show  $\bigcup (v \text{ ' } S) \subseteq \bigcup (t \text{ ' } S) \cap \text{closure } S$ 
    by clarify (meson IntD2 IntI UN_I closure_subset subsetD tv)
  have  $t \text{ } x \cap \text{closure } S \subseteq v \text{ } x$  if  $x \in S$  for  $x$ 
  proof -
    have  $t \text{ } x \cap \text{closure } S \subseteq \text{closure } (t \text{ } x \cap S)$ 
      by (simp add: open_Int_closure_subset that tv)
    also have ...  $\subseteq v \text{ } x$ 
      by (metis Int_commute closure_minimal compact_imp_closed that tv)
    finally show ?thesis .
  qed
  then show  $\bigcup (t \text{ ' } S) \cap \text{closure } S \subseteq \bigcup (v \text{ ' } S)$ 
    by blast
  qed
  finally have  $e: S = \bigcup (t \text{ ' } S) \cap \text{closure } S$  .
  show ?thesis
    by (rule that [OF o e])
  qed

```

```

lemma locally_compact_closedin_open:
  fixes  $S :: 'a :: \text{metric\_space set}$ 
  assumes locally_compact  $S$ 
  obtains  $T$  where open  $T$  closedin (top_of_set  $T$ )  $S$ 
  by (metis locally_compact_open_Int_closure [OF assms] closed_closure closedin_closed_Int)

```

```

lemma locally_compact_homeomorphism_projection_closed:
  assumes locally_compact  $S$ 
  obtains  $T$  and  $f :: 'a \Rightarrow 'a :: \text{euclidean\_space} \times 'b :: \text{euclidean\_space}$ 
  where closed  $T$  homeomorphism  $S \text{ } T \text{ } f \text{ } \text{fst}$ 
proof (cases closed  $S$ )
  case True
  show ?thesis
  proof
    show homeomorphism  $S (S \times \{0\}) (\lambda x. (x, 0)) \text{ } \text{fst}$ 
      by (auto simp: homeomorphism_def continuous_intros)
    qed (use True closed_Times in auto)
  next
  case False
  obtain  $U$  where open  $U$  and  $US: U \cap \text{closure } S = S$ 
    by (metis locally_compact_open_Int_closure [OF assms])
  with False have  $U_{\text{comp}}: -U \neq \{\}$ 
    using closure_eq by auto
  have [simp]:  $\text{closure } (-U) = -U$ 
    by (simp add:  $\langle \text{open } U \rangle \text{ } \text{closed\_Compl}$ )
  define  $f :: 'a \Rightarrow 'a \times 'b$  where  $f \equiv \lambda x. (x, \text{One } /_R \text{ setdist } \{x\} (-U))$ 
  have continuous_on  $U (\lambda x. (x, \text{One } /_R \text{ setdist } \{x\} (-U)))$ 

```

```

proof (intro continuous_intros continuous_on_setdist)
  show  $\forall x \in U. \text{setdist } \{x\} (-U) \neq 0$ 
    by (simp add: Ucomp setdist_eq_0_sing_1)
qed
then have homU: homeomorphism U (f'U) f fst
  by (auto simp: f_def homeomorphism_def image_iff continuous_intros)
have cloS: closedin (top_of_set U) S
  by (metis US closed_closure closedin_closed_Int)
have cont: isCont (( $\lambda x. \text{setdist } \{x\} (-U)$ ) o fst) z for z :: 'a  $\times$  'b
  by (rule continuous_at_compose continuous_intros continuous_at_setdist)+
have setdist1D:  $\text{setdist } \{a\} (-U) *_R b = \text{One} \implies \text{setdist } \{a\} (-U) \neq 0$  for
a::'a and b::'b
  by force
have *:  $r *_R b = \text{One} \implies b = (1 / r) *_R \text{One}$  for r and b::'b
  by (metis One_non_0 nonzero_divide_eq_eq real_vector.scale_eq_0_iff
real_vector.scale_scale scaleR_one)
have  $\bigwedge a b::'b. \text{setdist } \{a\} (-U) *_R b = \text{One} \implies (a,b) \in (\lambda x. (x, (1 / \text{setdist } \{x\} (-U)) *_R \text{One})) 'U$ 
  by (metis (mono_tags, lifting) * ComplI image_eqI setdist1D setdist_sing_in_set)
then have f' U = ( $\lambda z. (\text{setdist } \{\text{fst } z\} (-U) *_R \text{snd } z)$ ) - ' {One}
  by (auto simp: f_def setdist_eq_0_sing_1 field_simps Ucomp)
then have clfU: closed (f' U)
  by (force intro: continuous_intros cont [unfolded o_def] continuous_closed_vimage)
have closed (f' S)
  by (metis closedin_closed_trans [OF clfU] homeomorphism_imp_closed_map
[OF homU cloS])
then show ?thesis
  by (metis US homU homeomorphism_of_subsets inf_sup_ord(1) that)
qed

lemma locally_compact_closed_Int_open:
  fixes S :: 'a :: euclidean_space set
  shows locally_compact S  $\longleftrightarrow (\exists U V. \text{closed } U \wedge \text{open } V \wedge S = U \cap V)$  (is
?lhs = ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
    by (metis closed_closure inf_commute locally_compact_open_Int_closure)
  show ?rhs  $\implies$  ?lhs
    by (meson closed_imp_locally_compact locally_compact_Int open_imp_locally_compact)
qed

lemma lowerdim_embeddings:
  assumes DIM('a) < DIM('b)
  obtains f :: 'a::euclidean_space*real  $\Rightarrow$  'b::euclidean_space
    and g :: 'b  $\Rightarrow$  'a*real
    and j :: 'b
  where linear f linear g  $\bigwedge z. g(f z) = z j \in \text{Basis} \bigwedge x. f(x,0) \cdot j = 0$ 
proof -
  let ?B = Basis :: ('a*real) set

```

```

have b01: (0,1) ∈ ?B
  by (simp add: Basis_prod_def)
have DIM('a * real) ≤ DIM('b)
  by (simp add: Suc_leI assms)
then obtain basf :: 'a*real ⇒ 'b where sbf: basf ' ?B ⊆ Basis and injbf: inj_on
basf Basis
  by (metis finite_Basis card_le_inj)
define basg :: 'b ⇒ 'a * real where
  basg ≡ λi. if i ∈ basf ' Basis then inv_into Basis basf i else (0,1)
have bgf[simp]: basg (basf i) = i if i ∈ Basis for i
  using inv_into_f_f injbf that by (force simp: basg_def)
have sbg: basg ' Basis ⊆ ?B
  by (force simp: basg_def injbf b01)
define f :: 'a*real ⇒ 'b where f ≡ λu. ∑ j∈Basis. (u • basg j) *R j
define g :: 'b ⇒ 'a*real where g ≡ λz. (∑ i∈Basis. (z • basf i) *R i)
show ?thesis
proof
  show linear f
    unfolding f_def
    by (intro linear_compose_sum linearI ballI) (auto simp: algebra_simps)
  show linear g
    unfolding g_def
    by (intro linear_compose_sum linearI ballI) (auto simp: algebra_simps)
  have *: (∑ a ∈ Basis. a • basf b * (x • basg a)) = x • b if b ∈ Basis for x b
    using sbf that by auto
  show gf: g (f x) = x for x
  proof (rule euclidean_eqI)
    show ∧ b. b ∈ Basis ⇒ g (f x) • b = x • b
      using f_def g_def sbf by auto
  qed
  show basf(0,1) ∈ Basis
    using b01 sbf by auto
  then show f(x,0) • basf(0,1) = 0 for x
    unfolding f_def inner_sum_left
    using b01 inner_not_same_Basis
    by (fastforce intro: comm_monoid_add_class.sum.neutral)
qed
qed

```

proposition *locally_compact_homeomorphic_closed*:

```

fixes S :: 'a::euclidean_space set
assumes locally_compact S and dimlt: DIM('a) < DIM('b)
obtains T :: 'b::euclidean_space set where closed T S homeomorphic T
proof -
  obtain U :: ('a*real)set and h
    where closed U and homU: homeomorphism S U h fst
    using locally_compact_homeomorphism_projection_closed assms by metis
  obtain f :: 'a*real ⇒ 'b and g :: 'b ⇒ 'a*real
    where linear f linear g and gf [simp]: ∧ z. g (f z) = z

```

```

    using lowerdim_embeddings [OF dimlt] by metis
  then have inj f
    by (metis injI)
  have gfU:  $g \circ f \circ U = U$ 
    by (simp add: image_comp o_def)
  have S homeomorphic U
    using homU homeomorphic_def by blast
  also have ... homeomorphic  $f \circ U$ 
  proof (rule homeomorphicI [OF refl gfU])
    show continuous_on U f
      by (meson <inj f> <linear f> homeomorphism_cont2 linear_homeomorphism_image)
    show continuous_on  $(f \circ U) g$ 
      using <linear g> linear_continuous_on linear_conv_bounded_linear by blast
  qed (auto simp: o_def)
  finally show ?thesis
    using <closed U> <inj f> <linear f> closed_injective_linear_image that by blast
qed

```

```

lemma homeomorphic_convex_compact_lemma:
  fixes S :: 'a::euclidean_space set
  assumes convex S
    and compact S
    and cball 0 1  $\subseteq$  S
  shows S homeomorphic (cball (0::'a) 1)
proof (rule starlike_compact_projective_special[OF assms(2-3)])
  fix x u
  assume  $x \in S$  and  $0 \leq u$  and  $u < (1::real)$ 
  have open (ball  $(u *_R x) (1 - u)$ )
    by (rule open_ball)
  moreover have  $u *_R x \in \text{ball } (u *_R x) (1 - u)$ 
    unfolding centre_in_ball using <u < 1> by simp
  moreover have ball  $(u *_R x) (1 - u) \subseteq S$ 
  proof
    fix y
    assume  $y \in \text{ball } (u *_R x) (1 - u)$ 
    then have dist  $(u *_R x) y < 1 - u$ 
      unfolding mem_ball .
    with <u < 1> have inverse  $(1 - u) *_R (y - u *_R x) \in \text{cball } 0 1$ 
      by (simp add: dist_norm inverse_eq_divide norm_minus_commute)
    with assms(3) have inverse  $(1 - u) *_R (y - u *_R x) \in S$  ..
    with assms(1) have  $(1 - u) *_R ((y - u *_R x) /_R (1 - u)) + u *_R x \in S$ 
      using <x  $\in$  S> <0  $\leq$  u> <u < 1> [THEN less_imp_le] by (rule convexD_alt)
    then show  $y \in S$  using <u < 1>
      by simp
  qed
  ultimately have  $u *_R x \in \text{interior } S$  ..
  then show  $u *_R x \in S - \text{frontier } S$ 
    using frontier_def and interior_subset by auto

```

qed

proposition *homeomorphic_convex_compact_cball*:
 fixes $e :: \text{real}$
 and $S :: 'a::\text{euclidean\_space}$ set
 assumes S : *convex* S *compact* S *interior* $S \neq \{\}$ and $e > 0$
 shows S *homeomorphic* ($\text{cball } (b::'a) e$)
proof (*rule* *homeomorphic_trans*[*OF* _ *homeomorphic_balls*(2)])
 obtain a where $a \in \text{interior } S$
 using *assms* by *auto*
 then show S *homeomorphic* $\text{cball } (0::'a) 1$
 by (*metis* (*no_types*) *aff_dim_cball* S *compact_cball* *convex_cball*
homeomorphic_convex_lemma *interior_rel_interior_gen* *zero_less_one*)
qed (*use* $\langle e > 0 \rangle$ *in* *auto*)

corollary *homeomorphic_convex_compact*:
 fixes $S :: 'a::\text{euclidean\_space}$ set
 and $T :: 'a$ set
 assumes *convex* S *compact* S *interior* $S \neq \{\}$
 and *convex* T *compact* T *interior* $T \neq \{\}$
 shows S *homeomorphic* T
 using *assms*
 by (*meson* *zero_less_one* *homeomorphic_trans* *homeomorphic_convex_compact_cball*
homeomorphic_sym)

lemma *homeomorphic_closed_intervals*:
 fixes $a :: 'a::\text{euclidean\_space}$ and b and $c :: 'a::\text{euclidean\_space}$ and d
 assumes *box* a $b \neq \{\}$ and *box* c $d \neq \{\}$
 shows (*cbox* a b) *homeomorphic* (*cbox* c d)
 by (*simp* *add*: *assms* *homeomorphic_convex_compact*)

lemma *homeomorphic_closed_intervals_real*:
 fixes $a::\text{real}$ and b and $c::\text{real}$ and d
 assumes $a < b$ and $c < d$
 shows $\{a..b\}$ *homeomorphic* $\{c..d\}$
 using *assms* by (*auto* *intro*: *homeomorphic_convex_compact*)

6.19.4 Covering spaces and lifting results for them

definition *covering_space*
 $:: 'a::\text{topological\_space}$ set $\Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b::\text{topological\_space}$ set $\Rightarrow \text{bool}$
where
covering_space c p $S \equiv$
continuous_on c $p \wedge p \text{ ` } c = S \wedge$
 $(\forall x \in S. \exists T. x \in T \wedge \text{openin } (\text{top\_of\_set } S) T \wedge$
 $(\exists v. \bigcup v = c \cap p \text{ ` } T \wedge$
 $(\forall u \in v. \text{openin } (\text{top\_of\_set } c) u) \wedge$
pairwise disjoint $v \wedge$
 $(\forall u \in v. \exists q. \text{homeomorphism } u T p q)))$

lemma *covering_space_imp_continuous*: *covering_space c p S $\implies$ continuous_on c p*

by (*simp add: covering_space_def*)

lemma *covering_space_imp_surjective*: *covering_space c p S $\implies$ p ' c = S*

by (*simp add: covering_space_def*)

lemma *homeomorphism_imp_covering_space*: *homeomorphism S T f g $\implies$ covering_space S f T*

apply (*clarsimp simp add: homeomorphism_def covering_space_def*)

apply (*rule_tac x=T in exI, simp*)

apply (*rule_tac x={S} in exI, auto*)

done

lemma *covering_space_local_homeomorphism*:

assumes *covering_space c p S x $\in$ c*

obtains *T u q where x $\in$ T openin (top_of_set c) T*

p x $\in$ u openin (top_of_set S) u

homeomorphism T u p q

using *assms*

by (*clarsimp simp add: covering_space_def*) (*metis IntI UnionE vimage_eq*)

lemma *covering_space_local_homeomorphism_alt*:

assumes *p: covering_space c p S and y $\in$ S*

obtains *x T U q where p x = y*

x $\in$ T openin (top_of_set c) T

y $\in$ U openin (top_of_set S) U

homeomorphism T U p q

proof –

obtain *x where p x = y x $\in$ c*

using *assms covering_space_imp_surjective* **by** *blast*

show *?thesis*

using *that* $\langle p\ x = y \rangle$ **by** (*auto intro: covering_space_local_homeomorphism [OF p $\langle x \in c \rangle$]*)

qed

proposition *covering_space_open_map*:

fixes *S :: 'a :: metric_space set and T :: 'b :: metric_space set*

assumes *p: covering_space c p S and T: openin (top_of_set c) T*

shows *openin (top_of_set S) (p ' T)*

proof –

have *pce: p ' c = S*

and *covs:*

$\bigwedge x. x \in S \implies$

$\exists X\ VS. x \in X \wedge \text{openin } (\text{top\_of\_set } S) X \wedge$

$\bigcup VS = c \cap p^{-1} X \wedge$

$(\forall u \in VS. \text{openin } (\text{top\_of\_set } c) u) \wedge$

```

pairwise disjoint VS  $\wedge$ 
( $\forall u \in VS. \exists q. \text{homeomorphism } u \ X \ p \ q$ )
using p by (auto simp: covering_space_def)
have  $T \subseteq c$  by (metis openin_euclidean_subtopology_iff T)
have  $\exists X. \text{openin } (\text{top\_of\_set } S) \ X \wedge y \in X \wedge X \subseteq p^{-1} T$ 
  if  $y \in p^{-1} T$  for y
proof -
  have  $y \in S$  using  $\langle T \subseteq c \rangle$  pce that by blast
  obtain U VS where  $y \in U$  and  $U: \text{openin } (\text{top\_of\_set } S) \ U$ 
    and  $VS: \bigcup VS = c \cap p^{-1} U$ 
    and  $\text{openVS}: \forall V \in VS. \text{openin } (\text{top\_of\_set } c) \ V$ 
    and  $\text{homVS}: \bigwedge V. V \in VS \implies \exists q. \text{homeomorphism } V \ U \ p \ q$ 
  using covs [OF  $\langle y \in S \rangle$ ] by auto
  obtain x where  $x \in c \ p \ x \in U \ x \in T \ p \ x = y$ 
    using T [unfolded openin_euclidean_subtopology_iff]  $\langle y \in U \rangle \langle y \in p^{-1} T \rangle$ 
  by blast
  with VS obtain V where  $x \in V \ V \in VS$  by auto
  then obtain q where  $q: \text{homeomorphism } V \ U \ p \ q$  using homVS by blast
  then have ptV:  $p^{-1} (T \cap V) = U \cap q^{-1} (T \cap V)$ 
    using VS  $\langle V \in VS \rangle$  by (auto simp: homeomorphism_def)
  have ocv:  $\text{openin } (\text{top\_of\_set } c) \ V$ 
    by (simp add:  $\langle V \in VS \rangle \text{openVS}$ )
  have  $\text{openin } (\text{top\_of\_set } (q^{-1} U)) (T \cap V)$ 
    using q unfolding homeomorphism_def
  by (metis T inf.absorb_iff2 ocv openin_imp_subset openin_subtopology_Int
subtopology_subtopology)
  then have  $\text{openin } (\text{top\_of\_set } U) (U \cap q^{-1} (T \cap V))$ 
    using continuous_on_open homeomorphism_def q by blast
  then have os:  $\text{openin } (\text{top\_of\_set } S) (U \cap q^{-1} (T \cap V))$ 
    using openin_trans [of U] by (simp add: Collect_conj_eq U)
  show ?thesis
  proof (intro exI conjI)
    show  $\text{openin } (\text{top\_of\_set } S) (p^{-1} (T \cap V))$ 
      by (simp only: ptV os)
    qed (use  $\langle p \ x = y \rangle \langle x \in V \rangle \langle x \in T \rangle$  in auto)
  qed
  with openin_subopen show ?thesis by blast
qed

```

```

lemma covering_space_lift_unique_gen:
  fixes f :: 'a::topological_space  $\Rightarrow$  'b::topological_space
  fixes g1 :: 'a  $\Rightarrow$  'c::real_normed_vector
  assumes cov: covering_space c p S
    and eq:  $g1 \ a = g2 \ a$ 
    and f: continuous_on T f  $f^{-1} T \subseteq S$ 
    and g1: continuous_on T g1  $g1^{-1} T \subseteq c$ 
    and fg1:  $\bigwedge x. x \in T \implies f \ x = p(g1 \ x)$ 
    and g2: continuous_on T g2  $g2^{-1} T \subseteq c$ 
    and fg2:  $\bigwedge x. x \in T \implies f \ x = p(g2 \ x)$ 

```



```

    and  $u\_compt$ :  $U \in components\ T$  and  $a \in U\ x \in U$ 
    shows  $g1\ x = g2\ x$ 
  proof -
    have  $U \subseteq T$  by (rule in_components_subset [OF  $u\_compt$ ])
    define  $G12$  where  $G12 \equiv \{x \in U. g1\ x - g2\ x = 0\}$ 
    have connected  $U$  by (rule in_components_connected [OF  $u\_compt$ ])
    have  $contu$ : continuous_on  $U\ g1$  continuous_on  $U\ g2$ 
      using  $\langle U \subseteq T \rangle$  continuous_on_subset  $g1\ g2$  by blast+
    have  $o12$ : openin (top_of_set  $U$ )  $G12$ 
    unfolding  $G12\_def$ 
    proof (subst openin_subopen, clarify)
      fix  $z$ 
      assume  $z$ :  $z \in U\ g1\ z - g2\ z = 0$ 
      obtain  $v\ w\ q$  where  $g1\ z \in v$  and  $ocv$ : openin (top_of_set  $c$ )  $v$ 
        and  $p$  ( $g1\ z$ )  $\in w$  and  $osw$ : openin (top_of_set  $S$ )  $w$ 
        and  $hom$ : homeomorphism  $v\ w\ p\ q$ 
      proof (rule covering_space_local_homeomorphism [OF  $cov$ ])
        show  $g1\ z \in c$ 
          using  $\langle U \subseteq T \rangle\ \langle z \in U \rangle\ g1(2)$  by blast
      qed auto
      have  $g2\ z \in v$  using  $\langle g1\ z \in v \rangle\ z$  by auto
      have  $gg$ :  $U \cap g - 'v = U \cap g - '(v \cap g - 'U)$  for  $g$ 
        by auto
      have openin (top_of_set ( $g1 - 'U$ )) ( $v \cap g1 - 'U$ )
        using  $ocv\ \langle U \subseteq T \rangle\ g1$  by (fastforce simp add: openin_open)
      then have 1: openin (top_of_set  $U$ ) ( $U \cap g1 - 'v$ )
        unfolding  $gg$  by (blast intro:  $contu$  continuous_on_open [THEN iffD1,
rule_format])
      have openin (top_of_set ( $g2 - 'U$ )) ( $v \cap g2 - 'U$ )
        using  $ocv\ \langle U \subseteq T \rangle\ g2$  by (fastforce simp add: openin_open)
      then have 2: openin (top_of_set  $U$ ) ( $U \cap g2 - 'v$ )
        unfolding  $gg$  by (blast intro:  $contu$  continuous_on_open [THEN iffD1,
rule_format])
      let  $?T = (U \cap g1 - 'v) \cap (U \cap g2 - 'v)$ 
      show  $\exists T. openin (top_of_set U)\ T \wedge z \in T \wedge T \subseteq \{z \in U. g1\ z - g2\ z = 0\}$ 
    proof (intro exI conjI)
      show openin (top_of_set  $U$ )  $?T$ 
        using 1 2 by blast
      show  $z \in ?T$ 
        using  $z$  by (simp add:  $\langle g1\ z \in v \rangle\ \langle g2\ z \in v \rangle$ )
      show  $?T \subseteq \{z \in U. g1\ z - g2\ z = 0\}$ 
        using  $hom$ 
        by (clarsimp simp: homeomorphism_def) (metis  $\langle U \subseteq T \rangle\ fg1\ fg2\ subsetD$ )
    qed
  qed
  have  $c12$ : closedin (top_of_set  $U$ )  $G12$ 
    unfolding  $G12\_def$ 
    by (intro continuous_intros continuous_closedin_preimage_constant  $contu$ )

```

```

have G12 = {} ∨ G12 = U
  by (intro connected_clopen [THEN iffD1, rule_format] ⟨connected U⟩ conjI
o12 c12)
with eq ⟨a ∈ U⟩ have ∧x. x ∈ U ⇒ g1 x - g2 x = 0 by (auto simp: G12_def)
then show ?thesis
  using ⟨x ∈ U⟩ by force
qed

```

proposition *covering_space_lift_unique*:

```

fixes f :: 'a::topological_space ⇒ 'b::topological_space
fixes g1 :: 'a ⇒ 'c::real_normed_vector
assumes covering_space c p S
      g1 a = g2 a
      continuous_on T f f ' T ⊆ S
      continuous_on T g1 g1 ' T ⊆ c ∧x. x ∈ T ⇒ f x = p(g1 x)
      continuous_on T g2 g2 ' T ⊆ c ∧x. x ∈ T ⇒ f x = p(g2 x)
      connected T a ∈ T x ∈ T
shows g1 x = g2 x
  using covering_space_lift_unique_gen [of c p S] in_components_self assms
ex_in_conv
  by blast

```

lemma *covering_space_locally*:

```

fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
assumes loc: locally φ C and cov: covering_space C p S
      and pim: ∧T. [T ⊆ C; φ T] ⇒ ψ(p ' T)
shows locally ψ S
proof -
  have locally ψ (p ' C)
  proof (rule locally_open_map_image [OF loc])
    show continuous_on C p
      using cov covering_space_imp_continuous by blast
    show ∧T. openin (top_of_set C) T ⇒ openin (top_of_set (p ' C)) (p ' T)
      using cov covering_space_imp_surjective covering_space_open_map by blast
  qed (simp add: pim)
  then show ?thesis
    using covering_space_imp_surjective [OF cov] by metis
qed

```

proposition *covering_space_locally_eq*:

```

fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
assumes cov: covering_space C p S
      and pim: ∧T. [T ⊆ C; φ T] ⇒ ψ(p ' T)
      and qim: ∧q U. [U ⊆ S; continuous_on U q; ψ U] ⇒ φ(q ' U)
shows locally ψ S ⇔ locally φ C
      (is ?lhs = ?rhs)
proof
  assume L: ?lhs

```

```

show ?rhs
proof (rule locallyI)
  fix V x
  assume V: openin (top_of_set C) V and x ∈ V
  have p x ∈ p ' C
    by (metis IntE V ⟨x ∈ V⟩ imageI openin_open)
  then obtain T V where p x ∈ T
    and opeT: openin (top_of_set S) T
    and veg:  $\bigcup V = C \cap p - ' T$ 
    and ope:  $\forall U \in V. \text{openin } (top\_of\_set C) U$ 
    and hom:  $\forall U \in V. \exists q. \text{homeomorphism } U T p q$ 
  using cov unfolding covering_space_def by (blast intro: that)
  have x ∈  $\bigcup V$ 
    using V veg ⟨p x ∈ T⟩ ⟨x ∈ V⟩ openin_imp_subset by fastforce
  then obtain U where x ∈ U U ∈ V
    by blast
  then obtain q where opeU: openin (top_of_set C) U and q: homeomorphism
    U T p q
    using ope hom by blast
  with V have openin (top_of_set C) (U ∩ V)
    by blast
  then have UV: openin (top_of_set S) (p ' (U ∩ V))
    using cov covering_space_open_map by blast
  obtain W W' where opeW: openin (top_of_set S) W and ψ W' p x ∈ W
    W ⊆ W' and W'sub: W' ⊆ p ' (U ∩ V)
    using locallyE [OF L UV] ⟨x ∈ U⟩ ⟨x ∈ V⟩ by blast
  then have W ⊆ T
    by (metis Int_lower1 q homeomorphism_image1 image_Int_subset order_trans)
  show  $\exists U Z. \text{openin } (top\_of\_set C) U \wedge$ 
     $\varphi Z \wedge x \in U \wedge U \subseteq Z \wedge Z \subseteq V$ 
proof (intro exI conjI)
  have openin (top_of_set T) W
    by (meson opeW opeT openin_imp_subset openin_subset_trans ⟨W ⊆ T⟩)
  then have openin (top_of_set U) (q ' W)
    by (meson homeomorphism_imp_open_map homeomorphism_symD q)
  then show openin (top_of_set C) (q ' W)
    using opeU openin_trans by blast
  show  $\varphi (q ' W')$ 
    by (metis (mono_tags, lifting) Int_subset_iff UV W'sub ⟨ψ W'⟩ continu-
ous_on_subset dual_order.trans homeomorphism_def image_Int_subset openin_imp_subset
q qim)
  show x ∈ q ' W
    by (metis ⟨p x ∈ W⟩ ⟨x ∈ U⟩ homeomorphism_def imageI q)
  show q ' W ⊆ q ' W'
    using ⟨W ⊆ W'⟩ by blast
  have W' ⊆ p ' V
    using W'sub by blast
  then show q ' W' ⊆ V
    using W'sub homeomorphism_apply1 [OF q] by auto

```

```

      qed
    qed
  next
    assume ?rhs
    then show ?lhs
      using cov covering_space_locally_pim by blast
    qed

```

```

lemma covering_space_locally_compact_eq:
  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes covering_space C p S
  shows locally_compact S  $\longleftrightarrow$  locally_compact C
proof (rule covering_space_locally_eq [OF assms])
  show  $\bigwedge T. \llbracket T \subseteq C; \text{compact } T \rrbracket \Longrightarrow \text{compact } (p \text{ ` } T)$ 
    by (meson assms compact_continuous_image continuous_on_subset covering_space_imp_continuous)
qed (use compact_continuous_image in blast)

```

```

lemma covering_space_locally_connected_eq:
  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes covering_space C p S
  shows locally_connected S  $\longleftrightarrow$  locally_connected C
proof (rule covering_space_locally_eq [OF assms])
  show  $\bigwedge T. \llbracket T \subseteq C; \text{connected } T \rrbracket \Longrightarrow \text{connected } (p \text{ ` } T)$ 
    by (meson connected_continuous_image assms continuous_on_subset covering_space_imp_continuous)
qed (use connected_continuous_image in blast)

```

```

lemma covering_space_locally_path_connected_eq:
  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes covering_space C p S
  shows locally_path_connected S  $\longleftrightarrow$  locally_path_connected C
proof (rule covering_space_locally_eq [OF assms])
  show  $\bigwedge T. \llbracket T \subseteq C; \text{path_connected } T \rrbracket \Longrightarrow \text{path_connected } (p \text{ ` } T)$ 
    by (meson path_connected_continuous_image assms continuous_on_subset covering_space_imp_continuous)
qed (use path_connected_continuous_image in blast)

```

```

lemma covering_space_locally_compact:
  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes locally_compact C covering_space C p S
  shows locally_compact S
  using assms covering_space_locally_compact_eq by blast

```

```

lemma covering_space_locally_connected:
  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes locally_connected C covering_space C p S

```

shows *locally connected S*
using *assms covering_space_locally_connected_eq* **by** *blast*

lemma *covering_space_locally_path_connected*:
fixes $p :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$
assumes *locally_path_connected C covering_space C p S*
shows *locally_path_connected S*
using *assms covering_space_locally_path_connected_eq* **by** *blast*

proposition *covering_space_lift_homotopy*:
fixes $p :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$
and $h :: real \times 'c::real\_normed\_vector \Rightarrow 'b$
assumes *cov: covering_space C p S*
and *conth: continuous_on $(\{0..1\} \times U)$ h*
and *him: $h \restriction (\{0..1\} \times U) \subseteq S$*
and *heq: $\bigwedge y. y \in U \implies h(0, y) = p(f y)$*
and *conf: continuous_on U f and fim: $f \restriction U \subseteq C$*
obtains k **where** *continuous_on $(\{0..1\} \times U)$ k*
 $k \restriction (\{0..1\} \times U) \subseteq C$
 $\bigwedge y. y \in U \implies k(0, y) = f y$
 $\bigwedge z. z \in \{0..1\} \times U \implies h z = p(k z)$

proof –

have $\exists V k. \text{openin } (top\_of\_set U) V \wedge y \in V \wedge$
 $\text{continuous\_on } (\{0..1\} \times V) k \wedge k \restriction (\{0..1\} \times V) \subseteq C \wedge$
 $(\forall z \in V. k(0, z) = f z) \wedge (\forall z \in \{0..1\} \times V. h z = p(k z))$
if $y \in U$ **for** y

proof –

obtain UU **where** $UU: \bigwedge s. s \in S \implies s \in (UU s) \wedge \text{openin } (top\_of\_set S)$
 $(UU s) \wedge$

$(\exists \mathcal{V}. \bigcup \mathcal{V} = C \cap p - ' UU s \wedge$
 $(\forall U \in \mathcal{V}. \text{openin } (top\_of\_set C) U) \wedge$
 $\text{pairwise disjoint } \mathcal{V} \wedge$
 $(\forall U \in \mathcal{V}. \exists q. \text{homeomorphism } U (UU s) p q))$

using *cov unfolding covering_space_def* **by** *(metis (mono_tags))*

then have *ope: $\bigwedge s. s \in S \implies s \in (UU s) \wedge \text{openin } (top\_of\_set S) (UU s)$*
by *blast*

have $\exists k n i. \text{open } k \wedge \text{open } n \wedge$

$t \in k \wedge y \in n \wedge i \in S \wedge h \restriction ((\{0..1\} \cap k) \times (U \cap n)) \subseteq UU i$ **if** $t \in \{0..1\}$ **for** t

proof –

have *hinS: $h(t, y) \in S$*

using $\langle y \in U \rangle$ *him that* **by** *blast*

then have $(t, y) \in (\{0..1\} \times U) \cap h - ' UU(h(t, y))$

using $\langle y \in U \rangle$ $\langle t \in \{0..1\} \rangle$ **by** *(auto simp: ope)*

moreover have *ope_01U: openin $(top\_of\_set (\{0..1\} \times U)) ((\{0..1\} \times U) \cap h - ' UU(h(t, y)))$*

using *hinS ope continuous_on_open_gen [OF him] conth* **by** *blast*

ultimately obtain $V W$ **where** *opeV: open V and $t \in \{0..1\} \cap V$ $t \in \{0..1\} \cap V$*

```

    and opeW: open W and y ∈ U y ∈ W
    and VW: ({0..1} ∩ V) × (U ∩ W) ⊆ (({0..1} × U) ∩
h - ' UU(h(t, y)))
    by (rule Times_in_interior_subtopology) (auto simp: openin_open)
    then show ?thesis
    using hinS by blast
qed
then obtain K NN X where
    K: ⋀t. t ∈ {0..1} ⇒ open (K t)
    and NN: ⋀t. t ∈ {0..1} ⇒ open (NN t)
    and inUS: ⋀t. t ∈ {0..1} ⇒ t ∈ K t ∧ y ∈ NN t ∧ X t ∈ S
    and him: ⋀t. t ∈ {0..1} ⇒ h ' ({0..1} ∩ K t) × (U ∩ NN t) ⊆ UU
(X t)
    by (metis (mono_tags))
    obtain T where T ⊆ ((λi. K i × NN i) ' {0..1} finite T {0::real..1} × {y}
⊆ ⋃T
    proof (rule compactE)
    show compact ({0::real..1} × {y})
    by (simp add: compact_Times)
    show {0..1} × {y} ⊆ (⋃i∈{0..1}. K i × NN i)
    using K inUS by auto
    show ⋀B. B ∈ (λi. K i × NN i) ' {0..1} ⇒ open B
    using K NN by (auto simp: open_Times)
qed blast
then obtain tk where tk ⊆ {0..1} finite tk
    and tk: {0::real..1} × {y} ⊆ (⋃i ∈ tk. K i × NN i)
    by (metis (no_types, lifting) finite_subset_image)
then have tk ≠ {}
    by auto
define n where n = ⋂(NN ' tk)
have y ∈ n open n
    using inUS NN ⟨tk ⊆ {0..1}⟩ ⟨finite tk⟩
    by (auto simp: n_def open_INT subset_iff)
obtain δ where 0 < δ and δ: ⋀T. [T ⊆ {0..1}; diameter T < δ] ⇒ ∃ B ∈ K
' tk. T ⊆ B
    proof (rule Lebesgue_number_lemma [of {0..1} K ' tk])
    show K ' tk ≠ {}
    using ⟨tk ≠ {}⟩ by auto
    show {0..1} ⊆ ⋃(K ' tk)
    using tk by auto
    show ⋀B. B ∈ K ' tk ⇒ open B
    using ⟨tk ⊆ {0..1}⟩ K by auto
qed auto
obtain N::nat where N: N > 1 / δ
    using reals_Archimedean2 by blast
then have N > 0
    using ⟨0 < δ⟩ order.asym by force
have *: ∃ V k. openin (top_of_set U) V ∧ y ∈ V ∧
    continuous_on ({0..of_nat n / N} × V) k ∧

```

```

      k ' ( $\{0..of\_nat\ n / N\} \times V$ )  $\subseteq C \wedge$ 
      ( $\forall z \in V. k\ (0, z) = f\ z$ )  $\wedge$ 
      ( $\forall z \in \{0..of\_nat\ n / N\} \times V. h\ z = p\ (k\ z)$ ) if  $n \leq N$  for  $n$ 
    using that
  proof (induction n)
    case 0
    show ?case
      apply (rule_tac x=U in exI)
      apply (rule_tac x=f  $\circ$  snd in exI)
      apply (intro conjI  $\langle y \in U \rangle$  continuous_intros continuous_on_subset [OF
contf])
      using fim apply (auto simp: heq)
      done
    next
    case (Suc n)
    then obtain V k where opeUV: openin (top_of_set U) V
      and y  $\in V$ 
      and contk: continuous_on ( $\{0..n/N\} \times V$ ) k
      and kim: k ' ( $\{0..n/N\} \times V$ )  $\subseteq C$ 
      and keq:  $\bigwedge z. z \in V \implies k\ (0, z) = f\ z$ 
      and heq:  $\bigwedge z. z \in \{0..n/N\} \times V \implies h\ z = p\ (k\ z)$ 
      using Suc_leD by auto
    have n  $\leq N$ 
      using Suc.prem by auto
    obtain t where t  $\in tk$  and t:  $\{n/N .. (1 + real\ n) / N\} \subseteq K\ t$ 
    proof (rule bexE [OF  $\delta$ ])
      show  $\{n/N .. (1 + real\ n) / N\} \subseteq \{0..1\}$ 
        using Suc.prem by (auto simp: field_split_simps)
      show diameter_less: diameter  $\{n/N .. (1 + real\ n) / N\} < \delta$ 
        using  $\langle 0 < \delta \rangle\ N$  by (auto simp: field_split_simps)
    qed blast
    have t01: t  $\in \{0..1\}$ 
      using  $\langle t \in tk \rangle\ \langle tk \subseteq \{0..1\} \rangle$  by blast
    obtain  $\mathcal{V}$  where  $\mathcal{V}: \bigcup \mathcal{V} = C \cap p - ' UU\ (X\ t)$ 
      and opeC:  $\bigwedge U. U \in \mathcal{V} \implies openin\ (top\_of\_set\ C)\ U$ 
      and pairwise_disjnt  $\mathcal{V}$ 
      and homuu:  $\bigwedge U. U \in \mathcal{V} \implies \exists q. homeomorphism\ U\ (UU\ (X\ t))\ p\ q$ 
      using inUS [OF t01] UU by meson
    have n_div_N_in:  $n/N \in \{n/N .. (1 + real\ n) / N\}$ 
      using N by (auto simp: field_split_simps)
    with t have nN_in_kkt:  $n/N \in K\ t$ 
      by blast
    have k (n/N, y)  $\in C \cap p - ' UU\ (X\ t)$ 
    proof (simp, rule conjI)
      show k (n/N, y)  $\in C$ 
        using  $\langle y \in V \rangle\ kim\ keq$  by force
      have p (k (n/N, y)) = h (n/N, y)
        by (simp add:  $\langle y \in V \rangle\ heq$ )
      also have ...  $\in h\ ' ((\{0..1\} \cap K\ t) \times (U \cap NN\ t))$ 

```

```

    using ⟨y ∈ V⟩ t01 ⟨n ≤ N⟩
    by (simp add: nN_in_kkt ⟨y ∈ U⟩ inUS field_split_simps)
  also have ... ⊆ UU (X t)
    using him t01 by blast
  finally show p (k (n/N, y)) ∈ UU (X t) .
qed
with V have k (n/N, y) ∈ ⋃ V
  by blast
then obtain W where W: k (n/N, y) ∈ W and W ∈ V
  by blast
then obtain p' where opeC': openin (top_of_set C) W
  and hom': homeomorphism W (UU (X t)) p p'
  using homuu opeC by blast
then have W ⊆ C
  using openin_imp_subset by blast
define W' where W' = UU (X t)
have opeVW: openin (top_of_set V) (V ∩ (k ∘ Pair (n / N))) = W
proof (rule continuous_openin_preimage [OF _ _ opeC'])
  show continuous_on V (k ∘ Pair (n/N))
    by (intro continuous_intros continuous_on_subset [OF contk], auto)
  show (k ∘ Pair (n/N)) ' V ⊆ C
    using kim by (auto simp: ⟨y ∈ V⟩ W)
qed
obtain N' where opeUN': openin (top_of_set U) N'
  and y ∈ N' and kimw: k ' ({(n/N)} × N') ⊆ W
proof
  show openin (top_of_set U) (V ∩ (k ∘ Pair (n/N))) = W
    using opeUV opeVW openin_trans by blast
qed (use ⟨y ∈ V⟩ W in ⟨force+⟩)
obtain Q Q' where opeUQ: openin (top_of_set U) Q
  and cloUQ': closedin (top_of_set U) Q'
  and y ∈ Q Q ⊆ Q'
  and Q': Q' ⊆ (U ∩ NN(t)) ∩ N' ∩ V
proof -
  obtain VO VX where open VO open VX and VO: V = U ∩ VO and
  VX: N' = U ∩ VX
    using opeUV opeUN' by (auto simp: openin_open)
  then have open (NN(t) ∩ VO ∩ VX)
    using NN t01 by blast
  then obtain e where e > 0 and e: cball y e ⊆ NN(t) ∩ VO ∩ VX
    by (metis Int_iff ⟨N' = U ∩ VX⟩ ⟨V = U ∩ VO⟩ ⟨y ∈ N'⟩ ⟨y ∈ V⟩ inUS
    open_contains_cball t01)
  show ?thesis
proof
  show openin (top_of_set U) (U ∩ ball y e)
    by blast
  show closedin (top_of_set U) (U ∩ cball y e)
    using e by (auto simp: closedin_closed)
qed (use ⟨y ∈ U⟩ ⟨e > 0⟩ VO VX e in auto)

```



```

qed
then have  $y \in Q' \ Q \subseteq (U \cap NN(t)) \cap N' \cap V$ 
  by blast+
have  $neg: \{0..n/N\} \cup \{n/N..(1 + real\ n) / N\} = \{0..(1 + real\ n) / N\}$ 
  apply (auto simp: field_split_simps)
by (metis not_less of_nat_0_le_iff of_nat_0_less_iff order_trans zero_le_mult_iff)
then have  $negQ': \{0..n/N\} \times Q' \cup \{n/N..(1 + real\ n) / N\} \times Q' = \{0..(1 + real\ n) / N\} \times Q'$ 
  by blast
have  $cont: continuous\_on (\{0..(1 + real\ n) / N\} \times Q') (\lambda x. if\ x \in \{0..n/N\} \times Q' then\ k\ x\ else\ (p' \circ h)\ x)$ 
  unfolding  $negQ'$  [symmetric]
proof (rule continuous_on_cases_local, simp_all add:  $negQ'$  del: comp_apply)
  have  $\exists T. closed\ T \wedge \{0..n/N\} \times Q' = \{0..(1+n)/N\} \times Q' \cap T$ 
    using  $n\_div\_N\_in$ 
  by (rule_tac  $x = \{0 .. n/N\} \times UNIV$  in exI) (auto simp: closed_Times)
  then show  $closedin\ (top\_of\_set\ (\{0..(1 + real\ n) / N\} \times Q')) (\{0..n/N\} \times Q')$ 
    by (simp add: closedin_closed)
  have  $\exists T. closed\ T \wedge \{n/N..(1+n)/N\} \times Q' = \{0..(1+n)/N\} \times Q' \cap T$ 
  by (rule_tac  $x = \{n/N..(1+n)/N\} \times UNIV$  in exI) (auto simp: closed_Times order_trans [rotated])
  then show  $closedin\ (top\_of\_set\ (\{0..(1 + real\ n) / N\} \times Q')) (\{n/N..(1 + real\ n) / N\} \times Q')$ 
    by (simp add: closedin_closed)
show  $continuous\_on (\{0..n/N\} \times Q') k$ 
  using  $Q'$  by (auto intro: continuous_on_subset [OF contk])
have  $continuous\_on (\{n/N..(1 + real\ n) / N\} \times Q') h$ 
proof (rule continuous_on_subset [OF conth])
  show  $\{n/N..(1 + real\ n) / N\} \times Q' \subseteq \{0..1\} \times U$ 
  proof (clarsimp, intro conjI)
    fix  $a\ b$ 
    assume  $b \in Q'$  and  $a: n/N \leq a \leq (1 + real\ n) / N$ 
    have  $0 \leq n/N\ (1 + real\ n) / N \leq 1$ 
      using  $a\ Suc.premis$  by (auto simp: divide_simps)
    with  $a$  show  $0 \leq a \leq 1$ 
      by linarith+
    show  $b \in U$ 
      using  $\langle b \in Q' \rangle\ cloUQ'\ closedin\_imp\_subset$  by blast
  qed
qed
qed
moreover have  $continuous\_on\ (h\ '(\{n/N..(1 + real\ n) / N\} \times Q'))\ p'$ 
proof (rule continuous_on_subset [OF homeomorphism_cont2 [OF hom']])
  have  $h\ '(\{n/N..(1 + real\ n) / N\} \times Q') \subseteq h\ '(\{0..1\} \cap K\ t) \times (U \cap NN\ t)$ 
  proof (rule image_mono)
    show  $\{n/N..(1 + real\ n) / N\} \times Q' \subseteq (\{0..1\} \cap K\ t) \times (U \cap NN\ t)$ 
    proof (clarsimp, intro conjI)
      fix  $a::real$  and  $b$ 

```

```

    assume  $b \in Q'$   $n/N \leq a$   $a \leq (1 + \text{real } n) / N$ 
    show  $0 \leq a$ 
      by (meson  $\langle n/N \leq a \rangle$  divide_nonneg_nonneg of_nat_0_le_iff
order_trans)
    show  $a \leq 1$ 
      using Suc.premis  $\langle a \leq (1 + \text{real } n) / N \rangle$  order_trans by force
    show  $a \in K$  t
      using  $\langle a \leq (1 + \text{real } n) / N \rangle$   $\langle n/N \leq a \rangle$  t by auto
    show  $b \in U$ 
      using  $\langle b \in Q' \rangle$  cloUQ' closedin_imp_subset by blast
    show  $b \in NN$  t
      using  $Q' \langle b \in Q' \rangle$  by auto
  qed
qed
with him show  $h' (\{n/N..(1 + \text{real } n) / N\} \times Q') \subseteq UU (X t)$ 
  using t01 by blast
qed
ultimately show continuous_on  $(\{n/N..(1 + \text{real } n) / N\} \times Q') (p' \circ h)$ 
  by (rule continuous_on_compose)
have  $k (n/N, b) = p' (h (n/N, b))$  if  $b \in Q'$  for  $b$ 
proof -
  have  $k (n/N, b) \in W$ 
    using that  $Q'$  kimw by force
  then have  $k (n/N, b) = p' (p (k (n/N, b)))$ 
    by (simp add: homeomorphism_apply1 [OF hom])
  then show ?thesis
    using  $Q'$  that by (force simp: heq)
qed
then show  $\bigwedge x. x \in \{n/N..(1 + \text{real } n) / N\} \times Q' \wedge$ 
 $x \in \{0..n/N\} \times Q' \implies k x = (p' \circ h) x$ 
  by auto
qed
have  $h\_in\_UU: h (x, y) \in UU (X t)$  if  $y \in Q \neg x \leq n/N$   $0 \leq x$   $x \leq (1 +$ 
 $\text{real } n) / N$  for  $x y$ 
proof -
  have  $x \leq 1$ 
    using Suc.premis that order_trans by force
  moreover have  $x \in K$  t
    by (meson atLeastAtMost_iff le_less not_le subset_eq t that)
  moreover have  $y \in U$ 
    using  $\langle y \in Q \rangle$  opeUQ openin_imp_subset by blast
  moreover have  $y \in NN$  t
    using  $Q' \langle Q \subseteq Q' \rangle \langle y \in Q \rangle$  by auto
  ultimately have  $(x, y) \in ((\{0..1\} \cap K t) \times (U \cap NN t))$ 
    using that by auto
  then have  $h (x, y) \in h' ((\{0..1\} \cap K t) \times (U \cap NN t))$ 
    by blast
  also have  $\dots \subseteq UU (X t)$ 
    by (metis him t01)

```

```

    finally show ?thesis .
qed
let ?k = ( $\lambda x$ . if  $x \in \{0..n/N\} \times Q'$  then  $k\ x$  else  $(p' \circ h)\ x$ )
show ?case
proof (intro exI conjI)
  show continuous_on ( $\{0..real\ (Suc\ n) / N\} \times Q$ ) ?k
    using  $\langle Q \subseteq Q' \rangle$  by (auto intro: continuous_on_subset [OF cont])
  have  $\bigwedge x\ y. \llbracket x \leq n/N; y \in Q'; 0 \leq x \rrbracket \implies k\ (x, y) \in C$ 
    using kim  $Q'$  by force
  moreover have  $p'\ (h\ (x, y)) \in C$  if  $y \in Q \neg x \leq n/N\ 0 \leq x\ x \leq (1 +$ 
 $real\ n) / N$  for  $x\ y$ 
  proof (rule  $\langle W \subseteq C \rangle$  [THEN subsetD])
    show  $p'\ (h\ (x, y)) \in W$ 
      using homeomorphism_image2 [OF hom', symmetric]  $h\_in\_UU\ Q'$ 
 $\langle Q \subseteq Q' \rangle \langle W \subseteq C \rangle$  that by auto
  qed
  ultimately show ?k ' $(\{0..real\ (Suc\ n) / N\} \times Q) \subseteq C$ '
    using  $Q' \langle Q \subseteq Q' \rangle$  by force
  show  $\forall z \in Q. ?k\ (0, z) = f\ z$ 
    using  $Q'$  keq  $\langle Q \subseteq Q' \rangle$  by auto
  show  $\forall z \in \{0..real\ (Suc\ n) / N\} \times Q. h\ z = p\ (?k\ z)$ 
    using  $\langle Q \subseteq U \cap NN\ t \cap N' \cap V \rangle$  heq  $Q' \langle Q \subseteq Q' \rangle$ 
    by (auto simp: homeomorphism_apply2 [OF hom'] dest:  $h\_in\_UU$ )
  qed (auto simp:  $\langle y \in Q \rangle$  opeUQ)
qed
show ?thesis
  using  $*[OF\ order\_refl]\ N\ \langle 0 < \delta \rangle$  by (simp add: split: if_split_asm)
qed
then obtain  $V\ fs$  where  $opeV: \bigwedge y. y \in U \implies openin\ (top\_of\_set\ U)\ (V\ y)$ 
  and  $V: \bigwedge y. y \in U \implies y \in V\ y$ 
  and  $contfs: \bigwedge y. y \in U \implies continuous\_on\ (\{0..1\} \times V\ y)\ (fs\ y)$ 
  and  $*$ :  $\bigwedge y. y \in U \implies (fs\ y) ' $(\{0..1\} \times V\ y) \subseteq C \wedge$$ 
 $(\forall z \in V\ y. fs\ y\ (0, z) = f\ z) \wedge$ 
 $(\forall z \in \{0..1\} \times V\ y. h\ z = p(fs\ y\ z))$ 
  by (metis (mono_tags))
then have  $VU: \bigwedge y. y \in U \implies V\ y \subseteq U$ 
  by (meson openin_imp_subset)
obtain  $k$  where  $contk: continuous\_on\ (\{0..1\} \times U)\ k$ 
  and  $k: \bigwedge x\ i. \llbracket i \in U; x \in \{0..1\} \times U \cap \{0..1\} \times V\ i \rrbracket \implies k\ x = fs\ i\ x$ 
proof (rule pasting_lemma_exists)
  let  $?X = top\_of\_set\ (\{0..1::real\} \times U)$ 
  show  $topspace\ ?X \subseteq (\bigcup_{i \in U}. \{0..1\} \times V\ i)$ 
    using  $V$  by force
  show  $\bigwedge i. i \in U \implies openin\ (top\_of\_set\ (\{0..1\} \times U))\ (\{0..1\} \times V\ i)$ 
    by (simp add: Abstract_Topology.openin_Times opeV)
  show  $\bigwedge i. i \in U \implies continuous\_map$ 
 $(subtopology\ (top\_of\_set\ (\{0..1\} \times U))\ (\{0..1\} \times V\ i))\ euclidean\ (fs$ 
 $i)$ 
    by (metis contfs subtopology_subtopology continuous_map_iff_continuous

```

```

Times_Int_Times VU inf.absorb_iff2 inf.idem)
  show fs i x = fs j x if i ∈ U j ∈ U and x: x ∈ topspace ?X ∩ {0..1} × V i
    ∩ {0..1} × V j
    for i j x
  proof -
    obtain u y where x = (u, y) y ∈ V i y ∈ V j 0 ≤ u u ≤ 1
    using x by auto
    show ?thesis
  proof (rule covering_space_lift_unique [OF cov, of _ (0,y) _ {0..1} × {y}
h])
    show fs i (0, y) = fs j (0, y)
      using* V by (simp add: ⟨y ∈ V i⟩ ⟨y ∈ V j⟩ that)
    show conth_y: continuous_on ({0..1} × {y}) h
      using VU ⟨y ∈ V j⟩ that by (auto intro: continuous_on_subset [OF
conth])
    show h ' ({0..1} × {y}) ⊆ S
      using ⟨y ∈ V i⟩ assms(3) VU that by fastforce
    show continuous_on ({0..1} × {y}) (fs i)
      using continuous_on_subset [OF contfs] ⟨i ∈ U⟩
      by (simp add: ⟨y ∈ V i⟩ subset_iff)
    show fs i ' ({0..1} × {y}) ⊆ C
      using * ⟨y ∈ V i⟩ ⟨i ∈ U⟩ by fastforce
    show ∧x. x ∈ {0..1} × {y} ⇒ h x = p (fs i x)
      using * ⟨y ∈ V i⟩ ⟨i ∈ U⟩ by blast
    show continuous_on ({0..1} × {y}) (fs j)
      using continuous_on_subset [OF contfs] ⟨j ∈ U⟩
      by (simp add: ⟨y ∈ V j⟩ subset_iff)
    show fs j ' ({0..1} × {y}) ⊆ C
      using * ⟨y ∈ V j⟩ ⟨j ∈ U⟩ by fastforce
    show ∧x. x ∈ {0..1} × {y} ⇒ h x = p (fs j x)
      using * ⟨y ∈ V j⟩ ⟨j ∈ U⟩ by blast
    show connected ({0..1::real} × {y})
      using connected_Icc connected_Times connected_sing by blast
    show (0, y) ∈ {0..1::real} × {y}
      by force
    show x ∈ {0..1} × {y}
      using ⟨x = (u, y)⟩ x by blast
  qed
qed
qed force
show ?thesis
proof
  show k ' ({0..1} × U) ⊆ C
    using V*k VU by fastforce
  show ∧y. y ∈ U ⇒ k (0, y) = f y
    by (simp add: V*k)
  show ∧z. z ∈ {0..1} × U ⇒ h z = p (k z)
    using V*k by auto
qed (auto simp: contk)

```

qed

corollary *covering_space_lift_homotopy_alt:*

fixes $p :: 'a::\text{real\_normed\_vector} \Rightarrow 'b::\text{real\_normed\_vector}$
and $h :: 'c::\text{real\_normed\_vector} \times \text{real} \Rightarrow 'b$

assumes $\text{cov}: \text{covering\_space } C \text{ } p \text{ } S$

and $\text{conth}: \text{continuous\_on } (U \times \{0..1\}) \text{ } h$

and $\text{him}: h \text{ ' } (U \times \{0..1\}) \subseteq S$

and $\text{heq}: \bigwedge y. y \in U \Rightarrow h(y, 0) = p(f y)$

and $\text{contf}: \text{continuous\_on } U \text{ } f$ **and** $\text{fim}: f \text{ ' } U \subseteq C$

obtains k **where** $\text{continuous\_on } (U \times \{0..1\}) \text{ } k$

$k \text{ ' } (U \times \{0..1\}) \subseteq C$

$\bigwedge y. y \in U \Rightarrow k(y, 0) = f y$

$\bigwedge z. z \in U \times \{0..1\} \Rightarrow h z = p(k z)$

proof –

have $\text{continuous\_on } (\{0..1\} \times U) \text{ } (h \circ (\lambda z. (\text{snd } z, \text{fst } z)))$

by (*intro continuous_intros continuous_on_subset [OF conth]*) *auto*

then obtain k **where** $\text{contk}: \text{continuous\_on } (\{0..1\} \times U) \text{ } k$

and $\text{kim}: k \text{ ' } (\{0..1\} \times U) \subseteq C$

and $k0: \bigwedge y. y \in U \Rightarrow k(0, y) = f y$

and $\text{heqp}: \bigwedge z. z \in \{0..1\} \times U \Rightarrow (h \circ (\lambda z. \text{Pair } (\text{snd } z) (\text{fst } z)))$

$z = p(k z)$

apply (*rule covering_space_lift_homotopy [OF cov _ _ contf fim]*)

using him **by** (*auto simp: contf heq*)

show *?thesis*

proof

show $\text{continuous\_on } (U \times \{0..1\}) \text{ } (k \circ (\lambda z. (\text{snd } z, \text{fst } z)))$

by (*intro continuous_intros continuous_on_subset [OF contk]*) *auto*

qed (*use kim heqp in <auto simp: k0>*)

qed

corollary *covering_space_lift_homotopic_function:*

fixes $p :: 'a::\text{real\_normed\_vector} \Rightarrow 'b::\text{real\_normed\_vector}$ **and** $g :: 'c::\text{real\_normed\_vector} \Rightarrow 'a$

assumes $\text{cov}: \text{covering\_space } C \text{ } p \text{ } S$

and $\text{contg}: \text{continuous\_on } U \text{ } g$

and $\text{gim}: g \text{ ' } U \subseteq C$

and $\text{pgeq}: \bigwedge y. y \in U \Rightarrow p(g y) = f y$

and $\text{hom}: \text{homotopic\_with\_canon } (\lambda x. \text{True}) \text{ } U \text{ } S \text{ } f \text{ '}$

obtains g' **where** $\text{continuous\_on } U \text{ } g' \text{ image } g' \text{ } U \subseteq C \bigwedge y. y \in U \Rightarrow p(g' y) = f' y$

proof –

obtain h **where** $\text{conth}: \text{continuous\_on } (\{0..1::\text{real}\} \times U) \text{ } h$

and $\text{him}: h \text{ ' } (\{0..1\} \times U) \subseteq S$

and $h0: \bigwedge x. h(0, x) = f x$

and $h1: \bigwedge x. h(1, x) = f' x$

using hom **by** (*auto simp: homotopic_with_def*)

have $\bigwedge y. y \in U \Rightarrow h(0, y) = p(g y)$

by (*simp add: h0 pgeq*)

```

then obtain k where contk: continuous_on ( $\{0..1\} \times U$ ) k
and kim:  $k \restriction (\{0..1\} \times U) \subseteq C$ 
and k0:  $\bigwedge y. y \in U \implies k(0, y) = g \ y$ 
and heq:  $\bigwedge z. z \in \{0..1\} \times U \implies h \ z = p(k \ z)$ 
using covering_space_lift_homotopy [OF cov conth him _ contg gim] by metis
show ?thesis
proof
show continuous_on U (k o Pair 1)
by (meson contk atLeastAtMost_iff continuous_on_o_Pair order_refl zero_le_one)
show (k o Pair 1)  $\restriction U \subseteq C$ 
using kim by auto
show  $\bigwedge y. y \in U \implies p((k o Pair 1) \ y) = f' \ y$ 
by (auto simp: h1 heq [symmetric])
qed
qed

corollary covering_space_lift_inessential_function:
fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector and U :: 'c::real_normed_vector
set
assumes cov: covering_space C p S
and hom: homotopic_with_canon ( $\lambda x. \text{True}$ ) U S f ( $\lambda x. a$ )
obtains g where continuous_on U g g  $\restriction U \subseteq C$   $\bigwedge y. y \in U \implies p(g \ y) = f \ y$ 
proof (cases U = {})
case True
then show ?thesis
using that continuous_on_empty by blast
next
case False
then obtain b where b:  $b \in C$  p b = a
using covering_space_imp_surjective [OF cov] homotopic_with_imp_subset2
[OF hom]
by auto
then have gim: ( $\lambda y. b$ )  $\restriction U \subseteq C$ 
by blast
show ?thesis
proof (rule covering_space_lift_homotopic_function [OF cov continuous_on_const
gim])
show  $\bigwedge y. y \in U \implies p \ b = a$ 
using b by auto
qed (use that homotopic_with_symD [OF hom] in auto)
qed

```

6.19.5 Lifting of general functions to covering space

proposition *covering_space_lift_path_strong*:

```

fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
and f :: 'c::real_normed_vector  $\Rightarrow$  'b
assumes cov: covering_space C p S and a  $\in C$ 
and path g and pag:  $\text{path\_image } g \subseteq S$  and pas:  $\text{pathstart } g = p \ a$ 

```

```

    obtains h where path h path_image h  $\subseteq$  C pathstart h = a
    and  $\bigwedge t. t \in \{0..1\} \implies p(h\ t) = g\ t$ 
proof -
  obtain k:: real  $\times$  'c  $\Rightarrow$  'a
  where contk: continuous_on ( $\{0..1\} \times \{undefined\}$ ) k
    and kim: k ' ( $\{0..1\} \times \{undefined\}$ )  $\subseteq$  C
    and k0: k (0, undefined) = a
    and pk:  $\bigwedge z. z \in \{0..1\} \times \{undefined\} \implies p(k\ z) = (g \circ fst)\ z$ 
proof (rule covering_space_lift_homotopy [OF cov, of  $\{undefined\}\ g \circ fst$ ])
  show continuous_on ( $\{0..1::real\} \times \{undefined::'c\}$ ) (g  $\circ$  fst)
    using  $\langle$ path g $\rangle$  by (intro continuous_intros) (simp add: path_def)
  show (g  $\circ$  fst) ' ( $\{0..1\} \times \{undefined\}$ )  $\subseteq$  S
    using pag by (auto simp: path_image_def)
  show (g  $\circ$  fst) (0, y) = p a if y  $\in$   $\{undefined\}$  for y::'c
    by (metis comp_def fst_conv pas pathstart_def)
qed (use assms in auto)
show ?thesis
proof
  show path (k  $\circ$  ( $\lambda t. \text{Pair } t\ undefined$ ))
    unfolding path_def
    by (intro continuous_on_compose continuous_intros continuous_on_subset
[OF contk]) auto
  show path_image (k  $\circ$  ( $\lambda t. (t, undefined)$ ))  $\subseteq$  C
    using kim by (auto simp: path_image_def)
  show pathstart (k  $\circ$  ( $\lambda t. (t, undefined)$ )) = a
    by (auto simp: pathstart_def k0)
  show  $\bigwedge t. t \in \{0..1\} \implies p ((k \circ (\lambda t. (t, undefined)))\ t) = g\ t$ 
    by (auto simp: pk)
qed
qed

corollary covering_space_lift_path:
  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes cov: covering_space C p S and path g and pig: path_image g  $\subseteq$  S
  obtains h where path h path_image h  $\subseteq$  C  $\bigwedge t. t \in \{0..1\} \implies p(h\ t) = g\ t$ 
proof -
  obtain a where a  $\in$  C pathstart g = p a
  by (metis pig cov covering_space_imp_surjective imageE pathstart_in_path_image
subsetCE)
  show ?thesis
    using covering_space_lift_path_strong [OF cov  $\langle$ a  $\in$  C $\rangle$   $\langle$ path g $\rangle$  pig]
    by (metis  $\langle$ pathstart g = p a $\rangle$  that)
qed

proposition covering_space_lift_homotopic_paths:
  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes cov: covering_space C p S
    and path g1 and pig1: path_image g1  $\subseteq$  S

```

```

    and path g2 and pig2: path_image g2  $\subseteq S$ 
    and hom: homotopic_paths S g1 g2
    and path h1 and pih1: path_image h1  $\subseteq C$  and ph1:  $\bigwedge t. t \in \{0..1\} \implies$ 
p(h1 t) = g1 t
    and path h2 and pih2: path_image h2  $\subseteq C$  and ph2:  $\bigwedge t. t \in \{0..1\} \implies$ 
p(h2 t) = g2 t
    and h1h2: pathstart h1 = pathstart h2
    shows homotopic_paths C h1 h2
proof -
  obtain h :: real  $\times$  real  $\Rightarrow$  'b
  where conth: continuous_on ( $\{0..1\} \times \{0..1\}$ ) h
    and him: h ' ( $\{0..1\} \times \{0..1\}$ )  $\subseteq S$ 
    and h0:  $\bigwedge x. h (0, x) = g1 x$  and h1:  $\bigwedge x. h (1, x) = g2 x$ 
    and heq0:  $\bigwedge t. t \in \{0..1\} \implies h (t, 0) = g1 0$ 
    and heq1:  $\bigwedge t. t \in \{0..1\} \implies h (t, 1) = g1 1$ 
    using hom by (auto simp: homotopic_paths_def homotopic_with_def path-
start_def pathfinish_def)
  obtain k where kontk: continuous_on ( $\{0..1\} \times \{0..1\}$ ) k
    and kim: k ' ( $\{0..1\} \times \{0..1\}$ )  $\subseteq C$ 
    and kh2:  $\bigwedge y. y \in \{0..1\} \implies k (y, 0) = h2 0$ 
    and hpk:  $\bigwedge z. z \in \{0..1\} \times \{0..1\} \implies h z = p (k z)$ 
proof (rule covering_space_lift_homotopy_alt [OF cov conth him])
  show  $\bigwedge y. y \in \{0..1\} \implies h (y, 0) = p (h2 0)$ 
  by (metis atLeastAtMost_iff h1h2 heq0 order_refl pathstart_def ph1 zero_le_one)
qed (use path_image_def pih2 in <fastforce+>)
have contg1: continuous_on  $\{0..1\}$  g1 and contg2: continuous_on  $\{0..1\}$  g2
  using <path g1> <path g2> path_def by blast+
have g1im: g1 '  $\{0..1\} \subseteq S$  and g2im: g2 '  $\{0..1\} \subseteq S$ 
  using path_image_def pig1 pig2 by auto
have conth1: continuous_on  $\{0..1\}$  h1 and conth2: continuous_on  $\{0..1\}$  h2
  using <path h1> <path h2> path_def by blast+
have h1im: h1 '  $\{0..1\} \subseteq C$  and h2im: h2 '  $\{0..1\} \subseteq C$ 
  using path_image_def pih1 pih2 by auto
show ?thesis
  unfolding homotopic_paths pathstart_def pathfinish_def
proof (intro exI conjI ballI)
  show keqh1:  $k(0, x) = h1 x$  if  $x \in \{0..1\}$  for x
  proof (rule covering_space_lift_unique [OF cov _ contg1 g1im])
    show  $k (0, 0) = h1 0$ 
    by (metis atLeastAtMost_iff h1h2 kh2 order_refl pathstart_def zero_le_one)
    show continuous_on  $\{0..1\}$  ( $\lambda a. k (0, a)$ )
    by (intro continuous_intros continuous_on_compose2 [OF kontk]) auto
    show  $\bigwedge x. x \in \{0..1\} \implies g1 x = p (k (0, x))$ 
    by (metis atLeastAtMost_iff h0 hpk zero_le_one mem_Sigma_iff order_refl)
  qed (use conth1 h1im kim that in <auto simp: ph1>)
  show  $k(1, x) = h2 x$  if  $x \in \{0..1\}$  for x
  proof (rule covering_space_lift_unique [OF cov _ contg2 g2im])
    show  $k (1, 0) = h2 0$ 
    by (metis atLeastAtMost_iff kh2 order_refl zero_le_one)

```



```

  show continuous_on {0..1} (λa. k (1, a))
    by (intro continuous_intros continuous_on_compose2 [OF contk]) auto
  show ∧x. x ∈ {0..1} ⇒ g2 x = p (k (1, x))
    by (metis atLeastAtMost_iff h1 hpk mem_Sigma_iff order_refl zero_le_one)
qed (use conth2 h2im kim that in ⟨auto simp: ph2⟩)
show ∧t. t ∈ {0..1} ⇒ (k ∘ Pair t) 0 = h1 0
  by (metis comp_apply h1h2 kh2 pathstart_def)
show (k ∘ Pair t) 1 = h1 1 if t ∈ {0..1} for t
proof (rule covering_space_lift_unique
  [OF cov, of λa. (k ∘ Pair a) 1 0 λa. h1 1 {0..1} λx. g1 1])
  show (k ∘ Pair 0) 1 = h1 1
    using keqh1 by auto
  show continuous_on {0..1} (λa. (k ∘ Pair a) 1)
    by (auto intro!: continuous_intros continuous_on_compose2 [OF contk])
  show ∧x. x ∈ {0..1} ⇒ g1 1 = p ((k ∘ Pair x) 1)
    using heq1 hpk by auto
qed (use contk kim g1im h1im that in ⟨auto simp: ph1⟩)
qed (use contk kim in auto)
qed

```

corollary *covering_space_monodromy:*

```

  fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
  assumes cov: covering_space C p S
    and path g1 and pig1: path_image g1 ⊆ S
    and path g2 and pig2: path_image g2 ⊆ S
    and hom: homotopic_paths S g1 g2
    and path h1 and pih1: path_image h1 ⊆ C and ph1: ∧t. t ∈ {0..1} ⇒
p(h1 t) = g1 t
    and path h2 and pih2: path_image h2 ⊆ C and ph2: ∧t. t ∈ {0..1} ⇒
p(h2 t) = g2 t
    and h1h2: pathstart h1 = pathstart h2
  shows pathfinish h1 = pathfinish h2
using covering_space_lift_homotopic_paths [OF assms] homotopic_paths_imp_pathfinish
by blast

```

corollary *covering_space_lift_homotopic_path:*

```

  fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
  assumes cov: covering_space C p S
    and hom: homotopic_paths S f f'
    and path g and pig: path_image g ⊆ C
    and a: pathstart g = a and b: pathfinish g = b
    and pgeq: ∧t. t ∈ {0..1} ⇒ p(g t) = f t
  obtains g' where path g' path_image g' ⊆ C
    pathstart g' = a pathfinish g' = b ∧t. t ∈ {0..1} ⇒ p(g' t) = f' t
proof (rule covering_space_lift_path_strong [OF cov, of a f])
  show a ∈ C
    using a pig by auto

```

```

show path f' path_image f' ⊆ S
using hom homotopic_paths_imp_path homotopic_paths_imp_subset by blast+
show pathstart f' = p a
by (metis a atLeastAtMost_iff hom homotopic_paths_imp_pathstart order_refl
pathstart_def pgeq zero_le_one)
qed (metis (mono_tags, lifting) assms cov covering_space_monodromy hom ho-
motopic_paths_imp_path homotopic_paths_imp_subset pgeq pig)

```

proposition *covering_space_lift_general*:

```

fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
and f :: 'c::real_normed_vector ⇒ 'b
assumes cov: covering_space C p S and a ∈ C z ∈ U
and U: path_connected U locally_path_connected U
and contf: continuous_on U f and fim: f ' U ⊆ S
and feq: f z = p a
and hom: ⋀r. [path r; path_image r ⊆ U; pathstart r = z; pathfinish r = z]
⇒ ∃ q. path q ∧ path_image q ⊆ C ∧
pathstart q = a ∧ pathfinish q = a ∧
homotopic_paths S (f ∘ r) (p ∘ q)
obtains g where continuous_on U g g ' U ⊆ C g z = a ∧ y. y ∈ U ⇒ p(g y)
= f y
proof -
have *: ∃ g h. path g ∧ path_image g ⊆ U ∧
pathstart g = z ∧ pathfinish g = y ∧
path h ∧ path_image h ⊆ C ∧ pathstart h = a ∧
(∀ t ∈ {0..1}. p(h t) = f(g t))
if y ∈ U for y
proof -
obtain g where path g path_image g ⊆ U and pastg: pathstart g = z
and pafg: pathfinish g = y
using U ⟨z ∈ U⟩ ⟨y ∈ U⟩ by (force simp: path_connected_def)
obtain h where path h path_image h ⊆ C pathstart h = a
and ⋀t. t ∈ {0..1} ⇒ p(h t) = (f ∘ g) t
proof (rule covering_space_lift_path_strong [OF cov ⟨a ∈ C⟩])
show path (f ∘ g)
using ⟨path g⟩ ⟨path_image g ⊆ U⟩ contf continuous_on_subset path_continuous_image
by blast
show path_image (f ∘ g) ⊆ S
by (metis ⟨path_image g ⊆ U⟩ fim image_mono path_image_compose
subset_trans)
show pathstart (f ∘ g) = p a
by (simp add: feq pastg pathstart_compose)
qed auto
then show ?thesis
by (metis ⟨path g⟩ ⟨path_image g ⊆ U⟩ comp_apply pafg pastg)
qed
have ∃ l. ∀ g h. path g ∧ path_image g ⊆ U ∧ pathstart g = z ∧ pathfinish g =
y ∧

```

```

      path h ∧ path_image h ⊆ C ∧ pathstart h = a ∧
      (∀ t ∈ {0..1}. p(h t) = f(g t)) ⟶ pathfinish h = l for y
proof -
  have pathfinish h = pathfinish h'
  if g: path g path_image g ⊆ U pathstart g = z pathfinish g = y
  and h: path h path_image h ⊆ C pathstart h = a
  and phg: ∧t. t ∈ {0..1} ⟹ p(h t) = f(g t)
  and g': path g' path_image g' ⊆ U pathstart g' = z pathfinish g' = y
  and h': path h' path_image h' ⊆ C pathstart h' = a
  and phg': ∧t. t ∈ {0..1} ⟹ p(h' t) = f(g' t)
  for g h g' h'
proof -
  obtain q where path q and piq: path_image q ⊆ C and pastq: pathstart q
= a and pafiq: pathfinish q = a
    and homS: homotopic_paths S (f ∘ g +++ reversepath g') (p ∘ q)
  using g g' hom [of g +++ reversepath g'] by (auto simp: subset_path_image_join)
  have papq: path (p ∘ q)
  using homS homotopic_paths_imp_path by blast
  have pipq: path_image (p ∘ q) ⊆ S
  using homS homotopic_paths_imp_subset by blast
  obtain q' where path q' path_image q' ⊆ C
  and pathstart q' = pathstart q pathfinish q' = pathfinish q
  and pq'_eq: ∧t. t ∈ {0..1} ⟹ p (q' t) = (f ∘ g +++ reversepath
g') t
  using covering_space_lift_homotopic_path [OF cov homotopic_paths_sym
[OF homS] ⟨path q⟩ piq refl refl]
  by auto
  have q' t = (h ∘ (*R) 2) t if 0 ≤ t t ≤ 1/2 for t
  proof (rule covering_space_lift_unique [OF cov, of q' 0 h ∘ (*R) 2 {0..1/2}
f ∘ g ∘ (*R) 2 t])
    show q' 0 = (h ∘ (*R) 2) 0
    by (metis ⟨pathstart q' = pathstart q⟩ comp_def h(3) pastq pathstart_def
pth_4(2))
    show continuous_on {0..1/2} (f ∘ g ∘ (*R) 2)
    proof (intro continuous_intros continuous_on_path [OF ⟨path g⟩] contin-
uous_on_subset [OF contf])
      show g ' (*R) 2 ' {0..1/2} ⊆ U
      using g path_image_def by fastforce
      qed auto
      show (f ∘ g ∘ (*R) 2) ' {0..1/2} ⊆ S
      using g(2) path_image_def fim by fastforce
      show (h ∘ (*R) 2) ' {0..1/2} ⊆ C
      using h path_image_def by fastforce
      show q' ' {0..1/2} ⊆ C
      using ⟨path_image q' ⊆ C⟩ path_image_def by fastforce
      show ∧x. x ∈ {0..1/2} ⟹ (f ∘ g ∘ (*R) 2) x = p (q' x)
      by (auto simp: joinpaths_def pq'_eq)
      show ∧x. x ∈ {0..1/2} ⟹ (f ∘ g ∘ (*R) 2) x = p ((h ∘ (*R) 2) x)
      by (simp add: phg)

```

```

show continuous_on {0..1/2} q'
  by (simp add: continuous_on_path ⟨path q'⟩)
show continuous_on {0..1/2} (h ∘ (*R) 2)
  by (intro continuous_intros continuous_on_path [OF ⟨path h⟩]) auto
qed (use that in auto)
moreover have q' t = (reversepath h' ∘ (λt. 2 *R t - 1)) t if 1/2 < t ≤
1 for t
proof (rule covering_space_lift_unique [OF cov, of q' 1 reversepath h' ∘ (λt.
2 *R t - 1) {1/2<..1} f ∘ reversepath g' ∘ (λt. 2 *R t - 1) t])
show q' 1 = (reversepath h' ∘ (λt. 2 *R t - 1)) 1
  using h' ⟨pathfinish q' = pathfinish q⟩ pafiq
  by (simp add: pathstart_def pathfinish_def reversepath_def)
show continuous_on {1/2<..1} (f ∘ reversepath g' ∘ (λt. 2 *R t - 1))
  proof (intro continuous_intros continuous_on_path ⟨path g'⟩ continu-
ous_on_subset [OF conf])
    show reversepath g' ' (λt. 2 *R t - 1) ' {1/2<..1} ⊆ U
      using g' by (auto simp: path_image_def reversepath_def)
    qed (use g' in auto)
    show (f ∘ reversepath g' ∘ (λt. 2 *R t - 1)) ' {1/2<..1} ⊆ S
      using g'(2) path_image_def fim by (auto simp: image_subset_iff
path_image_def reversepath_def)
    show q' ' {1/2<..1} ⊆ C
      using ⟨path_image q' ⊆ C⟩ path_image_def by fastforce
    show (reversepath h' ∘ (λt. 2 *R t - 1)) ' {1/2<..1} ⊆ C
      using h' by (simp add: path_image_def reversepath_def subset_eq)
    show ∧x. x ∈ {1/2<..1} ⇒ (f ∘ reversepath g' ∘ (λt. 2 *R t - 1)) x =
p (q' x)
      by (auto simp: joinpaths_def pq'_eq)
    show ∧x. x ∈ {1/2<..1} ⇒
      (f ∘ reversepath g' ∘ (λt. 2 *R t - 1)) x = p ((reversepath h' ∘ (λt.
2 *R t - 1)) x)
      by (simp add: phg' reversepath_def)
    show continuous_on {1/2<..1} q'
      by (auto intro: continuous_on_path [OF ⟨path q'⟩])
    show continuous_on {1/2<..1} (reversepath h' ∘ (λt. 2 *R t - 1))
      by (intro continuous_intros continuous_on_path ⟨path h'⟩) (use h' in
auto)
    qed (use that in auto)
  ultimately have q' t = (h +++ reversepath h') t if 0 ≤ t ≤ 1 for t
    using that by (simp add: joinpaths_def)
  then have path(h +++ reversepath h')
    by (auto intro: path_eq [OF ⟨path q'⟩])
  then show ?thesis
    by (auto simp: ⟨path h⟩ ⟨path h'⟩)
qed
then show ?thesis by metis
qed
then obtain l :: 'c ⇒ 'a
  where l: ∧y g h. ⌊path g; path_image g ⊆ U; pathstart g = z; pathfinish

```

```

g = y;
      path h; path_image h  $\subseteq$  C; pathstart h = a;
       $\bigwedge t. t \in \{0..1\} \implies p(h\ t) = f(g\ t) \implies \text{pathfinish } h = l\ y$ 

  by metis
  show ?thesis
  proof
    show pleg: p (l y) = f y if y  $\in$  U for y
      using*[OF  $\langle y \in U \rangle$ ] by (metis l atLeastAtMost_iff order_refl pathfinish_def
zero_le_one)
    show l z = a
      using l [of linepath z z linepath a a] by (auto simp: assms)
    show LC: l ' U  $\subseteq$  C
      by (clarify dest!: *) (metis (full_types) l pathfinish_in_path_image subsetCE)
    have  $\exists T. \text{openin } (\text{top\_of\_set } U) \ T \wedge y \in T \wedge T \subseteq U \cap l - ' X$ 
      if X: openin (top_of_set C) X and y  $\in$  U l y  $\in$  X for X y
    proof -
      have X  $\subseteq$  C
        using X openin_euclidean_subtopology_iff by blast
      have f y  $\in$  S
        using fim  $\langle y \in U \rangle$  by blast
      then obtain W  $\mathcal{V}$ 
        where WV: f y  $\in$  W  $\wedge$  openin (top_of_set S) W  $\wedge$ 
          ( $\bigcup \mathcal{V} = C \cap p - ' W \wedge$ 
            ( $\forall U \in \mathcal{V}. \text{openin } (\text{top\_of\_set } C) \ U \wedge$ 
              pairwise disjoint  $\mathcal{V} \wedge$ 
              ( $\forall U \in \mathcal{V}. \exists q. \text{homeomorphism } U \ W \ p\ q$ ))
          using cov by (force simp: covering_space_def)
      then have l y  $\in \bigcup \mathcal{V}$ 
        using  $\langle X \subseteq C \rangle$  pleg that by auto
      then obtain W' where l y  $\in$  W' and W'  $\in \mathcal{V}$ 
        by blast
      with WV obtain p' where opeCW': openin (top_of_set C) W'
        and homUW': homeomorphism W' W p p'
      by blast
      then have contp': continuous_on W p' and p'im: p' ' W  $\subseteq$  W'
        using homUW' homeomorphism_image2 homeomorphism_cont2 by fast-
force+
      obtain V where y  $\in$  V y  $\in$  U and fimW: f ' V  $\subseteq$  W V  $\subseteq$  U
        and path_connected V and opeUV: openin (top_of_set U) V
      proof -
        have openin (top_of_set U) (U  $\cap$  f - ' W)
          using WV contf continuous_on_open_gen fim by auto
        then obtain UO where openin (top_of_set U) UO  $\wedge$  path_connected UO
 $\wedge y \in UO \wedge UO \subseteq U \cap f - ' W$ 
          using U WV  $\langle y \in U \rangle$  unfolding locally_path_connected by (meson IntI
vimage_eq)
        then show ?thesis
          by (meson  $\langle y \in U \rangle$  image_subset_iff_subset_vimage le_inf_iff that)
      qed
    qed
  end

```

```

have  $W' \subseteq C \ W \subseteq S$ 
  using  $opeCW' \ WV \ openin\_imp\_subset$  by auto
have  $p'im: p' \text{ -- } W \subseteq W'$ 
  using  $homUW' \ homeomorphism\_image2$  by fastforce
show ?thesis
proof (intro exI conjI)
  have  $openin \ (top\_of\_set \ S) \ (W \cap p' \text{ -- } (W' \cap X))$ 
  proof (rule openin_trans)
    show  $openin \ (top\_of\_set \ W) \ (W \cap p' \text{ -- } (W' \cap X))$ 
    using  $X \text{ -- } W' \subseteq C$  by (intro continuous_openin_preimage [OF contp'
 $p'im$ ]) (auto simp: openin_open)
    show  $openin \ (top\_of\_set \ S) \ W$ 
    using  $WV$  by blast
  qed
  then show  $openin \ (top\_of\_set \ U) \ (V \cap (U \cap (f \text{ -- } (W \cap (p' \text{ -- } (W' \cap X))))))$ 
  by (blast intro:  $opeUV \ openin\_subtopology\_self \ continuous\_openin\_preimage$ 
[OF contf fim])
  have  $p' \ (f \ y) \in X$ 
  using  $\langle l \ y \in W' \rangle \ homeomorphism\_apply1 \ [OF \ homUW'] \ pleq \ \langle y \in U \rangle \ \langle l$ 
 $y \in X \rangle$  by fastforce
  then show  $y \in V \cap (U \cap f \text{ -- } (W \cap p' \text{ -- } (W' \cap X)))$ 
  using  $\langle y \in U \rangle \ \langle y \in V \rangle \ WV \ p'im$  by auto
  show  $V \cap (U \cap f \text{ -- } (W \cap p' \text{ -- } (W' \cap X))) \subseteq U \cap l \text{ -- } X$ 
  proof (intro subsetI IntI; clarify)
    fix  $y'$ 
    assume  $y': y' \in V \ y' \in U \ f \ y' \in W \ p' \ (f \ y') \in W' \ p' \ (f \ y') \in X$ 
    then obtain  $\gamma$  where  $path \ \gamma \ path\_image \ \gamma \subseteq V \ pathstart \ \gamma = y \ pathfinish$ 
 $\gamma = y'$ 
    by (meson  $\langle path\_connected \ V \rangle \ \langle y \in V \rangle \ path\_connected\_def$ )
    obtain  $pp \ qq$  where  $pp: path \ pp \ path\_image \ pp \subseteq U \ pathstart \ pp = z$ 
 $pathfinish \ pp = y$ 
    and  $qq: path \ qq \ path\_image \ qq \subseteq C \ pathstart \ qq = a$ 
    and  $pqqeq: \bigwedge t. t \in \{0..1\} \implies p(qq \ t) = f(pp \ t)$ 
    using*[OF  $\langle y \in U \rangle$ ] by blast
    have  $finW: \bigwedge x. \llbracket 0 \leq x; x \leq 1 \rrbracket \implies f \ (\gamma \ x) \in W$ 
    using  $\langle path\_image \ \gamma \subseteq V \rangle$  by (auto simp: image_subset_iff path_image_def
 $finW \ [THEN \ subsetD]$ )
    have  $pathfinish \ (qq \ +++ \ (p' \circ f \circ \gamma)) = l \ y'$ 
    proof (rule l [of  $pp \ +++ \ \gamma \ y' \ qq \ +++ \ (p' \circ f \circ \gamma)$ ])
      show  $path \ (pp \ +++ \ \gamma)$ 
      by (simp add:  $\langle path \ \gamma \rangle \ \langle path \ pp \rangle \ \langle pathfinish \ pp = y \rangle \ \langle pathstart \ \gamma = y \rangle$ )
      show  $path\_image \ (pp \ +++ \ \gamma) \subseteq U$ 
      using  $\langle V \subseteq U \rangle \ \langle path\_image \ \gamma \subseteq V \rangle \ \langle path\_image \ pp \subseteq U \rangle$ 
 $not\_in\_path\_image\_join$  by blast
      show  $pathstart \ (pp \ +++ \ \gamma) = z$ 
      by (simp add:  $\langle pathstart \ pp = z \rangle$ )
      show  $pathfinish \ (pp \ +++ \ \gamma) = y'$ 
      by (simp add:  $\langle pathfinish \ \gamma = y' \rangle$ )

```

```

    have pathfinish qq = l y
      using ⟨path pp⟩ ⟨path qq⟩ ⟨path_image pp ⊆ U⟩ ⟨path_image qq ⊆ C⟩
    ⟨pathfinish pp = y⟩ ⟨pathstart pp = z⟩ ⟨pathstart qq = a⟩ l pqreq by blast
    also have ... = p' (f y)
      using ⟨l y ∈ W'⟩ homUW' homeomorphism_apply1 pleq that(2) by
fastforce
    finally have pathfinish qq = p' (f y) .
    then have paqq: pathfinish qq = pathstart (p' ∘ f ∘ γ)
      by (simp add: ⟨pathstart γ = y⟩ pathstart_compose)
    have continuous_on (path_image γ) (p' ∘ f)
    proof (rule continuous_on_compose)
      show continuous_on (path_image γ) f
        using ⟨path_image γ ⊆ V⟩ ⟨V ⊆ U⟩ contf continuous_on_subset
by blast
      show continuous_on (f ` path_image γ) p'
    proof (rule continuous_on_subset [OF contp'])
      show f ` path_image γ ⊆ W
        by (auto simp: path_image_def pathfinish_def pathstart_def finW)
      qed
    qed
    then show path (qq +++ (p' ∘ f ∘ γ))
      using ⟨path γ⟩ ⟨path qq⟩ paqq path_continuous_image path_join_imp
by blast
    show path_image (qq +++ (p' ∘ f ∘ γ)) ⊆ C
    proof (rule subset_path_image_join)
      show path_image qq ⊆ C
        by (simp add: ⟨path_image qq ⊆ C⟩)
      show path_image (p' ∘ f ∘ γ) ⊆ C
        by (metis ⟨W' ⊆ C⟩ ⟨path_image γ ⊆ V⟩ dual_order.trans fimW(1)
image_comp image_mono p'im path_image_compose)
      qed
    show pathstart (qq +++ (p' ∘ f ∘ γ)) = a
      by (simp add: ⟨pathstart qq = a⟩)
    show p ((qq +++ (p' ∘ f ∘ γ)) ξ) = f ((pp +++ γ) ξ) if ξ: ξ ∈ {0..1}
for ξ
    proof (simp add: joinpaths_def, safe)
      show p (qq (2*ξ)) = f (pp (2*ξ)) if ξ*2 ≤ 1
        using ⟨ξ ∈ {0..1}⟩ pqreq that by auto
      show p (p' (f (γ (2*ξ - 1)))) = f (γ (2*ξ - 1)) if ¬ ξ*2 ≤ 1
        using that ξ by (auto intro: homeomorphism_apply2 [OF homUW'
finW])
      qed
    qed
    with ⟨pathfinish γ = y'⟩ ⟨p' (f y') ∈ X⟩ show y' ∈ l - ` X
      unfolding pathfinish_join by (simp add: pathfinish_def)
    qed
  qed
  then show continuous_on U l

```

by (metis IntD1 IntD2 vimage_eq openin_subopen continuous_on_open_gen
[OF LC])

qed

qed

corollary *covering_space_lift_stronger*:

fixes $p :: 'a::\text{real\_normed\_vector} \Rightarrow 'b::\text{real\_normed\_vector}$

and $f :: 'c::\text{real\_normed\_vector} \Rightarrow 'b$

assumes $\text{cov}: \text{covering\_space } C \ p \ S \ a \in C \ z \in U$

and $U: \text{path\_connected } U \text{ locally\_path\_connected } U$

and $\text{contf}: \text{continuous\_on } U \ f$ and $\text{fim}: f \restriction U \subseteq S$

and $\text{feq}: f \ z = p \ a$

and $\text{hom}: \bigwedge r. \llbracket \text{path } r; \text{path\_image } r \subseteq U; \text{pathstart } r = z; \text{pathfinish } r = z \rrbracket$
 $\implies \exists b. \text{homotopic\_paths } S \ (f \circ r) \ (\text{linepath } b \ b)$

obtains g where $\text{continuous\_on } U \ g \ g \restriction U \subseteq C \ g \ z = a \ \bigwedge y. y \in U \implies p(g \ y)$
 $= f \ y$

proof (rule *covering_space_lift_general* [OF cov U contf fim feq])

fix r

assume $\text{path } r \ \text{path\_image } r \subseteq U \ \text{pathstart } r = z \ \text{pathfinish } r = z$

then obtain b where $b: \text{homotopic\_paths } S \ (f \circ r) \ (\text{linepath } b \ b)$

using hom by blast

then have $f \ (\text{pathstart } r) = b$

by (metis *homotopic_paths_imp_pathstart_pathstart_compose_pathstart_linepath*)

then have $\text{homotopic\_paths } S \ (f \circ r) \ (\text{linepath } (f \ z) \ (f \ z))$

by (simp add: $b \ \langle \text{pathstart } r = z \rangle$)

then have $\text{homotopic\_paths } S \ (f \circ r) \ (p \circ \text{linepath } a \ a)$

by (simp add: $o\_def \ feq \ \text{linepath\_def}$)

then show $\exists q. \text{path } q \ \wedge$

$\text{path\_image } q \subseteq C \ \wedge$

$\text{pathstart } q = a \ \wedge \ \text{pathfinish } q = a \ \wedge \ \text{homotopic\_paths } S \ (f \circ r) \ (p$

$\circ q)$

by (force simp: $\langle a \in C \rangle$)

qed auto

corollary *covering_space_lift_strong*:

fixes $p :: 'a::\text{real\_normed\_vector} \Rightarrow 'b::\text{real\_normed\_vector}$

and $f :: 'c::\text{real\_normed\_vector} \Rightarrow 'b$

assumes $\text{cov}: \text{covering\_space } C \ p \ S \ a \in C \ z \in U$

and $\text{scU}: \text{simply\_connected } U$ and $\text{lpcU}: \text{locally\_path\_connected } U$

and $\text{contf}: \text{continuous\_on } U \ f$ and $\text{fim}: f \restriction U \subseteq S$

and $\text{feq}: f \ z = p \ a$

obtains g where $\text{continuous\_on } U \ g \ g \restriction U \subseteq C \ g \ z = a \ \bigwedge y. y \in U \implies p(g \ y)$
 $= f \ y$

proof (rule *covering_space_lift_stronger* [OF cov _ lpcU contf fim feq])

show $\text{path\_connected } U$

using scU *simply_connected_eq_contractible_loop_some* by blast

fix r

assume $r: \text{path } r \ \text{path\_image } r \subseteq U \ \text{pathstart } r = z \ \text{pathfinish } r = z$

have $\text{linepath } (f \ z) \ (f \ z) = f \circ \text{linepath } z \ z$


```

    by (simp add: o_def linepath_def)
  then have homotopic_paths S (f ∘ r) (linepath (f z) (f z))
    by (metis r contf fim homotopic_paths_continuous_image scU simply_connected_eq_contractible_path)
  then show ∃ b. homotopic_paths S (f ∘ r) (linepath b b)
    by blast
qed blast

```

corollary *covering_space_lift*:

```

  fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
    and f :: 'c::real_normed_vector ⇒ 'b
  assumes cov: covering_space C p S
    and U: simply_connected U locally_path_connected U
    and contf: continuous_on U f and fim: f ' U ⊆ S
  obtains g where continuous_on U g g ' U ⊆ C ∧ y. y ∈ U ⇒ p(g y) = f y
proof (cases U = {})
  case True
    with that show ?thesis by auto
  next
  case False
    then obtain z where z ∈ U by blast
    then obtain a where a ∈ C f z = p a
      by (metis cov covering_space_imp_surjective fim image_iff image_subset_iff)
    then show ?thesis
      by (metis that covering_space_lift_strong [OF cov _ ⟨z ∈ U⟩ U contf fim])
qed

```

6.19.6 Homeomorphisms of arc images

lemma *homeomorphism_arc*:

```

  fixes g :: real ⇒ 'a::t2_space
  assumes arc g
  obtains h where homeomorphism {0..1} (path_image g) g h
using assms by (force simp: arc_def homeomorphism_compact path_def path_image_def)

```

lemma *homeomorphic_arc_image_interval*:

```

  fixes g :: real ⇒ 'a::t2_space and a::real
  assumes arc g a < b
  shows (path_image g) homeomorphic {a..b}
proof -
  have (path_image g) homeomorphic {0..1::real}
    by (meson assms(1) homeomorphic_def homeomorphic_sym homeomorphism_arc)
  also have ... homeomorphic {a..b}
    using assms by (force intro: homeomorphic_closed_intervals_real)
  finally show ?thesis .
qed

```

lemma *homeomorphic_arc_images*:

```

  fixes g :: real ⇒ 'a::t2_space and h :: real ⇒ 'b::t2_space
  assumes arc g arc h

```

```

    shows (path_image g) homeomorphic (path_image h)
  proof -
    have (path_image g) homeomorphic {0..1::real}
      by (meson assms homeomorphic_def homeomorphic_sym homeomorphism_arc)
    also have ... homeomorphic (path_image h)
      by (meson assms homeomorphic_def homeomorphism_arc)
    finally show ?thesis .
  qed

end

```

```

theory Equivalence_Lebesgue_Henstock_Integration
  imports
    Lebesgue_Measure
    Henstock_Kurzweil_Integration
    Complete_Measure
    Set_Integral
    Homeomorphism
    Cartesian_Euclidean_Space
begin

```

```

lemma LIMSEQ_if_less: ( $\lambda k. \text{if } i < k \text{ then } a \text{ else } b$ )  $\longrightarrow a$ 
  by (rule_tac k=Suc i in LIMSEQ_offset) auto

```

Note that the rhs is an implication. This lemma plays a specific role in one proof.

```

lemma le_left_mono:  $x \leq y \implies y \leq a \longrightarrow x \leq (a::'a::preorder)$ 
  by (auto intro: order_trans)

```

```

lemma ball_trans:
  assumes  $y \in \text{ball } z \text{ } q \text{ } r + q \leq s$  shows  $\text{ball } y \text{ } r \subseteq \text{ball } z \text{ } s$ 
  using assms by metric

```

```

lemma has_integral_implies_lebesgue_measurable_cbox:
  fixes  $f :: 'a :: euclidean\_space \Rightarrow \text{real}$ 
  assumes  $f: (f \text{ has\_integral } I) (\text{cbox } x \text{ } y)$ 
  shows  $f \in \text{lebesgue\_on } (\text{cbox } x \text{ } y) \rightarrow_M \text{borel}$ 
proof (rule cld_measure.borel_measurable_cld)
  let ?L = lebesgue_on (cbox x y)
  let ? $\mu$  = emeasure ?L
  let ? $\mu'$  = outer_measure_of ?L
  interpret L: finite_measure ?L
proof
  show ? $\mu$  (space ?L)  $\neq \infty$ 
    by (simp add: emeasure_restrict_space space_restrict_space emeasure_lborel_cbox_eq)
  qed

show cld_measure ?L

```

```

proof
  fix B A assume B ⊆ A A ∈ null_sets ?L
  then show B ∈ sets ?L
    using null_sets_completion_subset[OF ⟨B ⊆ A⟩, of lborel]
    by (auto simp add: null_sets_restrict_space sets_restrict_space_iff intro: )
  next
    fix A assume A ⊆ space ?L ∧ B. B ∈ sets ?L ⇒ ?μ B < ∞ ⇒ A ∩ B ∈
sets ?L
    from this(1) this(2)[of space ?L] show A ∈ sets ?L
      by (auto simp: Int_absorb2 less_top[symmetric])
  qed auto
  then interpret cld_measure ?L
  .

  have content_eq_L: A ∈ sets borel ⇒ A ⊆ cbox x y ⇒ content A = measure
?L A for A
    by (subst measure_restrict_space) (auto simp: measure_def)

  fix E and a b :: real assume E ∈ sets ?L a < b 0 < ?μ E ?μ E < ∞
  then obtain M :: real where ?μ E = M 0 < M
    by (cases ?μ E) auto
  define e where e = M / (4 + 2 / (b - a))
  from ⟨a < b⟩ ⟨0 < M⟩ have 0 < e
    by (auto intro!: divide_pos_pos simp: field_simps e_def)

  have e < M / (3 + 2 / (b - a))
    using ⟨a < b⟩ ⟨0 < M⟩
    unfolding e_def by (intro divide_strict_left_mono add_strict_right_mono
mult_pos_pos) (auto simp: field_simps)
  then have 2 * e < (b - a) * (M - e * 3)
    using ⟨0 < M⟩ ⟨0 < e⟩ ⟨a < b⟩ by (simp add: field_simps)

  have e_less_M: e < M / 1
    unfolding e_def using ⟨a < b⟩ ⟨0 < M⟩ by (intro divide_strict_left_mono)
(auto simp: field_simps)

  obtain d
    where gauge d
      and integral_f: ∀ p. p tagged_division_of cbox x y ∧ d fine p ⟶
norm ((∑ (x,k) ∈ p. content k *R f x) - I) < e
    using ⟨0 < e⟩ f unfolding has_integral by auto

  define C where C X m = X ∩ {x. ball x (1/Suc m) ⊆ d x} for X m
  have incseq (C X) for X
    unfolding C_def [abs_def]
    by (intro monoI Collect_mono conj_mono imp_refl le_left_mono subset_ball
divide_left_mono Int_mono) auto

  { fix X assume X ⊆ space ?L and eq: ?μ' X = ?μ E

```

```

have (SUP m. outer_measure_of ?L (C X m)) = outer_measure_of ?L (⋃ m.
C X m)
  using ⟨X ⊆ space ?L⟩ by (intro SUP_outer_measure_of_incseq ⟨incseq (C
X)⟩) (auto simp: C_def)
  also have (⋃ m. C X m) = X
  proof -
    { fix x
      obtain e where 0 < e ball x e ⊆ d x
      using gaugeD[OF ⟨gauge d⟩, of x] unfolding open_contains_ball by auto
      moreover
      obtain n where 1 / (1 + real n) < e
      using reals_Archimedean[OF ⟨0 < e⟩] by (auto simp: inverse_eq_divide)
      then have ball x (1 / (1 + real n)) ⊆ ball x e
      by (intro subset_ball) auto
      ultimately have ∃ n. ball x (1 / (1 + real n)) ⊆ d x
      by blast }
    then show ?thesis
    by (auto simp: C_def)
  qed
finally have (SUP m. outer_measure_of ?L (C X m)) = ?μ E
  using eq by auto
  also have ... > M - e
  using ⟨0 < M⟩ ⟨?μ E = M⟩ ⟨0 < e⟩ by (auto intro!: ennreal_lessI)
  finally have ∃ m. M - e < outer_measure_of ?L (C X m)
  unfolding less_SUP_iff by auto }
note C = this

let ?E = {x ∈ E. f x ≤ a} and ?F = {x ∈ E. b ≤ f x}

have ¬ (?μ' ?E = ?μ E ∧ ?μ' ?F = ?μ E)
proof
  assume eq: ?μ' ?E = ?μ E ∧ ?μ' ?F = ?μ E
  with C[of ?E] C[of ?F] ⟨E ∈ sets ?L⟩[THEN sets.sets_into_space] obtain ma
mb
  where M - e < outer_measure_of ?L (C ?E ma) M - e < outer_measure_of
?L (C ?F mb)
  by auto
  moreover define m where m = max ma mb
  ultimately have M_minus_e: M - e < outer_measure_of ?L (C ?E m) M
- e < outer_measure_of ?L (C ?F m)
  using
    incseqD[OF ⟨incseq (C ?E)⟩, of ma m, THEN outer_measure_of_mono]
    incseqD[OF ⟨incseq (C ?F)⟩, of mb m, THEN outer_measure_of_mono]
  by (auto intro: less_le_trans)
  define d' where d' x = d x ∩ ball x (1 / (3 * Suc m)) for x
  have gauge d'
  unfolding d'_def by (intro gauge_Int ⟨gauge d⟩ gauge_ball) auto
  then obtain p where p: p tagged_division_of cbox x y d' fine p
  by (rule fine_division_exists)

```

```

then have d fine p
  unfolding d'_def[abs_def] fine_def by auto

define s where s = {(x::'a, k). k ∩ (C ?E m) ≠ {} ∧ k ∩ (C ?F m) ≠ {}}
define T where T E k = (SOME x. x ∈ k ∩ C E m) for E k
let ?A = (λ(x, k). (T ?E k, k)) ' (p ∩ s) ∪ (p - s)
let ?B = (λ(x, k). (T ?F k, k)) ' (p ∩ s) ∪ (p - s)

{ fix X assume X_eq: X = ?E ∨ X = ?F
  let ?T = (λ(x, k). (T X k, k))
  let ?p = ?T ' (p ∩ s) ∪ (p - s)

  have in_s: (x, k) ∈ s ⟹ T X k ∈ k ∩ C X m for x k
    using someI_ex[of λx. x ∈ k ∩ C X m] X_eq unfolding ex_in_conv by
(auto simp: T_def s_def)

  { fix x k assume (x, k) ∈ p (x, k) ∈ s
    have k: k ⊆ ball x (1 / (3 * Suc m))
      using ⟨d' fine p⟩[THEN fineD, OF ⟨(x, k) ∈ p⟩] by (auto simp: d'_def)
    then have x ∈ ball (T X k) (1 / (3 * Suc m))
      using in_s[OF ⟨(x, k) ∈ s⟩] by (auto simp: C_def subset_eq dist_commute)
    then have ball x (1 / (3 * Suc m)) ⊆ ball (T X k) (1 / Suc m)
      by (rule ball_trans) (auto simp: field_split_simps)
    with k in_s[OF ⟨(x, k) ∈ s⟩] have k ⊆ d (T X k)
      by (auto simp: C_def) }
  then have d fine ?p
    using ⟨d fine p⟩ by (auto intro!: fineI)
  moreover
  have ?p tagged_division_of cbox x y
  proof (rule tagged_division_ofI)
    show finite ?p
      using p(1) by auto
  next
    fix z k assume *: (z, k) ∈ ?p
    then consider (z, k) ∈ p (z, k) ∉ s
      | x' where (x', k) ∈ p (x', k) ∈ s z = T X k
      by (auto simp: T_def)
    then have z ∈ k ∧ k ⊆ cbox x y ∧ (∃ a b. k = cbox a b)
      using p(1) by cases (auto dest: in_s)
    then show z ∈ k k ⊆ cbox x y ∃ a b. k = cbox a b
      by auto
  next
    fix z k z' k' assume (z, k) ∈ ?p (z', k') ∈ ?p (z, k) ≠ (z', k')
    with tagged_division_ofD(5)[OF p(1), of _ k _ k']
    show interior k ∩ interior k' = {}
      by (auto simp: T_def dest: in_s)
  next
    have {k. ∃ x. (x, k) ∈ ?p} = {k. ∃ x. (x, k) ∈ p}
      by (auto simp: T_def image_iff Bex_def)
  }

```

```

    then show  $\bigcup \{k. \exists x. (x, k) \in ?p\} = \text{cbox } x \ y$ 
      using  $p(1)$  by auto
  qed
  ultimately have  $I: \text{norm } ((\sum (x,k) \in ?p. \text{content } k *_R f \ x) - I) < e$ 
    using  $\text{integral\_f}$  by auto

  have  $(\sum (x,k) \in ?p. \text{content } k *_R f \ x) =$ 
     $(\sum (x,k) \in ?T \text{ ' } (p \cap s). \text{content } k *_R f \ x) + (\sum (x,k) \in p - s. \text{content } k$ 
 $*_R f \ x)$ 
    using  $p(1)[\text{THEN tagged\_division\_ofD}(1)]$ 
    by  $(\text{safe intro!}: \text{sum.union\_inter\_neutral}) (\text{auto simp: } s\_def \ T\_def)$ 
    also have  $(\sum (x,k) \in ?T \text{ ' } (p \cap s). \text{content } k *_R f \ x) = (\sum (x,k) \in p \cap s.$ 
 $\text{content } k *_R f \ (T \ X \ k))$ 
    proof  $(\text{subst sum.reindex\_nontrivial}, \text{safe})$ 
      fix  $x1 \ x2 \ k$  assume  $1: (x1, k) \in p \ (x1, k) \in s$  and  $2: (x2, k) \in p \ (x2, k)$ 
 $\in s$ 
      and  $\text{eq: content } k *_R f \ (T \ X \ k) \neq 0$ 
      with  $\text{tagged\_division\_ofD}(5)[\text{OF } p(1), \text{of } x1 \ k \ x2 \ k] \text{ tagged\_division\_ofD}(4)[\text{OF}$ 
 $p(1), \text{of } x1 \ k]$ 
      show  $x1 = x2$ 
      by  $(\text{auto simp: content\_eq\_0\_interior})$ 
    qed  $(\text{use } p \text{ in } \langle \text{auto intro!}: \text{sum.cong} \rangle)$ 
    finally have  $\text{eq: } (\sum (x,k) \in ?p. \text{content } k *_R f \ x) =$ 
 $(\sum (x,k) \in p \cap s. \text{content } k *_R f \ (T \ X \ k)) + (\sum (x,k) \in p - s. \text{content } k$ 
 $*_R f \ x) .$ 

  have  $\text{in\_T}: (x, k) \in s \implies T \ X \ k \in X$  for  $x \ k$ 
    using  $\text{in\_s}[\text{of } x \ k]$  by  $(\text{auto simp: } C\_def)$ 

  note  $I \text{ eq in\_T} \}$ 
  note  $\text{parts} = \text{this}$ 

  have  $p\_in\_L: (x, k) \in p \implies k \in \text{sets } ?L$  for  $x \ k$ 
    using  $\text{tagged\_division\_ofD}(3, 4)[\text{OF } p(1), \text{of } x \ k]$  by  $(\text{auto simp: sets\_restrict\_space})$ 

  have  $[\text{simp}]: \text{finite } p$ 
    using  $\text{tagged\_division\_ofD}(1)[\text{OF } p(1)]$  .

  have  $(M - 3 * e) * (b - a) \leq (\sum (x,k) \in p \cap s. \text{content } k) * (b - a)$ 
  proof  $(\text{intro mult\_right\_mono})$ 
    have  $\text{fin: } ?\mu \ (E \cap \bigcup \{k \in \text{snd}' p. k \cap C \ X \ m = \{\}\}) < \infty$  for  $X$ 
      using  $\langle ?\mu \ E < \infty \rangle$  by  $(\text{rule le\_less\_trans}[\text{rotated}]) (\text{auto intro!}: \text{emea-}$ 
 $\text{sure\_mono } \langle E \in \text{sets } ?L \rangle)$ 
    have  $\text{sets: } (E \cap \bigcup \{k \in \text{snd}' p. k \cap C \ X \ m = \{\}\}) \in \text{sets } ?L$  for  $X$ 
      using  $\text{tagged\_division\_ofD}(1)[\text{OF } p(1)]$  by  $(\text{intro sets.Diff } \langle E \in \text{sets } ?L \rangle$ 
 $\text{sets.finite\_Union sets.Int}) (\text{auto intro: } p\_in\_L)$ 
    { fix  $X$  assume  $X \subseteq E \ M - e < ?\mu' (C \ X \ m)$ 
      have  $M - e \leq ?\mu' (C \ X \ m)$ 
      by  $(\text{rule less\_imp\_le}) \text{fact}$ 

```

```

also have ... ≤ ?μ' (E - (E ∩ ⋃ {k ∈ snd'p. k ∩ C X m = {}}))
proof (intro outer_measure_of_mono subsetI)
  fix v assume v ∈ C X m
  then have v ∈ cbox x y v ∈ E
    using ⟨E ⊆ space ?L⟩ ⟨X ⊆ E⟩ by (auto simp: space_restrict_space
C_def)
  then obtain z k where (z, k) ∈ p v ∈ k
    using tagged_division_ofD(6)[OF p(1), symmetric] by auto
  then show v ∈ E - E ∩ (⋃ {k ∈ snd'p. k ∩ C X m = {}})
    using ⟨v ∈ C X m⟩ ⟨v ∈ E⟩ by auto
qed
also have ... = ?μ E - ?μ (E ∩ ⋃ {k ∈ snd'p. k ∩ C X m = {}})
  using ⟨E ∈ sets ?L⟩ fin[of X] sets[of X] by (auto intro!: emeasure_Diff)
finally have ?μ (E ∩ ⋃ {k ∈ snd'p. k ∩ C X m = {}}) ≤ e
  using ⟨0 < e⟩ e_less_M
  by (cases ?μ (E ∩ ⋃ {k ∈ snd'p. k ∩ C X m = {}})) (auto simp add: ⟨?μ
E = M⟩ ennreal_minus ennreal_le_iff2)
  note this }
note upper_bound = this

have ?μ (E ∩ ⋃ (snd'(p - s))) =
  ?μ ((E ∩ ⋃ {k ∈ snd'p. k ∩ C ?E m = {}}) ∪ (E ∩ ⋃ {k ∈ snd'p. k ∩ C ?F
m = {}}))
  by (intro arg_cong[where f=?μ]) (auto simp: s_def image_def Bex_def)
also have ... ≤ ?μ (E ∩ ⋃ {k ∈ snd'p. k ∩ C ?E m = {}}) + ?μ (E ∩
⋃ {k ∈ snd'p. k ∩ C ?F m = {}})
  using sets[of ?E] sets[of ?F] M_minus_e by (intro emeasure_subadditive)
auto
also have ... ≤ e + ennreal e
  using upper_bound[of ?E] upper_bound[of ?F] M_minus_e by (intro
add_mono) auto
finally have ?μ E - 2*e ≤ ?μ (E - (E ∩ ⋃ (snd'(p - s))))
  using ⟨0 < e⟩ ⟨E ∈ sets ?L⟩ tagged_division_ofD(1)[OF p(1)]
  by (subst emeasure_Diff)
  (auto simp: top_unique simp flip: ennreal_plus
intro!: sets.Int sets.finite_UN ennreal_mono_minus intro: p_in_L)
also have ... ≤ ?μ (⋃ x ∈ p ∩ s. snd x)
proof (safe intro!: emeasure_mono subsetI)
  fix v assume v ∈ E and not: v ∉ (⋃ x ∈ p ∩ s. snd x)
  then have v ∈ cbox x y
    using ⟨E ⊆ space ?L⟩ by (auto simp: space_restrict_space)
  then obtain z k where (z, k) ∈ p v ∈ k
    using tagged_division_ofD(6)[OF p(1), symmetric] by auto
  with not show v ∈ ⋃ (snd '(p - s))
    by (auto intro!: bexI[of _ (z, k)] elim: ballE[of _ _ (z, k)])
qed (auto intro!: sets.Int sets.finite_UN ennreal_mono_minus intro: p_in_L)
also have ... = measure ?L (⋃ x ∈ p ∩ s. snd x)
  by (auto intro!: emeasure_eq_ennreal_measure)
finally have M - 2 * e ≤ measure ?L (⋃ x ∈ p ∩ s. snd x)

```

```

    unfolding ⟨?μ E = M⟩ using ⟨0 < e⟩ by (simp add: ennreal_minus)
  also have measure ?L (⋃ x∈p ∩ s. snd x) = content (⋃ x∈p ∩ s. snd x)
    using tagged_division_ofD(1,3,4) [OF p(1)]
    by (intro content_eq_L[symmetric])
    (fastforce intro!: sets.finite_UN UN_least del: subsetI)+
  also have content (⋃ x∈p ∩ s. snd x) ≤ (∑ k∈p ∩ s. content (snd k))
  using p(1) by (auto simp: emeasure_lborel_cbox_eq intro!: measure_subadditive_finite
    dest!: p(1)[THEN tagged_division_ofD(4)])
  finally show M - 3 * e ≤ (∑ (x, y)∈p ∩ s. content y)
    using ⟨0 < e⟩ by (simp add: split_beta)
qed (use ⟨a < b⟩ in auto)
also have ... = (∑ (x,k) ∈ p ∩ s. content k * (b - a))
  by (simp add: sum_distrib_right split_beta')
also have ... ≤ (∑ (x,k) ∈ p ∩ s. content k * (f (T ?F k) - f (T ?E k)))
  using parts(3) by (auto intro!: sum_mono mult_left_mono diff_mono)
also have ... = (∑ (x,k) ∈ p ∩ s. content k * f (T ?F k)) - (∑ (x,k) ∈ p ∩
s. content k * f (T ?E k))
  by (auto intro!: sum.cong simp: field_simps sum_subtractf[symmetric])
also have ... = (∑ (x,k) ∈ ?B. content k *R f x) - (∑ (x,k) ∈ ?A. content k
*_R f x)
  by (subst (1 2) parts) auto
also have ... ≤ norm ((∑ (x,k) ∈ ?B. content k *R f x) - (∑ (x,k) ∈ ?A.
content k *R f x))
  by auto
also have ... ≤ e + e
  using parts(1)[of ?E] parts(1)[of ?F] by (intro norm_diff_triangle_le[of _
I]) auto
finally show False
  using ⟨2 * e < (b - a) * (M - e * 3)⟩ by (auto simp: field_simps)
qed
moreover have ?μ' ?E ≤ ?μ E ?μ' ?F ≤ ?μ E
  unfolding outer_measure_of_eq[OF ⟨E ∈ sets ?L⟩, symmetric] by (auto intro!:
outer_measure_of_mono)
ultimately show min (?μ' ?E) (?μ' ?F) < ?μ E
  unfolding min_less_iff_disj by (auto simp: less_le)
qed

```

lemma *has_integral_implies_lebesgue_measurable_real*:

fixes *f* :: 'a :: euclidean_space ⇒ real

assumes *f*: (*f* has_integral *I*) Ω

shows (λ*x*. *f* *x* * indicator Ω *x*) ∈ lebesgue →_M borel

proof –

define *B* :: nat ⇒ 'a set **where** *B* *n* = cbox (– real *n* *__R One) (real *n* *__R One)

for *n*

show (λ*x*. *f* *x* * indicator Ω *x*) ∈ lebesgue →_M borel

proof (rule measurable_piecewise_restrict)

have (⋃ *n*. box (– real *n* *__R One) (real *n* *__R One)) ⊆ ⋃ (*B* ‘ UNIV)

unfolding *B*_def **by** (intro UN_mono box_subset_cbox order_refl)

then show countable (range *B*) space lebesgue ⊆ ⋃ (*B* ‘ UNIV)


```

    by (auto simp: B_def UN_box_eq_UNIV)
  next
    fix  $\Omega'$  assume  $\Omega' \in \text{range } B$ 
    then obtain  $n$  where  $\Omega': \Omega' = B\ n$  by auto
    then show  $\Omega' \cap \text{space lebesgue} \in \text{sets lebesgue}$ 
      by (auto simp: B_def)

  have  $f \text{ integrable\_on } \Omega$ 
    using  $f$  by auto
  then have  $(\lambda x. f\ x * \text{indicator } \Omega\ x) \text{ integrable\_on } \Omega$ 
    by (auto simp: integrable_on_def cong: has_integral_cong)
  then have  $(\lambda x. f\ x * \text{indicator } \Omega\ x) \text{ integrable\_on } (\Omega \cup B\ n)$ 
    by (rule integrable_on_superset) auto
  then have  $(\lambda x. f\ x * \text{indicator } \Omega\ x) \text{ integrable\_on } B\ n$ 
    unfolding B_def by (rule integrable_on_subcbox) auto
  then show  $(\lambda x. f\ x * \text{indicator } \Omega\ x) \in \text{lebesgue\_on } \Omega' \rightarrow_M \text{borel}$ 
    unfolding B_def  $\Omega'$  by (auto intro: has_integral_implies_lebesgue_measurable_cbox)
simp: integrable_on_def)
qed
qed

```

```

lemma has_integral_implies_lebesgue_measurable:
  fixes  $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$ 
  assumes  $f: (f \text{ has\_integral } I) \Omega$ 
  shows  $(\lambda x. \text{indicator } \Omega\ x *_{\mathbb{R}} f\ x) \in \text{lebesgue} \rightarrow_M \text{borel}$ 
proof (intro borel_measurable_euclidean_space[where 'c='b, THEN iffD2] ballI)
  fix  $i :: 'b$  assume  $i \in \text{Basis}$ 
  have  $(\lambda x. (f\ x \cdot i) * \text{indicator } \Omega\ x) \in \text{borel\_measurable } (\text{completion } \text{lborel})$ 
    using has_integral_linear[OF  $f$  bounded_linear_inner_left, of  $i$ ]
    by (intro has_integral_implies_lebesgue_measurable_real) (auto simp: comp_def)
  then show  $(\lambda x. \text{indicator } \Omega\ x *_{\mathbb{R}} f\ x \cdot i) \in \text{borel\_measurable } (\text{completion } \text{lborel})$ 
    by (simp add: ac_simps)
qed

```

6.19.7 Equivalence Lebesgue integral on *lborel* and HK-integral

```

lemma has_integral_measure_lborel:
  fixes  $A :: 'a :: \text{euclidean\_space}$  set
  assumes  $A[\text{measurable}]: A \in \text{sets borel}$  and  $\text{finite}: \text{emeasure lborel } A < \infty$ 
  shows  $((\lambda x. 1) \text{ has\_integral measure lborel } A) \ A$ 
proof -
  { fix  $l\ u :: 'a$ 
    have  $((\lambda x. 1) \text{ has\_integral measure lborel } (\text{box } l\ u)) (\text{box } l\ u)$ 
    proof cases
      assume  $\forall b \in \text{Basis}. l \cdot b \leq u \cdot b$ 
      then show ?thesis
        using has_integral_const[of  $1 :: \text{real } l\ u$ ]
        by (simp flip: has_integral_restrict[OF box_subset_cbox] add: has_integral_spike_interior)
    next

```

```

    assume  $\neg (\forall b \in \text{Basis}. l \cdot b \leq u \cdot b)$ 
    then have  $\text{box } l \ u = \{\}$ 
      unfolding  $\text{box\_eq\_empty}$  by (auto simp: not_le intro: less_imp_le)
    then show ?thesis
      by simp
  qed }
note  $\text{has\_integral\_box} = \text{this}$ 

{ fix  $a \ b :: 'a$  let  $?M = \lambda A. \text{measure } \text{lborel } (A \cap \text{box } a \ b)$ 
  have  $\text{Int\_stable } (\text{range } (\lambda(a, b). \text{box } a \ b))$ 
    by (auto simp:  $\text{Int\_stable\_def}$   $\text{box\_Int\_box}$ )
  moreover have  $(\text{range } (\lambda(a, b). \text{box } a \ b)) \subseteq \text{Pow } \text{UNIV}$ 
    by auto
  moreover have  $A \in \text{sigma\_sets } \text{UNIV } (\text{range } (\lambda(a, b). \text{box } a \ b))$ 
    using  $A$  unfolding  $\text{borel\_eq\_box}$  by simp
  ultimately have  $((\lambda x. 1) \text{ has\_integral } ?M \ A) \ (A \cap \text{box } a \ b)$ 
    proof (induction rule:  $\text{sigma\_sets\_induct\_disjoint}$ )
      case (basic  $A$ ) then show ?case
        by (auto simp:  $\text{box\_Int\_box}$   $\text{has\_integral\_box}$ )
    next
      case empty then show ?case
        by simp
    next
      case (compl  $A$ )
      then have  $[\text{measurable}]: A \in \text{sets borel}$ 
        by (simp add:  $\text{borel\_eq\_box}$ )

      have  $((\lambda x. 1) \text{ has\_integral } ?M \ (\text{box } a \ b)) \ (\text{box } a \ b)$ 
        by (simp add:  $\text{has\_integral\_box}$ )
      moreover have  $((\lambda x. \text{if } x \in A \cap \text{box } a \ b \text{ then } 1 \text{ else } 0) \text{ has\_integral } ?M \ A)$ 
        (box  $a \ b$ )
        by (subst  $\text{has\_integral\_restrict}$ ) (auto intro: compl)
      ultimately have  $((\lambda x. 1 - (\text{if } x \in A \cap \text{box } a \ b \text{ then } 1 \text{ else } 0)) \text{ has\_integral } ?M \ (\text{box } a \ b) - ?M \ A) \ (\text{box } a \ b)$ 
        by (rule  $\text{has\_integral\_diff}$ )
      then have  $((\lambda x. (\text{if } x \in (\text{UNIV} - A) \cap \text{box } a \ b \text{ then } 1 \text{ else } 0)) \text{ has\_integral } ?M \ (\text{box } a \ b) - ?M \ A) \ (\text{box } a \ b)$ 
        by (rule  $\text{has\_integral\_cong}$  [THEN  $\text{iffD1}$ , rotated 1]) auto
      then have  $((\lambda x. 1) \text{ has\_integral } ?M \ (\text{box } a \ b) - ?M \ A) \ ((\text{UNIV} - A) \cap \text{box } a \ b)$ 
        by (subst (asm)  $\text{has\_integral\_restrict}$ ) auto
      also have  $?M \ (\text{box } a \ b) - ?M \ A = ?M \ (\text{UNIV} - A)$ 
        by (subst  $\text{measure\_Diff}$  [symmetric]) (auto simp:  $\text{emeasure\_lborel\_box\_eq\_Diff\_Int\_distrib2}$ )
      finally show ?case .
    next
      case (union  $F$ )
      then have  $[\text{measurable}]: \bigwedge i. F \ i \in \text{sets borel}$ 
        by (simp add:  $\text{borel\_eq\_box}$   $\text{subset\_eq}$ )

```

```

    have (( $\lambda x.$  if  $x \in \bigcup (F \text{ ' UNIV}) \cap \text{box } a \text{ } b$  then 1 else 0) has_integral ?M
      ( $\bigcup i.$  F i)) (box a b)
  proof (rule has_integral_monotone_convergence_increasing)
    let ?f =  $\lambda k x.$   $\sum i < k.$  if  $x \in F i \cap \text{box } a \text{ } b$  then 1 else 0 :: real
    show  $\bigwedge k.$  (?f k has_integral ( $\sum i < k.$  ?M (F i))) (box a b)
      using union.IH by (auto intro!: has_integral_sum simp del: Int_iff)
    show  $\bigwedge k x.$  ?f k x  $\leq$  ?f (Suc k) x
      by (intro sum_mono2) auto
    from union(1) have *:  $\bigwedge x i j.$   $x \in F i \implies x \in F j \longleftrightarrow j = i$ 
      by (auto simp add: disjoint_family_on_def)
    show ( $\lambda k.$  ?f k x)  $\longrightarrow$  (if  $x \in \bigcup (F \text{ ' UNIV}) \cap \text{box } a \text{ } b$  then 1 else 0) for
      x
    by (auto simp: * sum.If_cases Iio_Int_singleton if_distrib LIMSEQ_if_less
      cong: if_cong)
    have *: emeasure lborel (( $\bigcup x.$  F x)  $\cap$  box a b)  $\leq$  emeasure lborel (box a b)
      by (intro emeasure_mono) auto

    with union(1) show ( $\lambda k.$   $\sum i < k.$  ?M (F i))  $\longrightarrow$  ?M ( $\bigcup i.$  F i)
      unfolding sums_def[symmetric] UN_extend_simps
      by (intro measure_UNION) (auto simp: disjoint_family_on_def emeasure_lborel_box_eq top_unique)
    qed
    then show ?case
      by (subst (asm) has_integral_restrict) auto
    qed }
  note * = this

  show ?thesis
  proof (rule has_integral_monotone_convergence_increasing)
    let ?B =  $\lambda n::\text{nat}.$  box ( $-$  real n *R One) (real n *R One) :: 'a set
    let ?f =  $\lambda n::\text{nat}.$   $\lambda x.$  if  $x \in A \cap ?B n$  then 1 else 0 :: real
    let ?M =  $\lambda n.$  measure lborel (A  $\cap$  ?B n)

    show  $\bigwedge n::\text{nat}.$  (?f n has_integral ?M n) A
      using * by (subst has_integral_restrict) simp_all
    show  $\bigwedge k x.$  ?f k x  $\leq$  ?f (Suc k) x
      by (auto simp: box_def)
    { fix x assume x  $\in$  A
      moreover have ( $\lambda k.$  indicator (A  $\cap$  ?B k) x :: real)  $\longrightarrow$  indicator ( $\bigcup k::\text{nat}.$ 
        A  $\cap$  ?B k) x
        by (intro LIMSEQ_indicator_incseq) (auto simp: incseq_def box_def)
      ultimately show ( $\lambda k.$  if  $x \in A \cap ?B k$  then 1 else 0 :: real)  $\longrightarrow$  1
        by (simp add: indicator_def of_bool_def UN_box_eq UNIV) }

    have ( $\lambda n.$  emeasure lborel (A  $\cap$  ?B n))  $\longrightarrow$  emeasure lborel ( $\bigcup n::\text{nat}.$  A  $\cap$ 
      ?B n)
      by (intro Lim_emeasure_incseq) (auto simp: incseq_def box_def)
    also have ( $\lambda n.$  emeasure lborel (A  $\cap$  ?B n)) = ( $\lambda n.$  measure lborel (A  $\cap$  ?B n))
  proof (intro ext emeasure_eq ennreal_measure)

```

```

    fix n have emeasure lborel (A ∩ ?B n) ≤ emeasure lborel (?B n)
      by (intro emeasure_mono) auto
    then show emeasure lborel (A ∩ ?B n) ≠ top
      by (auto simp: top_unique)
    qed
  finally show (λn. measure lborel (A ∩ ?B n)) → measure lborel A
    using emeasure_eq_enreal_measure[of lborel A] finite
    by (simp add: UN_box_eq_UNIV less_top)
  qed
qed

lemma nn_integral_has_integral:
  fixes f::'a::euclidean_space ⇒ real
  assumes f: f ∈ borel_measurable borel ∧ x. 0 ≤ f x (∫+ x. f x ∂lborel) = ennreal
  r 0 ≤ r
  shows (f has_integral r) UNIV
using f proof (induct f arbitrary: r rule: borel_measurable_induct_real)
  case (set A)
  then have ((λx. 1) has_integral measure lborel A) A
    by (intro has_integral_measure_lborel) (auto simp: ennreal_indicator)
  with set show ?case
    by (simp add: ennreal_indicator measure_def) (simp add: indicator_def of_bool_def)
next
  case (mult g c)
  then have ennreal c * (∫+ x. g x ∂lborel) = ennreal r
    by (subst nn_integral_cmult[symmetric]) (auto simp: ennreal_mult)
  with ⟨0 ≤ r⟩ ⟨0 ≤ c⟩
  obtain r' where (c = 0 ∧ r = 0) ∨ (0 ≤ r' ∧ (∫+ x. ennreal (g x) ∂lborel) =
    ennreal r' ∧ r = c * r')
    by (cases ∫+ x. ennreal (g x) ∂lborel rule: ennreal_cases)
      (auto split: if_split_asm simp: ennreal_mult_top ennreal_mult[symmetric])
  with mult show ?case
    by (auto intro!: has_integral_cmult_real)
next
  case (add g h)
  then have (∫+ x. h x + g x ∂lborel) = (∫+ x. h x ∂lborel) + (∫+ x. g x ∂lborel)
    by (simp add: nn_integral_add)
  with add obtain a b where 0 ≤ a 0 ≤ b (∫+ x. h x ∂lborel) = ennreal a (∫+
    x. g x ∂lborel) = ennreal b r = a + b
    by (cases ∫+ x. h x ∂lborel ∫+ x. g x ∂lborel rule: ennreal2_cases)
      (auto simp: add_top nn_integral_add top_add simp flip: ennreal_plus)
  with add show ?case
    by (auto intro!: has_integral_add)
next
  case (seq U)
  note seq(1)[measurable] and f[measurable]

  have U_le_f: U i x ≤ f x for i x
    by (metis (no_types) LIMSEQ_le_const UNIV_I incseq_def le_fun_def seq.hyps(4))

```

seq.hyps(5) space_borel)

```

{ fix i
  have ( $\int^+ x. U\ i\ x\ \partial\text{lborel}$ )  $\leq$  ( $\int^+ x. f\ x\ \partial\text{lborel}$ )
    using seq(2) f(2) U_le_f by (intro nn_integral_mono) simp
  then obtain p where ( $\int^+ x. U\ i\ x\ \partial\text{lborel}$ ) = ennreal p  $p \leq r$   $0 \leq p$ 
    using seq(6) ‹ $0 \leq r$ › by (cases  $\int^+ x. U\ i\ x\ \partial\text{lborel}$  rule: ennreal_cases) (auto
simp: top_unique)
  moreover note seq
  ultimately have  $\exists p. (\int^+ x. U\ i\ x\ \partial\text{lborel}) = \text{ennreal } p \wedge 0 \leq p \wedge p \leq r \wedge (U$ 
i has_integral p) UNIV
    by auto }
  then obtain p where p:  $\bigwedge i. (\int^+ x. \text{ennreal } (U\ i\ x)\ \partial\text{lborel}) = \text{ennreal } (p\ i)$ 
    and bnd:  $\bigwedge i. p\ i \leq r \wedge i. 0 \leq p\ i$ 
    and U_int:  $\bigwedge i. (U\ i\ \text{has\_integral } (p\ i))\ \text{UNIV}$  by metis

  have int_eq:  $\bigwedge i. \text{integral UNIV } (U\ i) = p\ i$  using U_int by (rule integral_unique)

  have *:  $f\ \text{integrable\_on UNIV} \wedge (\lambda k. \text{integral UNIV } (U\ k)) \longrightarrow \text{integral UNIV } f$ 
  proof (rule monotone_convergence_increasing)
    show  $\bigwedge k. U\ k\ \text{integrable\_on UNIV}$  using U_int by auto
    show  $\bigwedge k\ x. x \in \text{UNIV} \implies U\ k\ x \leq U\ (\text{Suc } k)\ x$  using ‹incseq U› by (auto
simp: incseq_def le_fun_def)
    then show bounded (range ( $\lambda k. \text{integral UNIV } (U\ k)$ ))
      using bnd int_eq by (auto simp: bounded_real intro!: exI[of _ r])
    show  $\bigwedge x. x \in \text{UNIV} \implies (\lambda k. U\ k\ x) \longrightarrow f\ x$ 
      using seq by auto
  qed
  moreover have ( $\lambda i. (\int^+ x. U\ i\ x\ \partial\text{lborel})$ )  $\longrightarrow (\int^+ x. f\ x\ \partial\text{lborel})$ 
    using seq f(2) U_le_f by (intro nn_integral_dominated_convergence[where
w=f]) auto
  ultimately have  $\text{integral UNIV } f = r$ 
    by (auto simp add: bnd int_eq p seq intro: LIMSEQ_unique)
  with * show ?case
    by (simp add: has_integral_integral)
qed

```

lemma nn_integral_lborel_eq_integral:

fixes $f::'a::\text{euclidean\_space} \Rightarrow \text{real}$

assumes $f: f \in \text{borel\_measurable borel} \wedge x. 0 \leq f\ x\ (\int^+ x. f\ x\ \partial\text{lborel}) < \infty$

shows $(\int^+ x. f\ x\ \partial\text{lborel}) = \text{integral UNIV } f$

proof -

from f(3) obtain r where $(\int^+ x. f\ x\ \partial\text{lborel}) = \text{ennreal } r$ $0 \leq r$

by (cases $\int^+ x. f\ x\ \partial\text{lborel}$ rule: ennreal_cases) auto

then show ?thesis

using nn_integral_has_integral[OF f(1,2) r] by (simp add: integral_unique)

qed

```

lemma nn_integral_integrable_on:
  fixes f::'a::euclidean_space  $\Rightarrow$  real
  assumes f:  $f \in \text{borel\_measurable borel } \bigwedge x. 0 \leq f\ x \ (\int^+ x. f\ x\ \partial \text{lborel}) < \infty$ 
  shows  $f \text{ integrable\_on UNIV}$ 
proof -
  from f(3) obtain r where  $r: (\int^+ x. f\ x\ \partial \text{lborel}) = \text{ennreal } r \ 0 \leq r$ 
  by (cases  $\int^+ x. f\ x\ \partial \text{lborel}$  rule: ennreal_cases) auto
  then show ?thesis
  by (intro has_integral_integrable[where i=r] nn_integral_has_integral[where
r=r] f)
qed

lemma nn_integral_has_integral_lborel:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes f_borel:  $f \in \text{borel\_measurable borel}$  and nonneg:  $\bigwedge x. 0 \leq f\ x$ 
  assumes I:  $(f \text{ has\_integral } I) \text{ UNIV}$ 
  shows  $\text{integral}^N \text{ lborel } f = I$ 
proof -
  from f_borel have  $(\lambda x. \text{ennreal } (f\ x)) \in \text{borel\_measurable lborel}$  by auto
  from borel_measurable_implies_simple_function_sequence'[OF this]
  obtain F where F:  $\bigwedge i. \text{simple\_function lborel } (F\ i) \text{ incseq } F$ 
   $\bigwedge i\ x. F\ i\ x < \text{top } \bigwedge x. (\text{SUP } i. F\ i\ x) = \text{ennreal } (f\ x)$ 
  by blast
  then have [measurable]:  $\bigwedge i. F\ i \in \text{borel\_measurable lborel}$ 
  by (metis borel_measurable_simple_function)
  let ?B =  $\lambda i::\text{nat}. \text{box } (- (\text{real } i *_{\mathbb{R}} \text{One})) (\text{real } i *_{\mathbb{R}} \text{One}) :: 'a \text{ set}$ 

  have  $0 \leq I$ 
  using I by (rule has_integral_nonneg) (simp add: nonneg)

  have F_le_f:  $\text{enn2real } (F\ i\ x) \leq f\ x$  for i x
  using F(3,4)[where x=x] nonneg SUP_upper[of i UNIV  $\lambda i. F\ i\ x$ ]
  by (cases  $F\ i\ x$  rule: ennreal_cases) auto
  let ?F =  $\lambda i\ x. F\ i\ x * \text{indicator } (?B\ i)\ x$ 
  have  $(\int^+ x. \text{ennreal } (f\ x)\ \partial \text{lborel}) = (\text{SUP } i. \text{integral}^N \text{ lborel } (\lambda x. ?F\ i\ x))$ 
  proof (subst nn_integral_monotone_convergence_SUP[symmetric])
  { fix x
    obtain j where  $j: x \in ?B\ j$ 
    using UN_box_eq_UNIV by auto

    have  $\text{ennreal } (f\ x) = (\text{SUP } i. F\ i\ x)$ 
    using F(4)[of x] nonneg[of x] by (simp add: max_def)
    also have  $\dots = (\text{SUP } i. ?F\ i\ x)$ 
    proof (rule SUP_eq)
      fix i show  $\exists j \in \text{UNIV}. F\ i\ x \leq ?F\ j\ x$ 
      using j F(2)
      by (intro boxI[of _ max i j])
      (auto split: split_max split_indicator simp: incseq_def le_fun_def
box_def)
    }
  }

```

```

    qed (auto intro!: F split: split_indicator)
    finally have ennreal (f x) = (SUP i. ?F i x) . }
  then show ( $\int^+ x. \text{ennreal } (f x) \, \partial \text{lborel}$ ) = ( $\int^+ x. (\text{SUP } i. ?F i x) \, \partial \text{lborel}$ )
    by simp
  qed (insert F, auto simp: incseq_def le_fun_def box_def split: split_indicator)
  also have ...  $\leq$  ennreal I
  proof (rule SUP_least)
    fix i :: nat
    have finite_F: ( $\int^+ x. \text{ennreal } (\text{enn2real } (F i x) * \text{indicator } (?B i) x) \, \partial \text{lborel}$ )
  <  $\infty$ 
    proof (rule nn_integral_bound_simple_function)
      have emeasure_lborel {x  $\in$  space lborel.  $\text{ennreal } (\text{enn2real } (F i x) * \text{indicator } (?B i) x) \neq 0$ }  $\leq$ 
        emeasure_lborel (?B i)
      by (intro emeasure_mono) (auto split: split_indicator)
      then show emeasure_lborel {x  $\in$  space lborel.  $\text{ennreal } (\text{enn2real } (F i x) * \text{indicator } (?B i) x) \neq 0$ } <  $\infty$ 
        by (auto simp: less_top[symmetric] top_unique)
    qed (auto split: split_indicator
      intro!: F simple_function_compose1 [where g=enn2real] simple_function_ennreal)

    have int_F: ( $\lambda x. \text{enn2real } (F i x) * \text{indicator } (?B i) x$ ) integrable_on UNIV
      using F(4) finite_F
    by (intro nn_integral_integrable_on) (auto split: split_indicator simp: enn2real_nonneg)

    have ( $\int^+ x. F i x * \text{indicator } (?B i) x \, \partial \text{lborel}$ ) =
      ( $\int^+ x. \text{ennreal } (\text{enn2real } (F i x) * \text{indicator } (?B i) x) \, \partial \text{lborel}$ )
      using F(3,4)
    by (intro nn_integral_cong) (auto simp: image_iff eq_commute split: split_indicator)
    also have ... = ennreal (integral UNIV ( $\lambda x. \text{enn2real } (F i x) * \text{indicator } (?B i) x$ ))
      using F
    by (intro nn_integral_lborel_eq_integral[OF __ finite_F])
      (auto split: split_indicator intro: enn2real_nonneg)
    also have ...  $\leq$  ennreal I
      by (auto intro!: has_integral_le[OF integrable_integral[OF int_F] I] nonneg
        F_le_f
        simp:  $\langle 0 \leq I \rangle$  split: split_indicator )
    finally show ( $\int^+ x. F i x * \text{indicator } (?B i) x \, \partial \text{lborel}$ )  $\leq$  ennreal I .
  qed
  finally have ( $\int^+ x. \text{ennreal } (f x) \, \partial \text{lborel}$ ) <  $\infty$ 
    by (auto simp: less_top[symmetric] top_unique)
  from nn_integral_lborel_eq_integral[OF assms(1,2) this] I show ?thesis
    by (simp add: integral_unique)
  qed

lemma has_integral_iff_emeasure_lborel:
  fixes A :: 'a::euclidean_space set
  assumes A[measurable]: A  $\in$  sets borel and [simp]:  $0 \leq r$ 

```

```

shows (( $\lambda x. 1$ ) has_integral  $r$ )  $A \longleftrightarrow \text{emeasure } \text{lborel } A = \text{ennreal } r$ 
proof (cases emeasure lborel A =  $\infty$ )
  case emeasure_A: True
    have  $\neg (\lambda x. 1::\text{real}) \text{integrable\_on } A$ 
    proof
      assume int: ( $\lambda x. 1::\text{real}$ ) integrable_on A
      then have (indicator A::'a  $\Rightarrow$  real) integrable_on UNIV
      unfolding indicator_def of_bool_def integrable_restrict_UNIV .
      then obtain  $r$  where ((indicator A::'a  $\Rightarrow$  real) has_integral r) UNIV
      by auto
      from nn_integral_has_integral_lborel[OF __ this] emeasure_A show False
      by (simp add: ennreal_indicator)
    qed
  with emeasure_A show ?thesis
    by auto
next
  case False
  then have (( $\lambda x. 1$ ) has_integral measure lborel A)  $A$ 
  by (simp add: has_integral_measure_lborel less_top)
  with False show ?thesis
  by (auto simp: emeasure_eq_ennreal_measure has_integral_unique)
qed

```

```

lemma ennreal_max_0: ennreal ( $\max 0 x$ ) = ennreal x
by (auto simp: max_def ennreal_neg)

```

```

lemma has_integral_integral_real:
  fixes f::'a::euclidean_space  $\Rightarrow$  real
  assumes f: integrable lborel f
  shows (f has_integral (integralL lborel f)) UNIV
proof -
  from integrableE[OF f] obtain  $r$   $q$ 
  where  $0 \leq r \leq q$ 
  and  $r: (\int^+ x. \text{ennreal } (\max 0 (f x)) \, \partial \text{lborel}) = \text{ennreal } r$ 
  and  $q: (\int^+ x. \text{ennreal } (\max 0 (- f x)) \, \partial \text{lborel}) = \text{ennreal } q$ 
  and  $f: f \in \text{borel\_measurable lborel}$  and eq: integralL lborel f =  $r - q$ 
  unfolding ennreal_max_0 by auto
  then have (( $\lambda x. \max 0 (f x)$ ) has_integral r) UNIV (( $\lambda x. \max 0 (- f x)$ )
has_integral q) UNIV
  using nn_integral_has_integral[OF __  $r$ ] nn_integral_has_integral[OF __
 $q$ ] by auto
  note has_integral_diff[OF this]
  moreover have ( $\lambda x. \max 0 (f x) - \max 0 (- f x)$ ) =  $f$ 
  by auto
  ultimately show ?thesis
  by (simp add: eq)
qed

```

```

lemma has_integral_AE:

```



```

    assumes ae: AE x in lborel.  $x \in \Omega \longrightarrow f x = g x$ 
    shows (f has_integral x)  $\Omega = (g \text{ has\_integral } x) \Omega$ 
  proof -
    from ae obtain N
    where N:  $N \in \text{sets borel emeasure lborel } N = 0 \ \{x. \neg (x \in \Omega \longrightarrow f x = g x)\}$ 
 $\subseteq N$ 
    by (auto elim!: AE_E)
    then have not_N: AE x in lborel.  $x \notin N$ 
    by (simp add: AE_iff_measurable)
    show ?thesis
  proof (rule has_integral_spike_eq[symmetric])
    show  $\bigwedge x. x \in \Omega - N \implies f x = g x$  using N(3) by auto
    show negligible N
      unfolding negligible_def
    proof (intro allI)
      fix a b :: 'a
      let ?F =  $\lambda x::'a. \text{if } x \in \text{cbox } a \text{ b then indicator } N \text{ else } 0 :: \text{real}$ 
      have integrable lborel ?F = integrable lborel ( $\lambda x::'a. 0 :: \text{real}$ )
      using not_N N(1) by (intro integrable_cong_AE) auto
      moreover have (LINT x|lborel. ?F x) = (LINT x::'a|lborel.  $0 :: \text{real}$ )
      using not_N N(1) by (intro integral_cong_AE) auto
      ultimately have (?F has_integral 0) UNIV
      using has_integral_integral_real[of ?F] by simp
      then show (indicator N has_integral ( $0 :: \text{real}$ )) (cbox a b)
      unfolding has_integral_restrict_UNIV .
    qed
  qed
qed

```

lemma nn_integral_has_integral_lebesgue:

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes nonneg:  $\bigwedge x. 0 \leq f x$  and I: (f has_integral I)  $\Omega$ 
  shows integralN lborel ( $\lambda x. \text{indicator } \Omega \ x * f x$ ) = I
  proof -
    from I have ( $\lambda x. \text{indicator } \Omega \ x * f x$ )  $\in \text{lebesgue} \rightarrow_M \text{borel}$ 
    by (rule has_integral_implies_lebesgue_measurable)
    then obtain f' :: 'a  $\Rightarrow$  real
    where [measurable]:  $f' \in \text{borel} \rightarrow_M \text{borel}$  and eq: AE x in lborel. indicator  $\Omega$ 
 $x * f x = f' x$ 
    by (auto dest: completion_ex_borel_measurable_real)

    from I have (( $\lambda x. \text{abs} (\text{indicator } \Omega \ x * f x)$ ) has_integral I) UNIV
    using nonneg by (simp add: indicator_def of_bool_def if_distrib[of  $\lambda x. x * f$ 
    y for y] cong: if_cong)
    also have (( $\lambda x. \text{abs} (\text{indicator } \Omega \ x * f x)$ ) has_integral I) UNIV  $\longleftrightarrow$  (( $\lambda x. \text{abs}$ 
    ( $f' x$ )) has_integral I) UNIV
    using eq by (intro has_integral_AE) auto
    finally have integralN lborel ( $\lambda x. \text{abs} (f' x)$ ) = I
    by (rule nn_integral_has_integral_lborel[rotated 2]) auto
  
```

2190

```

    also have integralN lborel (λx. abs (f' x)) = integralN lborel (λx. abs (indicator
    Ω x * f x))
    using eq by (intro nn_integral_cong_AE) auto
    finally show ?thesis
    using nonneg by auto
qed

```

```

lemma has_integral_iff_nn_integral_lebesgue:
  assumes f: λx. 0 ≤ f x
  shows (f has_integral r) UNIV ⟷ (f ∈ lebesgue →M borel ∧ integralN lebesgue
  f = r ∧ 0 ≤ r) (is ?I = ?N)
proof
  assume ?I
  have 0 ≤ r
  using has_integral_nonneg[OF ‹?I›] f by auto
  then show ?N
  using nn_integral_has_integral_lebesgue[OF f ‹?I›]
    has_integral_implies_lebesgue_measurable[OF ‹?I›]
    by (auto simp: nn_integral_completion)
next
  assume ?N
  then obtain f' where f': f' ∈ borel →M borel AE x in lborel. f x = f' x
  by (auto dest: completion_ex_borel_measurable_real)
  moreover have (∫+ x. ennreal |f' x| ∂lborel) = (∫+ x. ennreal |f x| ∂lborel)
  using f' by (intro nn_integral_cong_AE) auto
  moreover have ((λx. |f' x|) has_integral r) UNIV ⟷ ((λx. |f x|) has_integral
  r) UNIV
  using f' by (intro has_integral_AE) auto
  moreover note nn_integral_has_integral[of λx. |f' x| r] ‹?N›
  ultimately show ?I
  using f by (auto simp: nn_integral_completion)
qed

```

```

context
  fixes f::'a::euclidean_space ⇒ 'b::euclidean_space
begin

```

```

lemma has_integral_integral_lborel:
  assumes f: integrable lborel f
  shows (f has_integral (integralL lborel f)) UNIV
proof -
  have ((λx. ∑ b∈Basis. (f x • b) *R b) has_integral (∑ b∈Basis. integralL lborel
  (λx. f x • b) *R b)) UNIV
  using f by (intro has_integral_sum finite_Basis ballI has_integral_scaleR_left
  has_integral_integral_real) auto
  also have eq_f: (λx. ∑ b∈Basis. (f x • b) *R b) = f
  by (simp add: fun_eq_iff euclidean_representation)
  also have (∑ b∈Basis. integralL lborel (λx. f x • b) *R b) = integralL lborel f
  using f by (subst (2) eq_f[symmetric]) simp

```

finally show ?thesis .
qed

lemma integrable_on_lborel: integrable lborel $f \implies f$ integrable_on UNIV
using has_integral_integral_lborel by auto

lemma integral_lborel: integrable lborel $f \implies \text{integral UNIV } f = (\int x. f x \partial \text{lborel})$
using has_integral_integral_lborel by auto

end

context
begin

private lemma has_integral_integral_lebesgue_real:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow \text{real}$
assumes f : integrable lebesgue f
shows $(f \text{ has\_integral } (\text{integral}^L \text{ lebesgue } f)) \text{ UNIV}$
proof –
obtain f' where $f': f' \in \text{borel} \rightarrow_M \text{borel AE } x \text{ in lborel. } f x = f' x$
using completion_ex_borel_measurable_real[OF borel_measurable_integrable[OF f]] by auto
moreover have $(\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial \text{lborel}) = (\int^+ x. \text{ennreal } (\text{norm } (f' x)) \partial \text{lborel})$
using f' by (intro nn_integral_cong_AE) auto
ultimately have integrable lborel f'
using f by (auto simp: integrable_iff_bounded nn_integral_completion cong: nn_integral_cong_AE)
note has_integral_integral_real[OF this]
moreover have $\text{integral}^L \text{ lebesgue } f = \text{integral}^L \text{ lebesgue } f'$
using $f' f$ by (intro integral_cong_AE) (auto intro: AE_completion measurable_completion)
moreover have $\text{integral}^L \text{ lebesgue } f' = \text{integral}^L \text{ lborel } f'$
using f' by (simp add: integral_completion)
moreover have $(f' \text{ has\_integral } \text{integral}^L \text{ lborel } f') \text{ UNIV} \longleftrightarrow (f \text{ has\_integral } \text{integral}^L \text{ lborel } f') \text{ UNIV}$
using f' by (intro has_integral_AE) auto
ultimately show ?thesis
by auto
qed

lemma has_integral_integral_lebesgue:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes f : integrable lebesgue f
shows $(f \text{ has\_integral } (\text{integral}^L \text{ lebesgue } f)) \text{ UNIV}$
proof –
have $((\lambda x. \sum_{b \in \text{Basis.}} (f x \cdot b) *_R b) \text{ has\_integral } (\sum_{b \in \text{Basis.}} \text{integral}^L \text{ lebesgue } (\lambda x. f x \cdot b) *_R b)) \text{ UNIV}$
using f by (intro has_integral_sum_finite_Basis ballI has_integral_scaleR_left

```

has_integral_integral_lebesgue_real) auto
  also have eq_f:  $(\lambda x. \sum_{b \in \text{Basis}} (f\ x \cdot b) *_{\mathbb{R}} b) = f$ 
    by (simp add: fun_eq_iff euclidean_representation)
  also have  $(\sum_{b \in \text{Basis}} \text{integral}^L \text{lebesgue } (\lambda x. f\ x \cdot b) *_{\mathbb{R}} b) = \text{integral}^L \text{lebesgue } f$ 
    using f by (subst (2) eq_f[symmetric]) simp
  finally show ?thesis .
qed

```

```

lemma has_integral_integral_lebesgue_on:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes integrable (lebesgue_on S) f  $S \in \text{sets lebesgue}$ 
  shows  $(f \text{ has\_integral } (\text{integral}^L (\text{lebesgue\_on } S) f))\ S$ 
proof -
  let ?f =  $\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0$ 
  have integrable_lebesgue  $(\lambda x. \text{indicat\_real } S\ x *_{\mathbb{R}} f\ x)$ 
    using indicator_scaleR_eq_if [of S _ f] assms
  by (metis (full_types) integrable_restrict_space sets.Int_space_eq2)
  then have integrable_lebesgue ?f
    using indicator_scaleR_eq_if [of S _ f] assms by auto
  then have  $(?f \text{ has\_integral } (\text{integral}^L \text{lebesgue } ?f))\ \text{UNIV}$ 
    by (rule has_integral_integral_lebesgue)
  then have  $(f \text{ has\_integral } (\text{integral}^L \text{lebesgue } ?f))\ S$ 
    using has_integral_restrict_UNIV by blast
  moreover
  have  $S \cap \text{space lebesgue} \in \text{sets lebesgue}$ 
    by (simp add: assms)
  then have  $(\text{integral}^L \text{lebesgue } ?f) = (\text{integral}^L (\text{lebesgue\_on } S) f)$ 
    by (simp add: integrable_restrict_space indicator_scaleR_eq_if)
  ultimately show ?thesis
    by auto
qed

```

```

lemma lebesgue_integral_eq_integral:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes integrable (lebesgue_on S) f  $S \in \text{sets lebesgue}$ 
  shows  $\text{integral}^L (\text{lebesgue\_on } S) f = \text{integral } S\ f$ 
  by (metis has_integral_integral_lebesgue_on assms integral_unique)

```

```

lemma integrable_on_lebesgue:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  shows  $\text{integrable lebesgue } f \Longrightarrow f \text{ integrable\_on UNIV}$ 
  using has_integral_integral_lebesgue by auto

```

```

lemma integral_lebesgue:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  shows  $\text{integrable lebesgue } f \Longrightarrow \text{integral UNIV } f = (\int x. f\ x\ \partial \text{lebesgue})$ 
  using has_integral_integral_lebesgue by auto

```

end

6.19.8 Absolute integrability (this is the same as Lebesgue integrability)

translations

$LBINT\ x.\ f == CONST\ lebesgue\_integral\ CONST\ lborel\ (\lambda x.\ f)$

translations

$LBINT\ x:A.\ f == CONST\ set\_lebesgue\_integral\ CONST\ lborel\ A\ (\lambda x.\ f)$

lemma *set_integral_reflect*:

fixes S and $f :: real \Rightarrow 'a :: \{banach, second\_countable\_topology\}$
 shows $(LBINT\ x : S.\ f\ x) = (LBINT\ x : \{x. -x \in S\}.\ f\ (-x))$
 unfolding *set_lebesgue_integral_def*
 by (subst *lborel_integral_real_affine*[where $c=-1$ and $t=0$])
 (auto intro!: *Bochner_Integration.integral_cong_split*: *split_indicator*)

lemma *borel_integrable_atLeastAtMost'*:

fixes $f :: real \Rightarrow 'a :: \{banach, second\_countable\_topology\}$
 assumes f : *continuous_on* $\{a..b\}$ f
 shows *set_integrable* *lborel* $\{a..b\}$ f
 unfolding *set_integrable_def*
 by (intro *borel_integrable_compact* *compact_Icc* f)

lemma *integral_FTC_atLeastAtMost*:

fixes $f :: real \Rightarrow 'a :: euclidean\_space$
 assumes $a \leq b$
 and F : $\bigwedge x. a \leq x \implies x \leq b \implies (F\ has\_vector\_derivative\ f\ x)$ (at x within $\{a..b\}$)
 and f : *continuous_on* $\{a..b\}$ f
 shows *integral^L* *lborel* $(\lambda x. indicator\ \{a..b\}\ x *_{\mathbb{R}} f\ x) = F\ b - F\ a$
 proof -
 let $?f = \lambda x. indicator\ \{a..b\}\ x *_{\mathbb{R}} f\ x$
 have $(?f\ has\_integral\ (\int x. ?f\ x\ \partial lborel))\ UNIV$
 using *borel_integrable_atLeastAtMost*[OF f]
 unfolding *set_integrable_def* by (rule *has_integral_integral_lborel*)
 moreover
 have $(f\ has\_integral\ F\ b - F\ a)\ \{a..b\}$
 by (intro *fundamental_theorem_of_calculus_ballI* *assms*) *auto*
 then have $(?f\ has\_integral\ F\ b - F\ a)\ \{a..b\}$
 by (subst *has_integral_cong*[where $g=f$]) *auto*
 then have $(?f\ has\_integral\ F\ b - F\ a)\ UNIV$
 by (intro *has_integral_on_superset*[where $T=UNIV$ and $S=\{a..b\}$]) *auto*
 ultimately show *integral^L* *lborel* $?f = F\ b - F\ a$
 by (rule *has_integral_unique*)
 qed

lemma *set_borel_integral_eq_integral*:

```

fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
assumes  $set\_integrable\ lborel\ S\ f$ 
shows  $f\ integrable\_on\ S\ LINT\ x : S \mid lborel.\ f\ x = integral\ S\ f$ 
proof –
  let  $?f = \lambda x.\ indicator\ S\ x *_{\mathbb{R}} f\ x$ 
  have  $(?f\ has\_integral\ LINT\ x : S \mid lborel.\ f\ x)\ UNIV$ 
    using  $assms\ has\_integral\ integral\_lborel$ 
    unfolding  $set\_integrable\_def\ set\_lebesgue\_integral\_def$  by  $blast$ 
  hence  $1: (f\ has\_integral\ (set\_lebesgue\_integral\ lborel\ S\ f))\ S$ 
    by  $(simp\ add: indicator\_scaleR\_eq\_if)$ 
  thus  $f\ integrable\_on\ S$ 
    by  $(auto\ simp\ add: integrable\_on\_def)$ 
  with  $1$  have  $(f\ has\_integral\ (integral\ S\ f))\ S$ 
    by  $(intro\ integrable\_integral,\ auto\ simp\ add: integrable\_on\_def)$ 
  thus  $LINT\ x : S \mid lborel.\ f\ x = integral\ S\ f$ 
    by  $(intro\ has\_integral\_unique\ [OF\ 1])$ 
qed

```

```

lemma  $has\_integral\_set\_lebesgue$ :
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $f: set\_integrable\ lebesgue\ S\ f$ 
  shows  $(f\ has\_integral\ (LINT\ x:S\ lebesgue.\ f\ x))\ S$ 
    using  $has\_integral\_integral\_lebesgue\ f$ 
  by  $(fastforce\ simp\ add: set\_integrable\_def\ set\_lebesgue\_integral\_def\ indicator\_def$ 
     $of\_bool\_def\ if\_distrib[of\ \lambda x.\ x *_{\mathbb{R}} f\ \_] \ cong: if\_cong)$ 

```

```

lemma  $set\_lebesgue\_integral\_eq\_integral$ :
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $f: set\_integrable\ lebesgue\ S\ f$ 
  shows  $f\ integrable\_on\ S\ LINT\ x:S \mid lebesgue.\ f\ x = integral\ S\ f$ 
    using  $has\_integral\_set\_lebesgue[OF\ f]$  by  $(auto\ simp: integral\_unique\ integrable\_on\_def)$ 

```

```

lemma  $lmeasurable\_iff\_has\_integral$ :
   $S \in lmeasurable \longleftrightarrow ((indicator\ S)\ has\_integral\ measure\ lebesgue\ S)\ UNIV$ 
  by  $(subst\ has\_integral\_iff\_nn\_integral\_lebesgue)$ 
   $(auto\ simp: ennreal\_indicator\ emeasure\_eq\_measure2\ borel\_measurable\_indicator\_iff$ 
   $intro!: fmeasurableI)$ 

```

abbreviation

```

   $absolutely\_integrable\_on :: ('a::euclidean\_space \Rightarrow 'b::{banach,\ second\_countable\_topology})$ 
   $\Rightarrow 'a\ set \Rightarrow bool$ 
   $(\text{infixr } absolutely\_integrable\_on\ 46)$ 
  where  $f\ absolutely\_integrable\_on\ s \equiv set\_integrable\ lebesgue\ s\ f$ 

```

```

lemma  $absolutely\_integrable\_zero$   $[simp]: (\lambda x.\ 0)\ absolutely\_integrable\_on\ S$ 
  by  $(simp\ add: set\_integrable\_def)$ 

```

```

lemma absolutely_integrable_on_def:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  shows  $f$  absolutely_integrable_on  $S \iff f$  integrable_on  $S \wedge (\lambda x. \text{norm } (f\ x))$ 
integrable_on  $S$ 
proof safe
  assume  $f$ :  $f$  absolutely_integrable_on  $S$ 
  then have  $nf$ : integrable lebesgue  $(\lambda x. \text{norm } (\text{indicator } S\ x *_{\mathbb{R}} f\ x))$ 
    using integrable_norm set_integrable_def by blast
  show  $f$  integrable_on  $S$ 
    by (rule set_lebesgue_integral_eq_integral[OF f])
  have  $(\lambda x. \text{norm } (\text{indicator } S\ x *_{\mathbb{R}} f\ x)) = (\lambda x. \text{if } x \in S \text{ then } \text{norm } (f\ x) \text{ else } 0)$ 
    by auto
  with integrable_on_lebesgue[OF nf] show  $(\lambda x. \text{norm } (f\ x))$  integrable_on  $S$ 
    by (simp add: integrable_restrict_UNIV)
next
  assume  $f$ :  $f$  integrable_on  $S$  and  $nf$ :  $(\lambda x. \text{norm } (f\ x))$  integrable_on  $S$ 
  show  $f$  absolutely_integrable_on  $S$ 
    unfolding set_integrable_def
  proof (rule integrableI_bounded)
    show  $(\lambda x. \text{indicator } S\ x *_{\mathbb{R}} f\ x) \in \text{borel\_measurable lebesgue}$ 
      using  $f$  has_integral_implies_lebesgue_measurable[of f _ S] by (auto simp:
integrable_on_def)
    show  $(\int^+ x. \text{ennreal } (\text{norm } (\text{indicator } S\ x *_{\mathbb{R}} f\ x))\ \partial \text{lebesgue}) < \infty$ 
      using  $nf$  nn_integral_has_integral_lebesgue[of  $\lambda x. \text{norm } (f\ x)$  _ S]
      by (auto simp: integrable_on_def nn_integral_completion)
  qed
qed

lemma integrable_on_lebesgue_on:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $f$ : integrable (lebesgue_on  $S$ )  $f$  and  $S$ :  $S \in \text{sets lebesgue}$ 
  shows  $f$  integrable_on  $S$ 
proof –
  have integrable lebesgue  $(\lambda x. \text{indicator } S\ x *_{\mathbb{R}} f\ x)$ 
    using  $S$  f inf_top.comm_neutral integrable_restrict_space by blast
  then show ?thesis
    using absolutely_integrable_on_def set_integrable_def by blast
qed

lemma absolutely_integrable_imp_integrable:
  assumes  $f$  absolutely_integrable_on  $S$   $S \in \text{sets lebesgue}$ 
  shows integrable (lebesgue_on  $S$ )  $f$ 
  by (meson assms integrable_restrict_space set_integrable_def sets.Int sets.top)

lemma absolutely_integrable_on_null [intro]:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  shows content (cbox  $a$   $b$ ) = 0  $\implies f$  absolutely_integrable_on (cbox  $a$   $b$ )
  by (auto simp: absolutely_integrable_on_def)

```

lemma *absolutely_integrable_on_open_interval*:
fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$
shows $f \text{ absolutely\_integrable\_on } \text{box } a \ b \longleftrightarrow$
 $f \text{ absolutely\_integrable\_on } \text{cbox } a \ b$
by (auto simp: integrable_on_open_interval absolutely_integrable_on_def)

lemma *absolutely_integrable_restrict_UNIV*:
 $(\lambda x. \text{ if } x \in S \text{ then } f \ x \text{ else } 0) \text{ absolutely\_integrable\_on } \text{UNIV} \longleftrightarrow f \text{ absolutely\_integrable\_on } S$
unfolding set_integrable_def
by (intro arg_cong2[where f=integrable]) auto

lemma *absolutely_integrable_onI*:
fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$
shows $f \text{ integrable\_on } S \implies (\lambda x. \text{ norm } (f \ x)) \text{ integrable\_on } S \implies f \text{ absolutely\_integrable\_on } S$
unfolding absolutely_integrable_on_def **by** auto

lemma *nonnegative_absolutely_integrable_1*:
fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow \text{real}$
assumes $f: f \text{ integrable\_on } A$ **and** $\bigwedge x. x \in A \implies 0 \leq f \ x$
shows $f \text{ absolutely\_integrable\_on } A$
by (rule absolutely_integrable_onI [OF f]) (use assms **in** <simp add: integrable_eq>)

lemma *absolutely_integrable_on_iff_nonneg*:
fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow \text{real}$
assumes $\bigwedge x. x \in S \implies 0 \leq f \ x$ **shows** $f \text{ absolutely\_integrable\_on } S \longleftrightarrow f \text{ integrable\_on } S$
proof –
{ **assume** $f \text{ integrable\_on } S$
then have $(\lambda x. \text{ if } x \in S \text{ then } f \ x \text{ else } 0) \text{ integrable\_on } \text{UNIV}$
by (simp add: integrable_restrict_UNIV)
then have $(\lambda x. \text{ if } x \in S \text{ then } f \ x \text{ else } 0) \text{ absolutely\_integrable\_on } \text{UNIV}$
using < $f \text{ integrable\_on } S$ > absolutely_integrable_restrict_UNIV **assms** nonnegative_absolutely_integrable_1 **by** blast
then have $f \text{ absolutely\_integrable\_on } S$
using absolutely_integrable_restrict_UNIV **by** blast
}
then show ?thesis
unfolding absolutely_integrable_on_def **by** auto
qed

lemma *absolutely_integrable_on_scaleR_iff*:
fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$
shows
 $(\lambda x. c *_{\text{R}} f \ x) \text{ absolutely\_integrable\_on } S \longleftrightarrow$
 $c = 0 \vee f \text{ absolutely\_integrable\_on } S$
proof (cases $c=0$)
case False


```

then show ?thesis
unfolding absolutely_integrable_on_def
by (simp add: norm_mult)
qed auto

```

```

lemma absolutely_integrable_spike:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes f absolutely_integrable_on T and S: negligible S  $\wedge x. x \in T - S \implies$ 
    g x = f x
  shows g absolutely_integrable_on T
  using assms integrable_spike
  unfolding absolutely_integrable_on_def by metis

```

```

lemma absolutely_integrable_negligible:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes negligible S
  shows f absolutely_integrable_on S
  using assms by (simp add: absolutely_integrable_on_def integrable_negligible)

```

```

lemma absolutely_integrable_spike_eq:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes negligible S  $\wedge x. x \in T - S \implies g x = f x$ 
  shows (f absolutely_integrable_on T  $\longleftrightarrow$  g absolutely_integrable_on T)
  using assms by (blast intro: absolutely_integrable_spike sym)

```

```

lemma absolutely_integrable_spike_set_eq:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes negligible {x  $\in$  S - T. f x  $\neq$  0} negligible {x  $\in$  T - S. f x  $\neq$  0}
  shows (f absolutely_integrable_on S  $\longleftrightarrow$  f absolutely_integrable_on T)
proof -
  have ( $\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0$ ) absolutely_integrable_on UNIV  $\longleftrightarrow$ 
    ( $\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0$ ) absolutely_integrable_on UNIV
  proof (rule absolutely_integrable_spike_eq)
    show negligible ({x  $\in$  S - T. f x  $\neq$  0}  $\cup$  {x  $\in$  T - S. f x  $\neq$  0})
    by (rule negligible_Un [OF assms])
  qed auto
  with absolutely_integrable_restrict_UNIV show ?thesis
  by blast
qed

```

```

lemma absolutely_integrable_spike_set:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes f: f absolutely_integrable_on S and neg: negligible {x  $\in$  S - T. f x  $\neq$ 
    0} negligible {x  $\in$  T - S. f x  $\neq$  0}
  shows f absolutely_integrable_on T
  using absolutely_integrable_spike_set_eq f neg by blast

```

```

lemma absolutely_integrable_reflect[simp]:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space

```

```

shows  $(\lambda x. f(-x))$  absolutely_integrable_on cbox  $(-b)$   $(-a) \longleftrightarrow f$  absolutely_integrable_on
cbox  $a$   $b$ 
unfolding absolutely_integrable_on_def
by (metis (mono_tags, lifting) integrable_eq integrable_reflect)

```

```

lemma absolutely_integrable_reflect_real[simp]:
  fixes  $f :: \text{real} \Rightarrow 'b::\text{euclidean\_space}$ 
shows  $(\lambda x. f(-x))$  absolutely_integrable_on  $\{-b .. -a\} \longleftrightarrow f$  absolutely_integrable_on
 $\{a..b::\text{real}\}$ 
unfolding box_real[symmetric] by (rule absolutely_integrable_reflect)

```

```

lemma absolutely_integrable_on_subcbox:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
shows  $\llbracket f \text{ absolutely\_integrable\_on } S; \text{ cbox } a \ b \subseteq S \rrbracket \Longrightarrow f$  absolutely_integrable_on
cbox  $a$   $b$ 
by (meson absolutely_integrable_on_def integrable_on_subcbox)

```

```

lemma absolutely_integrable_on_subinterval:
  fixes  $f :: \text{real} \Rightarrow 'b::\text{euclidean\_space}$ 
shows  $\llbracket f \text{ absolutely\_integrable\_on } S; \{a..b\} \subseteq S \rrbracket \Longrightarrow f$  absolutely_integrable_on
 $\{a..b\}$ 
using absolutely_integrable_on_subcbox by fastforce

```

```

lemma integrable_subinterval:
  fixes  $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
assumes integrable (lebesgue_on  $\{a..b\}$ )  $f$ 
and  $\{c..d\} \subseteq \{a..b\}$ 
shows integrable (lebesgue_on  $\{c..d\}$ )  $f$ 
proof (rule absolutely_integrable_imp_integrable)
show  $f$  absolutely_integrable_on  $\{c..d\}$ 
proof -
  have  $f$  integrable_on  $\{c..d\}$ 
    using assms integrable_on_lebesgue_on integrable_on_subinterval by fast-
force
  moreover have  $(\lambda x. \text{norm } (f \ x))$  integrable_on  $\{c..d\}$ 
proof (rule integrable_on_subinterval)
  show  $(\lambda x. \text{norm } (f \ x))$  integrable_on  $\{a..b\}$ 
    by (simp add: assms integrable_on_lebesgue_on)
qed (use assms in auto)
ultimately show ?thesis
  by (auto simp: absolutely_integrable_on_def)
qed
qed auto

```

```

lemma indefinite_integral_continuous_real:
  fixes  $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
assumes integrable (lebesgue_on  $\{a..b\}$ )  $f$ 
shows continuous_on  $\{a..b\}$   $(\lambda x. \text{integral}^L (\text{lebesgue\_on } \{a..x\}) \ f)$ 
proof -

```

```

have f integrable_on {a..b}
  by (simp add: assms integrable_on_lebesgue_on)
then have continuous_on {a..b} ( $\lambda x. \text{integral } \{a..x\} f$ )
  using indefinite_integral_continuous_1 by blast
moreover have  $\text{integral}^L (\text{lebesgue\_on } \{a..x\}) f = \text{integral } \{a..x\} f$  if  $a \leq x \leq b$  for  $x$ 
proof -
  have  $\{a..x\} \subseteq \{a..b\}$ 
  using that by auto
  then have integrable ( $\text{lebesgue\_on } \{a..x\}$ )  $f$ 
  using integrable_subinterval assms by blast
  then show  $\text{integral}^L (\text{lebesgue\_on } \{a..x\}) f = \text{integral } \{a..x\} f$ 
  by (simp add: lebesgue_integral_eq_integral)
qed
ultimately show ?thesis
  by (metis (no_types, lifting) atLeastAtMost_iff continuous_on_cong)
qed

```

```

lemma lmeasurable_iff_integrable_on:  $S \in \text{lmeasurable} \longleftrightarrow (\lambda x. 1::\text{real}) \text{ integrable\_on } S$ 
  by (subst absolutely_integrable_on_iff_nonneg[symmetric])
  (simp_all add: lmeasurable_iff_integrable_set_integrable_def)

```

```

lemma lmeasure_integral_UNIV:  $S \in \text{lmeasurable} \implies \text{measure lebesgue } S = \text{integral UNIV (indicator } S)$ 
  by (simp add: lmeasurable_iff_has_integral integral_unique)

```

```

lemma lmeasure_integral:  $S \in \text{lmeasurable} \implies \text{measure lebesgue } S = \text{integral } S$ 
( $\lambda x. 1::\text{real}$ )
  by (fastforce simp add: lmeasure_integral_UNIV indicator_def [abs_def] of_bool_def
lmeasurable_iff_integrable_on)

```

```

lemma integrable_on_const:  $S \in \text{lmeasurable} \implies (\lambda x. c) \text{ integrable\_on } S$ 
  unfolding lmeasurable_iff_integrable
  by (metis (mono_tags, lifting) integrable_eq integrable_on_scaleR_left lmeasurable_iff_integrable lmeasurable_iff_integrable_on scaleR_one)

```

```

lemma integral_indicator:
  assumes  $(S \cap T) \in \text{lmeasurable}$ 
  shows  $\text{integral } T (\text{indicator } S) = \text{measure lebesgue } (S \cap T)$ 
proof -
  have  $\text{integral UNIV (indicator } (S \cap T)) = \text{integral UNIV } (\lambda a. \text{if } a \in S \cap T \text{ then } 1::\text{real} \text{ else } 0)$ 
  by (simp add: indicator_def [abs_def] of_bool_def)
  moreover have  $(\text{indicator } (S \cap T) \text{ has\_integral measure lebesgue } (S \cap T))$ 
  UNIV
  using assms by (simp add: lmeasurable_iff_has_integral)
  ultimately have  $\text{integral UNIV } (\lambda x. \text{if } x \in S \cap T \text{ then } 1 \text{ else } 0) = \text{measure lebesgue } (S \cap T)$ 

```

```

    by (metis (no_types) integral_unique)
  moreover have integral T ( $\lambda a.$  if  $a \in S$  then  $1::\text{real}$  else  $0$ ) = integral ( $S \cap T \cap \text{UNIV}$ ) ( $\lambda a.$   $1$ )
  by (simp add: Henstock_Kurzweil_Integration.integral_restrict_Int)
  moreover have integral T (indicat_real S) = integral T ( $\lambda a.$  if  $a \in S$  then  $1$  else  $0$ )
  by (simp add: indicator_def [abs_def] of_bool_def)
  ultimately show ?thesis
  by (simp add: assms lmeasure_integral)
qed

```

```

lemma measurable_integrable:
  fixes S :: 'a::euclidean_space set
  shows S  $\in$  lmeasurable  $\longleftrightarrow$  (indicat_real S) integrable_on UNIV
  by (auto simp: lmeasurable_iff_integrable absolutely_integrable_on_iff_nonneg [symmetric] set_integrable_def)

```

```

lemma integrable_on_indicator:
  fixes S :: 'a::euclidean_space set
  shows indicat_real S integrable_on T  $\longleftrightarrow$  ( $S \cap T$ )  $\in$  lmeasurable
  unfolding measurable_integrable
  unfolding integrable_restrict_UNIV [of T, symmetric]
  by (fastforce simp add: indicator_def elim: integrable_eq)

```

```

lemma
  assumes  $\mathcal{D}$ :  $\mathcal{D}$  division_of S
  shows lmeasurable_division: S  $\in$  lmeasurable (is ?l)
    and content_division: ( $\sum k \in \mathcal{D}. \text{measure lebesgue } k$ ) = measure lebesgue S (is ?m)
proof -
  { fix d1 d2 assume *: d1  $\in$   $\mathcal{D}$  d2  $\in$   $\mathcal{D}$  d1  $\neq$  d2
    then obtain a b c d where d1 = cbox a b d2 = cbox c d
      using division_ofD(4)[OF  $\mathcal{D}$ ] by blast
    with division_ofD(5)[OF  $\mathcal{D}$  *]
    have d1  $\in$  sets lborel d2  $\in$  sets lborel d1  $\cap$  d2  $\subseteq$  (cbox a b - box a b)  $\cup$  (cbox c d - box c d)
      by auto
    moreover have (cbox a b - box a b)  $\cup$  (cbox c d - box c d)  $\in$  null_sets lborel
      by (intro null_sets.Un null_sets_cbox_Diff_box)
    ultimately have d1  $\cap$  d2  $\in$  null_sets lborel
      by (blast intro: null_sets_subset) }
  then show ?l ?m
    unfolding division_ofD(6)[OF  $\mathcal{D}$ , symmetric]
    using division_ofD(1,4)[OF  $\mathcal{D}$ ]
    by (auto intro!: measure_Union_AE[symmetric] simp: completion.AE_iff_null_sets Int_def[symmetric] pairwise_def null_sets_def)
qed

```

```

lemma has_measure_limit:

```

```

assumes  $S \in \text{lmeasurable}$   $e > 0$ 
obtains  $B$  where  $B > 0$ 
   $\bigwedge a \ b. \text{ball } 0 \ B \subseteq \text{cbox } a \ b \implies |\text{measure lebesgue } (S \cap \text{cbox } a \ b) - \text{measure lebesgue } S| < e$ 
using assms unfolding lmeasurable_iff_has_integral_has_integral_alt'
by (force simp: integral_indicator integrable_on_indicator)

```

```

lemma lmeasurable_iff_indicator_has_integral:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  shows  $S \in \text{lmeasurable} \wedge m = \text{measure lebesgue } S \longleftrightarrow (\text{indicat\_real } S \text{ has\_integral } m)$  UNIV
  by (metis has_integral_iff lmeasurable_iff_has_integral measurable_integrable)

```

```

lemma has_measure_limit_iff:
  fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$ 
  shows  $S \in \text{lmeasurable} \wedge m = \text{measure lebesgue } S \longleftrightarrow$ 
     $(\forall e > 0. \exists B > 0. \forall a \ b. \text{ball } 0 \ B \subseteq \text{cbox } a \ b \longrightarrow$ 
       $(S \cap \text{cbox } a \ b) \in \text{lmeasurable} \wedge |\text{measure lebesgue } (S \cap \text{cbox } a \ b) - m|$ 
       $< e)$  (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
    by (meson has_measure_limit fmeasurable.Int lmeasurable_cbox)
next
  assume RHS [rule_format]: ?rhs
  then show ?lhs
    apply (simp add: lmeasurable_iff_indicator_has_integral_has_integral') [where
      i=m]
    by (metis (full_types) integral_indicator integrable_integral integrable_on_indicator)
qed

```

6.19.9 Applications to Negligibility

```

lemma negligible_iff_null_sets:  $\text{negligible } S \longleftrightarrow S \in \text{null\_sets lebesgue}$ 
proof
  assume negligible S
  then have  $(\text{indicator } S \text{ has\_integral } (0::\text{real}))$  UNIV
    by (auto simp: negligible)
  then show  $S \in \text{null\_sets lebesgue}$ 
    by (subst (asm) has_integral_iff_nn_integral_lebesgue)
    (auto simp: borel_measurable_indicator_iff nn_integral_0_iff AE AE_iff_null_sets indicator_eq_0_iff)
next
  assume  $S: S \in \text{null\_sets lebesgue}$ 
  show negligible S
    unfolding negligible_def
  proof (safe intro!: has_integral_iff_nn_integral_lebesgue[THEN iffD2])
     $\text{has\_integral\_restrict\_UNIV}[\textbf{where } s=\text{cbox } \_\_, \textbf{ THEN iffD1}]$ 
    fix  $a \ b$ 
    show  $(\lambda x. \text{if } x \in \text{cbox } a \ b \text{ then indicator } S \ x \text{ else } 0) \in \text{lebesgue} \rightarrow_M \text{borel}$ 

```

```

    using S by (auto intro!: measurable_Iff)
    then show  $(\int^+ x. \text{ennreal } (\text{if } x \in \text{cbox } a \ b \text{ then indicator } S \ x \text{ else } 0) \ \partial \text{lebesgue})$ 
    = ennreal 0
    using S[THEN AE_not_in] by (auto intro!: nn_integral_0_iff_AE[THEN
iffD2])
    qed auto
  qed

```

corollary *eventually_ae_filter_negligible:*

eventually P (ae_filter lebesgue) $\longleftrightarrow (\exists N. \text{negligible } N \wedge \{x. \neg P \ x\} \subseteq N)$
by (auto simp: completion.AE_iff_null_sets negligible_iff_null_sets null_sets_completion_subset)

lemma *starlike_negligible:*

```

  assumes closed S
    and eq1:  $\bigwedge c \ x. (a + c *_R x) \in S \implies 0 \leq c \implies a + x \in S \implies c = 1$ 
    shows negligible S
  proof -
    have negligible ((+) (-a) ' S)
    proof (subst negligible_on_intervals, intro allI)
      fix u v
      show negligible ((+) (-a) ' S  $\cap$  cbox u v)
        using <closed S> eq1 by (auto simp add: negligible_iff_null_sets algebra_simps
        intro!: closed_translation_subtract starlike_negligible_compact cong: image_cong_simp)
      (metis add_diff_eq diff_add_cancel scale_right_diff_distrib)
    qed
    then show ?thesis
      by (rule negligible_translation_rev)
  qed

```

lemma *starlike_negligible_strong:*

```

  assumes closed S
    and star:  $\bigwedge c \ x. [0 \leq c; c < 1; a+x \in S] \implies a + c *_R x \notin S$ 
    shows negligible S
  proof -
    show ?thesis
    proof (rule starlike_negligible [OF <closed S>, of a])
      fix c x
      assume cx:  $a + c *_R x \in S$   $0 \leq c$   $a + x \in S$ 
      with star have  $\neg (c < 1)$  by auto
      moreover have  $\neg (c > 1)$ 
        using star [of  $1/c$   $c *_R x$ ] cx by force
      ultimately show  $c = 1$  by arith
    qed
  qed

```

lemma *negligible_hyperplane:*

assumes $a \neq 0 \vee b \neq 0$ **shows** negligible $\{x. a \cdot x = b\}$

```

proof -
  obtain  $x$  where  $x: a \cdot x \neq b$ 
    using assms by (metis euclidean_all_zero_iff inner_zero_right)
  moreover have  $c = 1$  if  $a \cdot (x + c *_R w) = b$   $a \cdot (x + w) = b$  for  $c \ w$ 
    using that
    by (metis (no_types, opaque_lifting) add_diff_eq diff_0 diff_minus_eq_add
inner_diff_right inner_scaleR_right mult_cancel_right2 right_minus_eq  $x$ )
  ultimately
  show ?thesis
    using starlike_negligible [OF closed_hyperplane, of  $x$ ] by simp
qed

```

lemma *negligible_lowdim*:

```

fixes  $S :: 'N :: euclidean\_space$  set
assumes  $\dim S < DIM('N)$ 
shows negligible  $S$ 

```

```

proof -
  obtain  $a$  where  $a \neq 0$  and  $a: \text{span } S \subseteq \{x. a \cdot x = 0\}$ 
    using lowdim_subset_hyperplane [OF assms] by blast
  have negligible (span  $S$ )
    using  $\langle a \neq 0 \rangle$  a_negligible_hyperplane by (blast intro: negligible_subset)
  then show ?thesis
    using span_base by (blast intro: negligible_subset)
qed

```

proposition *negligible_convex_frontier*:

```

fixes  $S :: 'N :: euclidean\_space$  set
assumes convex  $S$ 
shows negligible(frontier  $S$ )

```

```

proof -
  have nf: negligible(frontier  $S$ ) if convex  $S$   $0 \in S$  for  $S :: 'N$  set
  proof -
    obtain  $B$  where  $B \subseteq S$  and indB: independent  $B$ 
      and spanB:  $S \subseteq \text{span } B$  and cardB:  $\text{card } B = \dim S$ 
      by (metis basis_exists)
    consider  $\dim S < DIM('N) \mid \dim S = DIM('N)$ 
    using dim_subset_UNIV le_eq_less_or_eq by auto
    then show ?thesis
  proof cases
    case 1
    show ?thesis
      by (rule negligible_subset [of closure  $S$ ])
      (simp_all add: frontier_def negligible_lowdim 1)
    next
    case 2
    obtain  $a$  where  $a \in \text{interior } (\text{convex hull insert } 0 B)$ 
    proof (rule interior_simplex_nonempty [OF indB])
      show finite  $B$ 
        by (simp add: indB independent_finite)
    qed
  qed

```

```

    show card B = DIM('N)
    by (simp add: cardB 2)
qed
then have a: a ∈ interior S
    by (metis ‹B ⊆ S› ‹0 ∈ S› ‹convex S› insert_absorb insert_subset inte-
rior_mono subset_hull)
show ?thesis
proof (rule starlike_negligible_strong [where a=a])
  fix c::real and x
  have eq: a + c *R x = (a + x) - (1 - c) *R ((a + x) - a)
    by (simp add: algebra_simps)
  assume 0 ≤ c < 1 a + x ∈ frontier S
  then show a + c *R x ∉ frontier S
    using eq mem_interior_closure_convex_shrink [OF ‹convex S› a, of _
1 - c]
    unfolding frontier_def
    by (metis Diff_iff add_diff_cancel_left' add_diff_eq diff_gt_0_iff_gt
group_cancel.rule0 not_le)
  qed auto
qed
qed
show ?thesis
proof (cases S = {})
  case True then show ?thesis by auto
next
  case False
  then obtain a where a ∈ S by auto
  show ?thesis
    using nf [of (λx. -a + x) ' S]
    by (metis ‹a ∈ S› add_left_inverse assms convex_translation_eq frontier_translation
image_eqI negligible_translation_rev)
  qed
qed

corollary negligible_sphere: negligible (sphere a e)
  using frontier_cball negligible_convex_frontier convex_cball
  by (blast intro: negligible_subset)

lemma non_negligible_UNIV [simp]: ¬ negligible UNIV
  unfolding negligible_iff_null_sets by (auto simp: null_sets_def)

lemma negligible_interval:
  negligible (cbox a b) ⟷ box a b = {} negligible (box a b) ⟷ box a b = {}
  by (auto simp: negligible_iff_null_sets null_sets_def prod_nonneg inner_diff_left
box_eq_empty
not_le emeasure_lborel_cbox_eq emeasure_lborel_box_eq
intro: eq_refl antisym less_imp_le)

proposition open_not_negligible:

```



```

  assumes open S S ≠ {}
  shows ¬ negligible S
proof
  assume neg: negligible S
  obtain a where a ∈ S
    using ⟨S ≠ {}⟩ by blast
  then obtain e where e > 0 cball a e ⊆ S
    using ⟨open S⟩ open_contains_cball_eq by blast
  let ?p = a - (e / DIM('a)) *R One let ?q = a + (e / DIM('a)) *R One
  have cbox ?p ?q ⊆ cball a e
  proof (clarsimp simp: mem_box dist_norm)
    fix x
    assume ∀i∈Basis. ?p · i ≤ x · i ∧ x · i ≤ ?q · i
    then have ax: |(a - x) · i| ≤ e / real DIM('a) if i ∈ Basis for i
      using that by (auto simp: algebra_simps)
    have norm (a - x) ≤ (∑ i∈Basis. |(a - x) · i|)
      by (rule norm_le_l1)
    also have ... ≤ DIM('a) * (e / real DIM('a))
      by (intro sum_bounded_above ax)
    also have ... = e
      by simp
    finally show norm (a - x) ≤ e .
  qed
  then have negligible (cbox ?p ?q)
    by (meson ⟨cball a e ⊆ S⟩ neg negligible_subset)
  with ⟨e > 0⟩ show False
    by (simp add: negligible_interval box_eq_empty algebra_simps field_split_simps
mult_le_0_iff)
qed

lemma negligible_convex_interior:
  convex S ⟹ (negligible S ⟷ interior S = {})
  by (metis Diff_empty_closure_subset frontier_def interior_subset negligible_convex_frontier
negligible_subset open_interior open_not_negligible)

lemma measure_eq_0_null_sets: S ∈ null_sets M ⟹ measure M S = 0
  by (auto simp: measure_def null_sets_def)

lemma negligible_imp_measure0: negligible S ⟹ measure lebesgue S = 0
  by (simp add: measure_eq_0_null_sets negligible_iff_null_sets)

lemma negligible_iff_emeasure0: S ∈ sets lebesgue ⟹ (negligible S ⟷ emeasure
lebesgue S = 0)
  by (auto simp: measure_eq_0_null_sets negligible_iff_null_sets)

lemma negligible_iff_measure0: S ∈ lmeasurable ⟹ (negligible S ⟷ measure
lebesgue S = 0)
  by (metis (full_types) completion.null_sets_outer negligible_iff_null_sets neg-
ligible_imp_measure0 order_refl)

```

lemma *negligible_imp_sets*: $\text{negligible } S \implies S \in \text{sets lebesgue}$
by (*simp add: negligible_iff_null_sets null_setsD2*)

lemma *negligible_imp_measurable*: $\text{negligible } S \implies S \in \text{lmeasurable}$
by (*simp add: fmeasurableI_null_sets negligible_iff_null_sets*)

lemma *negligible_iff_measure*: $\text{negligible } S \longleftrightarrow S \in \text{lmeasurable} \wedge \text{measure lebesgue } S = 0$
by (*fastforce simp: negligible_iff_measure0 negligible_imp_measurable dest: negligible_imp_measure0*)

lemma *negligible_outer*:
 $\text{negligible } S \longleftrightarrow (\forall e > 0. \exists T. S \subseteq T \wedge T \in \text{lmeasurable} \wedge \text{measure lebesgue } T < e)$ (**is** $\_ = ?rhs$)
proof
assume *negligible S then show ?rhs*
by (*metis negligible_iff_measure order_refl*)
next
assume *?rhs then show negligible S*
by (*meson completion.null_sets_outer negligible_iff_null_sets*)
qed

lemma *negligible_outer_le*:
 $\text{negligible } S \longleftrightarrow (\forall e > 0. \exists T. S \subseteq T \wedge T \in \text{lmeasurable} \wedge \text{measure lebesgue } T \leq e)$ (**is** $\_ = ?rhs$)
proof
assume *negligible S then show ?rhs*
by (*metis dual_order.strict_implies_order negligible_imp_measurable negligible_imp_measure0 order_refl*)
next
assume *?rhs then show negligible S*
by (*metis le_less_trans negligible_outer field_lbound_gt_zero*)
qed

lemma *negligible_UNIV*: $\text{negligible } S \longleftrightarrow (\text{indicat\_real } S \text{ has\_integral } 0)$ *UNIV*
(is $\_ = ?rhs$)
by (*metis lmeasurable_iff_indicator_has_integral negligible_iff_measure*)

lemma *sets_negligible_symdiff*:
 $\llbracket S \in \text{sets lebesgue}; \text{negligible}((S - T) \cup (T - S)) \rrbracket \implies T \in \text{sets lebesgue}$
by (*metis Diff_Diff_Int Int_Diff_Un inf_commute negligible_Un_eq negligible_imp_sets sets.Diff sets.Un*)

lemma *lmeasurable_negligible_symdiff*:
 $\llbracket S \in \text{lmeasurable}; \text{negligible}((S - T) \cup (T - S)) \rrbracket \implies T \in \text{lmeasurable}$
using *integrable_spike_set_eq lmeasurable_iff_integrable_on* **by** *blast*

lemma *measure_Un3_negligible*:
assumes *meas*: $S \in \text{lmeasurable}$ $T \in \text{lmeasurable}$ $U \in \text{lmeasurable}$
and *neg*: $\text{negligible}(S \cap T)$ $\text{negligible}(S \cap U)$ $\text{negligible}(T \cap U)$ **and** $V: S \cup T \cup U = V$
shows $\text{measure lebesgue } V = \text{measure lebesgue } S + \text{measure lebesgue } T + \text{measure lebesgue } U$
proof –
have [*simp*]: $\text{measure lebesgue } (S \cap T) = 0$
using *neg*(1) *negligible_imp_measure0* **by** *blast*
have [*simp*]: $\text{measure lebesgue } (S \cap U \cup T \cap U) = 0$
using *neg*(2) *neg*(3) *negligible_Un negligible_imp_measure0* **by** *blast*
have $\text{measure lebesgue } V = \text{measure lebesgue } (S \cup T \cup U)$
using *V* **by** *simp*
also have $\dots = \text{measure lebesgue } S + \text{measure lebesgue } T + \text{measure lebesgue } U$
by (*simp add: measure_Un3 meas fmeasurable.Un Int_Un_distrib2*)
finally show ?thesis .
qed

lemma *measure_translate_add*:
assumes *meas*: $S \in \text{lmeasurable}$ $T \in \text{lmeasurable}$
and $U: S \cup ((+)a \text{ ` } T) = U$  **and** *neg*:  $\text{negligible}(S \cap ((+)a \text{ ` } T))$
shows $\text{measure lebesgue } S + \text{measure lebesgue } T = \text{measure lebesgue } U$
proof –
have [*simp*]: $\text{measure lebesgue } (S \cap (+) a \text{ ` } T) = 0$
using *neg negligible_imp_measure0* **by** *blast*
have $\text{measure lebesgue } (S \cup ((+)a \text{ ` } T)) = \text{measure lebesgue } S + \text{measure lebesgue } T$
by (*simp add: measure_Un3 meas measurable_translation measure_translation fmeasurable.Un*)
then show ?thesis
using *U* **by** *auto*
qed

lemma *measure_negligible_symdiff*:
assumes *S*: $S \in \text{lmeasurable}$
and *neg*: $\text{negligible } (S - T \cup (T - S))$
shows $\text{measure lebesgue } T = \text{measure lebesgue } S$
proof –
have $\text{measure lebesgue } (S - T) = 0$
using *neg negligible_Un_eq negligible_imp_measure0* **by** *blast*
then show ?thesis
by (*metis S Un_commute add.right_neutral lmeasurable_negligible_symdiff measure_Un2 neg negligible_Un_eq negligible_imp_measure0*)
qed

lemma *measure_closure*:
assumes *bounded S* **and** *neg*: $\text{negligible } (\text{frontier } S)$
shows $\text{measure lebesgue } (\text{closure } S) = \text{measure lebesgue } S$

proof –

have $\text{measure lebesgue } (\text{frontier } S) = 0$
 by $(\text{metis neg negligible\_imp\_measure0})$
 then show $?thesis$
 by $(\text{metis assms lmeasurable\_iff\_integrable\_on\_eq\_iff\_diff\_eq\_0 has\_integral\_interior integrable\_on\_def integral\_unique lmeasurable\_interior lmeasure\_integral measure\_frontier})$
qed

lemma *measure_interior*:

$\llbracket \text{bounded } S; \text{negligible}(\text{frontier } S) \rrbracket \implies \text{measure lebesgue } (\text{interior } S) = \text{measure lebesgue } S$
 using *measure_closure measure_frontier negligible_imp_measure0* **by** *fastforce*

lemma *measurable_Jordan*:

assumes *bounded S* **and** *neg: negligible (frontier S)*
 shows $S \in \text{lmeasurable}$

proof –

have $\text{closure } S \in \text{lmeasurable}$
 by $(\text{metis lmeasurable\_closure } \langle \text{bounded } S \rangle)$
 moreover have $\text{interior } S \in \text{lmeasurable}$
 by $(\text{simp add: lmeasurable\_interior } \langle \text{bounded } S \rangle)$
 moreover have $\text{interior } S \subseteq S$
 by $(\text{simp add: interior\_subset})$
 ultimately show $?thesis$
 using *assms* **by** $(\text{metis (full\_types) closure\_subset completion.complete\_sets\_sandwich\_fmeasurable measure\_closure measure\_interior})$
qed

lemma *measurable_convex*: $\llbracket \text{convex } S; \text{bounded } S \rrbracket \implies S \in \text{lmeasurable}$

by $(\text{simp add: measurable\_Jordan negligible\_convex\_frontier})$

lemma *content_cball_conv_ball*: $\text{content } (\text{cball } c \ r) = \text{content } (\text{ball } c \ r)$

proof –

have $\text{ball } c \ r - \text{cball } c \ r \cup (\text{cball } c \ r - \text{ball } c \ r) = \text{sphere } c \ r$
 by *auto*
 hence $\text{measure lebesgue } (\text{cball } c \ r) = \text{measure lebesgue } (\text{ball } c \ r)$
 using *negligible_sphere[of c r]* **by** $(\text{intro measure\_negligible\_syndiff})$ *simp_all*
 thus $?thesis$ **by** *simp*
qed

6.19.10 Negligibility of image under non-injective linear map

lemma *negligible_Union_nat*:

assumes $\bigwedge n::\text{nat. negligible}(S \ n)$

shows $\text{negligible}(\bigcup n. S \ n)$

proof –

have $\text{negligible } (\bigcup_{m \leq k. S \ m})$ **for** k

using *assms* **by** *blast*

then have $0: \text{integral UNIV } (\text{indicat\_real } (\bigcup_{m \leq k. S \ m})) = 0$

```

    and 1: (indicat_real ( $\bigcup m \leq k. S\ m$ )) integrable_on UNIV for k
    by (auto simp: negligible has_integral_iff)
    have 2:  $\bigwedge k\ x. \text{indicat\_real } (\bigcup m \leq k. S\ m)\ x \leq (\text{indicat\_real } (\bigcup m \leq \text{Suc } k. S\ m)\ x)$ 
    by (auto simp add: indicator_def)
    have 3:  $\bigwedge x. (\bigwedge k. \text{indicat\_real } (\bigcup m \leq k. S\ m)\ x) \longrightarrow (\text{indicat\_real } (\bigcup n. S\ n)\ x)$ 
    by (force simp: indicator_def eventually_sequentially intro: tendsto_eventually)
    have 4: bounded (range ( $\bigwedge k. \text{integral UNIV } (\text{indicat\_real } (\bigcup m \leq k. S\ m))$ ))
    by (simp add: 0)
    have *:  $\text{indicat\_real } (\bigcup n. S\ n) \text{ integrable\_on UNIV } \wedge$ 
       $(\bigwedge k. \text{integral UNIV } (\text{indicat\_real } (\bigcup m \leq k. S\ m))) \longrightarrow (\text{integral UNIV } (\text{indicat\_real } (\bigcup n. S\ n)))$ 
    by (intro monotone_convergence_increasing 1 2 3 4)
    then have  $\text{integral UNIV } (\text{indicat\_real } (\bigcup n. S\ n)) = (0::\text{real})$ 
    using LIMSEQ_unique by (auto simp: 0)
    then show ?thesis
    using * by (simp add: negligible_UNIV has_integral_iff)
qed

```

```

lemma negligible_linear_singular_image:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'n
  assumes linear f  $\neg$  inj f
  shows negligible (f ' S)
proof -
  obtain a where  $a \neq 0 \wedge S. f\ ' S \subseteq \{x. a \cdot x = 0\}$ 
  using assms linear_singular_image_hyperplane by blast
  then show negligible (f ' S)
  by (metis negligible_hyperplane negligible_subset)
qed

```

```

lemma measure_negligible_finite_Union:
  assumes finite  $\mathcal{F}$ 
  and meas:  $\bigwedge S. S \in \mathcal{F} \implies S \in \text{lmeasurable}$ 
  and disjointish: pairwise ( $\lambda S\ T. \text{negligible } (S \cap T)$ )  $\mathcal{F}$ 
  shows  $\text{measure lebesgue } (\bigcup \mathcal{F}) = (\sum S \in \mathcal{F}. \text{measure lebesgue } S)$ 
  using assms
proof (induction)
  case empty
  then show ?case
  by auto
next
  case (insert S  $\mathcal{F}$ )
  then have  $S \in \text{lmeasurable} \cup \mathcal{F} \in \text{lmeasurable pairwise } (\lambda S\ T. \text{negligible } (S \cap T))\ \mathcal{F}$ 
  by (simp_all add: fmeasurable.finite_Union insert.hyps(1) insert.premis(1) pairwise_insert_subsetI)
  then show ?case

```

```

proof (simp add: measure_Un3 insert)
  have *:  $\bigwedge T. T \in (\cap) S \text{ ' } \mathcal{F} \implies \text{negligible } T$ 
    using insert by (force simp: pairwise_def)
  have negligible( $S \cap \bigcup \mathcal{F}$ )
    unfolding Int_Union
    by (rule negligible_Union) (simp_all add: * insert.hyps(1))
  then show measure lebesgue ( $S \cap \bigcup \mathcal{F}$ ) = 0
    using negligible_imp_measure0 by blast
qed
qed

```

```

lemma measure_negligible_finite_Union_image:
  assumes finite S
    and meas:  $\bigwedge x. x \in S \implies f x \in \text{lmeasurable}$ 
    and djointish: pairwise ( $\lambda x y. \text{negligible } (f x \cap f y)$ ) S
  shows measure lebesgue ( $\bigcup (f \text{ ' } S)$ ) = ( $\sum_{x \in S}. \text{measure lebesgue } (f x)$ )
proof -
  have measure lebesgue ( $\bigcup (f \text{ ' } S)$ ) = sum (measure lebesgue) (f \text{ ' } S)
    using assms by (auto simp: pairwise_mono pairwise_image intro: measure_negligible_finite_Union)
  also have ... = sum (measure lebesgue  $\circ$  f) S
    using djointish [unfolded pairwise_def] by (metis inf.idem negligible_imp_measure0
sum.reindex_nontrivial [OF <finite S>])
  also have ... = ( $\sum_{x \in S}. \text{measure lebesgue } (f x)$ )
    by simp
  finally show ?thesis .
qed

```

6.19.11 Negligibility of a Lipschitz image of a negligible set

The bound will be eliminated by a sort of onion argument

```

lemma locally_Lipschitz_negl_bounded:
  fixes f :: 'M::euclidean_space  $\Rightarrow$  'N::euclidean_space
  assumes MleN: DIM('M)  $\leq$  DIM('N) 0 < B bounded S negligible S
    and lips:  $\bigwedge x. x \in S$ 
       $\implies \exists T. \text{open } T \wedge x \in T \wedge$ 
        ( $\forall y \in S \cap T. \text{norm}(f y - f x) \leq B * \text{norm}(y - x)$ )
  shows negligible (f \text{ ' } S)
    unfolding negligible_iff_null_sets
proof (clarsimp simp: completion.null_sets_outer)
  fix e::real
  assume 0 < e
  have S  $\in$  lmeasurable
    using <negligible S> by (simp add: negligible_iff_null_sets fmeasurableI_null_sets)
  then have S  $\in$  sets lebesgue
    by blast
  have e22: 0 < e/2 / (2 * B * real DIM('M))  $\wedge$  DIM('N)
    using <0 < e> <0 < B> by (simp add: field_split_simps)
  obtain T where open T S  $\subseteq$  T (T - S)  $\in$  lmeasurable
    measure lebesgue (T - S) < e/2 / (2 * B * DIM('M))  $\wedge$  DIM('N)

```

```

    using sets_lebesgue_outer_open [OF ‹ $S \in \text{sets lebesgue}$ › e22]
  by (metis emeasure_eq_measure2 ennreal_leI linorder_not_le)
then have  $T: \text{measure lebesgue } T \leq e/2 / (2 * B * \text{DIM}('M)) \wedge \text{DIM}('N)$ 
  using ‹negligible  $S$ › by (simp add: measure_Diff_null_set negligible_iff_null_sets)
have  $\exists r. 0 < r \wedge r \leq 1/2 \wedge$ 
   $(x \in S \longrightarrow (\forall y. \text{norm}(y - x) < r$ 
     $\longrightarrow y \in T \wedge (y \in S \longrightarrow \text{norm}(f y - f x) \leq B * \text{norm}(y - x))))$ 
  for  $x$ 
proof (cases  $x \in S$ )
case True
  obtain  $U$  where open  $U$   $x \in U$  and  $U: \bigwedge y. y \in S \cap U \implies \text{norm}(f y - f x)$ 
 $\leq B * \text{norm}(y - x)$ 
  using lips [OF ‹ $x \in S$ ›] by auto
  have  $x \in T \cap U$ 
  using ‹ $S \subseteq T$ › ‹ $x \in U$ › ‹ $x \in S$ › by auto
  then obtain  $\varepsilon$  where  $0 < \varepsilon$  ball  $x \varepsilon \subseteq T \cap U$ 
  by (metis ‹open  $T$ › ‹open  $U$ › openE open_Int)
  then show ?thesis
  by (rule_tac  $x = \min(1/2) \varepsilon$  in exI) (simp add: U_dist_norm_norm_minus_commute
subset_iff)
next
case False
  then show ?thesis
  by (rule_tac  $x = 1/4$  in exI) auto
qed
then obtain  $R$  where  $R12: \bigwedge x. 0 < R x \wedge R x \leq 1/2$ 
  and  $RT: \bigwedge x y. \llbracket x \in S; \text{norm}(y - x) < R x \rrbracket \implies y \in T$ 
  and  $RB: \bigwedge x y. \llbracket x \in S; y \in S; \text{norm}(y - x) < R x \rrbracket \implies \text{norm}(f y -$ 
 $f x) \leq B * \text{norm}(y - x)$ 
  by metis+
  then have gaugeR: gauge  $(\lambda x. \text{ball } x (R x))$ 
  by (simp add: gauge_def)
  obtain  $c$  where  $c: S \subseteq \text{cbox } (-c *_{\mathbb{R}} \text{One}) (c *_{\mathbb{R}} \text{One}) \text{ box } (-c *_{\mathbb{R}} \text{One}:: 'M) (c$ 
 $*_{\mathbb{R}} \text{One}) \neq \{\}$ 
  proof -
  obtain  $B$  where  $B: \bigwedge x. x \in S \implies \text{norm } x \leq B$ 
  using ‹bounded  $S$ › bounded_iff by blast
  show ?thesis
  proof (rule_tac  $c = \text{abs } B + 1$  in that)
  show  $S \subseteq \text{cbox } (-(|B| + 1) *_{\mathbb{R}} \text{One}) (|B| + 1) *_{\mathbb{R}} \text{One}$ 
  using norm_bound_Basis_le Basis_le_norm
  by (fastforce simp: mem_box dest!: B intro: order_trans)
  show box  $(-(|B| + 1) *_{\mathbb{R}} \text{One}) (|B| + 1) *_{\mathbb{R}} \text{One}) \neq \{\}$ 
  by (simp add: box_eq_empty(1))
  qed
qed
qed
obtain  $\mathcal{D}$  where countable  $\mathcal{D}$ 
  and  $D_{\text{sub}}: \bigcup \mathcal{D} \subseteq \text{cbox } (-c *_{\mathbb{R}} \text{One}) (c *_{\mathbb{R}} \text{One})$ 
  and  $c_{\text{box}}: \bigwedge K. K \in \mathcal{D} \implies \text{interior } K \neq \{\} \wedge (\exists c d. K = \text{cbox } c d)$ 

```

and *pw*: *pairwise* ($\lambda A B. \text{interior } A \cap \text{interior } B = \{\}$) $\mathcal{D}$
and *Ksub*: $\bigwedge K. K \in \mathcal{D} \implies \exists x \in S \cap K. K \subseteq (\lambda x. \text{ball } x (R x)) x$
and *exN*: $\bigwedge u v. \text{cbox } u v \in \mathcal{D} \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (2 * c) / 2^n$
and $S \subseteq \bigcup \mathcal{D}$
using *covering_lemma* [*OF c gaugeR*] **by** *force*
have $\exists u v z. K = \text{cbox } u v \wedge \text{box } u v \neq \{\} \wedge z \in S \wedge z \in \text{cbox } u v \wedge \text{cbox } u v \subseteq \text{ball } z (R z)$ **if** $K \in \mathcal{D}$ **for** K
proof –
obtain $u v$ **where** $K = \text{cbox } u v$
using $\langle K \in \mathcal{D} \rangle$ **cbox** **by** *blast*
with *that* **show** *?thesis*
by (*metis Int_iff interior_cbox cbox Ksub*)
qed
then obtain $uf vf zf$
where $uvz: \bigwedge K. K \in \mathcal{D} \implies K = \text{cbox } (uf K) (vf K) \wedge \text{box } (uf K) (vf K) \neq \{\} \wedge zf K \in S \wedge zf K \in \text{cbox } (uf K) (vf K) \wedge \text{cbox } (uf K) (vf K) \subseteq \text{ball } (zf K) (R (zf K))$
by *metis*
define *prj1* **where** $prj1 \equiv \lambda x::'M. x \cdot (\text{SOME } i. i \in \text{Basis})$
define *fbx* **where** $fbx \equiv \lambda D. \text{cbox } (f(zf D) - (B * \text{DIM}('M) * (prj1(vf D - uf D)))) *_R \text{One}::'N)$
 $(f(zf D) + (B * \text{DIM}('M) * prj1(vf D - uf D))) *_R \text{One})$
have $vu\_pos: 0 < prj1 (vf X - uf X)$ **if** $X \in \mathcal{D}$ **for** X
using uvz [*OF that*] **by** (*simp add: prj1_def box_ne_empty SOME_Basis inner_diff_left*)
have $prj1\_idem: prj1 (vf X - uf X) = (vf X - uf X) \cdot i$ **if** $X \in \mathcal{D} i \in \text{Basis}$ **for** $X i$
proof –
have $\text{cbox } (uf X) (vf X) \in \mathcal{D}$
using $uvz \langle X \in \mathcal{D} \rangle$ **by** *auto*
with *exN* **obtain** n **where** $\bigwedge i. i \in \text{Basis} \implies vf X \cdot i - uf X \cdot i = (2 * c) / 2^n$
by *blast*
then show *?thesis*
by (*simp add: \langle i \in Basis \rangle SOME_Basis inner_diff prj1_def*)
qed
have *countbl*: *countable* ($fbx \text{ ` } \mathcal{D}$)
using $\langle \text{countable } \mathcal{D} \rangle$ **by** *blast*
have $(\sum k \in fbx \text{ ` } \mathcal{D}'. \text{measure lebesgue } k) \leq e/2$ **if** $\mathcal{D}' \subseteq \mathcal{D}$ *finite* $\mathcal{D}'$ **for** $\mathcal{D}'$
proof –
have *BM_ge0*: $0 \leq B * (\text{DIM}('M) * prj1 (vf X - uf X))$ **if** $X \in \mathcal{D}'$ **for** X
using $\langle 0 < B \rangle \langle \mathcal{D}' \subseteq \mathcal{D} \rangle$ *that* vu_pos **by** *fastforce*
have $\{\} \notin \mathcal{D}'$
using $\text{cbox } \langle \mathcal{D}' \subseteq \mathcal{D} \rangle$ *interior_empty* **by** *blast*
have $(\sum k \in fbx \text{ ` } \mathcal{D}'. \text{measure lebesgue } k) \leq \text{sum } (\text{measure lebesgue } o fbx) \mathcal{D}'$
by (*rule sum_image_le* [*OF \langle finite \mathcal{D}' \rangle*]) (*force simp: fbx_def*)


```

also have ... ≤ (∑ X ∈  $\mathcal{D}'$ . (2 * B * DIM('M)) ^ DIM('N) * measure lebesgue X)
proof (rule sum_mono)
  fix X assume X ∈  $\mathcal{D}'$ 
  then have X ∈  $\mathcal{D}$  using  $\langle \mathcal{D}' \subseteq \mathcal{D} \rangle$  by blast
  then have ufuf: cbox (uf X) (vf X) = X
    using uvz by blast
  have prj1 (vf X - uf X) ^ DIM('M) = (∏ i::'M ∈ Basis. prj1 (vf X - uf
X))
    by (rule prod_constant [symmetric])
  also have ... = (∏ i ∈ Basis. vf X · i - uf X · i)
    by (simp add:  $\langle X \in \mathcal{D} \rangle$  inner_diff_left prj1_idem cong: prod.cong)
  finally have prj1_eq: prj1 (vf X - uf X) ^ DIM('M) = (∏ i ∈ Basis. vf X ·
i - uf X · i) .
  have uf X ∈ cbox (uf X) (vf X) vf X ∈ cbox (uf X) (vf X)
    using uvz [OF  $\langle X \in \mathcal{D} \rangle$ ] by (force simp: mem_box)+
  moreover have cbox (uf X) (vf X) ⊆ ball (zf X) (1/2)
    by (meson R12 order_trans subset_ball uvz [OF  $\langle X \in \mathcal{D} \rangle$ ])
  ultimately have uf X ∈ ball (zf X) (1/2) vf X ∈ ball (zf X) (1/2)
    by auto
  then have dist (vf X) (uf X) ≤ 1
    unfolding mem_ball
    by (metis dist_commute dist_triangle_half_l dual_order.order_iff_strict)
  then have 1: prj1 (vf X - uf X) ≤ 1
    unfolding prj1_def dist_norm using Basis_le_norm SOME_Basis or-
der_trans by fastforce
  have 0: 0 ≤ prj1 (vf X - uf X)
    using  $\langle X \in \mathcal{D} \rangle$  prj1_def vu_pos by fastforce
  have (measure lebesgue ∘ fbx) X ≤ (2 * B * DIM('M)) ^ DIM('N) * content
(cbox (uf X) (vf X))
    apply (simp add: fbx_def content_cbox_cases algebra_simps BM_ge0  $\langle X \in \mathcal{D}' \rangle$ 
 $\langle 0 < B \rangle$  flip: prj1_eq)
    using MleN 0 1 uvz  $\langle X \in \mathcal{D} \rangle$ 
    by (fastforce simp add: box_ne_empty power_decreasing)
  also have ... = (2 * B * DIM('M)) ^ DIM('N) * measure lebesgue X
    by (subst (3) ufuf[symmetric]) simp
  finally show (measure lebesgue ∘ fbx) X ≤ (2 * B * DIM('M)) ^ DIM('N)
* measure lebesgue X .
qed
also have ... = (2 * B * DIM('M)) ^ DIM('N) * sum (measure lebesgue)  $\mathcal{D}'$ 
  by (simp add: sum_distrib_left)
also have ... ≤ e/2
proof -
  have  $\bigwedge K. K \in \mathcal{D}' \implies \exists a b. K = \text{cbox } a b$ 
    using cbox_that by blast
  then have div:  $\mathcal{D}'$  division_of  $\bigcup \mathcal{D}'$ 
    using pairwise_subset [OF pw  $\langle \mathcal{D}' \subseteq \mathcal{D} \rangle$ ] unfolding pairwise_def
    by (force simp:  $\langle \text{finite } \mathcal{D}' \rangle$   $\langle \{\} \notin \mathcal{D}' \rangle$  division_of_def)
  have le_meat: measure lebesgue ( $\bigcup \mathcal{D}'$ ) ≤ measure lebesgue T

```

```

proof (rule measure_mono_fmeasurable)
  show  $(\bigcup \mathcal{D}') \in \text{sets lebesgue}$ 
    using div lmeasurable_division by auto
  have  $\bigcup \mathcal{D}' \subseteq \bigcup \mathcal{D}$ 
    using  $\langle \mathcal{D}' \subseteq \mathcal{D} \rangle$  by blast
  also have  $\dots \subseteq T$ 
  proof (clarify)
    fix  $x D$ 
    assume  $x \in D \ D \in \mathcal{D}$ 
    show  $x \in T$ 
      using Ksub [OF  $\langle D \in \mathcal{D} \rangle$ ]
      by (metis  $\langle x \in D \rangle$  Int_iff dist_norm mem_ball norm_minus_commute
subsetD RT)
    qed
  finally show  $\bigcup \mathcal{D}' \subseteq T$  .
  show  $T \in \text{lmeasurable}$ 
    using  $\langle S \in \text{lmeasurable} \rangle \ \langle S \subseteq T \rangle \ \langle T - S \in \text{lmeasurable} \rangle$  fmeasurable_Diff_D by blast
  qed
  have  $\text{sum (measure lebesgue) } \mathcal{D}' = \text{sum content } \mathcal{D}'$ 
    using  $\langle \mathcal{D}' \subseteq \mathcal{D} \rangle$  cbox by (force intro: sum.cong)
  then have  $(2 * B * \text{DIM}('M)) \wedge \text{DIM}('N) * \text{sum (measure lebesgue) } \mathcal{D}' =$ 
 $(2 * B * \text{real DIM}('M)) \wedge \text{DIM}('N) * \text{measure lebesgue} (\bigcup \mathcal{D}')$ 
    using content_division [OF div] by auto
  also have  $\dots \leq (2 * B * \text{real DIM}('M)) \wedge \text{DIM}('N) * \text{measure lebesgue } T$ 
    using  $\langle 0 < B \rangle$ 
    by (intro mult_left_mono [OF le_meat]) (force simp add: algebra_simps)
  also have  $\dots \leq e/2$ 
    using  $T \ \langle 0 < B \rangle$  by (simp add: field_simps)
  finally show ?thesis .
  qed
finally show ?thesis .
qed
then have  $e2: \text{sum (measure lebesgue) } \mathcal{G} \leq e/2$  if  $\mathcal{G} \subseteq \text{fbx } ' \mathcal{D}$  finite  $\mathcal{G}$  for  $\mathcal{G}$ 
  by (metis finite_subset_image that)
show  $\exists W \in \text{lmeasurable}. f \restriction S \subseteq W \wedge \text{measure lebesgue } W < e$ 
proof (intro bexI conjI)
  have  $\exists X \in \mathcal{D}. f \restriction y \in \text{fbx } X$  if  $y \in S$  for  $y$ 
  proof –
    obtain  $X$  where  $y \in X \ X \in \mathcal{D}$ 
      using  $\langle S \subseteq \bigcup \mathcal{D} \rangle \ \langle y \in S \rangle$  by auto
    then have  $y: y \in \text{ball}(zf \ X) \ (R(zf \ X))$ 
      using uvz by fastforce
    have conj_le_eq:  $z - b \leq y \wedge y \leq z + b \longleftrightarrow \text{abs}(y - z) \leq b$  for  $z \ y \ b::\text{real}$ 
      by auto
    have yin:  $y \in \text{cbox} \ (uf \ X) \ (vf \ X)$  and zin:  $(zf \ X) \in \text{cbox} \ (uf \ X) \ (vf \ X)$ 
      using uvz  $\langle X \in \mathcal{D} \rangle \ \langle y \in X \rangle$  by auto
    have norm  $(y - zf \ X) \leq (\sum i \in \text{Basis}. |(y - zf \ X) \cdot i|)$ 
      by (rule norm_le_l1)
  qed

```

```

also have ...  $\leq$  real DIM('M) * prj1 (vf X - uf X)
proof (rule sum_bounded_above)
  fix j::'M assume j: j  $\in$  Basis
  show  $|(y - zf\ X) \cdot j| \leq prj1\ (vf\ X - uf\ X)$ 
    using yin zin j
  by (fastforce simp add: mem_box prj1_idem [OF  $\langle X \in \mathcal{D} \rangle j$ ] inner_diff_left)
qed
finally have nole: norm (y - zf X)  $\leq DIM('M) * prj1\ (vf\ X - uf\ X)$ 
  by simp
have fle:  $|f\ y \cdot i - f(zf\ X) \cdot i| \leq B * DIM('M) * prj1\ (vf\ X - uf\ X)$  if i  $\in$ 
Basis for i
proof -
  have  $|f\ y \cdot i - f(zf\ X) \cdot i| = |(f\ y - f(zf\ X)) \cdot i|$ 
    by (simp add: algebra_simps)
  also have ...  $\leq norm\ (f\ y - f(zf\ X))$ 
    by (simp add: Basis_le_norm that)
  also have ...  $\leq B * norm(y - zf\ X)$ 
    by (metis uvz RB  $\langle X \in \mathcal{D} \rangle dist\_commute\ dist\_norm\ mem\_ball\ \langle y \in S \rangle$ 
y)
  also have ...  $\leq B * real\ DIM('M) * prj1\ (vf\ X - uf\ X)$ 
    using  $\langle 0 < B \rangle$  by (simp add: nole)
  finally show ?thesis .
qed
show ?thesis
  by (rule_tac x=X in boxI)
  (auto simp: fbx_def prj1_idem mem_box conj_le_eq inner_add inner_diff
fle  $\langle X \in \mathcal{D} \rangle$ )
qed
then show  $f \restriction S \subseteq (\bigcup D \in \mathcal{D}. fbx\ D)$  by auto
next
have 1:  $\bigwedge D. D \in \mathcal{D} \implies fbx\ D \in lmeasurable$ 
  by (auto simp: fbx_def)
have 2:  $I' \subseteq \mathcal{D} \implies finite\ I' \implies measure\ lebesgue\ (\bigcup D \in I'. fbx\ D) \leq e/2$  for
I'
  by (rule order_trans[OF measure_Union_le e2]) (auto simp: fbx_def)
show  $(\bigcup D \in \mathcal{D}. fbx\ D) \in lmeasurable$ 
  by (intro fmeasurable_UN_bound[OF  $\langle countable\ \mathcal{D} \rangle 1\ 2$ ])
have measure lebesgue  $(\bigcup D \in \mathcal{D}. fbx\ D) \leq e/2$ 
  by (intro measure_UN_bound[OF  $\langle countable\ \mathcal{D} \rangle 1\ 2$ ])
then show measure lebesgue  $(\bigcup D \in \mathcal{D}. fbx\ D) < e$ 
  using  $\langle 0 < e \rangle$  by linarith
qed
qed

proposition negligible_locally_Lipschitz_image:
fixes f :: 'M::euclidean_space  $\Rightarrow$  'N::euclidean_space
assumes MleN: DIM('M)  $\leq DIM('N)$  negligible S
and lips:  $\bigwedge x. x \in S \implies \exists T\ B. open\ T \wedge x \in T \wedge$ 

```

```

      (∀ y ∈ S ∩ T. norm(f y - f x) ≤ B * norm(y - x))
    shows negligible (f ' S)
  proof -
    let ?S = λn. ({x ∈ S. norm x ≤ n ∧
      (∃ T. open T ∧ x ∈ T ∧
        (∀ y ∈ S ∩ T. norm (f y - f x) ≤ (real n + 1) * norm (y
- x))))})
    have negfn: f ' ?S n ∈ null_sets lebesgue for n::nat
      unfolding negligible_iff_null_sets[symmetric]
      apply (rule_tac B = real n + 1 in locally_Lipschitz_negl_bounded)
      by (auto simp: MleN bounded_iff intro: negligible_subset [OF ⟨negligible S⟩])
    have S = (⋃ n. ?S n)
    proof (intro set_eqI iffI)
      fix x assume x ∈ S
      with lips obtain T B where T: open T x ∈ T
        and B: ∧y. y ∈ S ∩ T ⇒ norm(f y - f x) ≤ B * norm(y
- x)
      by metis+
      have no: norm (f y - f x) ≤ (nat ⌈max B (norm x)⌉ + 1) * norm (y - x) if
y ∈ S ∩ T for y
      proof -
        have B * norm(y - x) ≤ (nat ⌈max B (norm x)⌉ + 1) * norm (y - x)
        by (meson max.cobounded1 mult_right_mono nat_ceiling_le_eq nat_le_iff_add
norm_ge_zero order_trans)
        then show ?thesis
          using B order_trans that by blast
      qed
      have norm x ≤ real (nat ⌈max B (norm x)⌉)
      by linarith
      then have x ∈ ?S (nat (ceiling (max B (norm x))))
      using T no by (force simp: ⟨x ∈ S⟩ algebra_simps)
      then show x ∈ (⋃ n. ?S n) by force
    qed auto
    then show ?thesis
      by (rule ssubst) (auto simp: image_Union negligible_iff_null_sets intro: negfn)
  qed

corollary negligible_differentiable_image_negligible:
  fixes f :: 'M::euclidean_space ⇒ 'N::euclidean_space
  assumes MleN: DIM('M) ≤ DIM('N) negligible S
  and diff_f: f differentiable_on S
  shows negligible (f ' S)
proof -
  have ∃ T B. open T ∧ x ∈ T ∧ (∀ y ∈ S ∩ T. norm(f y - f x) ≤ B * norm(y
- x))
  if x ∈ S for x
  proof -
    obtain f' where linear f'
    and f': ∧e. e > 0 ⇒

```

```

      
$$\exists d > 0. \forall y \in S. \text{norm } (y - x) < d \longrightarrow$$


$$\text{norm } (f y - f x - f' (y - x)) \leq e * \text{norm } (y - x)$$

    using diff_f ‹x ∈ S›
  by (auto simp: linear_linear differentiable_on_def differentiable_def has_derivative_within_alt)
  obtain B where B > 0 and B:  $\forall x. \text{norm } (f' x) \leq B * \text{norm } x$ 
    using linear_bounded_pos ‹linear f'› by blast
  obtain d where d > 0
    and d:  $\bigwedge y. \llbracket y \in S; \text{norm } (y - x) < d \rrbracket \implies$ 

$$\text{norm } (f y - f x - f' (y - x)) \leq \text{norm } (y - x)$$

    using f' [of 1] by (force simp:)
  show ?thesis
  proof (intro exI conjI ballI)
    show  $\text{norm } (f y - f x) \leq (B + 1) * \text{norm } (y - x)$ 
      if y ∈ S ∩ ball x d for y
    proof -
      have  $\text{norm } (f y - f x) - B * \text{norm } (y - x) \leq \text{norm } (f y - f x) - \text{norm } (f' (y - x))$ 
      by (simp add: B)
      also have  $\dots \leq \text{norm } (f y - f x - f' (y - x))$ 
      by (rule norm_triangle_ineq2)
      also have  $\dots \leq \text{norm } (y - x)$ 
      by (metis IntE d dist_norm mem_ball norm_minus_commute that)
      finally show ?thesis
      by (simp add: algebra_simps)
    qed
  qed (use ‹d > 0› in auto)
  qed
  with negligible_locally_Lipschitz_image assms show ?thesis by metis
  qed

corollary negligible_differentiable_image_lowdim:
  fixes f :: 'M::euclidean_space  $\Rightarrow$  'N::euclidean_space
  assumes MlessN:  $\text{DIM}('M) < \text{DIM}('N)$  and diff_f: f differentiable_on S
  shows negligible (f ‹S)
  proof -
    have  $x \leq \text{DIM}('M) \implies x \leq \text{DIM}('N)$  for x
    using MlessN by linarith
    obtain lift :: 'M * real  $\Rightarrow$  'N and drop :: 'N  $\Rightarrow$  'M * real and j :: 'N
      where linear_lift linear_drop and dropl [simp]:  $\bigwedge z. \text{drop } (\text{lift } z) = z$ 
      and j ∈ Basis and j:  $\bigwedge x. \text{lift}(x, 0) \cdot j = 0$ 
    using lowerdim_embeddings [OF MlessN] by metis
    have negligible (( $\lambda x. \text{lift } (x, 0)$ ) ‹S)
    proof -
      have negligible {x. x · j = 0}
      by (metis ‹j ∈ Basis› negligible_standard_hyperplane)
      moreover have ( $\lambda m. \text{lift } (m, 0)$ ) ‹S  $\subseteq$  {n. n · j = 0}
      using j by force
      ultimately show ?thesis
      using negligible_subset by auto
    qed
  qed

```

```

qed
moreover
have f ∘ fst ∘ drop differentiable_on (λx. lift (x, 0)) ‘ S
  using diff_f
  apply (clarsimp simp add: differentiable_on_def)
  apply (intro differentiable_chain_within linear_imp_differentiable [OF ⟨linear
drop⟩]
        linear_imp_differentiable [OF linear_fst])
  apply (force simp: image_comp o_def)
done
moreover
have f = f ∘ fst ∘ drop ∘ (λx. lift (x, 0))
  by (simp add: o_def)
ultimately show ?thesis
  by (metis (no_types) image_comp negligible_differentiable_image_negligible
order_refl)
qed

```

6.19.12 Measurability of countable unions and intersections of various kinds.

```

lemma
  assumes S:  $\bigwedge n. S\ n \in lmeasurable$ 
  and leB:  $\bigwedge n. \text{measure lebesgue } (S\ n) \leq B$ 
  and nest:  $\bigwedge n. S\ n \subseteq S(Suc\ n)$ 
  shows measurable_nested_Union:  $(\bigcup n. S\ n) \in lmeasurable$ 
  and measure_nested_Union:  $(\lambda n. \text{measure lebesgue } (S\ n)) \longrightarrow \text{measure lebesgue } (\bigcup n. S\ n)$  (is ?Lim)
proof -
  have indicat_real ( $\bigcup (range\ S)$ ) integrable_on UNIV  $\wedge$ 
    ( $\lambda n. \text{integral UNIV } (\text{indicat\_real } (S\ n))$ )
     $\longrightarrow \text{integral UNIV } (\text{indicat\_real } (\bigcup (range\ S)))$ 
  proof (rule monotone_convergence_increasing)
    show  $\bigwedge n. (\text{indicat\_real } (S\ n)) \text{ integrable\_on UNIV}$ 
      using S measurable_integrable by blast
    show  $\bigwedge n\ x::'a. \text{indicat\_real } (S\ n)\ x \leq (\text{indicat\_real } (S\ (Suc\ n))\ x)$ 
      by (simp add: indicator_leI nest rev_subsetD)
    have  $\bigwedge x. (\exists n. x \in S\ n) \longrightarrow (\forall_F\ n \text{ in sequentially. } x \in S\ n)$ 
      by (metis eventually_sequentiallyI lift_Suc_mono_le nest subsetCE)
    then
    show  $\bigwedge x. (\lambda n. \text{indicat\_real } (S\ n)\ x) \longrightarrow (\text{indicat\_real } (\bigcup (S\ ' UNIV))\ x)$ 
      by (simp add: indicator_def tendsto_eventually)
    show bounded (range ( $\lambda n. \text{integral UNIV } (\text{indicat\_real } (S\ n))$ ))
    using leB by (auto simp: lmeasure_integral_UNIV [symmetric] S bounded_iff)
  qed
  then have  $(\bigcup n. S\ n) \in lmeasurable \wedge ?Lim$ 
    by (simp add: lmeasure_integral_UNIV S cong: conj_cong) (simp add: measurable_integrable)
  then show  $(\bigcup n. S\ n) \in lmeasurable\ ?Lim$ 

```

by auto
qed

lemma

assumes $S: \bigwedge n. S\ n \in \text{lmeasurable}$
 and $djointish: \text{pairwise } (\lambda m\ n. \text{negligible } (S\ m \cap S\ n))\ UNIV$
 and $leB: \bigwedge n. (\sum k \leq n. \text{measure lebesgue } (S\ k)) \leq B$
 shows $\text{measurable\_countable\_negligible\_Union}: (\bigcup n. S\ n) \in \text{lmeasurable}$
 and $\text{measure\_countable\_negligible\_Union}: (\lambda n. (\text{measure lebesgue } (S\ n)))$
 $\text{sums measure lebesgue } (\bigcup n. S\ n) \text{ (is ?Sums)}$
 proof -
 have 1: $\bigcup (S\ ' \{..n\}) \in \text{lmeasurable}$ for n
 using S by blast
 have 2: $\text{measure lebesgue } (\bigcup (S\ ' \{..n\})) \leq B$ for n
 proof -
 have $\text{measure lebesgue } (\bigcup (S\ ' \{..n\})) \leq (\sum k \leq n. \text{measure lebesgue } (S\ k))$
 by (simp add: $S\ \text{fmeasurableD measure\_UNION\_le}$)
 with leB show ?thesis
 using order_trans by blast
 qed
 have 3: $\bigwedge n. \bigcup (S\ ' \{..n\}) \subseteq \bigcup (S\ ' \{..Suc\ n\})$
 by (simp add: SUP_subset_mono)
 have $eqS: (\bigcup n. S\ n) = (\bigcup n. \bigcup (S\ ' \{..n\}))$
 using atLeastAtMost_iff by blast
 also have $(\bigcup n. \bigcup (S\ ' \{..n\})) \in \text{lmeasurable}$
 by (intro $\text{measurable\_nested\_Union [OF 1 2]}$ 3)
 finally show $(\bigcup n. S\ n) \in \text{lmeasurable}$.
 have $eqm: (\sum i \leq n. \text{measure lebesgue } (S\ i)) = \text{measure lebesgue } (\bigcup (S\ ' \{..n\}))$
 for n
 using assms by (simp add: $\text{measure\_negligible\_finite\_Union\_image pairwise\_mono}$)
 have $(\lambda n. (\text{measure lebesgue } (S\ n))) \text{ sums measure lebesgue } (\bigcup n. \bigcup (S\ ' \{..n\}))$
 by (simp add: $\text{sums\_def' eqm atLeast0AtMost}$) (intro $\text{measure\_nested\_Union [OF 1 2]}$ 3)
 then show ?Sums
 by (simp add: eqS)
 qed

lemma $\text{negligible\_countable\_Union [intro]}$:

assumes $\text{countable } \mathcal{F}$ and $\text{meas}: \bigwedge S. S \in \mathcal{F} \implies \text{negligible } S$
 shows $\text{negligible } (\bigcup \mathcal{F})$
 proof (cases $\mathcal{F} = \{\}$)
 case False
 then show ?thesis
 by (metis $\text{from\_nat\_into range\_from\_nat\_into assms negligible\_Union\_nat}$)
 qed simp

lemma

assumes $S: \bigwedge n. (S\ n) \in \text{lmeasurable}$

```

    and djointish: pairwise ( $\lambda m n. \text{negligible } (S\ m \cap S\ n)$ ) UNIV
    and bo: bounded ( $\bigcup n. S\ n$ )
    shows measurable_countable_negligible_Union_bounded: ( $\bigcup n. S\ n$ )  $\in$  lmeasurable
    and measure_countable_negligible_Union_bounded: ( $\lambda n. (\text{measure lebesgue } (S\ n)) \text{ sums measure lebesgue } (\bigcup n. S\ n)$ ) (is ?Sums)
    proof -
      obtain a b where ab: ( $\bigcup n. S\ n$ )  $\subseteq$  cbox a b
      using bo bounded_subset_cbox_symmetric by metis
      then have B: ( $\sum k \leq n. \text{measure lebesgue } (S\ k)$ )  $\leq$  measure lebesgue (cbox a b)
    for n
      proof -
        have ( $\sum k \leq n. \text{measure lebesgue } (S\ k)$ ) = measure lebesgue ( $\bigcup (S\ ' \{..n\})$ )
        using measure_negligible_finite_Union_image [OF _ _ pairwise_subset]
      djointish
        by (metis S finite_atMost_subset_UNIV)
      also have ...  $\leq$  measure lebesgue (cbox a b)
      proof (rule measure_mono_fmeasurable)
        show  $\bigcup (S\ ' \{..n\}) \in \text{sets lebesgue}$  using S by blast
      qed (use ab in auto)
      finally show ?thesis .
    qed
    show ( $\bigcup n. S\ n$ )  $\in$  lmeasurable
    by (rule measurable_countable_negligible_Union [OF S djointish B])
    show ?Sums
    by (rule measure_countable_negligible_Union [OF S djointish B])
  qed

lemma measure_countable_Union_approachable:
  assumes countable  $\mathcal{D}$   $e > 0$  and measD:  $\bigwedge d. d \in \mathcal{D} \implies d \in \text{lmeasurable}$ 
  and B:  $\bigwedge D'. \llbracket D' \subseteq \mathcal{D}; \text{finite } D' \rrbracket \implies \text{measure lebesgue } (\bigcup D') \leq B$ 
  obtains  $D'$  where  $D' \subseteq \mathcal{D}$  finite  $D'$  measure lebesgue ( $\bigcup \mathcal{D}$ )  $- e < \text{measure lebesgue } (\bigcup D')$ 
  proof (cases  $\mathcal{D} = \{\}$ )
    case True
      then show ?thesis
        by (simp add:  $\langle e > 0 \rangle$  that)
    next
      case False
        let ?S =  $\lambda n. \bigcup k \leq n. \text{from\_nat\_into } \mathcal{D}\ k$ 
        have ( $\lambda n. \text{measure lebesgue } (?S\ n)$ )  $\longrightarrow$  measure lebesgue ( $\bigcup n. ?S\ n$ )
        proof (rule measure_nested_Union)
          show ?S  $n \in \text{lmeasurable}$  for n
            by (simp add: False fmeasurable.finite_UN from_nat_into measD)
          show measure lebesgue (?S n)  $\leq B$  for n
            by (metis (mono_tags, lifting) B False finite_atMost_finite_imageI from_nat_into image_iff subsetI)
          show ?S n  $\subseteq$  ?S (Suc n) for n
            by force
        qed
      qed
  qed

```



```

qed
  then obtain N where N:  $\bigwedge n. n \geq N \implies \text{dist } (\text{measure lebesgue } (?S \ n))$ 
    ( $\text{measure lebesgue } (\bigcup n. ?S \ n)) < e$ 
    using metric_LIMSEQ_D  $\langle e > 0 \rangle$  by blast
  show ?thesis
  proof
    show from_nat_into  $\mathcal{D}$   $\langle \{..N\} \subseteq \mathcal{D}$ 
      by (auto simp: False from_nat_into)
    have eq:  $(\bigcup n. \bigcup k \leq n. \text{from\_nat\_into } \mathcal{D} \ k) = (\bigcup \mathcal{D})$ 
      using  $\langle \text{countable } \mathcal{D} \rangle$  False
      by (auto intro: from_nat_into dest: from_nat_into_surj [OF  $\langle \text{countable } \mathcal{D} \rangle$ ])
    show  $\text{measure lebesgue } (\bigcup \mathcal{D}) - e < \text{measure lebesgue } (\bigcup (\text{from\_nat\_into } \mathcal{D} \ \{..N\}))$ 
      using N [OF order_refl]
      by (auto simp: eq algebra_simps dist_norm)
  qed auto
qed

```

6.19.13 Negligibility is a local property

lemma *locally_negligible_alt*:

$\text{negligible } S \iff (\forall x \in S. \exists U. \text{openin } (\text{top\_of\_set } S) \ U \wedge x \in U \wedge \text{negligible } U)$
 (is $\_ = ?rhs$)

proof

assume *negligible S*

then show *?rhs*

using *openin_subtopology_self* by blast

next

assume *?rhs*

then obtain *U* where *ope*: $\bigwedge x. x \in S \implies \text{openin } (\text{top\_of\_set } S) \ (U \ x)$

and *cov*: $\bigwedge x. x \in S \implies x \in U \ x$

and *neg*: $\bigwedge x. x \in S \implies \text{negligible } (U \ x)$

by *metis*

obtain $\mathcal{F}$ where $\mathcal{F} \subseteq U \text{ ' } S$ *countable* $\mathcal{F}$ and *eq*: $\bigcup \mathcal{F} = \bigcup (U \text{ ' } S)$

using *ope* by (force *intro*: *Lindelof_openin* [of $U \text{ ' } S$])

then have *negligible* $(\bigcup \mathcal{F})$

by (*metis imageE neg negligible_countable_Union subset_eq*)

with *eq* have *negligible* $(\bigcup (U \text{ ' } S))$

by *metis*

moreover have $S \subseteq \bigcup (U \text{ ' } S)$

using *cov* by blast

ultimately show *negligible S*

using *negligible_subset* by blast

qed

lemma *locally_negligible*: *locally negligible S* $\iff$ *negligible S*

unfolding *locally_def*

by (*metis locally_negligible_alt negligible_subset openin_imp_subset openin_subtopology_self*)

6.19.14 Integral bounds

lemma *set_integral_norm_bound*:
fixes $f :: \_ \Rightarrow 'a :: \{\text{banach, second\_countable\_topology}\}$
shows $\text{set\_integrable } M \ k \ f \implies \text{norm } (\text{LINT } x:k|M. f \ x) \leq \text{LINT } x:k|M. \text{norm } (f \ x)$
using *integral_norm_bound*[of $M \ \lambda x. \text{indicator } k \ x \ *_R \ f \ x$] **by** (*simp add: set_lebesgue_integral_def*)

lemma *set_integral_finite_UN_AE*:
fixes $f :: \_ \Rightarrow \_ :: \{\text{banach, second\_countable\_topology}\}$
assumes *finite I*
and $ae: \bigwedge i \ j. i \in I \implies j \in I \implies AE \ x \text{ in } M. (x \in A \ i \wedge x \in A \ j) \longrightarrow i = j$
and [*measurable*]: $\bigwedge i. i \in I \implies A \ i \in \text{sets } M$
and $f: \bigwedge i. i \in I \implies \text{set\_integrable } M \ (A \ i) \ f$
shows $\text{LINT } x:(\bigcup i \in I. A \ i)|M. f \ x = (\sum i \in I. \text{LINT } x:A \ i|M. f \ x)$
using $\langle \text{finite } I \rangle \text{ order\_refl}$ [of I]
proof (*induction I rule: finite_subset_induct*)
case (*insert i I'*)
have $AE \ x \text{ in } M. (\forall j \in I'. x \in A \ i \longrightarrow x \notin A \ j)$
proof (*intro AE_ball_countable[THEN iffD2] ballI*)
fix j **assume** $j \in I'$
with $\langle I' \subseteq I \rangle \langle i \notin I' \rangle$ **have** $i \neq j \ j \in I$
by *auto*
then show $AE \ x \text{ in } M. x \in A \ i \longrightarrow x \notin A \ j$
using ae [of $i \ j$] $\langle i \in I \rangle$ **by** *auto*
qed (*use* $\langle \text{finite } I' \rangle$ **in** $\langle \text{rule countable\_finite} \rangle$)
then have $AE \ x \in A \ i \text{ in } M. \forall xa \in I'. x \notin A \ xa$
by *auto*
with *insert.hyps insert.IH*[*symmetric*]
show *?case*
by (*auto intro!: set_integral_Un_AE sets.finite_UN f set_integrable_UN*)
qed (*simp add: set_lebesgue_integral_def*)

lemma *set_integrable_norm*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second\_countable\_topology}\}$
assumes $f: \text{set\_integrable } M \ k \ f$ **shows** $\text{set\_integrable } M \ k \ (\lambda x. \text{norm } (f \ x))$
using *integrable_norm f* **by** (*force simp add: set_integrable_def*)

lemma *absolutely_integrable_bounded_variation*:
fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$
assumes $f: f \text{ absolutely\_integrable\_on } UNIV$
obtains B **where** $\forall d. d \text{ division\_of } (\bigcup d) \longrightarrow \text{sum } (\lambda k. \text{norm } (\text{integral } k \ f)) \ d \leq B$
proof (*rule that*[of *integral UNIV* $(\lambda x. \text{norm } (f \ x))$]; *safe*)
fix $d :: 'a \text{ set set}$ **assume** $d: d \text{ division\_of } \bigcup d$
have $*$: $k \in d \implies f \text{ absolutely\_integrable\_on } k$ **for** k
using f [*THEN set_integrable_subset, of k*] *division_ofD*(2,4)[*OF d, of k*] **by** *auto*
note $d' = \text{division\_ofD}$ [*OF d*]

```

have ( $\sum k \in d. \text{norm} (\text{integral } k \ f)$ ) = ( $\sum k \in d. \text{norm} (\text{LINT } x:k | \text{lebesgue}. f \ x)$ )
by (intro sum.cong refl arg_cong[where  $f = \text{norm}$ ] set_lebesgue_integral_eq_integral(2)[symmetric]
*)
also have ...  $\leq$  ( $\sum k \in d. \text{LINT } x:k | \text{lebesgue}. \text{norm} (f \ x)$ )
by (intro sum_mono set_integral_norm_bound *)
also have ... = ( $\sum k \in d. \text{integral } k \ (\lambda x. \text{norm} (f \ x))$ )
by (intro sum.cong refl set_lebesgue_integral_eq_integral(2) set_integrable_norm
*)
also have ...  $\leq \text{integral} (\bigcup d) (\lambda x. \text{norm} (f \ x))$ 
using integrable_on_subdivision[OF d] assms f unfolding absolutely_integrable_on_def
by (subst integral_combine_division_topdown[OF _ d]) auto
also have ...  $\leq \text{integral UNIV} (\lambda x. \text{norm} (f \ x))$ 
using integrable_on_subdivision[OF d] assms unfolding absolutely_integrable_on_def
by (intro integral_subset_le) auto
finally show ( $\sum k \in d. \text{norm} (\text{integral } k \ f)$ )  $\leq \text{integral UNIV} (\lambda x. \text{norm} (f \ x))$  .
qed

```

lemma absdiff_norm_less:

```

assumes sum ( $\lambda x. \text{norm} (f \ x - g \ x)$ )  $S < e$ 
shows  $|\text{sum} (\lambda x. \text{norm}(f \ x)) \ S - \text{sum} (\lambda x. \text{norm}(g \ x)) \ S| < e$  (is ?lhs  $< e$ )
proof -
have ?lhs  $\leq$  ( $\sum i \in S. |\text{norm} (f \ i) - \text{norm} (g \ i)|$ )
by (metis (no_types) sum_abs sum_subtractf)
also have ...  $\leq$  ( $\sum x \in S. \text{norm} (f \ x - g \ x)$ )
by (simp add: norm_triangle_ineq3 sum_mono)
also have ...  $< e$ 
using assms(1) by blast
finally show ?thesis .
qed

```

proposition bounded_variation_absolutely_integrable_interval:

```

fixes f :: 'n::euclidean_space  $\Rightarrow$  'm::euclidean_space
assumes f: f integrable_on cbox a b
and *:  $\bigwedge d. d \text{ division\_of } (cbox \ a \ b) \implies \text{sum} (\lambda K. \text{norm}(\text{integral } K \ f)) \ d \leq B$ 
shows f absolutely_integrable_on cbox a b
proof -
let ?f =  $\lambda d. \sum K \in d. \text{norm} (\text{integral } K \ f)$  and ?D =  $\{d. d \text{ division\_of } (cbox \ a \ b)\}$ 
have D_1: ?D  $\neq \{\}$ 
by (rule elementary_interval[of a b]) auto
have D_2: bdd_above (?f`?D)
by (metis * mem_Collect_eq bdd_aboveI2)
note D = D_1 D_2
let ?S = SUP  $x \in ?D. ?f \ x$ 
have *:  $\exists \gamma. \text{gauge } \gamma \wedge$ 
 $(\forall p. p \text{ tagged\_division\_of } cbox \ a \ b \wedge$ 
 $\gamma \text{ fine } p \implies$ 
 $\text{norm} ((\sum (x,k) \in p. \text{content } k * \text{norm} (f \ x)) - ?S) < e)$ 
if e:  $e > 0$  for e

```

```

proof -
  have  $?S - e/2 < ?S$  using  $\langle e > 0 \rangle$  by simp
  then obtain  $d$  where  $d$ :  $d$  division_of (cbox  $a$   $b$ )  $?S - e/2 < (\sum_{k \in d. \text{norm}}$ 
(integral  $k$   $f$ ))
    unfolding less_cSUP_iff[OF  $D$ ] by auto
    note  $d' = \text{division\_of} D[\text{OF } \text{this}(1)]$ 

  have  $\exists e > 0. \forall i \in d. x \notin i \longrightarrow \text{ball } x \ e \cap i = \{\}$  for  $x$ 
  proof -
    have  $\exists d' > 0. \forall x' \in \bigcup \{i \in d. x \notin i\}. d' \leq \text{dist } x \ x'$ 
    proof (rule separate_point_closed)
      show closed  $(\bigcup \{i \in d. x \notin i\})$ 
        using  $d'$  by force
      show  $x \notin \bigcup \{i \in d. x \notin i\}$ 
        by auto
    qed
    then show  $?thesis$ 
      by force
    qed
  then obtain  $k$  where  $k$ :  $\bigwedge x. 0 < k \ x \ \bigwedge i \ x. \llbracket i \in d; x \notin i \rrbracket \Longrightarrow \text{ball } x \ (k \ x) \cap$ 
 $i = \{\}$ 
    by metis
  have  $e/2 > 0$ 
    using  $e$  by auto
  with Henstock_lemma[OF  $f$ ]
  obtain  $\gamma$  where  $g$ : gauge  $\gamma$ 
     $\bigwedge p. \llbracket p \text{ tagged\_partial\_division\_of } \text{cbox } a \ b; \gamma \text{ fine } p \rrbracket$ 
 $\Longrightarrow (\sum (x,k) \in p. \text{norm } (\text{content } k \ *_{\mathbb{R}} f \ x - \text{integral } k \ f)) < e/2$ 
    by (metis (no_types, lifting))
  let  $?g = \lambda x. \gamma \ x \cap \text{ball } x \ (k \ x)$ 
  show  $?thesis$ 
  proof (intro exI conjI allI impI)
    show gauge  $?g$ 
      using  $g(1) \ k(1)$  by (auto simp: gauge_def)
  next
    fix  $p$ 
    assume  $p$  tagged_division_of (cbox  $a$   $b$ )  $\wedge ?g$  fine  $p$ 
    then have  $p$ :  $p$  tagged_division_of cbox  $a$   $b$   $\gamma$  fine  $p$   $(\lambda x. \text{ball } x \ (k \ x))$  fine  $p$ 
      by (auto simp: fine_Int)
    note  $p' = \text{tagged\_division\_of} D[\text{OF } p(1)]$ 
    define  $p'$  where  $p' = \{(x,k) \mid x \ k. \exists i \ l. x \in i \wedge i \in d \wedge (x,l) \in p \wedge k = i \cap$ 
 $l\}$ 
    have  $gp'$ :  $\gamma$  fine  $p'$ 
      using  $p(2)$  by (auto simp: p'_def fine_def)
    have  $p''$ :  $p'$  tagged_division_of (cbox  $a$   $b$ )
    proof (rule tagged_division_ofI)
      show finite  $p'$ 
      proof (rule finite_subset)
        show  $p' \subseteq (\lambda(k, x, l). (x, k \cap l)) \text{ `` } (d \times p)$ 

```

```

    by (force simp: p'_def image_iff)
  show finite (( $\lambda(k, x, l). (x, k \cap l)$ ) ' ( $d \times p$ ))
    by (simp add: d'(1) p'(1))
qed
next
fix x K
assume (x, K)  $\in p'$ 
then have  $\exists i l. x \in i \wedge i \in d \wedge (x, l) \in p \wedge K = i \cap l$ 
  unfolding p'_def by auto
then obtain i l where il:  $x \in i \wedge i \in d \wedge (x, l) \in p \wedge K = i \cap l$  by blast
show  $x \in K$  and  $K \subseteq \text{cbox } a b$ 
  using p'(2-3)[OF il(3)] il by auto
show  $\exists a b. K = \text{cbox } a b$ 
unfolding il using p'(4)[OF il(3)] d'(4)[OF il(2)] by (meson Int_interval)
next
fix x1 K1
assume (x1, K1)  $\in p'$ 
then have  $\exists i l. x1 \in i \wedge i \in d \wedge (x1, l) \in p \wedge K1 = i \cap l$ 
  unfolding p'_def by auto
then obtain i1 l1 where il1:  $x1 \in i1 \wedge i1 \in d \wedge (x1, l1) \in p \wedge K1 = i1 \cap l1$ 
by blast
fix x2 K2
assume (x2, K2)  $\in p'$ 
then have  $\exists i l. x2 \in i \wedge i \in d \wedge (x2, l) \in p \wedge K2 = i \cap l$ 
  unfolding p'_def by auto
then obtain i2 l2 where il2:  $x2 \in i2 \wedge i2 \in d \wedge (x2, l2) \in p \wedge K2 = i2 \cap l2$ 
by blast
assume (x1, K1)  $\neq (x2, K2)$ 
then have interior i1  $\cap$  interior i2 = {}  $\vee$  interior l1  $\cap$  interior l2 = {}
  using d'(5)[OF il1(2) il2(2)] p'(5)[OF il1(3) il2(3)] by (auto simp: il1
il2)
then show interior K1  $\cap$  interior K2 = {}
  unfolding il1 il2 by auto
next
have *:  $\forall (x, X) \in p'. X \subseteq \text{cbox } a b$ 
  unfolding p'_def using d' by blast
show  $\bigcup \{K. \exists x. (x, K) \in p'\} = \text{cbox } a b$ 
proof
  show  $\bigcup \{k. \exists x. (x, k) \in p'\} \subseteq \text{cbox } a b$ 
    using * by auto
next
show  $\text{cbox } a b \subseteq \bigcup \{k. \exists x. (x, k) \in p'\}$ 
proof
  fix y
  assume y:  $y \in \text{cbox } a b$ 
  obtain x L where xl:  $(x, L) \in p \wedge y \in L$ 
    using y unfolding p'(6)[symmetric] by auto
  obtain I where i:  $I \in d \wedge y \in I$ 
    using y unfolding d'(6)[symmetric] by auto

```

```

have x ∈ I
  using fineD[OF p(3) xl(1)] using k(2) i xl by auto
then show y ∈ ⋃ {K. ∃ x. (x, K) ∈ p'}
proof -
  obtain x l where xl: (x, l) ∈ p y ∈ l
    using y unfolding p'(6)[symmetric] by auto
  obtain i where i: i ∈ d y ∈ i
    using y unfolding d'(6)[symmetric] by auto
  have x ∈ i
    using fineD[OF p(3) xl(1)] using k(2) i xl by auto
  then show ?thesis
    unfolding p'_def by (rule_tac X=i ∩ l in UnionI) (use i xl in auto)
qed
qed
qed
qed
then have sum_less_e2: (∑ (x,K) ∈ p'. norm (content K *R f x - integral
K f)) < e/2
  using g(2) gp' tagged_division_of_def by blast

have in_p': (x, I ∩ L) ∈ p' if x: (x, L) ∈ p I ∈ d and y: y ∈ I y ∈ L
  for x I L y
proof -
  have x ∈ I
    using fineD[OF p(3) that(1)] k(2)[OF ⟨I ∈ d⟩] y by auto
  with x have (∃ i l. x ∈ i ∧ i ∈ d ∧ (x, l) ∈ p ∧ I ∩ L = i ∩ l)
    by blast
  then have (x, I ∩ L) ∈ p'
    by (simp add: p'_def)
  with y show ?thesis by auto
qed
moreover
have Ex_p_p': ∃ y i l. (x, K) = (y, i ∩ l) ∧ (y, l) ∈ p ∧ i ∈ d ∧ i ∩ l ≠ {}
  if xK: (x, K) ∈ p' for x K
proof -
  obtain i l where il: x ∈ i i ∈ d (x, l) ∈ p K = i ∩ l
    using xK unfolding p'_def by auto
  then show ?thesis
    using p'(2) by fastforce
qed
ultimately have p'alt: p' = {(x, I ∩ L) | x I L. (x, L) ∈ p ∧ I ∈ d ∧ I ∩ L
≠ {}}
  by auto
have sum_p': (∑ (x,K) ∈ p'. norm (integral K f)) = (∑ k ∈ snd ' p'. norm
(integral k f))
proof (rule sum.over_tagged_division_lemma[OF p''])
  show ⋀ u v. box u v = {} ⟹ norm (integral (cbox u v) f) = 0
    by (auto intro: integral_null simp: content_eq_0_interior)
qed

```

```

have snd_p_div: snd ' p division_of cbox a b
  by (rule division_of_tagged_division[OF p(1)])
note snd_p = division_ofD[OF snd_p_div]
have fin_d_sndp: finite (d × snd ' p)
  by (simp add: d'(1) snd_p(1))

have *:  $\bigwedge sni sni' sf sf'. [|sf' - sni'| < e/2; ?S - e/2 < sni; sni' \leq ?S; sni \leq sni'; sf' = sf] \implies |sf - ?S| < e$ 
  by arith
show norm (( $\sum (x,k) \in p. \text{content } k *_R \text{norm } (f x)$ ) - ?S) < e
  unfolding real_norm_def
proof (rule *)
show |(  $\sum (x,K) \in p'. \text{norm } (\text{content } K *_R f x)$ ) - ( $\sum (x,k) \in p'. \text{norm } (\text{integral } k f)$ )| < e/2
  using p'' sum_less_e2 unfolding split_def by (force intro!: absd-iff_norm_less)
show ( $\sum (x,k) \in p'. \text{norm } (\text{integral } k f)$ )  $\leq ?S$ 
  by (auto simp: sum_p' division_of_tagged_division[OF p''] D intro!: cSUP_upper)
show ( $\sum k \in d. \text{norm } (\text{integral } k f)$ )  $\leq (\sum (x,k) \in p'. \text{norm } (\text{integral } k f))$ 
proof -
have *:  $\{k \cap l \mid k \cap l. k \in d \wedge l \in \text{snd ' p}\} = (\lambda(k,l). k \cap l) ' (d \times \text{snd ' p})$ 
  by auto
have ( $\sum K \in d. \text{norm } (\text{integral } K f)$ )  $\leq (\sum i \in d. \sum l \in \text{snd ' p}. \text{norm } (\text{integral } (i \cap l) f))$ 
proof (rule sum_mono)
fix K assume k:  $K \in d$ 
from d'(4)[OF this] obtain u v where uv:  $K = \text{cbox } u v$  by metis
define d' where  $d' = \{\text{cbox } u v \cap l \mid l. l \in \text{snd ' p} \wedge \text{cbox } u v \cap l \neq \{\}\}$ 
have uvab:  $\text{cbox } u v \subseteq \text{cbox } a b$ 
  using d(1) k uv by blast
have d'_div:  $d' \text{ division\_of cbox } u v$ 
  unfolding d'_def by (rule division_inter_1 [OF snd_p_div uvab])
moreover have norm ( $\sum i \in d'. \text{integral } i f$ )  $\leq (\sum k \in d'. \text{norm } (\text{integral } k f))$ 
  by (simp add: sum_norm_le)
moreover have f integrable_on K
  using f integrable_on_subcbox uv uvab by blast
moreover have d' division_of K
  using d'_div uv by blast
ultimately have norm ( $\text{integral } K f$ )  $\leq \text{sum } (\lambda k. \text{norm } (\text{integral } k f)) d'$ 
  by (simp add: integral_combine_division_topdown)
also have ... = ( $\sum I \in \{K \cap L \mid L. L \in \text{snd ' p}\}. \text{norm } (\text{integral } I f)$ )
proof (rule sum_mono_neutral_left)
show finite  $\{K \cap L \mid L. L \in \text{snd ' p}\}$ 
  by (simp add: snd_p(1))
show  $\forall i \in \{K \cap L \mid L. L \in \text{snd ' p}\} - d'. \text{norm } (\text{integral } i f) = 0$ 
   $d' \subseteq \{K \cap L \mid L. L \in \text{snd ' p}\}$ 
  using d'_def image_eqI uv by auto

```

```

qed
also have ... = (∑ l ∈ snd ' p. norm (integral (K ∩ l) f))
  unfolding Setcompr_eq_image
proof (rule sum.reindex_nontrivial [unfolded o_def])
  show finite (snd ' p)
    using snd_p(1) by blast
  show norm (integral (K ∩ l) f) = 0
    if l ∈ snd ' p y ∈ snd ' p l ≠ y K ∩ l = K ∩ y for l y
  proof -
    have interior (K ∩ l) ⊆ interior (l ∩ y)
      by (metis Int_lower2 interior_mono le_inf_iff that(4))
    then have interior (K ∩ l) = {}
      by (simp add: snd_p(5) that)
    moreover from d'(4)[OF k] snd_p(4)[OF that(1)]
    obtain u1 v1 u2 v2
      where uv: K = cbox u1 u2 l = cbox v1 v2 by metis
    ultimately show ?thesis
      using that integral_null
      unfolding uv Int_interval content_eq_0_interior
      by (metis (mono_tags, lifting) norm_eq_zero)
  qed
qed
finally show norm (integral K f) ≤ (∑ l ∈ snd ' p. norm (integral (K ∩
l) f)) .
qed
also have ... = (∑ (i,l) ∈ d × snd ' p. norm (integral (i ∩ l) f))
  by (simp add: sum.cartesian_product)
also have ... = (∑ x ∈ d × snd ' p. norm (integral (case_prod (∩) x) f))
  by (force simp: split_def intro!: sum.cong)
also have ... = (∑ k ∈ {i ∩ l | i l. i ∈ d ∧ l ∈ snd ' p}. norm (integral k
f))
proof -
  have eq0: (integral (l1 ∩ k1) f) = 0
    if l1 ∩ k1 = l2 ∩ k2 (l1, k1) ≠ (l2, k2)
      l1 ∈ d (j1,k1) ∈ p l2 ∈ d (j2,k2) ∈ p
    for l1 l2 k1 k2 j1 j2
  proof -
    obtain u1 v1 u2 v2 where uv: l1 = cbox u1 u2 k1 = cbox v1 v2
      using ⟨(j1, k1) ∈ p⟩ ⟨l1 ∈ d⟩ d'(4) p'(4) by blast
    have l1 ≠ l2 ∨ k1 ≠ k2
      using that by auto
    then have interior k1 ∩ interior k2 = {} ∨ interior l1 ∩ interior l2
      = {}

    by (meson d'(5) old.prod.inject p'(5) that(3) that(4) that(5) that(6))
    moreover have interior (l1 ∩ k1) = interior (l2 ∩ k2)
      by (simp add: that(1))
    ultimately have interior(l1 ∩ k1) = {}
      by auto
    then show ?thesis

```



```

      unfolding uv Int_interval content_eq_0_interior[symmetric] by auto
    qed
  show ?thesis
    unfolding *
      apply (rule sum.reindex_nontrivial [OF fin_d_sndp, symmetric,
unfolded o_def])
      apply clarsimp
      by (metis eq0 fst_conv snd_conv)
    qed
  also have ... = ( $\sum (x,k) \in p'. \text{norm } (\text{integral } k \ f)$ )
    unfolding sum_p'
  proof (rule sum.mono_neutral_right)
    show finite { $i \cap l \mid i \in d \wedge l \in \text{snd } 'p\}$ 
      by (metis * finite_imageI [OF fin_d_sndp])
    show  $\text{snd } 'p' \subseteq \{i \cap l \mid i \in d \wedge l \in \text{snd } 'p\}$ 
      by (clarsimp simp: p'_def) (metis image_eqI snd_conv)
    show  $\forall i \in \{i \cap l \mid i \in d \wedge l \in \text{snd } 'p\} - \text{snd } 'p'. \text{norm } (\text{integral } i \ f)$ 
      = 0
      by clarsimp (metis Henstock_Kurzweil_Integration.integral_empty
disjoint_iff image_eqI in_p' snd_conv)
    qed
  finally show ?thesis .
  qed
  show ( $\sum (x,k) \in p'. \text{norm } (\text{content } k *_R f \ x)$ ) = ( $\sum (x,k) \in p. \text{content } k *_R$ 
norm (f x))
  proof -
    let ?S = {(x, i  $\cap$  l)  $\mid$  x i l. (x, l)  $\in$  p  $\wedge$  i  $\in$  d}
    have *: ?S = ( $\lambda (x,l,i). (\text{fst } xl, \text{snd } xl \cap i)$ ) ' (p  $\times$  d)
      by force
    have fin_pd: finite (p  $\times$  d)
      using finite_cartesian_product [OF p'(1) d'(1)] by metis
    have ( $\sum (x,k) \in p'. \text{norm } (\text{content } k *_R f \ x)$ ) = ( $\sum (x,k) \in ?S. |\text{content}$ 
k| * norm (f x))
      unfolding norm_scaleR
    proof (rule sum.mono_neutral_left)
      show finite {(x, i  $\cap$  l)  $\mid$  x i l. (x, l)  $\in$  p  $\wedge$  i  $\in$  d}
        by (simp add: * fin_pd)
      qed (use p'_alt in <force+>)
    also have ... = ( $\sum ((x,l),i) \in p \times d. |\text{content } (l \cap i)| * \text{norm } (f \ x)$ )
    proof -
      have  $|\text{content } (l1 \cap k1)| * \text{norm } (f \ x1) = 0$ 
        if (x1, l1)  $\in$  p (x2, l2)  $\in$  p k1  $\in$  d k2  $\in$  d
          x1 = x2 l1  $\cap$  k1 = l2  $\cap$  k2 x1  $\neq$  x2  $\vee$  l1  $\neq$  l2  $\vee$  k1  $\neq$  k2
        for x1 l1 k1 x2 l2 k2
      proof -
        obtain u1 v1 u2 v2 where uv: k1 = cbox u1 u2 l1 = cbox v1 v2
          by (meson <(x1, l1)  $\in$  p> <k1  $\in$  d> d(1) division_ofD(4) p'(4))
        have l1  $\neq$  l2  $\vee$  k1  $\neq$  k2
          using that by auto

```

```

then have interior k1 ∩ interior k2 = {} ∨ interior l1 ∩ interior l2
= {}
  using that p'(5) d'(5) by (metis snd_conv)
moreover have interior (l1 ∩ k1) = interior (l2 ∩ k2)
  unfolding that ..
ultimately have interior (l1 ∩ k1) = {}
  by auto
then show |content (l1 ∩ k1)| * norm (f x1) = 0
  unfolding uv Int_interval content_eq_0_interior[symmetric] by auto
qed
then show ?thesis
  unfolding *
  apply (subst sum.reindex_nontrivial [OF fin_pd])
  unfolding split_paired_all o_def split_def prod.inject
  by force+
qed
also have ... = (∑ (x,k) ∈ p. content k *R norm (f x))
proof -
  have sumeq: (∑ i ∈ d. content (l ∩ i) * norm (f x)) = content l * norm
(f x)
    if (x, l) ∈ p for x l
  proof -
    note xl = p'(2-4)[OF that]
    then obtain u v where uv: l = cbox u v by blast
    have (∑ i ∈ d. |content (l ∩ i)|) = (∑ k ∈ d. content (k ∩ cbox u v))
      by (simp add: Int_commute uv)
    also have ... = sum content {k ∩ cbox u v | k. k ∈ d}
    proof -
      have eq0: content (k ∩ cbox u v) = 0
        if k ∈ d y ∈ d k ≠ y and eq: k ∩ cbox u v = y ∩ cbox u v for k y
      proof -
        from d'(4)[OF that(1)] d'(4)[OF that(2)]
        obtain α β where α: k ∩ cbox u v = cbox α β
          by (meson Int_interval)
        have {} = interior ((k ∩ y) ∩ cbox u v)
          by (simp add: d'(5) that)
        also have ... = interior (y ∩ (k ∩ cbox u v))
          by auto
        also have ... = interior (k ∩ cbox u v)
          unfolding eq by auto
        finally show ?thesis
          unfolding α content_eq_0_interior ..
      qed
    then show ?thesis
      unfolding Setcompr_eq_image
      by (fastforce intro: sum.reindex_nontrivial [OF ‹finite d›, unfolded
o_def, symmetric])
  qed
  also have ... = sum content {cbox u v ∩ k | k. k ∈ d ∧ cbox u v ∩ k

```

```

≠ {}
  proof (rule sum.mono_neutral_right)
    show finite {k ∩ cbox u v | k. k ∈ d}
      by (simp add: d'(1))
    qed (fastforce simp: inf.commute)+
    finally have (∑ i ∈ d. |content (l ∩ i)|) = content (cbox u v)
      using additive_content_division[OF division_inter_1[OF d(1)]] uv
xl(2) by auto
    then show (∑ i ∈ d. content (l ∩ i) * norm (f x)) = content l * norm
(f x)
      unfolding sum_distrib_right[symmetric] using uv by auto
    qed
    show ?thesis
      by (subst sum_Sigma_product[symmetric]) (auto intro!: sumeq sum.cong
p' d')
    qed
    finally show ?thesis .
  qed
  qed (rule d)
  qed
  then show ?thesis
    using absolutely_integrable_onI [OF f has_integral_integrable] has_integral[of
_ ?S]
    by blast
  qed

```

```

lemma bounded_variation_absolutely_integrable:
  fixes f :: 'n::euclidean_space ⇒ 'm::euclidean_space
  assumes f_integrable_on UNIV
    and ∀ d. d division_of (∪ d) ⟶ sum (λk. norm (integral k f)) d ≤ B
  shows f_absolutely_integrable_on UNIV
proof (rule absolutely_integrable_onI, fact)
  let ?f = λD. ∑ k ∈ D. norm (integral k f) and ?D = {d. d division_of (∪ d)}
  define SDF where SDF ≡ SUP d ∈ ?D. ?f d
  have D_1: ?D ≠ {}
    by (rule elementary_interval) auto
  have D_2: bdd_above (?f ` ?D)
    using assms(2) by auto
  have f_int: ∧ a b. f absolutely_integrable_on cbox a b
    using assms integrable_on_subcbox
    by (blast intro!: bounded_variation_absolutely_integrable_interval)
  have ∃ B > 0. ∀ a b. ball 0 B ⊆ cbox a b ⟶
    |integral (cbox a b) (λx. norm (f x)) - SDF| < e
    if 0 < e for e
  proof -
    have ∃ y ∈ ?f ` ?D. ¬ y ≤ SDF - e
      proof (rule ccontr)

```

```

    assume  $\neg ?thesis$ 
    then have  $SDF \leq SDF - e$ 
      unfolding  $SDF\_def$ 
      by (metis (mono_tags)  $D\_1$   $cSUP\_least\_image\_eqI$ )
    then show  $False$ 
      using that by auto
  qed
then obtain  $d$   $K$  where  $ddiv$ :  $d$   $division\_of$   $\bigcup d$  and  $K = ?f\ d\ SDF - e < K$ 
  by (auto simp add:  $image\_iff\_not\_le$ )
then have  $d$ :  $SDF - e < ?f\ d$ 
  by auto
note  $d' = division\_ofD[OF\ ddiv]$ 
have bounded  $(\bigcup d)$ 
  using  $ddiv$  by blast
then obtain  $K$  where  $K$ :  $0 < K\ \forall x \in \bigcup d. norm\ x \leq K$ 
  using bounded_pos by blast
show ?thesis
proof (intro conjI impI allI exI)
  fix  $a\ b :: 'n$ 
  assume  $ab$ :  $ball\ 0\ (K + 1) \subseteq cbox\ a\ b$ 
  have *:  $\bigwedge s\ s1. \llbracket SDF - e < s1; s1 \leq s; s < SDF + e \rrbracket \implies |s - SDF| < e$ 
    by arith
  show  $|integral\ (cbox\ a\ b)\ (\lambda x. norm\ (f\ x)) - SDF| < e$ 
    unfolding  $real\_norm\_def$ 
  proof (rule * [OF  $d$ ])
    have  $?f\ d \leq sum\ (\lambda k. integral\ k\ (\lambda x. norm\ (f\ x)))\ d$ 
    proof (intro sum_mono)
      fix  $k$  assume  $k \in d$ 
      with  $d'(4)$   $f\_int$  show  $norm\ (integral\ k\ f) \leq integral\ k\ (\lambda x. norm\ (f\ x))$ 
    by (force simp:  $absolutely\_integrable\_on\_def\ integral\_norm\_bound\_integral$ )
    qed
    also have  $\dots = integral\ (\bigcup d)\ (\lambda x. norm\ (f\ x))$ 
      by (metis (full_types)  $absolutely\_integrable\_on\_def\ d'(4)\ ddiv\ f\_int$ 
 $integral\_combine\_division\_bottomup$ )
    also have  $\dots \leq integral\ (cbox\ a\ b)\ (\lambda x. norm\ (f\ x))$ 
    proof -
      have  $\bigcup d \subseteq cbox\ a\ b$ 
      using  $K(2)\ ab$  by fastforce
      then show ?thesis
        using  $integrable\_on\_subdivision[OF\ ddiv]\ f\_int[of\ a\ b]$  unfolding
 $absolutely\_integrable\_on\_def$ 
        by (auto intro!:  $integral\_subset\_le$ )
    qed
    finally show  $?f\ d \leq integral\ (cbox\ a\ b)\ (\lambda x. norm\ (f\ x))$  .
  next
    have  $e/2 > 0$ 
    using  $\langle e > 0 \rangle$  by auto
  moreover
  have  $f$ :  $f$   $integrable\_on\ cbox\ a\ b\ (\lambda x. norm\ (f\ x))\ integrable\_on\ cbox\ a\ b$ 

```

```

    using f_int by (auto simp: absolutely_integrable_on_def)
  ultimately obtain d1 where gauge d1
    and d1:  $\bigwedge p. \llbracket p \text{ tagged\_division\_of } (cbox\ a\ b); d1 \text{ fine } p \rrbracket \implies$ 
       $norm\ ((\sum (x,k) \in p. content\ k *_R norm\ (f\ x)) - integral\ (cbox\ a\ b)\ (\lambda x. norm\ (f\ x))) < e/2$ 
    unfolding has_integral_integral_has_integral by meson
  obtain d2 where gauge d2
    and d2:  $\bigwedge p. \llbracket p \text{ tagged\_partial\_division\_of } (cbox\ a\ b); d2 \text{ fine } p \rrbracket \implies$ 
       $(\sum (x,k) \in p. norm\ (content\ k *_R f\ x - integral\ k\ f)) < e/2$ 
    by (blast intro: Henstock_lemma [OF f(1)  $\langle e/2 > 0 \rangle$ ])
  obtain p where
    p:  $p \text{ tagged\_division\_of } (cbox\ a\ b)\ d1 \text{ fine } p\ d2 \text{ fine } p$ 
    by (rule fine_division_exists [OF gauge_Int [OF  $\langle gauge\ d1 \rangle \langle gauge\ d2 \rangle$ ],
of a b])
    (auto simp add: fine_Int)
  have *:  $\bigwedge sf\ sf'\ si\ di. \llbracket sf' = sf; si \leq SDF; |sf - si| < e/2; |sf' - di| < e/2 \rrbracket \implies di < SDF + e$ 
    by arith
  have integral (cbox a b)  $(\lambda x. norm\ (f\ x)) < SDF + e$ 
  proof (rule *)
    show  $|(\sum (x,k) \in p. norm\ (content\ k *_R f\ x)) - (\sum (x,k) \in p. norm\ (integral\ k\ f))| < e/2$ 
      unfolding split_def
      proof (rule absdiff_norm_less)
        show  $(\sum p \in p. norm\ (content\ (snd\ p) *_R f\ (fst\ p) - integral\ (snd\ p)\ f)) < e/2$ 
          using d2[of p] p(1,3) by (auto simp: tagged_division_of_def split_def)
      qed
      show  $|(\sum (x,k) \in p. content\ k *_R norm\ (f\ x)) - integral\ (cbox\ a\ b)\ (\lambda x. norm\ (f\ x))| < e/2$ 
        using d1[OF p(1,2)] by (simp only: real_norm_def)
        show  $(\sum (x,k) \in p. content\ k *_R norm\ (f\ x)) = (\sum (x,k) \in p. norm\ (content\ k *_R f\ x))$ 
          by (auto simp: split_paired_all sum.cong [OF refl])
        have  $(\sum (x,k) \in p. norm\ (integral\ k\ f)) = (\sum k \in snd\ ' p. norm\ (integral\ k\ f))$ 
          apply (rule sum.over_tagged_division_lemma[OF p(1)])
          by (metis Henstock_Kurzweil_Integration.integral_empty integral_open_interval norm_zero)
        also have  $\dots \leq SDF$ 
          using partial_division_of_tagged_division[of p cbox a b] p(1)
          by (auto simp: SDF_def tagged_partial_division_of_def intro!: cSUP_upper2 D_1 D_2)
        finally show  $(\sum (x,k) \in p. norm\ (integral\ k\ f)) \leq SDF$  .
      qed
    then show integral (cbox a b)  $(\lambda x. norm\ (f\ x)) < SDF + e$ 
      by simp
  qed
qed (use K in auto)

```

```

qed
moreover have  $\bigwedge a\ b. (\lambda x. \text{norm } (f\ x)) \text{ integrable\_on } \text{cbox } a\ b$ 
  using absolutely_integrable_on_def f_int by auto
ultimately
have  $((\lambda x. \text{norm } (f\ x)) \text{ has\_integral } SDF) \text{ UNIV}$ 
  by (auto simp: has_integral_alt')
then show  $(\lambda x. \text{norm } (f\ x)) \text{ integrable\_on } \text{UNIV}$ 
  by blast
qed

```

6.19.15 Outer and inner approximation of measurable sets by well-behaved sets.

proposition *measurable_outer_intervals_bounded:*

assumes $S \in \text{lmeasurable } S \subseteq \text{cbox } a\ b \ e > 0$

obtains $\mathcal{D}$

where countable $\mathcal{D}$

$\bigwedge K. K \in \mathcal{D} \implies K \subseteq \text{cbox } a\ b \wedge K \neq \{\}$ $\wedge (\exists c\ d. K = \text{cbox } c\ d)$
pairwise $(\lambda A\ B. \text{interior } A \cap \text{interior } B = \{\}) \ \mathcal{D}$
 $\bigwedge u\ v. \text{cbox } u\ v \in \mathcal{D} \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot i - a \cdot i) / 2^n$
 $\bigwedge K. \llbracket K \in \mathcal{D}; \text{box } a\ b \neq \{\} \rrbracket \implies \text{interior } K \neq \{\}$
 $S \subseteq \bigcup \mathcal{D} \cup \mathcal{D} \in \text{lmeasurable } \text{measure lebesgue } (\bigcup \mathcal{D}) \leq \text{measure lebesgue } S$

+ e

proof (*cases* $\text{box } a\ b = \{\}$)

case *True*

show *?thesis*

proof (*cases* $\text{cbox } a\ b = \{\}$)

case *True*

with *assms* **have** $[simp]: S = \{\}$

by *auto*

show *?thesis*

proof

show *countable* $\{\}$

by *simp*

qed (*use* $\langle e > 0 \rangle$ *in* *auto*)

next

case *False*

show *?thesis*

proof

show *countable* $\{\text{cbox } a\ b\}$

by *simp*

show $\bigwedge u\ v. \text{cbox } u\ v \in \{\text{cbox } a\ b\} \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot i - a \cdot i) / 2^n$

using *False* **by** (*force* *simp: eq_cbox intro: exI* [**where** $x=0$])

show $\text{measure lebesgue } (\bigcup \{\text{cbox } a\ b\}) \leq \text{measure lebesgue } S + e$

using *assms* **by** (*simp* *add: sum_content.box_empty_imp* [*OF* *True*])

qed (*use* *assms* $\langle \text{cbox } a\ b \neq \{\} \rangle$ *in* *auto*)

qed

next

```

case False
let ?μ = measure lebesgue
have S ∩ cbox a b ∈ lmeasurable
  using ⟨S ∈ lmeasurable⟩ by blast
then have indS_int: (indicator S has_integral (?μ S)) (cbox a b)
  by (metis integral_indicator ⟨S ⊆ cbox a b⟩ has_integral_integrable_integral
inf.orderE integrable_on_indicator)
with ⟨e > 0⟩ obtain γ where gauge γ and γ:
  ⋀D. [D tagged_division_of (cbox a b); γ fine D] ⇒ norm ((∑ (x,K)∈D.
content(K) *R indicator S x) - ?μ S) < e
  by (force simp: has_integral)
have inteq: integral (cbox a b) (indicator S) = integral UNIV (indicator S)
  using assms by (metis has_integral_iff indS_int lmeasure_integral_UNIV)
obtain D where D: countable D ∪ D ⊆ cbox a b
  and cbox: ⋀K. K ∈ D ⇒ interior K ≠ {} ∧ (∃ c d. K = cbox c d)
  and djointish: pairwise (λA B. interior A ∩ interior B = {}) D
  and covered: ⋀K. K ∈ D ⇒ ∃ x ∈ S ∩ K. K ⊆ γ x
  and close: ⋀u v. cbox u v ∈ D ⇒ ∃ n. ∀ i ∈ Basis. v·i - u·i = (b·i -
a·i)/2n
  and covers: S ⊆ ⋃D
  using covering_lemma [of S a b γ] ⟨gauge γ⟩ ⟨box a b ≠ {}⟩ assms by force
show ?thesis
proof
show ⋀K. K ∈ D ⇒ K ⊆ cbox a b ∧ K ≠ {} ∧ (∃ c d. K = cbox c d)
  by (meson Sup_le_iff D(2) cbox_interior_empty)
have negl_int: negligible(K ∩ L) if K ∈ D L ∈ D K ≠ L for K L
proof -
  have interior K ∩ interior L = {}
    using djointish pairwiseD that by fastforce
  moreover obtain u v x y where K = cbox u v L = cbox x y
    using cbox ⟨K ∈ D⟩ ⟨L ∈ D⟩ by blast
  ultimately show ?thesis
    by (simp add: Int_interval box_Int_box negligible_interval(1))
qed
have fincase: ⋃F ∈ lmeasurable ∧ ?μ (⋃F) ≤ ?μ S + e if finite F F ⊆ D
for F
proof -
  obtain t where t: ⋀K. K ∈ F ⇒ t K ∈ S ∩ K ∧ K ⊆ γ(t K)
    using covered ⟨F ⊆ D⟩ subsetD by metis
  have ∀ K ∈ F. ∀ L ∈ F. K ≠ L ⟶ interior K ∩ interior L = {}
    using that djointish by (simp add: pairwise_def) (metis subsetD)
  with cbox that D have Fdiv: F division_of (⋃F)
    by (fastforce simp: division_of_def dest: cbox)
  then have 1: ⋃F ∈ lmeasurable
    by blast
  have norme: ⋀p. [p tagged_division_of cbox a b; γ fine p]
    ⇒ norm ((∑ (x,K)∈p. content K * indicator S x) - integral (cbox a b)
(indicator S)) < e
    by (auto simp: lmeasure_integral_UNIV assms inteq dest: γ)

```

```

have  $\forall x K y L. (x, K) \in (\lambda K. (t K, K)) \text{ ' } \mathcal{F} \wedge (y, L) \in (\lambda K. (t K, K)) \text{ ' } \mathcal{F} \wedge$ 
 $(x, K) \neq (y, L) \longrightarrow \text{interior } K \cap \text{interior } L = \{\}$ 
using that djointish by (clarsimp simp: pairwise_def) (metis subsetD)
with that D have tagged:  $(\lambda K. (t K, K)) \text{ ' } \mathcal{F} \text{ tagged\_partial\_division\_of}$ 
cbox a b
by (auto simp: tagged_partial_division_of_def dest: t cbox)
have fine:  $\gamma \text{ fine } (\lambda K. (t K, K)) \text{ ' } \mathcal{F}$ 
using t by (auto simp: fine_def)
have *:  $y \leq ?\mu S \implies |x - y| \leq e \implies x \leq ?\mu S + e$  for  $x y$ 
by arith
have  $?\mu (\bigcup \mathcal{F}) \leq ?\mu S + e$ 
proof (rule *)
have  $(\sum K \in \mathcal{F}. ?\mu (K \cap S)) = ?\mu (\bigcup C \in \mathcal{F}. C \cap S)$ 
proof (rule measure_negligible_finite_Union_image [OF <finite F>, sym-
metric])
show  $\bigwedge K. K \in \mathcal{F} \implies K \cap S \in \text{lmeasurable}$ 
using F div <S ∈ lmeasurable> by blast
show pairwise  $(\lambda K y. \text{negligible } (K \cap S \cap (y \cap S))) \mathcal{F}$ 
unfolding pairwise_def
by (metis inf.commute inf_sup_aci(3) negligible_Int subsetCE negl_int
 $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$ )
qed
also have  $\dots = ?\mu (\bigcup \mathcal{F} \cap S)$ 
by simp
also have  $\dots \leq ?\mu S$ 
by (simp add: 1 <S ∈ lmeasurable> fmeasurableD measure_mono_fmeasurable
sets.Int)
finally show  $(\sum K \in \mathcal{F}. ?\mu (K \cap S)) \leq ?\mu S$  .
next
have  $?\mu (\bigcup \mathcal{F}) = \text{sum } ?\mu \mathcal{F}$ 
by (metis F div content_division)
also have  $\dots = (\sum K \in \mathcal{F}. \text{content } K)$ 
using F div by (force intro: sum.cong)
also have  $\dots = (\sum x \in \mathcal{F}. \text{content } x * \text{indicator } S (t x))$ 
using t by auto
finally have eq1:  $?\mu (\bigcup \mathcal{F}) = (\sum x \in \mathcal{F}. \text{content } x * \text{indicator } S (t x))$  .
have eq2:  $(\sum K \in \mathcal{F}. ?\mu (K \cap S)) = (\sum K \in \mathcal{F}. \text{integral } K (\text{indicator } S))$ 
apply (rule sum.cong [OF refl])
by (metis integral_indicator F div <S ∈ lmeasurable> division_ofD(4)
fmeasurable.Int inf.commute lmeasurable_cbox)
have  $|\sum (x, K) \in (\lambda K. (t K, K)) \text{ ' } \mathcal{F}. \text{content } K * \text{indicator } S x - \text{integral } K$ 
 $(\text{indicator } S)| \leq e$ 
using Henstock_lemma_part1 [of indicator S::'a⇒real, OF _ <e > 0>
 $\langle \text{gauge } \gamma \rangle \text{ _ tagged fine}]$ 
indS_int norme by auto
then show  $|\sum ?\mu (\bigcup \mathcal{F}) - (\sum K \in \mathcal{F}. ?\mu (K \cap S))| \leq e$ 
by (simp add: eq1 eq2 comm_monoid_add_class.sum.reindex inj_on_def
sum_subtractf)
qed

```



```

    with 1 show ?thesis by blast
  qed
  have  $\bigcup \mathcal{D} \in \text{lmeasurable} \wedge ?\mu (\bigcup \mathcal{D}) \leq ?\mu S + e$ 
  proof (cases finite  $\mathcal{D}$ )
    case True
    with fincase show ?thesis
    by blast
  next
    case False
    let ?T = from_nat_into  $\mathcal{D}$ 
    have T: bij_betw ?T UNIV  $\mathcal{D}$ 
    by (simp add: False  $\mathcal{D}(1)$  bij_betw_from_nat_into)
    have TM:  $\bigwedge n. ?T n \in \text{lmeasurable}$ 
    by (metis False cbox finite.emptyI from_nat_into lmeasurable_cbox)
    have TN:  $\bigwedge m n. m \neq n \implies \text{negligible} (?T m \cap ?T n)$ 
    by (simp add: False  $\mathcal{D}(1)$  from_nat_into infinite_imp_nonempty negl_int)
    have TB:  $(\sum_{k \leq n}. ?\mu (?T k)) \leq ?\mu S + e$  for  $n$ 
    proof -
      have  $(\sum_{k \leq n}. ?\mu (?T k)) = ?\mu (\bigcup (?T ' \{..n\}))$ 
      by (simp add: pairwise_def TM TN measure_negligible_finite_Union_image)
      also have  $?\mu (\bigcup (?T ' \{..n\})) \leq ?\mu S + e$ 
      using fincase [of ?T '  $\{..n\}$ ] T by (auto simp: bij_betw_def)
      finally show ?thesis .
    qed
    have  $\bigcup \mathcal{D} \in \text{lmeasurable}$ 
    by (metis lmeasurable_compact T  $\mathcal{D}(2)$  bij_betw_def cbox compact_cbox
    countable_Un_Int(1) fmeasurableD fmeasurableI2 rangeI)
    moreover
    have  $?\mu (\bigcup x. \text{from\_nat\_into } \mathcal{D} x) \leq ?\mu S + e$ 
    proof (rule measure_countable_Union_le [OF TM])
      show  $?\mu (\bigcup_{x \leq n}. \text{from\_nat\_into } \mathcal{D} x) \leq ?\mu S + e$  for  $n$ 
      by (metis (mono_tags, lifting) False fincase finite.emptyI finite_atMost
      finite_imageI from_nat_into imageE subsetI)
    qed
    ultimately show ?thesis by (metis T bij_betw_def)
  qed
  then show  $\bigcup \mathcal{D} \in \text{lmeasurable measure lebesgue } (\bigcup \mathcal{D}) \leq ?\mu S + e$  by blast+
  qed (use  $\mathcal{D}$  cbox djointish close covers in auto)
qed

```

6.19.16 Transformation of measure by linear maps

lemma *emeasure_lebesgue_ball_conv_unit_ball*:

fixes $c :: 'a :: \text{euclidean\_space}$

assumes $r \geq 0$

shows $\text{emeasure lebesgue } (\text{ball } c r) =$

$\text{ennreal } (r \wedge \text{DIM}('a)) * \text{emeasure lebesgue } (\text{ball } (0 :: 'a) 1)$

proof (cases $r = 0$)

case *False*

```

with assms have r:  $r > 0$  by auto
have emeasure lebesgue (( $\lambda x. c + x$ ) ' ( $\lambda x. r *_{\mathbb{R}} x$ ) ' ball (0 :: 'a) 1) =
   $r \wedge \text{DIM}('a) * \text{emeasure lebesgue (ball (0 :: 'a) 1)}$ 
  unfolding image_image using emeasure_lebesgue_affine[of r c ball 0 1] assms
  by (simp add: add_ac)
also have ( $\lambda x. r *_{\mathbb{R}} x$ ) ' ball 0 1 = ball (0 :: 'a) r
  using r by (subst ball_scale) auto
also have ( $\lambda x. c + x$ ) ' ... = ball c r
  by (subst image_add_ball) (simp_all add: algebra_simps)
finally show ?thesis by simp
qed auto

```

```

lemma content_ball_conv_unit_ball:
  fixes c :: 'a :: euclidean_space
  assumes  $r \geq 0$ 
  shows content (ball c r) =  $r \wedge \text{DIM}('a) * \text{content (ball (0 :: 'a) 1)}$ 
proof -
  have ennreal (content (ball c r)) = emeasure lebesgue (ball c r)
    using emeasure_lborel_ball_finite[of c r] by (subst emeasure_eq_ennreal_measure)
  auto
  also have ... = ennreal ( $r \wedge \text{DIM}('a) * \text{emeasure lebesgue (ball (0 :: 'a) 1)}$ )
    using assms by (intro emeasure_lebesgue_ball_conv_unit_ball) auto
  also have ... = ennreal ( $r \wedge \text{DIM}('a) * \text{content (ball (0 :: 'a) 1)}$ )
    using emeasure_lborel_ball_finite[of 0 :: 'a 1] assms
    by (subst emeasure_eq_ennreal_measure) (auto simp: ennreal_mult')
  finally show ?thesis
    using assms by (subst (asm) ennreal_inj) auto
qed

```

```

lemma measurable_linear_image_interval:
  linear f  $\implies f$  ' (cbox a b)  $\in \text{lmeasurable}$ 
  by (metis bounded_linear_image linear_linear bounded_cbox closure_bounded_linear_image
    closure_cbox compact_closure lmeasurable_compact)

```

```

proposition measure_linear_sufficient:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'n
  assumes linear f and S:  $S \in \text{lmeasurable}$ 
  and im:  $\bigwedge a b. \text{measure lebesgue (f ' (cbox a b))} = m * \text{measure lebesgue (cbox a b)}$ 
  shows  $f$  '  $S \in \text{lmeasurable} \wedge m * \text{measure lebesgue } S = \text{measure lebesgue (f ' S)}$ 
  using le_less_linear [of 0 m]
proof
  assume  $m < 0$ 
  then show ?thesis
    using im [of 0 One] by auto
next
  assume  $m \geq 0$ 
  let ? $\mu$  = measure lebesgue
  show ?thesis

```

```

proof (cases inj f)
  case False
    then have  $?μ (f \text{ ` } S) = 0$ 
      using  $\langle \text{linear } f \rangle$  negligible_imp_measure0 negligible_linear_singular_image
by blast
    then have  $m * ?μ (\text{cbox } 0 \text{ (One)}) = 0$ 
      by (metis False  $\langle \text{linear } f \rangle$  cbox_borel content_unit in measure_completion
negligible_imp_measure0 negligible_linear_singular_image sets_lborel)
    then show ?thesis
      using  $\langle \text{linear } f \rangle$  negligible_linear_singular_image negligible_imp_measure0
False
      by (auto simp: lmeasurable_iff_has_integral negligible_UNIV)
  next
    case True
      then obtain  $h$  where linear  $h$  and  $hf: \bigwedge x. h (f x) = x$  and  $fh: \bigwedge x. f (h x)$ 
       $= x$ 
        using  $\langle \text{linear } f \rangle$  linear_injective_isomorphism by blast
        have  $fBS: (f \text{ ` } S) \in \text{lmeasurable} \wedge m * ?μ S = ?μ (f \text{ ` } S)$ 
          if bounded  $S$   $S \in \text{lmeasurable}$  for  $S$ 
        proof –
          obtain  $a b$  where  $S \subseteq \text{cbox } a b$ 
            using  $\langle \text{bounded } S \rangle$  bounded_subset_cbox_symmetric by metis
          have  $fUD: (f \text{ ` } \bigcup \mathcal{D}) \in \text{lmeasurable} \wedge ?μ (f \text{ ` } \bigcup \mathcal{D}) = (m * ?μ (\bigcup \mathcal{D}))$ 
            if countable  $\mathcal{D}$ 
            and  $\text{cbox}: \bigwedge K. K \in \mathcal{D} \implies K \subseteq \text{cbox } a b \wedge K \neq \{\} \wedge (\exists c d. K = \text{cbox } c$ 
d)
              and  $\text{intint}: \text{pairwise } (\lambda A B. \text{interior } A \cap \text{interior } B = \{\}) \mathcal{D}$ 
            for  $\mathcal{D}$ 
          proof –
            have  $\text{conv}: \bigwedge K. K \in \mathcal{D} \implies \text{convex } K$ 
              using  $\text{cbox convex\_box}(1)$  by blast
            have  $\text{neg}: \text{negligible } (g \text{ ` } K \cap g \text{ ` } L)$  if linear  $g$   $K \in \mathcal{D}$   $L \in \mathcal{D}$   $K \neq L$ 
              for  $K L$  and  $g :: 'n \Rightarrow 'n$ 
            proof (cases inj g)
              case True
                have negligible  $(\text{frontier}(g \text{ ` } K \cap g \text{ ` } L) \cup \text{interior}(g \text{ ` } K \cap g \text{ ` } L))$ 
                  proof (rule negligible_Un)
                    show negligible  $(\text{frontier } (g \text{ ` } K \cap g \text{ ` } L))$ 
                      by (simp add: negligible_convex_frontier convex_Int conv convex_linear_image that)
                  next
                    have  $\forall p N. \text{pairwise } p N = (\forall Na. (Na :: 'n \text{ set}) \in N \longrightarrow (\forall Nb. Nb \in N$ 
 $\wedge Na \neq Nb \longrightarrow p Na Nb))$ 
                      by (metis pairwise_def)
                    then have  $\text{interior } K \cap \text{interior } L = \{\}$ 
                      using  $\text{intint that}(2) \text{ that}(3) \text{ that}(4)$  by presburger
                    then show negligible  $(\text{interior } (g \text{ ` } K \cap g \text{ ` } L))$ 
                      by (metis True empty_imp_negligible image_Int image_empty interior_Int interior_injective_linear_image that(1))
              case False
                show negligible  $(g \text{ ` } K \cap g \text{ ` } L)$ 
                  by (metis False negligible_image)
            qed
        qed

```

```

    qed
    moreover have  $g \text{ ' } K \cap g \text{ ' } L \subseteq \text{frontier } (g \text{ ' } K \cap g \text{ ' } L) \cup \text{interior } (g \text{ ' } K \cap g \text{ ' } L)$ 
    by (metis Diff_partition Int_commute calculation closure_Un_frontier
    frontier_def inf.absorb_iff2 inf_bot_right inf_sup_absorb negligible_Un_eq open_interior
    open_not_negligible sup_commute)
    ultimately show ?thesis
    by (rule negligible_subset)
  next
  case False
  then show ?thesis
  by (simp add: negligible_Int negligible_linear_singular_image ⟨linear g⟩)
  qed
  have negf: negligible ( $(f \text{ ' } K) \cap (f \text{ ' } L)$ )
  and negid: negligible ( $K \cap L$ ) if  $K \in \mathcal{D}$   $L \in \mathcal{D}$   $K \neq L$  for  $K L$ 
  using neg [OF ⟨linear f⟩] neg [OF linear_id] that by auto
  show ?thesis
  proof (cases finite  $\mathcal{D}$ )
  case True
  then have  $?\mu (\bigcup x \in \mathcal{D}. f \text{ ' } x) = (\sum x \in \mathcal{D}. ?\mu (f \text{ ' } x))$ 
  using ⟨linear f⟩ cbox measurable_linear_image_interval negf
  by (blast intro: measure_negligible_finite_Union_image [unfolded pairwise_def])
  also have  $\dots = (\sum k \in \mathcal{D}. m * ?\mu k)$ 
  by (metis (no_types, lifting) cbox im sum.cong)
  also have  $\dots = m * ?\mu (\bigcup \mathcal{D})$ 
  unfolding sum_distrib_left [symmetric]
  by (metis True cbox lmeasurable_cbox measure_negligible_finite_Union
  [unfolded pairwise_def] negid)
  finally show ?thesis
  by (metis True ⟨linear f⟩ cbox image_Union fmeasurable.finite_UN
  measurable_linear_image_interval)
  next
  case False
  with ⟨countable  $\mathcal{D}$ ⟩ obtain  $X :: \text{nat} \Rightarrow 'n$  set where  $S$ :  $\text{bij\_betw } X \text{ UNIV } \mathcal{D}$ 
  using bij_betw_from_nat_into by blast
  then have eq:  $(\bigcup \mathcal{D}) = (\bigcup n. X n) (f \text{ ' } \bigcup \mathcal{D}) = (\bigcup n. f \text{ ' } X n)$ 
  by (auto simp: bij_betw_def)
  have meas:  $\bigwedge K. K \in \mathcal{D} \implies K \in \text{lmeasurable}$ 
  using cbox by blast
  with  $S$  have 1:  $\bigwedge n. X n \in \text{lmeasurable}$ 
  by (auto simp: bij_betw_def)
  have 2: pairwise  $(\lambda m n. \text{negligible } (X m \cap X n)) \text{ UNIV}$ 
  using  $S$  unfolding bij_betw_def pairwise_def by (metis injD negid
  range_eqI)
  have bounded  $(\bigcup \mathcal{D})$ 
  by (meson Sup_least bounded_cbox bounded_subset cbox)
  then have 3: bounded  $(\bigcup n. X n)$ 

```

```

      using S unfolding bij_betw_def by blast
    have  $(\bigcup n. X\ n) \in \text{lmeasurable}$ 
      by (rule measurable_countable_negligible_Union_bounded [OF 1 2 3])
    with S have f1:  $\bigwedge n. f\ ' (X\ n) \in \text{lmeasurable}$ 
    unfolding bij_betw_def by (metis assms(1) cbox measurable_linear_image_interval
rangeI)
    have f2: pairwise  $(\lambda m\ n. \text{negligible } (f\ ' (X\ m) \cap f\ ' (X\ n)))\ \text{UNIV}$ 
      using S unfolding bij_betw_def pairwise_def by (metis injD negf
rangeI)
    have bounded  $(\bigcup \mathcal{D})$ 
      by (meson Sup_least bounded_cbox bounded_subset cbox)
    then have f3: bounded  $(\bigcup n. f\ ' X\ n)$ 
      using S unfolding bij_betw_def
      by (metis bounded_linear_image linear_linear assms(1) image_Union
range_composition)
    have  $(\lambda n. ?\mu\ (X\ n))\ \text{sums}\ ?\mu\ (\bigcup n. X\ n)$ 
      by (rule measure_countable_negligible_Union_bounded [OF 1 2 3])
    have meq:  $?\mu\ (\bigcup n. f\ ' X\ n) = m * ?\mu\ (\bigcup (X\ ' \text{UNIV}))$ 
    proof (rule sums_unique2 [OF measure_countable_negligible_Union_bounded
[OF f1 f2 f3]])
      have m:  $\bigwedge n. ?\mu\ (f\ ' X\ n) = (m * ?\mu\ (X\ n))$ 
        using S unfolding bij_betw_def by (metis cbox im rangeI)
      show  $(\lambda n. ?\mu\ (f\ ' X\ n))\ \text{sums}\ (m * ?\mu\ (\bigcup (X\ ' \text{UNIV})))$ 
        unfolding m
        using measure_countable_negligible_Union_bounded [OF 1 2 3]
sums_mult by blast
      qed
    show ?thesis
      using measurable_countable_negligible_Union_bounded [OF f1 f2 f3]
meq
      by (auto simp: eq [symmetric])
    qed
  qed
show ?thesis
  unfolding completion.fmeasurable_measure_inner_outer_le
proof (intro conjI allI impI)
  fix e :: real
  assume e > 0
  have 1:  $\text{cbox } a\ b - S \in \text{lmeasurable}$ 
    by (simp add: fmeasurable.Diff that)
  have 2:  $0 < e / (1 + |m|)$ 
    using  $\langle e > 0 \rangle$  by (simp add: field_split_simps abs_add_one_gt_zero)
  obtain  $\mathcal{D}$ 
  where countable  $\mathcal{D}$ 
    and cbox:  $\bigwedge K. K \in \mathcal{D} \implies K \subseteq \text{cbox } a\ b \wedge K \neq \{\} \wedge (\exists c\ d. K = \text{cbox } c\ d)$ 
    and intdisj: pairwise  $(\lambda A\ B. \text{interior } A \cap \text{interior } B = \{\})\ \mathcal{D}$ 
    and DD:  $\text{cbox } a\ b - S \subseteq \bigcup \mathcal{D} \cup \mathcal{D} \in \text{lmeasurable}$ 
    and le:  $?\mu\ (\bigcup \mathcal{D}) \leq ?\mu\ (\text{cbox } a\ b - S) + e / (1 + |m|)$ 

```

```

    by (rule measurable_outer_intervals_bounded [of cbox a b - S a b e / (1
+ |m|)]); use 1 2 pairwise_def in force)
  show  $\exists T \in \text{lmeasurable}. T \subseteq f' S \wedge m * ?\mu S - e \leq ?\mu T$ 
  proof (intro bexI conjI)
    show  $f' (cbox a b) - f' (\bigcup \mathcal{D}) \subseteq f' S$ 
    using  $\langle cbox a b - S \subseteq \bigcup \mathcal{D} \rangle$  by force
    have  $m * ?\mu S - e \leq m * (?\mu S - e / (1 + |m|))$ 
    using  $\langle m \geq 0 \rangle \langle e > 0 \rangle$  by (simp add: field_simps)
    also have  $\dots \leq ?\mu (f' cbox a b) - ?\mu (f' (\bigcup \mathcal{D}))$ 
    proof -
      have  $?\mu (cbox a b - S) = ?\mu (cbox a b) - ?\mu S$ 
      by (simp add: measurable_measure_Diff  $\langle S \subseteq cbox a b \rangle$  fmeasurableD
that(2))
      then have  $(?\mu S - e / (1 + m)) \leq (\text{content } (cbox a b) - ?\mu (\bigcup \mathcal{D}))$ 
      using  $\langle m \geq 0 \rangle$  le by auto
      then show ?thesis
      using  $\langle m \geq 0 \rangle \langle e > 0 \rangle$ 
      by (simp add: mult_left_mono in fUD [OF  $\langle \text{countable } \mathcal{D} \rangle$  cbox intdisj]
flip: right_diff_distrib)
    qed
    also have  $\dots = ?\mu (f' cbox a b - f' (\bigcup \mathcal{D}))$ 
    proof (rule measurable_measure_Diff [symmetric])
      show  $f' cbox a b \in \text{lmeasurable}$ 
      by (simp add: assms(1) measurable_linear_image_interval)
      show  $f' \bigcup \mathcal{D} \in \text{sets lebesgue}$ 
      by (simp add:  $\langle \text{countable } \mathcal{D} \rangle$  cbox fUD fmeasurableD intdisj)
      show  $f' \bigcup \mathcal{D} \subseteq f' cbox a b$ 
      by (simp add: Sup_le_iff cbox image_mono)
    qed
    finally show  $m * ?\mu S - e \leq ?\mu (f' cbox a b - f' (\bigcup \mathcal{D}))$  .
    show  $f' cbox a b - f' (\bigcup \mathcal{D}) \in \text{lmeasurable}$ 
    by (simp add: fUD  $\langle \text{countable } \mathcal{D} \rangle \langle \text{linear } f \rangle$  cbox fmeasurable.Diff intdisj
measurable_linear_image_interval)
  qed
next
fix e :: real
assume e > 0
have em:  $0 < e / (1 + |m|)$ 
using  $\langle e > 0 \rangle$  by (simp add: field_split_simps abs_add_one_gt_zero)
obtain  $\mathcal{D}$ 
where countable  $\mathcal{D}$ 
and cbox:  $\bigwedge K. K \in \mathcal{D} \implies K \subseteq cbox a b \wedge K \neq \{\}$   $\wedge (\exists c d. K = cbox$ 
c d)
and intdisj: pairwise  $(\lambda A B. \text{interior } A \cap \text{interior } B = \{\})$   $\mathcal{D}$ 
and DD:  $S \subseteq \bigcup \mathcal{D} \cup \mathcal{D} \in \text{lmeasurable}$ 
and le:  $?\mu (\bigcup \mathcal{D}) \leq ?\mu S + e / (1 + |m|)$ 
by (rule measurable_outer_intervals_bounded [of S a b e / (1 + |m|)]); use
 $\langle S \in \text{lmeasurable} \rangle \langle S \subseteq cbox a b \rangle$  em in force)
show  $\exists U \in \text{lmeasurable}. f' S \subseteq U \wedge ?\mu U \leq m * ?\mu S + e$ 

```

```

proof (intro bexI conjI)
  show  $f' S \subseteq f' (\bigcup \mathcal{D})$ 
    by (simp add: DD(1) image_mono)
  have  $? \mu (f' \bigcup \mathcal{D}) \leq m * (? \mu S + e / (1 + |m|))$ 
    using  $\langle m \geq 0 \rangle$  le_mult_left_mono
  by (auto simp: fUD  $\langle$ countable  $\mathcal{D} \rangle$   $\langle$ linear  $f \rangle$  cbox fmeasurable.Diff intdisj
measurable_linear_image_interval)
  also have  $\dots \leq m * ? \mu S + e$ 
    using  $\langle m \geq 0 \rangle \langle e > 0 \rangle$  by (simp add: fUD [OF  $\langle$ countable  $\mathcal{D} \rangle$  cbox
intdisj] field_simps)
  finally show  $? \mu (f' \bigcup \mathcal{D}) \leq m * ? \mu S + e$  .
  show  $f' \bigcup \mathcal{D} \in \text{lmeasurable}$ 
    by (simp add:  $\langle$ countable  $\mathcal{D} \rangle$  cbox fUD intdisj)
qed
qed
qed
show ?thesis
  unfolding has_measure_limit_iff
proof (intro allI impI)
  fix  $e :: \text{real}$ 
  assume  $e > 0$ 
  obtain  $B$  where  $B > 0$  and  $B$ :
     $\bigwedge a b. \text{ball } 0 B \subseteq \text{cbox } a b \implies |? \mu (S \cap \text{cbox } a b) - ? \mu S| < e / (1 + |m|)$ 
    using has_measure_limit [OF  $S$ ]  $\langle e > 0 \rangle$  by (metis abs_add_one_gt_zero
zero_less_divide_iff)
  obtain  $c d :: 'n$  where  $cd: \text{ball } 0 B \subseteq \text{cbox } c d$ 
    by (metis bounded_subset_cbox_symmetric bounded_ball)
  with  $B$  have less:  $|? \mu (S \cap \text{cbox } c d) - ? \mu S| < e / (1 + |m|)$  .
  obtain  $D$  where  $D > 0$  and  $D$ :  $\text{cbox } c d \subseteq \text{ball } 0 D$ 
    by (metis bounded_cbox_bounded_subset_ballD)
  obtain  $C$  where  $C > 0$  and  $C$ :  $\bigwedge x. \text{norm } (f x) \leq C * \text{norm } x$ 
    using linear_bounded_pos  $\langle$ linear  $f \rangle$  by blast
  have  $f' S \cap \text{cbox } a b \in \text{lmeasurable} \wedge$ 
     $|? \mu (f' S \cap \text{cbox } a b) - m * ? \mu S| < e$ 
    if  $\text{ball } 0 (D * C) \subseteq \text{cbox } a b$  for  $a b$ 
proof -
  have bounded  $(S \cap h' \text{cbox } a b)$ 
    by (simp add: bounded_linear_image linear_linear  $\langle$ linear  $h \rangle$  bounded_Int)
  moreover have Shab:  $S \cap h' \text{cbox } a b \in \text{lmeasurable}$ 
by (simp add:  $S \langle$ linear  $h \rangle$  fmeasurable.Int measurable_linear_image_interval)
  moreover have fim:  $f' (S \cap h' (\text{cbox } a b)) = (f' S) \cap \text{cbox } a b$ 
    by (auto simp: hf_rev_image_eqI fh)
  ultimately have 1:  $(f' S) \cap \text{cbox } a b \in \text{lmeasurable}$ 
    and 2:  $? \mu ((f' S) \cap \text{cbox } a b) = m * ? \mu (S \cap h' \text{cbox } a b)$ 
    using fBS [of  $S \cap (h' (\text{cbox } a b))$ ] by auto
  have *:  $\llbracket z - m \rrbracket < e; z \leq w; w \leq m \rrbracket \implies |w - m| \leq e$ 
    for  $w z m$  and  $e :: \text{real}$  by auto
  have meas_adiff:  $|? \mu (S \cap h' \text{cbox } a b) - ? \mu S| \leq e / (1 + |m|)$ 
proof (rule * [OF less])

```

```

show ?μ (S ∩ cbox c d) ≤ ?μ (S ∩ h ‘ cbox a b)
proof (rule measure_mono_fmeasurable [OF __ Shab])
  have f ‘ ball 0 D ⊆ ball 0 (C * D)
    using C ‘ C > 0 ›
    apply (clarsimp simp: algebra_simps)
by (meson le_less_trans linordered_comm_semiring_strict_class.comm_mult_strict_left_mon
then have f ‘ ball 0 D ⊆ cbox a b
  by (metis mult.commute order_trans that)
have ball 0 D ⊆ h ‘ cbox a b
  by (metis ‘f ‘ ball 0 D ⊆ cbox a b› hf_image_subset_iff subsetI)
then show S ∩ cbox c d ⊆ S ∩ h ‘ cbox a b
  using D by blast
next
  show S ∩ cbox c d ∈ sets lebesgue
  using S_fmeasurable_cbox by blast
qed
next
show ?μ (S ∩ h ‘ cbox a b) ≤ ?μ S
  by (simp add: S Shab_fmeasurableD measure_mono_fmeasurable)
qed
have |?μ (f ‘ S ∩ cbox a b) - m * ?μ S| ≤ |?μ S - ?μ (S ∩ h ‘ cbox a b)|
* m
  by (metis 2 ‘m ≥ 0› abs_minus_commute abs_mult_pos mult.commute
order_refl right_diff_distrib')
also have ... ≤ e / (1 + m) * m
  by (metis ‘m ≥ 0› abs_minus_commute abs_of_nonneg meas_adiff
mult.commute mult_left_mono)
also have ... < e
  using ‘e > 0› ‘m ≥ 0› by (simp add: field_simps)
finally have |?μ (f ‘ S ∩ cbox a b) - m * ?μ S| < e .
with 1 show ?thesis by auto
qed
then show ∃ B > 0. ∀ a b. ball 0 B ⊆ cbox a b ⟶
  f ‘ S ∩ cbox a b ∈ lmeasurable ∧
  |?μ (f ‘ S ∩ cbox a b) - m * ?μ S| < e
using ‘C > 0› ‘D > 0› by (metis mult_zero_left mult_less_iff1)
qed
qed
qed

```

6.19.17 Lemmas about absolute integrability

```

lemma absolutely_integrable_linear:
  fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
  and h :: 'n::euclidean_space ⇒ 'p::euclidean_space
  shows f absolutely_integrable_on s ⟹ bounded_linear h ⟹ (h ∘ f) absolutely_integrable_on s
  using integrable_bounded_linear[of h lebesgue λx. indicator s x *R f x]
  by (simp add: linear_simps[of h] set_integrable_def)

```


lemma *absolutely_integrable_sum*:

fixes $f :: 'a \Rightarrow 'n::\text{euclidean\_space} \Rightarrow 'm::\text{euclidean\_space}$
assumes *finite* T **and** $\bigwedge a. a \in T \implies (f\ a)\ \text{absolutely\_integrable\_on}\ S$
shows $(\lambda x. \text{sum } (\lambda a. f\ a\ x)\ T)\ \text{absolutely\_integrable\_on}\ S$
using *assms* **by** *induction auto*

lemma *absolutely_integrable_integrable_bound*:

fixes $f :: 'n::\text{euclidean\_space} \Rightarrow 'm::\text{euclidean\_space}$
assumes $le: \bigwedge x. x \in S \implies \text{norm } (f\ x) \leq g\ x$ **and** $f: f\ \text{integrable\_on}\ S$ **and** $g: g\ \text{integrable\_on}\ S$
shows $f\ \text{absolutely\_integrable\_on}\ S$
unfolding *set_integrable_def*
proof (*rule Bochner_Integration.integrable_bound*)
have $g\ \text{absolutely\_integrable\_on}\ S$
unfolding *absolutely_integrable_on_def*
proof
show $(\lambda x. \text{norm } (g\ x))\ \text{integrable\_on}\ S$
using *le norm_ge_zero[of f _]*
by (*intro integrable_spike_finite[OF _ _ g, of {}]*)
(auto intro!: abs_of_nonneg intro: order_trans simp del: norm_ge_zero)
qed fact
then show *integrable lebesgue* $(\lambda x. \text{indicat\_real}\ S\ x\ *_R\ g\ x)$
by (*simp add: set_integrable_def*)
show $(\lambda x. \text{indicat\_real}\ S\ x\ *_R\ f\ x) \in \text{borel\_measurable lebesgue}$
using f **by** (*auto intro: has_integral_implies_lebesgue_measurable simp: integrable_on_def*)
qed (*use le in <force intro!: always_eventually split: split_indicator>*)

corollary *absolutely_integrable_on_const* [*simp*]:

fixes $c :: 'a::\text{euclidean\_space}$
assumes $S \in \text{lmeasurable}$
shows $(\lambda x. c)\ \text{absolutely\_integrable\_on}\ S$
by (*metis (full_types) assms absolutely_integrable_integrable_bound integrable_on_const order_refl*)

lemma *absolutely_integrable_continuous*:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
shows *continuous_on* $(\text{cbox}\ a\ b)\ f \implies f\ \text{absolutely\_integrable\_on}\ \text{cbox}\ a\ b$
using *absolutely_integrable_integrable_bound*
by (*simp add: absolutely_integrable_on_def continuous_on_norm integrable_continuous*)

lemma *absolutely_integrable_continuous_real*:

fixes $f :: \text{real} \Rightarrow 'b::\text{euclidean\_space}$
shows *continuous_on* $\{a..b\}\ f \implies f\ \text{absolutely\_integrable\_on}\ \{a..b\}$
by (*metis absolutely_integrable_continuous box_real(2)*)

lemma *continuous_imp_integrable*:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$

```

    assumes continuous_on (cbox a b) f
    shows integrable (lebesgue_on (cbox a b)) f
  proof -
    have f absolutely_integrable_on cbox a b
      by (simp add: absolutely_integrable_continuous assms)
    then show ?thesis
      by (simp add: integrable_restrict_space set_integrable_def)
  qed

```

```

lemma continuous_imp_integrable_real:
  fixes f :: real  $\Rightarrow$  'b::euclidean_space
  assumes continuous_on {a..b} f
  shows integrable (lebesgue_on {a..b}) f
  by (metis assms continuous_imp_integrable interval_cbox)

```

6.19.18 Componentwise

```

proposition absolutely_integrable_componentwise_iff:
  shows f absolutely_integrable_on A  $\longleftrightarrow$  ( $\forall b \in \text{Basis}. (\lambda x. f\ x \cdot b)$  absolutely_integrable_on A)
  proof -
    have *: ( $\lambda x. \text{norm } (f\ x)$ ) integrable_on A  $\longleftrightarrow$  ( $\forall b \in \text{Basis}. (\lambda x. \text{norm } (f\ x \cdot b))$  integrable_on A) (is ?lhs = ?rhs)
    if f integrable_on A
    proof
      assume ?lhs
      then show ?rhs
        by (metis absolutely_integrable_on_def Topology_Euclidean_Space.norm_nth_le absolutely_integrable_integrable_bound integrable_component that)
    next
      assume R: ?rhs
      have f absolutely_integrable_on A
      proof (rule absolutely_integrable_integrable_bound)
        show ( $\lambda x. \sum i \in \text{Basis}. \text{norm } (f\ x \cdot i)$ ) integrable_on A
          using R by (force intro: integrable_sum)
      qed (use that norm_le_l1 in auto)
      then show ?lhs
        using absolutely_integrable_on_def by auto
    qed
  show ?thesis
    unfolding absolutely_integrable_on_def
    by (simp add: integrable_componentwise_iff [symmetric] ball_conj_distrib * cong: conj_cong)
  qed

```

```

lemma absolutely_integrable_componentwise:
  shows ( $\bigwedge b. b \in \text{Basis} \implies (\lambda x. f\ x \cdot b)$  absolutely_integrable_on A)  $\implies$  f absolutely_integrable_on A
  using absolutely_integrable_componentwise_iff by blast

```

lemma *absolutely_integrable_component*:

f *absolutely_integrable_on* $A \implies (\lambda x. f\ x \cdot (b :: 'b :: euclidean\_space))$ *absolutely_integrable_on* A
by (*drule* *absolutely_integrable_linear*[*OF* $\_\text{bounded\_linear\_inner\_left}$ [*of* b]])
(*simp* *add*: *o_def*)

lemma *absolutely_integrable_scaleR_left*:

fixes $f :: 'n :: euclidean\_space \Rightarrow 'm :: euclidean\_space$
assumes f *absolutely_integrable_on* S
shows $(\lambda x. c *_{\mathbb{R}} f\ x)$ *absolutely_integrable_on* S
proof $-$
have $(\lambda x. c *_{\mathbb{R}} x)$ *o* f *absolutely_integrable_on* S
by (*simp* *add*: *absolutely_integrable_linear* *assms* *bounded_linear_scaleR_right*)
then show *?thesis*
using *assms* **by** *blast*
qed

lemma *absolutely_integrable_scaleR_right*:

assumes f *absolutely_integrable_on* S
shows $(\lambda x. f\ x *_{\mathbb{R}} c)$ *absolutely_integrable_on* S
using *assms* **by** *blast*

lemma *absolutely_integrable_norm*:

fixes $f :: 'a :: euclidean\_space \Rightarrow 'b :: euclidean\_space$
assumes f *absolutely_integrable_on* S
shows $(\text{norm} \circ f)$ *absolutely_integrable_on* S
using *assms* **by** (*simp* *add*: *absolutely_integrable_on_def* *o_def*)

lemma *absolutely_integrable_abs*:

fixes $f :: 'a :: euclidean\_space \Rightarrow 'b :: euclidean\_space$
assumes f *absolutely_integrable_on* S
shows $(\lambda x. \sum_{i \in \text{Basis}} |f\ x \cdot i| *_{\mathbb{R}} i)$ *absolutely_integrable_on* S
(is *?g* *absolutely_integrable_on* S)
proof $-$
have $*$: $(\lambda y. \sum_{j \in \text{Basis}} \text{if } j = i \text{ then } y *_{\mathbb{R}} j \text{ else } 0) \circ$
 $(\lambda x. \text{norm} (\sum_{j \in \text{Basis}} \text{if } j = i \text{ then } (x \cdot i) *_{\mathbb{R}} j \text{ else } 0)) \circ f$
absolutely_integrable_on S
if $i \in \text{Basis}$ **for** i
proof $-$
have *bounded_linear* $(\lambda y. \sum_{j \in \text{Basis}} \text{if } j = i \text{ then } y *_{\mathbb{R}} j \text{ else } 0)$
by (*simp* *add*: *linear_linear_algebra_simps* *linearI*)
moreover have $(\lambda x. \text{norm} (\sum_{j \in \text{Basis}} \text{if } j = i \text{ then } (x \cdot i) *_{\mathbb{R}} j \text{ else } 0)) \circ f$
absolutely_integrable_on S
using *assms* $\langle i \in \text{Basis} \rangle$
unfolding *o_def*
by (*intro* *absolutely_integrable_norm* [*unfolded* *o_def*])
(auto simp: algebra_simps dest: absolutely_integrable_component)

```

    ultimately show ?thesis
    by (subst comp_assoc) (blast intro: absolutely_integrable_linear)
qed
have eq: ?g =
  (λx. ∑ i∈Basis. ((λy. ∑ j∈Basis. if j = i then y *R j else 0) ∘
    (λx. norm(∑ j∈Basis. if j = i then (x · i) *R j else 0)) ∘ f) x)
  by (simp)
show ?thesis
  unfolding eq
  by (rule absolutely_integrable_sum) (force simp: intro!: *)+
qed

```

```

lemma abs_absolutely_integrableI_1:
  fixes f :: 'a :: euclidean_space ⇒ real
  assumes f: f integrable_on A and (λx. |f x|) integrable_on A
  shows f absolutely_integrable_on A
  by (rule absolutely_integrable_integrable_bound [OF _ assms]) auto

```

```

lemma abs_absolutely_integrableI:
  assumes f: f integrable_on S and fcomp: (λx. ∑ i∈Basis. |f x · i| *R i) inte-
    grable_on S
  shows f absolutely_integrable_on S
proof -
  have (λx. (f x · i) *R i) absolutely_integrable_on S if i ∈ Basis for i
  proof -
    have (λx. |f x · i|) integrable_on S
    using assms integrable_component [OF fcomp, where y=i] that by simp
    then have (λx. f x · i) absolutely_integrable_on S
    using abs_absolutely_integrableI_1 f integrable_component by blast
    then show ?thesis
    by (rule absolutely_integrable_scaleR_right)
  qed
qed
then have (λx. ∑ i∈Basis. (f x · i) *R i) absolutely_integrable_on S
  by (simp add: absolutely_integrable_sum)
then show ?thesis
  by (simp add: euclidean_representation)
qed

```

```

lemma absolutely_integrable_abs_iff:
  f absolutely_integrable_on S ⟷
  f integrable_on S ∧ (λx. ∑ i∈Basis. |f x · i| *R i) integrable_on S
  (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
    using absolutely_integrable_abs absolutely_integrable_on_def by blast
next
  assume ?rhs

```

moreover
have $(\lambda x. \text{if } x \in S \text{ then } \sum_{i \in \text{Basis}} |f x \cdot i| *_R i \text{ else } 0) = (\lambda x. \sum_{i \in \text{Basis}} |(f x \in S \text{ then } f x \text{ else } 0) \cdot i| *_R i)$
by force
ultimately show ?lhs
by (simp only: absolutely_integrable_restrict_UNIV [of S, symmetric] integrable_restrict_UNIV [of S, symmetric] abs_absolutely_integrableI)
qed

lemma absolutely_integrable_max:

fixes $f :: 'n::\text{euclidean\_space} \Rightarrow 'm::\text{euclidean\_space}$
assumes $f \text{ absolutely\_integrable\_on } S \ g \text{ absolutely\_integrable\_on } S$
shows $(\lambda x. \sum_{i \in \text{Basis}} \max (f x \cdot i) (g x \cdot i) *_R i) \text{ absolutely\_integrable\_on } S$
proof –
have $(\lambda x. \sum_{i \in \text{Basis}} \max (f x \cdot i) (g x \cdot i) *_R i) =$
 $(\lambda x. (1/2) *_R (f x + g x + (\sum_{i \in \text{Basis}} |f x \cdot i - g x \cdot i| *_R i)))$
proof (rule ext)
fix x
have $(\sum_{i \in \text{Basis}} \max (f x \cdot i) (g x \cdot i) *_R i) = (\sum_{i \in \text{Basis}} ((f x \cdot i + g x \cdot i + |f x \cdot i - g x \cdot i|) / 2) *_R i)$
by (force intro: sum.cong)
also have $\dots = (1 / 2) *_R (\sum_{i \in \text{Basis}} (f x \cdot i + g x \cdot i + |f x \cdot i - g x \cdot i|) *_R i)$
by (simp add: scaleR_right.sum)
also have $\dots = (1 / 2) *_R (f x + g x + (\sum_{i \in \text{Basis}} |f x \cdot i - g x \cdot i| *_R i))$
by (simp add: sum.distrib algebra_simps euclidean_representation)
finally
show $(\sum_{i \in \text{Basis}} \max (f x \cdot i) (g x \cdot i) *_R i) =$
 $(1 / 2) *_R (f x + g x + (\sum_{i \in \text{Basis}} |f x \cdot i - g x \cdot i| *_R i)) .$
qed
moreover have $(\lambda x. (1 / 2) *_R (f x + g x + (\sum_{i \in \text{Basis}} |f x \cdot i - g x \cdot i| *_R i)))$
 $\text{absolutely\_integrable\_on } S$
using absolutely_integrable_abs [OF set_integral_diff(1) [OF assms]]
by (intro set_integral_add absolutely_integrable_scaleR_left assms) (simp add: algebra_simps)
ultimately show ?thesis **by** metis
qed

corollary absolutely_integrable_max_1:

fixes $f :: 'n::\text{euclidean\_space} \Rightarrow \text{real}$
assumes $f \text{ absolutely\_integrable\_on } S \ g \text{ absolutely\_integrable\_on } S$
shows $(\lambda x. \max (f x) (g x)) \text{ absolutely\_integrable\_on } S$
using absolutely_integrable_max [OF assms] **by** simp

lemma absolutely_integrable_min:

fixes $f :: 'n::\text{euclidean\_space} \Rightarrow 'm::\text{euclidean\_space}$
assumes $f \text{ absolutely\_integrable\_on } S \ g \text{ absolutely\_integrable\_on } S$

shows $(\lambda x. \sum_{i \in \text{Basis}} \min (f x \cdot i) (g x \cdot i) *_R i)$
 $\text{absolutely\_integrable\_on } S$
proof –
have $(\lambda x. \sum_{i \in \text{Basis}} \min (f x \cdot i) (g x \cdot i) *_R i) =$
 $(\lambda x. (1/2) *_R (f x + g x - (\sum_{i \in \text{Basis}} |f x \cdot i - g x \cdot i| *_R i)))$
proof (rule ext)
fix x
have $(\sum_{i \in \text{Basis}} \min (f x \cdot i) (g x \cdot i) *_R i) = (\sum_{i \in \text{Basis}} ((f x \cdot i + g x \cdot$
 $i - |f x \cdot i - g x \cdot i|) / 2) *_R i)$
by (force intro: sum.cong)
also have $\dots = (1 / 2) *_R (\sum_{i \in \text{Basis}} (f x \cdot i + g x \cdot i - |f x \cdot i - g x \cdot i|$
 $*_R i))$
by (simp add: scaleR_right.sum)
also have $\dots = (1 / 2) *_R (f x + g x - (\sum_{i \in \text{Basis}} |f x \cdot i - g x \cdot i| *_R i))$
by (simp add: sum.distrib sum_subtractf algebra_simps euclidean_representation)
finally
show $(\sum_{i \in \text{Basis}} \min (f x \cdot i) (g x \cdot i) *_R i) =$
 $(1 / 2) *_R (f x + g x - (\sum_{i \in \text{Basis}} |f x \cdot i - g x \cdot i| *_R i)) .$
qed
moreover have $(\lambda x. (1 / 2) *_R (f x + g x - (\sum_{i \in \text{Basis}} |f x \cdot i - g x \cdot i| *_R$
 $i)))$
 $\text{absolutely\_integrable\_on } S$
using $\text{absolutely\_integrable\_abs } [OF \text{ set\_integral\_diff}(1) [OF \text{ assms}]]$
by (intro set_integral_add set_integral_diff absolutely_integrable_scaleR_left
 $\text{assms})$
 $(\text{simp add: algebra\_simps})$
ultimately show ?thesis **by** metis
qed

corollary $\text{absolutely\_integrable\_min}_1$:

fixes $f :: 'n :: \text{euclidean\_space} \Rightarrow \text{real}$
assumes $f \text{ absolutely\_integrable\_on } S$ $g \text{ absolutely\_integrable\_on } S$
shows $(\lambda x. \min (f x) (g x)) \text{ absolutely\_integrable\_on } S$
using $\text{absolutely\_integrable\_min } [OF \text{ assms}]$ **by** simp

lemma $\text{nonnegative\_absolutely\_integrable}$:

fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$
assumes $f \text{ integrable\_on } A$ **and** $\text{comp: } \bigwedge x b. \llbracket x \in A; b \in \text{Basis} \rrbracket \implies 0 \leq f x \cdot b$
shows $f \text{ absolutely\_integrable\_on } A$

proof –

have $(\lambda x. (f x \cdot i) *_R i) \text{ absolutely\_integrable\_on } A$ **if** $i \in \text{Basis}$ **for** i

proof –

have $(\lambda x. f x \cdot i) \text{ integrable\_on } A$

by (simp add: assms(1) integrable_component)

then have $(\lambda x. f x \cdot i) \text{ absolutely\_integrable\_on } A$

by (metis that comp nonnegative_absolutely_integrable_1)

then show ?thesis

by (rule absolutely_integrable_scaleR_right)

qed

```

then have ( $\lambda x. \sum_{i \in \text{Basis}. (f\ x \cdot i) *_{\mathbb{R}} i}$ ) absolutely_integrable_on A
by (simp add: absolutely_integrable_sum)
then show ?thesis
by (simp add: euclidean_representation)
qed

```

lemma *absolutely_integrable_component_ubound*:

```

fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: euclidean_space
assumes f: f integrable_on A and g: g absolutely_integrable_on A
and comp:  $\bigwedge x\ b. \llbracket x \in A; b \in \text{Basis} \rrbracket \Longrightarrow f\ x \cdot b \leq g\ x \cdot b$ 
shows f absolutely_integrable_on A
proof -
have ( $\lambda x. g\ x - (g\ x - f\ x)$ ) absolutely_integrable_on A
proof (rule set_integral_diff [OF g nonnegative_absolutely_integrable])
show ( $\lambda x. g\ x - f\ x$ ) integrable_on A
using Henstock_Kurzweil_Integration.integrable_diff absolutely_integrable_on_def
f g by blast
qed (simp add: comp inner_diff_left)
then show ?thesis
by simp
qed

```

lemma *absolutely_integrable_component_lbound*:

```

fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: euclidean_space
assumes f: f absolutely_integrable_on A and g: g integrable_on A
and comp:  $\bigwedge x\ b. \llbracket x \in A; b \in \text{Basis} \rrbracket \Longrightarrow f\ x \cdot b \leq g\ x \cdot b$ 
shows g absolutely_integrable_on A
proof -
have ( $\lambda x. f\ x + (g\ x - f\ x)$ ) absolutely_integrable_on A
proof (rule set_integral_add [OF f nonnegative_absolutely_integrable])
show ( $\lambda x. g\ x - f\ x$ ) integrable_on A
using Henstock_Kurzweil_Integration.integrable_diff absolutely_integrable_on_def
f g by blast
qed (simp add: comp inner_diff_left)
then show ?thesis
by simp
qed

```

lemma *integrable_on_1_iff*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  real^1
shows f integrable_on S  $\longleftrightarrow (\lambda x. f\ x\ \$\ 1)$  integrable_on S
by (auto simp: integrable_componentwise_iff [of f] Basis_vec_def cart_eq_inner_axis)

```

lemma *integral_on_1_eq*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  real^1
shows integral S f = vec (integral S ( $\lambda x. f\ x\ \$\ 1$ ))
by (cases f integrable_on S) (simp_all add: integrable_on_1_iff vec_eq_iff not_integrable_integral)

```

lemma *absolutely_integrable_on_1_iff*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  real1
shows f absolutely_integrable_on S  $\longleftrightarrow$  ( $\lambda x. f\ x\ \$\ 1$ ) absolutely_integrable_on S
unfolding absolutely_integrable_on_def
by (auto simp: integrable_on_1_iff norm_real)

```

```

lemma absolutely_integrable_absolutely_integrable_lbound:
fixes f :: 'm::euclidean_space  $\Rightarrow$  real
assumes f: f integrable_on S and g: g absolutely_integrable_on S
and *:  $\bigwedge x. x \in S \implies g\ x \leq f\ x$ 
shows f absolutely_integrable_on S
by (rule absolutely_integrable_component_lbound [OF g f]) (simp add: *)

```

```

lemma absolutely_integrable_absolutely_integrable_ubound:
fixes f :: 'm::euclidean_space  $\Rightarrow$  real
assumes fg: f integrable_on S g absolutely_integrable_on S
and *:  $\bigwedge x. x \in S \implies f\ x \leq g\ x$ 
shows f absolutely_integrable_on S
by (rule absolutely_integrable_component_ubound [OF fg]) (simp add: *)

```

```

lemma has_integral_vec1_I_cbox:
fixes f :: real1  $\Rightarrow$  'a::real_normed_vector
assumes (f has_integral y) (cbox a b)
shows ((f  $\circ$  vec) has_integral y) {a$1..b$1}
proof -
have (( $\lambda x. f(\text{vec } x)$ ) has_integral (1 / 1) *R y) (( $\lambda x. x\ \$\ 1$ ) ' cbox a b)
proof (rule has_integral_twiddle)
show  $\exists w z::\text{real}^1. \text{vec } ' \text{cbox } u\ v = \text{cbox } w\ z$ 
content (vec ' cbox u v :: (real1) set) = 1 * content (cbox u v) for u v
unfolding vec_cbox_1_eq
by (auto simp: content_cbox_if_cart_interval_eq_empty_cart)
show  $\exists w z. (\lambda x. x\ \$\ 1) ' \text{cbox } u\ v = \text{cbox } w\ z$  for u v :: real1
using vec_nth_cbox_1_eq by blast
qed (auto simp: continuous_vec assms)
then show ?thesis
by (simp add: o_def)
qed

```

```

lemma has_integral_vec1_I:
fixes f :: real1  $\Rightarrow$  'a::real_normed_vector
assumes (f has_integral y) S
shows (f  $\circ$  vec has_integral y) (( $\lambda x. x\ \$\ 1$ ) ' S)
proof -
have *:  $\exists z. ((\lambda x. \text{if } x \in (\lambda x. x\ \$\ 1) ' S \text{ then } (f \circ \text{vec})\ x \text{ else } 0) \text{ has\_integral } z)$ 
{a..b}  $\wedge$  norm (z - y) < e
if int:  $\bigwedge a\ b. \text{ball } 0\ B \subseteq \text{cbox } a\ b \implies$ 
( $\exists z. ((\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) \text{ has\_integral } z) (\text{cbox } a\ b) \wedge$ 
norm (z - y) < e)
and B: ball 0 B  $\subseteq$  {a..b} for e B a b

```



```

proof –
  have [simp]: ( $\exists y \in S. x = y$  $ 1)  $\longleftrightarrow$   $\text{vec } x \in S$  for  $x$ 
  by force
  have  $B'$ :  $\text{ball } (0 :: \text{real}^1) B \subseteq \text{cbox } (\text{vec } a) (\text{vec } b)$ 
    using  $B$  by (simp add: Basis_vec_def cart_eq_inner_axis [symmetric]
mem_box norm_real subset_iff)
  show ?thesis
    using int [OF  $B'$ ] by (auto simp: image_iff o_def cong: if_cong dest!:
has_integral_vec1_I_cbox)
  qed
  show ?thesis
    using assms
    apply (subst has_integral_alt)
    apply (subst (asm) has_integral_alt)
    apply (simp add: has_integral_vec1_I_cbox split: if_split_asm)
    subgoal by (metis vector_one_nth)
    subgoal
      apply (erule all_forward imp_forward ex_forward asm_rl)+
      by (blast intro!: *)+
    done
qed

lemma has_integral_vec1_nth_cbox:
  fixes  $f :: \text{real} \Rightarrow 'a :: \text{real\_normed\_vector}$ 
  assumes ( $f$  has_integral  $y$ ) { $a..b$ }
  shows ( $(\lambda x :: \text{real}^1. f(x\$1))$  has_integral  $y$ ) ( $\text{cbox } (\text{vec } a) (\text{vec } b)$ )
proof –
  have ( $(\lambda x :: \text{real}^1. f(x\$1))$  has_integral  $(1 / 1) *_{\mathbb{R}} y$ ) ( $\text{vec } ' \text{cbox } a b$ )
  proof (rule has_integral_twiddle)
    show  $\exists w z :: \text{real}. (\lambda x. x \$ 1) ' \text{cbox } u v = \text{cbox } w z$ 
       $\text{content } ((\lambda x. x \$ 1) ' \text{cbox } u v) = 1 * \text{content } (\text{cbox } u v)$  for  $u v :: \text{real}^1$ 
    unfolding vec_cbox_1_eq by (auto simp: content_cbox_if_cart_interval_eq_empty_cart)
    show  $\exists w z :: \text{real}^1. \text{vec } ' \text{cbox } u v = \text{cbox } w z$  for  $u v :: \text{real}$ 
      using vec_cbox_1_eq by auto
  qed (auto simp: continuous_vec assms)
  then show ?thesis
    using vec_cbox_1_eq by auto
qed

lemma has_integral_vec1_D_cbox:
  fixes  $f :: \text{real}^1 \Rightarrow 'a :: \text{real\_normed\_vector}$ 
  assumes ( $(f \circ \text{vec})$  has_integral  $y$ ) { $a\$1..b\$1$ }
  shows ( $f$  has_integral  $y$ ) ( $\text{cbox } a b$ )
  by (metis (mono_tags, lifting) assms comp_apply has_integral_eq has_integral_vec1_nth_cbox
vector_one_nth)

lemma has_integral_vec1_D:
  fixes  $f :: \text{real}^1 \Rightarrow 'a :: \text{real\_normed\_vector}$ 
  assumes ( $(f \circ \text{vec})$  has_integral  $y$ ) ( $(\lambda x. x \$ 1) ' S$ )

```

```

    shows (f has_integral y) S
  proof -
    have *:  $\exists z. ((\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ has\_integral } z) (cbox\ a\ b) \wedge \text{norm } (z - y) < e$ 
    if int:  $\bigwedge a\ b. \text{ball } 0\ B \subseteq \{a..b\} \implies (\exists z. ((\lambda x. \text{if } x \in (\lambda x. x \$ 1) \text{ ' } S \text{ then } (f \circ \text{vec})\ x \text{ else } 0) \text{ has\_integral } z) \{a..b\} \wedge \text{norm } (z - y) < e)$ 
    and B:  $\text{ball } 0\ B \subseteq cbox\ a\ b$  for  $e\ B$  and  $a\ b::\text{real}^1$ 
  proof -
    have B':  $\text{ball } 0\ B \subseteq \{a\$1..b\$1\}$ 
  proof (clarsimp)
    fix t
    assume  $|t| < B$  then show  $a\ \$\ 1 \leq t \wedge t \leq b\ \$\ 1$ 
    using subsetD [OF B]
    by (metis (mono_tags, opaque_lifting) mem_ball_0 mem_box_cart(2) norm_real_vec_component)
  qed
    have eq:  $(\lambda x. \text{if } \text{vec } x \in S \text{ then } f(\text{vec } x) \text{ else } 0) = (\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) \circ \text{vec}$ 
    by force
    have [simp]:  $(\exists y \in S. x = y\ \$\ 1) \longleftrightarrow \text{vec } x \in S$  for  $x$ 
    by force
    show ?thesis
    using int [OF B'] by (auto simp: image_iff eq cong: if_cong dest!: has_integral_vec1_D_cbox)
  qed
    show ?thesis
    using assms
    apply (subst has_integral_alt)
    apply (subst (asm) has_integral_alt)
    apply (simp add: has_integral_vec1_D_cbox eq_cbox split: if_split_asm, blast)
    apply (intro conjI impI)
    subgoal by (metis vector_one_nth)
    apply (erule thin_rl)
    apply (erule all_forward ex_forward conj_forward)+
    by (blast intro!: *)+
  qed

```

lemma *integral_vec1_eq*:

```

  fixes f ::  $\text{real}^1 \Rightarrow 'a::\text{real\_normed\_vector}$ 
  shows  $\text{integral } S\ f = \text{integral } ((\lambda x. x \$ 1) \text{ ' } S) (f \circ \text{vec})$ 
  using has_integral_vec1_I [of f] has_integral_vec1_D [of f]
  by (metis has_integral_iff not_integrable_integral)

```

lemma *absolutely_integrable_drop*:

```

  fixes f ::  $\text{real}^1 \Rightarrow 'b::\text{euclidean\_space}$ 
  shows  $f \text{ absolutely\_integrable\_on } S \longleftrightarrow (f \circ \text{vec}) \text{ absolutely\_integrable\_on } (\lambda x. x \$ 1) \text{ ' } S$ 
  unfolding absolutely_integrable_on_def integrable_on_def

```

```

proof safe
  fix y r
  assume (f has_integral y) S ((λx. norm (f x)) has_integral r) S
  then show ∃ y. (f ∘ vec has_integral y) ((λx. x $ 1) ' S)
    ∃ y. ((λx. norm ((f ∘ vec) x)) has_integral y) ((λx. x $ 1) ' S)
    by (force simp: o_def dest!: has_integral_vec1_I)+
next
  fix y :: 'b and r :: real
  assume (f ∘ vec has_integral y) ((λx. x $ 1) ' S)
    ((λx. norm ((f ∘ vec) x)) has_integral r) ((λx. x $ 1) ' S)
  then show ∃ y. (f has_integral y) S ∃ y. ((λx. norm (f x)) has_integral y) S
    by (force simp: o_def intro: has_integral_vec1_D)+
qed

```

6.19.19 Dominated convergence

```

lemma dominated_convergence:
  fixes f :: nat ⇒ 'n::euclidean_space ⇒ 'm::euclidean_space
  assumes f: ∧k. (f k) integrable_on S and h: h integrable_on S
  and le: ∧k x. x ∈ S ⇒ norm (f k x) ≤ h x
  and conv: ∧x. x ∈ S ⇒ (λk. f k x) ⟶ g x
  shows g integrable_on S (λk. integral S (f k)) ⟶ integral S g
proof -
  have 3: h absolutely_integrable_on S
    unfolding absolutely_integrable_on_def
  proof
    show (λx. norm (h x)) integrable_on S
    proof (intro integrable_spike_finite[OF _ _ h, of {}] ballI)
      fix x assume x ∈ S - {} then show norm (h x) = h x
      by (metis Diff_empty abs_of_nonneg bot_set_def le norm_ge_zero order_trans real_norm_def)
    qed auto
  qed fact
  have 2: set_borel_measurable lebesgue S (f k) for k
    unfolding set_borel_measurable_def
    using f by (auto intro: has_integral_implies_lebesgue_measurable simp: integrable_on_def)
  then have 1: set_borel_measurable lebesgue S g
    unfolding set_borel_measurable_def
    by (rule borel_measurable_LIMSEQ_metric) (use conv in ⟨auto split: split_indicator⟩)
  have 4: AE x in lebesgue. (λi. indicator S x *R f i x) ⟶ indicator S x *R g
    by (auto simp: absolutely_integrable_on_def)
  then show g integrable_on S
    using 1 2 3 4 unfolding set_borel_measurable_def set_integrable_def
    by (rule integrable_dominated_convergence)
then show g integrable_on S
  by (auto simp: absolutely_integrable_on_def)

```

```

have (λk. (LINT x:S|lebesgue. f k x)) → (LINT x:S|lebesgue. g x)
  unfolding set_borel_measurable_def set_lebesgue_integral_def
  using 1 2 3 4 unfolding set_borel_measurable_def set_lebesgue_integral_def
set_integrable_def
  by (rule integral_dominated_convergence)
then show (λk. integral S (f k)) → integral S g
  using g absolutely_integrable_integrable_bound[OF le f h]
  by (subst (asm) (1 2) set_lebesgue_integral_eq_integral) auto
qed

```

```

lemma has_integral_dominated_convergence:
  fixes f :: nat ⇒ 'n::euclidean_space ⇒ 'm::euclidean_space
  assumes ∧k. (f k has_integral y k) S h integrable_on S
    ∧k. ∀x∈S. norm (f k x) ≤ h x ∀x∈S. (λk. f k x) → g x
    and x: y → x
  shows (g has_integral x) S
proof -
  have int_f: ∧k. (f k) integrable_on S
    using assms by (auto simp: integrable_on_def)
  have (g has_integral (integral S g)) S
    by (metis assms(2-4) dominated_convergence(1) has_integral_integral int_f)
  moreover have integral S g = x
  proof (rule LIMSEQ_unique)
    show (λi. integral S (f i)) → x
      using integral_unique[OF assms(1)] x by simp
    show (λi. integral S (f i)) → integral S g
      by (metis assms(2) assms(3) assms(4) dominated_convergence(2) int_f)
  qed
  ultimately show ?thesis
    by simp
qed

```

```

lemma dominated_convergence_integrable_1:
  fixes f :: nat ⇒ 'n::euclidean_space ⇒ real
  assumes f: ∧k. f k absolutely_integrable_on S
    and h: h integrable_on S
    and normg: ∧x. x ∈ S ⇒ norm(g x) ≤ (h x)
    and lim: ∧x. x ∈ S ⇒ (λk. f k x) → g x
  shows g integrable_on S
proof -
  have habs: h absolutely_integrable_on S
    using h normg nonnegative_absolutely_integrable_1 norm_ge_zero order_trans
  by blast
  let ?f = λn x. (min (max (- h x) (f n x)) (h x))
  have h0: h x ≥ 0 if x ∈ S for x
    using normg that by force
  have leh: norm (?f k x) ≤ h x if x ∈ S for k x
    using h0 that by force
  have limf: (λk. ?f k x) → g x if x ∈ S for x

```

```

proof -
  have  $\bigwedge e y. |f y x - g x| < e \implies |\min (\max (- h x) (f y x)) (h x) - g x| < e$ 
    using h0 [OF that] normg [OF that] by simp
  then show ?thesis
    using lim [OF that] by (auto simp add: tendsto_iff dist_norm elim!: eventually_mono)
qed
show ?thesis
proof (rule dominated_convergence [of ?f S h g])
  have  $(\lambda x. - h x)$  absolutely_integrable_on S
    using habs unfolding set_integrable_def by auto
  then show  $?f k$  integrable_on S for k
    by (intro set_lebesgue_integral_eq_integral absolutely_integrable_min_1 absolutely_integrable_max_1 f habs)
  qed (use assms leh limf in auto)
qed

lemma dominated_convergence_integrable:
  fixes f :: nat  $\Rightarrow$  'n::euclidean_space  $\Rightarrow$  'm::euclidean_space
  assumes f:  $\bigwedge k. f k$  absolutely_integrable_on S
    and h: h integrable_on S
    and normg:  $\bigwedge x. x \in S \implies \text{norm}(g x) \leq (h x)$ 
    and lim:  $\bigwedge x. x \in S \implies (\lambda k. f k x) \longrightarrow g x$ 
  shows g integrable_on S
  using f
  unfolding integrable_componentwise_iff [of g] absolutely_integrable_componentwise_iff
  [where f = f k for k]
proof clarify
  fix b :: 'm
  assume fb [rule_format]:  $\bigwedge k. \forall b \in \text{Basis}. (\lambda x. f k x \cdot b)$  absolutely_integrable_on S
  and b: b  $\in$  Basis
  show  $(\lambda x. g x \cdot b)$  integrable_on S
  proof (rule dominated_convergence_integrable_1 [OF fb h])
    fix x
    assume x  $\in$  S
    show norm (g x  $\cdot$  b)  $\leq$  h x
      using norm_nth_le  $\langle x \in S \rangle$  b normg order.trans by blast
    show  $(\lambda k. f k x \cdot b) \longrightarrow g x \cdot b$ 
      using  $\langle x \in S \rangle$  b lim tendsto_componentwise_iff by fastforce
  qed (use b in auto)
qed

lemma dominated_convergence_absolutely_integrable:
  fixes f :: nat  $\Rightarrow$  'n::euclidean_space  $\Rightarrow$  'm::euclidean_space
  assumes f:  $\bigwedge k. f k$  absolutely_integrable_on S
    and h: h integrable_on S
    and normg:  $\bigwedge x. x \in S \implies \text{norm}(g x) \leq (h x)$ 
    and lim:  $\bigwedge x. x \in S \implies (\lambda k. f k x) \longrightarrow g x$ 
  shows g absolutely_integrable_on S

```

```

proof –
  have  $g$  integrable_on  $S$ 
    by (rule dominated_convergence_integrable [OF assms])
  with assms show ?thesis
    by (blast intro: absolutely_integrable_integrable_bound [where  $g=h$ ])
qed

```

proposition *integral_countable_UN*:

```

fixes  $f :: \text{real}^m \Rightarrow \text{real}^n$ 
assumes  $f$ :  $f$  absolutely_integrable_on  $(\bigcup (\text{range } s))$ 
and  $s$ :  $\bigwedge m. s\ m \in \text{sets lebesgue}$ 
shows  $\bigwedge n. f$  absolutely_integrable_on  $(\bigcup_{m \leq n} s\ m)$ 
and  $(\lambda n. \text{integral } (\bigcup_{m \leq n} s\ m) f) \longrightarrow \text{integral } (\bigcup (s \text{ ' } \text{UNIV})) f$  (is ?F
 $\longrightarrow ?I$ )

```

```

proof –
  show  $fU$ :  $f$  absolutely_integrable_on  $(\bigcup_{m \leq n} s\ m)$  for  $n$ 
    using assms by (blast intro: set_integrable_subset [OF  $f$ ])
  have  $fint$ :  $f$  integrable_on  $(\bigcup (\text{range } s))$ 
    using absolutely_integrable_on_def  $f$  by blast
  let  $?h = \lambda x. \text{if } x \in \bigcup (s \text{ ' } \text{UNIV}) \text{ then norm}(f\ x) \text{ else } 0$ 
  have  $(\lambda n. \text{integral UNIV } (\lambda x. \text{if } x \in (\bigcup_{m \leq n} s\ m) \text{ then } f\ x \text{ else } 0))$ 
     $\longrightarrow \text{integral UNIV } (\lambda x. \text{if } x \in \bigcup (s \text{ ' } \text{UNIV}) \text{ then } f\ x \text{ else } 0)$ 
  proof (rule dominated_convergence)
    show  $(\lambda x. \text{if } x \in (\bigcup_{m \leq n} s\ m) \text{ then } f\ x \text{ else } 0)$  integrable_on UNIV for  $n$ 
      unfolding integrable_restrict_UNIV
      using  $fU$  absolutely_integrable_on_def by blast
    show  $(\lambda x. \text{if } x \in \bigcup (s \text{ ' } \text{UNIV}) \text{ then norm}(f\ x) \text{ else } 0)$  integrable_on UNIV
      by (metis (no_types) absolutely_integrable_on_def integrable_restrict_UNIV)
    show  $\bigwedge x. (\lambda n. \text{if } x \in (\bigcup_{m \leq n} s\ m) \text{ then } f\ x \text{ else } 0)$ 
       $\longrightarrow (\text{if } x \in \bigcup (s \text{ ' } \text{UNIV}) \text{ then } f\ x \text{ else } 0)$ 
      by (force intro: tendsto_eventually_eventually_sequentiallyI)
  qed auto
  then show ?F  $\longrightarrow ?I$ 
    by (simp only: integral_restrict_UNIV)
qed

```

6.19.20 Fundamental Theorem of Calculus for the Lebesgue integral

For the positive integral we replace continuity with Borel-measurability.

```

lemma
  fixes  $f :: \text{real} \Rightarrow \text{real}$ 
  assumes [measurable]:  $f \in \text{borel\_measurable borel}$ 
  assumes  $f$ :  $\bigwedge x. x \in \{a..b\} \implies \text{DERIV } F\ x \text{ :> } f\ x \wedge x. x \in \{a..b\} \implies 0 \leq f\ x$ 
and  $a \leq b$ 
  shows  $nn\_integral\_FTC\_Icc$ :  $(\int^+ x. ennreal (f\ x) * \text{indicator } \{a..b\} x\ \partial \text{lborel})$ 
 $= F\ b - F\ a$  (is ?nn)
  and has_bochner_integral_FTC_Icc_nonneg:

```

```

      has_bochner_integral lborel ( $\lambda x. f\ x * \text{indicator}\ \{a \dots b\}\ x$ ) ( $F\ b - F\ a$ ) (is
?has)
    and integral_FTC_Icc_nonneg: ( $\int x. f\ x * \text{indicator}\ \{a \dots b\}\ x\ \partial \text{lborel}$ ) =  $F\ b$ 
-  $F\ a$  (is ?eq)
    and integrable_FTC_Icc_nonneg: integrable lborel ( $\lambda x. f\ x * \text{indicator}\ \{a \dots b\}\ x$ ) (is ?int)
  proof -
    have *: ( $\lambda x. f\ x * \text{indicator}\ \{a..b\}\ x$ )  $\in$  borel_measurable borel  $\wedge x. 0 \leq f\ x * \text{indicator}\ \{a..b\}\ x$ 
    using f(2) by (auto split: split_indicator)

    have F_mono:  $a \leq x \implies x \leq y \implies y \leq b \implies F\ x \leq F\ y$  for  $x\ y$ 
    using f by (intro DERIV_nonneg_imp_nondecreasing[of  $x\ y\ F$ ]) (auto intro:
order_trans)

    have (f has_integral  $F\ b - F\ a$ ) { $a..b$ }
    by (intro fundamental_theorem_of_calculus)
      (auto simp: has_field_derivative_iff_has_vector_derivative[symmetric]
        intro: has_field_derivative_subset[OF f(1)]  $\langle a \leq b \rangle$ )
    then have i: ( $\lambda x. f\ x * \text{indicator}\ \{a \dots b\}\ x$ ) has_integral  $F\ b - F\ a$  UNIV
    unfolding indicator_def of_bool_def if_distrib[where  $f = \lambda x. a * x$  for  $a$ ]
    by (simp cong del: if_weak_cong del: atLeastAtMost_iff)
    then have nn: ( $\int^+ x. f\ x * \text{indicator}\ \{a \dots b\}\ x\ \partial \text{lborel}$ ) =  $F\ b - F\ a$ 
    by (rule nn_integral_has_integral_lborel[OF *])
    then show ?has
    by (rule has_bochner_integral_nn_integral[rotated 3]) (simp_all add: * F_mono
 $\langle a \leq b \rangle$ )
    then show ?eq ?int
    unfolding has_bochner_integral_iff by auto
    show ?nn
    by (subst nn[symmetric])
      (auto intro!: nn_integral_cong simp add: ennreal_mult f split: split_indicator)
  qed

```

lemma

```

  fixes f :: real  $\Rightarrow$  'a :: euclidean_space
  assumes  $a \leq b$ 
  assumes  $\wedge x. a \leq x \implies x \leq b \implies (F\ \text{has\_vector\_derivative}\ f\ x)$  (at  $x$  within
{ $a \dots b$ })
  assumes cont: continuous_on { $a \dots b$ } f
  shows has_bochner_integral_FTC_Icc:
    has_bochner_integral lborel ( $\lambda x. \text{indicator}\ \{a \dots b\}\ x *_R f\ x$ ) ( $F\ b - F\ a$ ) (is
?has)
    and integral_FTC_Icc: ( $\int x. \text{indicator}\ \{a \dots b\}\ x *_R f\ x\ \partial \text{lborel}$ ) =  $F\ b - F\ a$ 
(is ?eq)
  proof -
    let ?f =  $\lambda x. \text{indicator}\ \{a \dots b\}\ x *_R f\ x$ 
    have int: integrable lborel ?f
    using borel_integrable_compact[OF _ cont] by auto

```

```

have (f has_integral F b - F a) {a..b}
  using assms(1,2) by (intro fundamental_theorem_of_calculus) auto
moreover
have (f has_integral integralL lborel ?f) {a..b}
  using has_integral_integral_lborel[OF int]
  unfolding indicator_def of_bool_def if_distrib[where f= $\lambda x. x *_R a$  for a]
  by (simp cong del: if_weak_cong del: atLeastAtMost_iff)
ultimately show ?eq
  by (auto dest: has_integral_unique)
then show ?has
  using int by (auto simp: has_bochner_integral_iff)
qed

lemma
  fixes f :: real  $\Rightarrow$  real
  assumes a  $\leq$  b
  assumes deriv:  $\bigwedge x. a \leq x \implies x \leq b \implies \text{DERIV } F x :> f x$ 
  assumes cont:  $\bigwedge x. a \leq x \implies x \leq b \implies \text{isCont } f x$ 
  shows has_bochner_integral_FTC_Icc_real:
    has_bochner_integral lborel ( $\lambda x. f x * \text{indicator } \{a .. b\} x$ ) (F b - F a) (is
    ?has)
    and integral_FTC_Icc_real: ( $\int x. f x * \text{indicator } \{a .. b\} x \partial \text{lborel}$ ) = F b -
    F a (is ?eq)
  proof -
    have 1:  $\bigwedge x. a \leq x \implies x \leq b \implies (F \text{ has\_vector\_derivative } f x) \text{ (at } x \text{ within } \{a .. b\})$ 
    .. b})
    unfolding has_field_derivative_iff_has_vector_derivative[symmetric]
    using deriv by (auto intro: DERIV_subset)
    have 2: continuous_on {a .. b} f
    using cont by (intro continuous_at_imp_continuous_on) auto
    show ?has ?eq
    using has_bochner_integral_FTC_Icc[OF  $\langle a \leq b \rangle$  1 2] integral_FTC_Icc[OF
     $\langle a \leq b \rangle$  1 2]
    by (auto simp: mult.commute)
  qed

lemma nn_integral_FTC_atLeast:
  fixes f :: real  $\Rightarrow$  real
  assumes f_borel: f  $\in$  borel_measurable borel
  assumes f:  $\bigwedge x. a \leq x \implies \text{DERIV } F x :> f x$ 
  assumes nonneg:  $\bigwedge x. a \leq x \implies 0 \leq f x$ 
  assumes lim: (F  $\longrightarrow$  T) at_top
  shows ( $\int^+ x. \text{ennreal } (f x) * \text{indicator } \{a ..\} x \partial \text{lborel}$ ) = T - F a
  proof -
    let ?f =  $\lambda(i::\text{nat}) (x::\text{real}). \text{ennreal } (f x) * \text{indicator } \{a..a + \text{real } i\} x$ 
    let ?fR =  $\lambda x. \text{ennreal } (f x) * \text{indicator } \{a ..\} x$ 

    have F_mono:  $a \leq x \implies x \leq y \implies F x \leq F y$  for x y
    using f nonneg by (intro DERIV_nonneg_imp_nondecreasing[of x y F]) (auto

```



```

intro: order_trans)
  then have  $F\_le\_T$ :  $a \leq x \implies F\ x \leq T$  for  $x$ 
    by (intro tendsto_lowerbound[OF lim])
      (auto simp: eventually_at_top_linorder)

  have  $(SUP\ i.\ ?f\ i\ x) = ?fR\ x$  for  $x$ 
  proof (rule LIMSEQ_unique[OF LIMSEQ_SUP])
    obtain  $n$  where  $x - a < real\ n$ 
      using reals_Archimedean2[of  $x - a$ ] ..
    then have eventually  $(\lambda n.\ ?f\ n\ x = ?fR\ x)$  sequentially
      by (auto intro!: eventually_sequentiallyI[where  $c=n$ ] split: split_indicator)
    then show  $(\lambda n.\ ?f\ n\ x) \longrightarrow ?fR\ x$ 
      by (rule tendsto_eventually)
  qed (auto simp: nonneg_incseq_def le_fun_def split: split_indicator)
  then have  $integral^N\ lborel\ ?fR = (\int^+ x.\ (SUP\ i.\ ?f\ i\ x)\ \partial lborel)$ 
    by simp
  also have  $\dots = (SUP\ i.\ (\int^+ x.\ ?f\ i\ x\ \partial lborel))$ 
  proof (rule nn_integral_monotone_convergence_SUP)
    show incseq  $?f$ 
      using nonneg by (auto simp: incseq_def le_fun_def split: split_indicator)
    show  $\bigwedge i.\ (?f\ i) \in borel\_measurable\ lborel$ 
      using  $f\_borel$  by auto
  qed
  also have  $\dots = (SUP\ i.\ ennreal\ (F\ (a + real\ i) - F\ a))$ 
    by (subst nn_integral_FTC_Icc[OF  $f\_borel\ f\_nonneg$ ]) auto
  also have  $\dots = T - F\ a$ 
  proof (rule LIMSEQ_unique[OF LIMSEQ_SUP])
    have  $(\lambda x.\ F\ (a + real\ x)) \longrightarrow T$ 
      by (auto intro: filterlim_compose[OF lim filterlim_tendsto_add_at_top]
        filterlim_real_sequentially)
    then show  $(\lambda n.\ ennreal\ (F\ (a + real\ n) - F\ a)) \longrightarrow ennreal\ (T - F\ a)$ 
      by (simp add:  $F\_mono\ F\_le\_T\ tendsto\_diff$ )
  qed (auto simp: incseq_def intro!: ennreal_le_iff[THEN iffD2]  $F\_mono$ )
  finally show  $?thesis$  .
qed

lemma integral_power:
   $a \leq b \implies (\int x.\ x^k * indicator\ \{a..b\}\ x\ \partial lborel) = (b^{Suc\ k} - a^{Suc\ k}) / Suc\ k$ 
  proof (subst integral_FTC_Icc_real)
    fix  $x$  show  $DERIV\ (\lambda x.\ x^{Suc\ k} / Suc\ k)\ x :=> x^k$ 
      by (intro derivative_eq_intros) auto
  qed (auto simp: field_simps simp del: of_nat_Suc)

```

6.19.21 Integration by parts

```

lemma integral_by_parts_integrable:
  fixes  $f\ g\ F\ G::real \Rightarrow real$ 
  assumes  $a \leq b$ 
  assumes  $cont\_f[intro]: !!x.\ a \leq x \implies x \leq b \implies isCont\ f\ x$ 

```

```

assumes cont_g[intro]: !!x.  $a \leq x \implies x \leq b \implies \text{isCont } g \ x$ 
assumes [intro]: !!x. DERIV F x :> f x
assumes [intro]: !!x. DERIV G x :> g x
shows integrable lborel ( $\lambda x. ((F \ x) * (g \ x) + (f \ x) * (G \ x)) * \text{indicator } \{a .. b\}$ 
x)
by (auto intro!: borel_integrable_atLeastAtMost continuous_intros) (auto intro!:
DERIV_isCont)

```

lemma *integral_by_parts*:

```

fixes f g F G::real  $\Rightarrow$  real
assumes [arith]:  $a \leq b$ 
assumes cont_f[intro]: !!x.  $a \leq x \implies x \leq b \implies \text{isCont } f \ x$ 
assumes cont_g[intro]: !!x.  $a \leq x \implies x \leq b \implies \text{isCont } g \ x$ 
assumes [intro]: !!x. DERIV F x :> f x
assumes [intro]: !!x. DERIV G x :> g x
shows ( $\int x. (F \ x * g \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}$ )
      =  $F \ b * G \ b - F \ a * G \ a - \int x. (f \ x * G \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}$ 
proof -
have ( $\int x. (F \ x * g \ x + f \ x * G \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}$ )
      = LBINT x. F x * g x * indicat_real {a..b} x + f x * G x * indicat_real
{a..b} x
by (meson vector_space_over_itself.scale_left_distrib)
also have ... = ( $\int x. (F \ x * g \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}$ ) +  $\int x. (f \ x * G \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}$ 
proof (intro Bochner_Integration.integral_add borel_integrable_atLeastAtMost
cont_f cont_g continuous_intros)
show  $\bigwedge x. \llbracket a \leq x; x \leq b \rrbracket \implies \text{isCont } F \ x \ \bigwedge x. \llbracket a \leq x; x \leq b \rrbracket \implies \text{isCont } G \ x$ 
using DERIV_isCont by blast+
qed
finally have ( $\int x. (F \ x * g \ x + f \ x * G \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}$ ) =
      ( $\int x. (F \ x * g \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}$ ) +  $\int x. (f \ x * G \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}$  .
moreover have ( $\int x. (F \ x * g \ x + f \ x * G \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}$ ) =
       $F \ b * G \ b - F \ a * G \ a$ 
proof (intro integral_FTC_Icc_real derivative_eq_intros cont_f cont_g continuous_intros)
show  $\bigwedge x. \llbracket a \leq x; x \leq b \rrbracket \implies \text{isCont } F \ x \ \bigwedge x. \llbracket a \leq x; x \leq b \rrbracket \implies \text{isCont } G \ x$ 
using DERIV_isCont by blast+
qed auto
ultimately show ?thesis by auto
qed

```

lemma *integral_by_parts'*:

```

fixes f g F G::real  $\Rightarrow$  real
assumes  $a \leq b$ 
assumes !!x.  $a \leq x \implies x \leq b \implies \text{isCont } f \ x$ 
assumes !!x.  $a \leq x \implies x \leq b \implies \text{isCont } g \ x$ 
assumes !!x. DERIV F x :> f x
assumes !!x. DERIV G x :> g x

```

shows $(\int x. \text{indicator } \{a .. b\} x *_R (F x * g x) \partial \text{lborel})$
 $= F b * G b - F a * G a - \int x. \text{indicator } \{a .. b\} x *_R (f x * G x)$
 ∂lborel
 using *integral_by_parts*[OF *assms*] by (simp add: *ac_simps*)

lemma *has_bochner_integral_even_function*:

fixes $f :: \text{real} \Rightarrow 'a :: \{\text{banach, second\_countable\_topology}\}$
assumes $f: \text{has\_bochner\_integral lborel } (\lambda x. \text{indicator } \{0.. \} x *_R f x) x$
assumes *even*: $\bigwedge x. f (-x) = f x$
shows *has_bochner_integral lborel f* $(2 *_R x)$
proof –
have *indicator*: $\bigwedge x::\text{real}. \text{indicator } \{..0\} (-x) = \text{indicator } \{0..\} x$
by (*auto split: split_indicator*)
have *has_bochner_integral lborel* $(\lambda x. \text{indicator } \{.. 0\} x *_R f x) x$
by (*subst lborel_has_bochner_integral_real_affine_iff*[**where** $c=-1$ **and** $t=0$])
 (*auto simp: indicator even f*)
with f **have** *has_bochner_integral lborel* $(\lambda x. \text{indicator } \{0..\} x *_R f x + \text{indicator } \{.. 0\} x *_R f x) (x + x)$
by (*rule has_bochner_integral_add*)
then have *has_bochner_integral lborel f* $(x + x)$
by (*rule has_bochner_integral_discrete_difference*[**where** $X=\{0\}$, *THEN iffD1*, *rotated 4*])
 (*auto split: split_indicator*)
then show *?thesis*
by (*simp add: scaleR_2*)
qed

lemma *has_bochner_integral_odd_function*:

fixes $f :: \text{real} \Rightarrow 'a :: \{\text{banach, second\_countable\_topology}\}$
assumes $f: \text{has\_bochner\_integral lborel } (\lambda x. \text{indicator } \{0..\} x *_R f x) x$
assumes *odd*: $\bigwedge x. f (-x) = -f x$
shows *has_bochner_integral lborel f* 0
proof –
have *indicator*: $\bigwedge x::\text{real}. \text{indicator } \{..0\} (-x) = \text{indicator } \{0..\} x$
by (*auto split: split_indicator*)
have *has_bochner_integral lborel* $(\lambda x. - \text{indicator } \{.. 0\} x *_R f x) x$
by (*subst lborel_has_bochner_integral_real_affine_iff*[**where** $c=-1$ **and** $t=0$])
 (*auto simp: indicator odd f*)
from *has_bochner_integral_minus*[OF *this*]
have *has_bochner_integral lborel* $(\lambda x. \text{indicator } \{.. 0\} x *_R f x) (-x)$
by *simp*
with f **have** *has_bochner_integral lborel* $(\lambda x. \text{indicator } \{0..\} x *_R f x + \text{indicator } \{.. 0\} x *_R f x) (x + -x)$
by (*rule has_bochner_integral_add*)
then have *has_bochner_integral lborel f* $(x + -x)$
by (*rule has_bochner_integral_discrete_difference*[**where** $X=\{0\}$, *THEN iffD1*, *rotated 4*])
 (*auto split: split_indicator*)
then show *?thesis*

```

    by simp
qed

lemma has_integral_0_closure_imp_0:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes f: continuous_on (closure S) f
    and nonneg_interior:  $\bigwedge x. x \in S \implies 0 \leq f x$ 
    and pos:  $0 < \text{emeasure lborel } S$ 
    and finite:  $\text{emeasure lborel } S < \infty$ 
    and regular:  $\text{emeasure lborel } (\text{closure } S) = \text{emeasure lborel } S$ 
    and opn: open S
  assumes int: (f has_integral 0) (closure S)
  assumes x:  $x \in \text{closure } S$ 
  shows f x = 0
proof -
  have zero:  $\text{emeasure lborel } (\text{frontier } S) = 0$ 
    using finite closure_subset regular
    unfolding frontier_def
    by (subst emeasure_Diff) (auto simp: frontier_def interior_open ⟨open S⟩)
  have nonneg:  $0 \leq f x$  if  $x \in \text{closure } S$  for x
    using continuous_ge_on_closure[OF f that nonneg_interior] by simp
  have 0 = integral (closure S) f
    by (blast intro: int sym)
  also
  note intl = has_integral_integrable[OF int]
  have af: f absolutely_integrable_on (closure S)
    using nonneg
    by (intro absolutely_integrable_onI intl integrable_eq[OF intl]) simp
  then have integral (closure S) f = set_lebesgue_integral lebesgue (closure S) f
    by (intro set_lebesgue_integral_eq_integral(2)[symmetric])
  also have ... = 0  $\longleftrightarrow$  (AE x in lebesgue. indicator (closure S) x *R f x = 0)
    unfolding set_lebesgue_integral_def
  proof (rule integral_nonneg_eq_0_iff_AE)
    show integrable lebesgue ( $\lambda x. \text{indicator\_real } (\text{closure } S) x *_{\mathbb{R}} f x$ )
      by (metis af set_integrable_def)
  qed (use nonneg in ⟨auto simp: indicator_def⟩)
  also have ...  $\longleftrightarrow$  (AE x in lebesgue.  $x \in \{x. x \in \text{closure } S \longrightarrow f x = 0\}$ )
    by (auto simp: indicator_def)
  finally have (AE x in lebesgue.  $x \in \{x. x \in \text{closure } S \longrightarrow f x = 0\}$ ) by simp
  moreover have (AE x in lebesgue.  $x \in - \text{frontier } S$ )
    using zero
    by (auto simp: eventually_ae_filter null_sets_def intro!: exI[where x=frontier S])
  ultimately have ae: AE x in lebesgue.  $x \in \{x \in \text{closure } S. f x = 0\}$  (is ?th)
    by eventually_elim (use closure_subset in ⟨auto simp: ⟩)
  have closed {0::real} by simp
  with continuous_on_closed_vimage[OF closed_closure, of S f]
  have closed (f - ' {0}  $\cap$  closure S) by blast

```

```

then have  $\text{closed } \{x \in \text{closure } S. f\ x = 0\}$  by (auto simp: vimage_def Int_def
conj_commute)
with  $\langle \text{open } S \rangle$  have  $x \in \{x \in \text{closure } S. f\ x = 0\}$  if  $x \in S$  for  $x$  using  $ae\ that$ 
by (rule mem_closed_if_AE_lebesgue_open)
then have  $f\ x = 0$  if  $x \in S$  for  $x$  using  $that$  by auto
from continuous_constant_on_closure[OF  $f\ this\ \langle x \in \text{closure } S \rangle$ ]
show  $f\ x = 0$  .
qed

```

```

lemma has_integral_0_cbox_imp_0:
fixes  $f :: 'a::euclidean\_space \Rightarrow real$ 
assumes  $f: \text{continuous\_on } (cbox\ a\ b)\ f$ 
and  $\text{nonneg\_interior: } \bigwedge x. x \in \text{box } a\ b \implies 0 \leq f\ x$ 
assumes  $\text{int: } (f\ \text{has\_integral } 0)\ (cbox\ a\ b)$ 
assumes  $\text{ne: } \text{box } a\ b \neq \{\}$ 
assumes  $x: x \in cbox\ a\ b$ 
shows  $f\ x = 0$ 
proof -
have  $0 < \text{emeasure } lborel\ (\text{box } a\ b)$ 
using  $\text{ne } x$  unfolding  $\text{emeasure\_lborel\_box\_eq}$ 
by (force intro!: prod_pos simp: mem_box algebra_simps)
then show ?thesis using  $\text{assms}$ 
by (intro has_integral_0_closure_imp_0[of  $\text{box } a\ b\ f\ x$ ])
(auto simp: emeasure_lborel_box_eq emeasure_lborel_cbox_eq algebra_simps
mem_box)
qed

```

6.19.22 Various common equivalent forms of function measurability

```

lemma indicator_sum_eq:
fixes  $m::real$  and  $f :: 'a \Rightarrow real$ 
assumes  $|m| \leq 2^{-(2*n)}\ m/2^n \leq f\ x\ f\ x < (m+1)/2^n\ m \in \mathbb{Z}$ 
shows  $(\sum k::real \mid k \in \mathbb{Z} \wedge |k| \leq 2^{-(2*n)}. \quad$ 
 $k/2^n * \text{indicator } \{y. k/2^n \leq f\ y \wedge f\ y < (k+1)/2^n\} x) = m/2^n$ 
 $(\text{is sum } ?f\ ?S = \_))$ 
proof -
have  $\text{sum } ?f\ ?S = \text{sum } (\lambda k. k/2^n * \text{indicator } \{y. k/2^n \leq f\ y \wedge f\ y <$ 
 $(k+1)/2^n\} x)\ \{m\}$ 
proof (rule comm_monoid_add_class.sum_mono_neutral_right)
show  $\text{finite } ?S$ 
by (rule finite_abs_int_segment)
show  $\{m\} \subseteq \{k \in \mathbb{Z}. |k| \leq 2^{-(2*n)}\}$ 
using  $\text{assms}$  by auto
show  $\forall i \in \{k \in \mathbb{Z}. |k| \leq 2^{-(2*n)}\} - \{m\}. ?f\ i = 0$ 
using  $\text{assms}$  by (auto simp: indicator_def Ints_def abs_le_iff field_split_simps)
qed
also have  $\dots = m/2^n$ 
using  $\text{assms}$  by (auto simp: indicator_def not_less)

```

finally show ?thesis .
qed

lemma measurable_on_sf_limit_lemma1:

fixes $f :: 'a::euclidean\_space \Rightarrow \text{real}$
 assumes $\bigwedge a b. \{x \in S. a \leq f\ x \wedge f\ x < b\} \in \text{sets}(\text{lebesgue\_on } S)$
 obtains g where $\bigwedge n. g\ n \in \text{borel\_measurable}(\text{lebesgue\_on } S)$
 $\bigwedge n. \text{finite}(\text{range}(g\ n))$
 $\bigwedge x. (\lambda n. g\ n\ x) \longrightarrow f\ x$

proof

show $(\lambda x. \text{sum}(\lambda k::\text{real}. k/2^n * \text{indicator}\ \{y. k/2^n \leq f\ y \wedge f\ y < (k+1)/2^n\} x)$

$\{k \in \mathbb{Z}. |k| \leq 2^n\}) \in \text{borel\_measurable}(\text{lebesgue\_on } S)$

(is ?g $\in$ _) for n

proof -

have $\bigwedge k. \llbracket k \in \mathbb{Z}; |k| \leq 2^n \rrbracket$

$\implies \text{Measurable.pred}(\text{lebesgue\_on } S) (\lambda x. k / (2^n) \leq f\ x \wedge f\ x < (k+1)$

$/ (2^n))$

using assms by (force simp: pred_def space_restrict_space)

then show ?thesis

by (simp add: field_class.field_divide_inverse)

qed

show finite (range (?g n)) for n

proof -

have $\text{range} (?g\ n) \subseteq (\lambda k. k/2^n) \cdot \{k \in \mathbb{Z}. |k| \leq 2^n\}$

proof clarify

fix x

show ?g $n\ x \in (\lambda k. k/2^n) \cdot \{k \in \mathbb{Z}. |k| \leq 2^n\}$

proof (cases $\exists k::\text{real}. k \in \mathbb{Z} \wedge |k| \leq 2^n \wedge k/2^n \leq (f\ x) \wedge (f\ x) < (k+1)/2^n$)

case True

then show ?thesis

apply clarify

by (subst indicator_sum_eq) auto

next

case False

then have ?g $n\ x = 0$ by auto

then show ?thesis by force

qed

qed

moreover have finite $((\lambda k::\text{real}. (k/2^n)) \cdot \{k \in \mathbb{Z}. |k| \leq 2^n\})$

by (simp add: finite_abs_int_segment)

ultimately show ?thesis

using finite_subset by blast

qed

show $(\lambda n. ?g\ n\ x) \longrightarrow f\ x$ for x

proof (rule LIMSEQ_I)

fix $e::\text{real}$

assume $e > 0$

```

obtain  $N1$  where  $N1: |f\ x| < 2^{-N1}$ 
  using real_arch_pow by fastforce
obtain  $N2$  where  $N2: (1/2)^{-N2} < e$ 
  using real_arch_pow_inv  $\langle e > 0 \rangle$  by force
have norm  $(\text{?g } n\ x - f\ x) < e$  if  $n: n \geq \max\ N1\ N2$  for  $n$ 
proof -
  define  $m$  where  $m \equiv \text{floor}(2^n * (f\ x))$ 
  have  $1 \leq |2^n| * e$ 
    using  $n\ N2\ \langle e > 0 \rangle$  less_eq_real_def less_le_trans by (fastforce simp add:
field_split_simps)
  then have  $*$ :  $\llbracket x \leq y; y < x + 1 \rrbracket \implies \text{abs}(x - y) < |2^n| * e$  for  $x\ y::\text{real}$ 
    by linarith
  have  $|2^n| * |m/2^n - f\ x| = |2^n * (m/2^n - f\ x)|$ 
    by (simp add: abs_mult)
  also have  $\dots = |\text{real\_of\_int } \lfloor 2^n * f\ x \rfloor - f\ x * 2^n|$ 
    by (simp add: algebra_simps m_def)
  also have  $\dots < |2^n| * e$ 
    by (rule  $*$ ; simp add: mult.commute)
  finally have  $|2^n| * |m/2^n - f\ x| < |2^n| * e$  .
  then have  $m$ :  $|m/2^n - f\ x| < e$ 
    by simp
  have  $|\text{real\_of\_int } m| \leq 2^{-(2*n)}$ 
proof (cases  $f\ x < 0$ )
  case True
    then have  $-|f\ x| \leq \lfloor (2::\text{real})^{-N1} \rfloor$ 
      using  $N1$  le_floor_iff minus_le_iff by fastforce
    with  $n\ \text{True}$  have  $|\text{real\_of\_int } \lfloor f\ x \rfloor| \leq 2^{-N1}$ 
      by linarith
    also have  $\dots \leq 2^n$ 
      using  $n$  by (simp add: m_def)
    finally have  $|\text{real\_of\_int } \lfloor f\ x \rfloor| * 2^n \leq 2^n * 2^n$ 
      by simp
  moreover
    have  $|\text{real\_of\_int } \lfloor 2^n * f\ x \rfloor| \leq |\text{real\_of\_int } \lfloor f\ x \rfloor| * 2^n$ 
proof -
    have  $|\text{real\_of\_int } \lfloor 2^n * f\ x \rfloor| = -(\text{real\_of\_int } \lfloor 2^n * f\ x \rfloor)$ 
      using True by (simp add: abs_if mult_less_0_iff)
    also have  $\dots \leq -(\text{real\_of\_int } (\lfloor (2::\text{real})^{-n} \rfloor * \lfloor f\ x \rfloor))$ 
      using le_mult_floor_Ints  $[\text{of } (2::\text{real})^n]$  by simp
    also have  $\dots \leq |\text{real\_of\_int } \lfloor f\ x \rfloor| * 2^n$ 
      using True
      by simp
    finally show ?thesis .
  qed
ultimately show ?thesis
  by (metis (no_types, opaque_lifting) m_def order_trans power2_eq_square
power_even_eq)
next
  case False

```

```

with n N1 have f x ≤ 2^n
  by (simp add: not_less) (meson less_eq_real_def one_le_numeral or-
der_trans power_increasing)
moreover have 0 ≤ m
  using False m_def by force
ultimately show ?thesis
  by (metis abs_of_nonneg floor_mono le_floor_iff m_def of_int_0_le_iff
power2_eq_square power_mult mult_le_cancel_iff1 zero_less_numeral mult.commute
zero_less_power)
qed
then have ?g n x = m/2^n
  by (rule indicator_sum_eq) (auto simp add: m_def field_split_simps,
linarith)
then have norm (?g n x - f x) = norm (m/2^n - f x)
  by simp
also have ... < e
  by (simp add: me)
finally show ?thesis .
qed
then show ∃ no. ∀ n ≥ no. norm (?g n x - f x) < e
  by blast
qed
qed

```

```

lemma borel_measurable_simple_function_limit:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  shows f ∈ borel_measurable (lebesgue_on S) ⟷
    (∃ g. (∀ n. (g n) ∈ borel_measurable (lebesgue_on S)) ∧
    (∀ n. finite (range (g n))) ∧ (∀ x. (λn. g n x) ⟶ f x))
proof -
  have ∃ g. (∀ n. (g n) ∈ borel_measurable (lebesgue_on S)) ∧
    (∀ n. finite (range (g n))) ∧ (∀ x. (λn. g n x) ⟶ f x)
  if f: ⋀ a i. i ∈ Basis ⟹ {x ∈ S. f x · i < a} ∈ sets (lebesgue_on S)
proof -
  have ∃ g. (∀ n. (g n) ∈ borel_measurable (lebesgue_on S)) ∧
    (∀ n. finite (image (g n) UNIV)) ∧
    (∀ x. ((λn. g n x) ⟶ f x · i) if i ∈ Basis for i)
  proof (rule measurable_on_sf_limit_lemma1 [of S λx. f x · i])
    show {x ∈ S. a ≤ f x · i ∧ f x · i < b} ∈ sets (lebesgue_on S) for a b
  proof -
    have {x ∈ S. a ≤ f x · i ∧ f x · i < b} = {x ∈ S. f x · i < b} - {x ∈ S. a
    > f x · i}
    by auto
    also have ... ∈ sets (lebesgue_on S)
    using f that by blast
    finally show ?thesis .
  qed
qed blast

```



```

then obtain  $g$  where  $g$ :
   $\bigwedge i \ n. i \in \text{Basis} \implies g \ i \ n \in \text{borel\_measurable} \ (\text{lebesgue\_on } S)$ 
   $\bigwedge i \ n. i \in \text{Basis} \implies \text{finite}(\text{range } (g \ i \ n))$ 
   $\bigwedge i \ x. i \in \text{Basis} \implies ((\lambda n. g \ i \ n \ x) \longrightarrow f \ x \cdot i)$ 
by metis
show ?thesis
proof (intro conjI allI exI)
  show  $(\lambda x. \sum_{i \in \text{Basis}} g \ i \ n \ x \cdot_R i) \in \text{borel\_measurable} \ (\text{lebesgue\_on } S)$  for
n
    by (intro borel_measurable_sum borel_measurable_scaleR) (auto intro: g)
  show finite (range  $(\lambda x. \sum_{i \in \text{Basis}} g \ i \ n \ x \cdot_R i)$ ) for  $n$ 
  proof -
    have range  $(\lambda x. \sum_{i \in \text{Basis}} g \ i \ n \ x \cdot_R i) \subseteq (\lambda h. \sum_{i \in \text{Basis}} h \ i \cdot_R i) \text{ ` } \text{PiE}$ 
Basis  $(\lambda i. \text{range } (g \ i \ n))$ 
    proof clarify
      fix  $x$ 
      show  $(\sum_{i \in \text{Basis}} g \ i \ n \ x \cdot_R i) \in (\lambda h. \sum_{i \in \text{Basis}} h \ i \cdot_R i) \text{ ` } (\Pi_E i \in \text{Basis}.$ 
range  $(g \ i \ n))$ 
      by (rule_tac  $x = \lambda i \in \text{Basis}. g \ i \ n \ x$  in image_eqI) auto
    qed
    moreover have finite  $(\text{PiE Basis } (\lambda i. \text{range } (g \ i \ n)))$ 
      by (simp add: g finite_PiE)
    ultimately show ?thesis
      by (metis (mono_tags, lifting) finite_surj)
  qed
  show  $(\lambda n. \sum_{i \in \text{Basis}} g \ i \ n \ x \cdot_R i) \longrightarrow f \ x$  for  $x$ 
  proof -
    have  $(\lambda n. \sum_{i \in \text{Basis}} g \ i \ n \ x \cdot_R i) \longrightarrow (\sum_{i \in \text{Basis}} (f \ x \cdot i) \cdot_R i)$ 
      by (auto intro!: tendsto_sum tendsto_scaleR g)
    moreover have  $(\sum_{i \in \text{Basis}} (f \ x \cdot i) \cdot_R i) = f \ x$ 
      using euclidean_representation by blast
    ultimately show ?thesis
      by metis
  qed
qed
qed
qed
moreover have  $f \in \text{borel\_measurable} \ (\text{lebesgue\_on } S)$ 
  if meas_g:  $\bigwedge n. g \ n \in \text{borel\_measurable} \ (\text{lebesgue\_on } S)$ 
  and fin:  $\bigwedge n. \text{finite} \ (\text{range } (g \ n))$ 
  and to_f:  $\bigwedge x. (\lambda n. g \ n \ x) \longrightarrow f \ x$  for  $g$ 
  by (rule borel_measurable_LIMSEQ_metric [OF meas_g to_f])
  ultimately show ?thesis
    using borel_measurable_vimage_halfspace_component_lt by blast
qed

```

6.19.23 Lebesgue sets and continuous images

proposition *lebesgue_regular_inner*:

assumes $S \in \text{sets lebesgue}$

```

obtains  $K \ C$  where negligible  $K \ \bigwedge n::nat. \text{compact}(C \ n) \ S = (\bigcup n. C \ n) \cup K$ 
proof –
  have  $\exists T. \text{closed } T \wedge T \subseteq S \wedge (S - T) \in \text{lmeasurable} \wedge \text{emeasure lebesgue } (S$ 
–  $T) < \text{ennreal } ((1/2)^{\wedge} n)$  for  $n$ 
    using sets_lebesgue_inner_closed assms
    by (metis sets_lebesgue_inner_closed zero_less_divide_1_iff zero_less_numeral
zero_less_power)
    then obtain  $C$  where  $\text{clo}: \bigwedge n. \text{closed } (C \ n)$  and  $\text{subS}: \bigwedge n. C \ n \subseteq S$ 
      and  $\text{mea}: \bigwedge n. (S - C \ n) \in \text{lmeasurable}$ 
      and  $\text{less}: \bigwedge n. \text{emeasure lebesgue } (S - C \ n) < \text{ennreal } ((1/2)^{\wedge} n)$ 
      by metis
    have  $\exists F. (\forall n::nat. \text{compact}(F \ n)) \wedge (\bigcup n. F \ n) = C \ m$  for  $m::nat$ 
      by (metis clo_closed_Union_compact_subsets)
    then obtain  $D :: [nat, nat] \Rightarrow 'a \text{ set}$  where  $D: \bigwedge m \ n. \text{compact}(D \ m \ n) \ \bigwedge m.$ 
–  $(\bigcup n. D \ m \ n) = C \ m$ 
      by metis
    let  $?C = \text{from\_nat\_into } (\bigcup m. \text{range } (D \ m))$ 
    have countable  $(\bigcup m. \text{range } (D \ m))$ 
      by blast
    have  $\text{range } (\text{from\_nat\_into } (\bigcup m. \text{range } (D \ m))) = (\bigcup m. \text{range } (D \ m))$ 
      using range_from_nat_into by simp
    then have  $CD: \exists m \ n. ?C \ k = D \ m \ n$  for  $k$ 
      by (metis mono_tags, lifting) UN_iff_rangeE_range_eqI
    show thesis
  proof
    show negligible  $(S - (\bigcup n. C \ n))$ 
    proof (clarsimp simp: negligible_outer_le)
      fix  $e :: \text{real}$ 
      assume  $e > 0$ 
      then obtain  $n$  where  $n: (1/2)^{\wedge} n < e$ 
        using real_arch_pow_inv [of e 1/2] by auto
      show  $\exists T. S - (\bigcup n. C \ n) \subseteq T \wedge T \in \text{lmeasurable} \wedge \text{measure lebesgue } T \leq e$ 
      proof (intro exI conjI)
        show  $S - (\bigcup n. C \ n) \subseteq S - C \ n$ 
          by blast
        show  $S - C \ n \in \text{lmeasurable}$ 
          by (simp add: mea)
        show  $\text{measure lebesgue } (S - C \ n) \leq e$ 
          using less [of n] n
          by (simp add: emeasure_eq_measure2 less_le mea)
      qed
    qed
    show compact  $(?C \ n)$  for  $n$ 
      using CD D by metis
    show  $S = (\bigcup n. ?C \ n) \cup (S - (\bigcup n. C \ n))$  (is  $\_ = ?rhs$ )
    proof
      show  $S \subseteq ?rhs$ 
        using  $D$  by fastforce
      show  $?rhs \subseteq S$ 

```

```

    using subS D CD by auto (metis Sup_upper range_eqI subsetCE)
  qed
qed
qed

lemma sets_lebesgue_continuous_image:
  assumes T:  $T \in \text{sets lebesgue}$  and conf:  $\text{continuous\_on } S f$ 
  and negim:  $\bigwedge T. [\text{negligible } T; T \subseteq S] \implies \text{negligible}(f \restriction T)$  and  $T \subseteq S$ 
  shows  $f \restriction T \in \text{sets lebesgue}$ 
proof -
  obtain K C where negligible K and com:  $\bigwedge n::\text{nat}. \text{compact}(C n)$  and Teq:  $T = (\bigcup n. C n) \cup K$ 
  using lebesgue_regular_inner [OF T] by metis
  then have comf:  $\bigwedge n::\text{nat}. \text{compact}(f \restriction C n)$ 
  by (metis Un_subset_iff Union_upper  $\langle T \subseteq S \rangle$  compact_continuous_image
  conf continuous_on_subset rangeI)
  have  $((\bigcup n. f \restriction C n) \cup f \restriction K) \in \text{sets lebesgue}$ 
  proof (rule sets.Un)
    have  $K \subseteq S$ 
    using Teq  $\langle T \subseteq S \rangle$  by blast
    show  $(\bigcup n. f \restriction C n) \in \text{sets lebesgue}$ 
    proof (rule sets.countable_Union)
      show  $\text{range } (\lambda n. f \restriction C n) \subseteq \text{sets lebesgue}$ 
      using borel_compact comf by (auto simp: borel_compact)
    qed auto
    show  $f \restriction K \in \text{sets lebesgue}$ 
    by (simp add:  $\langle K \subseteq S \rangle$   $\langle \text{negligible } K \rangle$  negim negligible_imp_sets)
  qed
  then show ?thesis
  by (simp add: Teq image_Un image_Union)
qed

```

```

lemma differentiable_image_in_sets_lebesgue:
  fixes f ::  $'m::\text{euclidean\_space} \Rightarrow 'n::\text{euclidean\_space}$ 
  assumes S:  $S \in \text{sets lebesgue}$  and dim:  $\text{DIM}('m) \leq \text{DIM}('n)$  and f:  $f$  differentiable_on S
  shows  $f \restriction S \in \text{sets lebesgue}$ 
proof (rule sets_lebesgue_continuous_image [OF S])
  show continuous_on S f
  by (meson differentiable_imp_continuous_on f)
  show  $\bigwedge T. [\text{negligible } T; T \subseteq S] \implies \text{negligible}(f \restriction T)$ 
  using differentiable_on_subset f
  by (auto simp: intro!: negligible_differentiable_image_negligible [OF dim])
qed auto

```

```

lemma sets_lebesgue_on_continuous_image:
  assumes S:  $S \in \text{sets lebesgue}$  and X:  $X \in \text{sets}(\text{lebesgue\_on } S)$  and conf:
  continuous_on S f
  and negim:  $\bigwedge T. [\text{negligible } T; T \subseteq S] \implies \text{negligible}(f \restriction T)$ 

```

```

  shows  $f \restriction X \in \text{sets } (\text{lebesgue\_on } (f \restriction S))$ 
proof -
  have  $X \subseteq S$ 
  by (metis  $S \ X \ \text{sets.Int\_space\_eq2} \ \text{sets\_restrict\_space\_iff}$ )
  moreover have  $f \restriction S \in \text{sets lebesgue}$ 
  using  $S \ \text{contf\_negim\_sets\_lebesgue\_continuous\_image}$  by blast
  moreover have  $f \restriction X \in \text{sets lebesgue}$ 
  by (metis  $S \ X \ \text{contf\_negim\_sets\_lebesgue\_continuous\_image} \ \text{sets\_restrict\_space\_iff} \ \text{space\_restrict\_space} \ \text{space\_restrict\_space2}$ )
  ultimately show ?thesis
  by (auto simp:  $\text{sets\_restrict\_space\_iff}$ )
qed

```

```

lemma differentiable_image_in_sets_lebesgue_on:
  fixes  $f :: 'm::\text{euclidean\_space} \Rightarrow 'n::\text{euclidean\_space}$ 
  assumes  $S: S \in \text{sets lebesgue}$  and  $X: X \in \text{sets } (\text{lebesgue\_on } S)$  and  $\text{dim}: \text{DIM}('m) \leq \text{DIM}('n)$ 
  and  $f: f \text{ differentiable\_on } S$ 
  shows  $f \restriction X \in \text{sets } (\text{lebesgue\_on } (f \restriction S))$ 
proof (rule  $\text{sets\_lebesgue\_on\_continuous\_image} \ [OF \ S \ X]$ )
  show  $\text{continuous\_on } S \ f$ 
  by (meson  $\text{differentiable\_imp\_continuous\_on} \ f$ )
  show  $\bigwedge T. \llbracket \text{negligible } T; T \subseteq S \rrbracket \implies \text{negligible } (f \restriction T)$ 
  using  $\text{differentiable\_on\_subset} \ f$ 
  by (auto simp:  $\text{intro!} \ \text{negligible\_differentiable\_image\_negligible} \ [OF \ \text{dim}]$ )
qed

```

6.19.24 Affine lemmas

```

lemma borel_measurable_affine:
  fixes  $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$ 
  assumes  $f: f \in \text{borel\_measurable lebesgue}$  and  $c \neq 0$ 
  shows  $(\lambda x. f(t + c *_{\mathbb{R}} x)) \in \text{borel\_measurable lebesgue}$ 
proof -
  { fix  $a \ b$ 
    have  $\{x. f \ x \in \text{cbox } a \ b\} \in \text{sets lebesgue}$ 
    using  $f \ \text{cbox\_borel\_lebesgue\_measurable\_vimage\_borel}$  by blast
    then have  $(\lambda x. (x - t) /_{\mathbb{R}} c) \restriction \{x. f \ x \in \text{cbox } a \ b\} \in \text{sets lebesgue}$ 
  }
  proof (rule  $\text{differentiable\_image\_in\_sets\_lebesgue}$ )
    show  $(\lambda x. (x - t) /_{\mathbb{R}} c) \text{ differentiable\_on } \{x. f \ x \in \text{cbox } a \ b\}$ 
    unfolding  $\text{differentiable\_on\_def} \ \text{differentiable\_def}$ 
    by (rule  $\langle c \neq 0 \rangle \ \text{derivative\_eq\_intros} \ \text{strip} \ \text{exI} \mid \text{simp}$ ) +
    qed auto
  moreover
  have  $\{x. f(t + c *_{\mathbb{R}} x) \in \text{cbox } a \ b\} = (\lambda x. (x - t) /_{\mathbb{R}} c) \restriction \{x. f \ x \in \text{cbox } a \ b\}$ 
  using  $\langle c \neq 0 \rangle$  by (auto simp:  $\text{image\_def}$ )
  ultimately have  $\{x. f(t + c *_{\mathbb{R}} x) \in \text{cbox } a \ b\} \in \text{sets lebesgue}$ 
  by (auto simp:  $\text{borel\_measurable\_vimage\_closed\_interval}$ ) }
  then show ?thesis

```

by (subst lebesgue_on_UNIV_eq [symmetric]; auto simp: borel_measurable_vimage_closed_interval)
qed

lemma *lebesgue_integrable_real_affine*:
 fixes $f :: \text{real} \Rightarrow 'a :: \text{euclidean\_space}$
 assumes $f: \text{integrable lebesgue } f$ and $c \neq 0$
 shows $\text{integrable lebesgue } (\lambda x. f(t + c * x))$
proof –
 have $(\lambda x. \text{norm } (f x)) \in \text{borel\_measurable lebesgue}$
 by (simp add: borel_measurable_integrable f)
 then show ?thesis
 using assms borel_measurable_affine [of f c]
 unfolding integrable_iff_bounded
 by (subst (asm) nn_integral_real_affine_lebesgue[where $c=c$ and $t=t$]) (auto
 simp: ennreal_mult_less_top)
 qed

lemma *lebesgue_integrable_real_affine_iff*:
 fixes $f :: \text{real} \Rightarrow 'a :: \text{euclidean\_space}$
 shows $c \neq 0 \implies \text{integrable lebesgue } (\lambda x. f(t + c * x)) \longleftrightarrow \text{integrable lebesgue } f$
 using lebesgue_integrable_real_affine[of f c t]
 lebesgue_integrable_real_affine[of $\lambda x. f(t + c * x)$ $1/c - t/c$]
 by (auto simp: field_simps)

lemma *lebesgue_integral_real_affine*:
 fixes $f :: \text{real} \Rightarrow 'a :: \text{euclidean\_space}$ and $c :: \text{real}$
 assumes $c: c \neq 0$ shows $(\int x. f x \partial \text{lebesgue}) = |c| *_R (\int x. f(t + c * x) \partial \text{lebesgue})$
proof cases
 have $(\lambda x. t + c * x) \in \text{lebesgue} \rightarrow_M \text{lebesgue}$
 using lebesgue_affine_measurable[where $c = \lambda x::\text{real}. c$] $\langle c \neq 0 \rangle$ by simp
 moreover
 assume integrable lebesgue f
 ultimately show ?thesis
 by (subst lebesgue_real_affine[OF c, of t]) (auto simp: integral_density integral_distr)
 next
 assume $\neg \text{integrable lebesgue } f$ with c show ?thesis
 by (simp add: lebesgue_integrable_real_affine_iff not_integrable_integral_eq)
 qed

lemma *has_bochner_integral_lebesgue_real_affine_iff*:
 fixes $i :: 'a :: \text{euclidean\_space}$
 shows $c \neq 0 \implies$
 $\text{has\_bochner\_integral lebesgue } f \ i \longleftrightarrow$
 $\text{has\_bochner\_integral lebesgue } (\lambda x. f(t + c * x)) \ (i /_R |c|)$
 unfolding has_bochner_integral_iff lebesgue_integrable_real_affine_iff
 by (simp_all add: lebesgue_integral_real_affine[symmetric] divideR_right cong: conj_cong)

```

lemma has_bochner_integral_reflect_real_lemma[intro]:
  fixes  $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
  assumes has_bochner_integral (lebesgue_on  $\{a..b\}$ )  $f$   $i$ 
  shows has_bochner_integral (lebesgue_on  $\{-b..-a\}$ )  $(\lambda x. f(-x))$   $i$ 
proof -
  have eq: indicat_real  $\{a..b\}$   $(-x) *_R f(-x) = \text{indicat\_real } \{-b..-a\} x *_R f(-x)$  for  $x$ 
    by (auto simp: indicator_def)
  have  $i$ : has_bochner_integral lebesgue  $(\lambda x. \text{indicator } \{a..b\} x *_R f x)$   $i$ 
    using assms by (auto simp: has_bochner_integral_restrict_space)
  then have has_bochner_integral lebesgue  $(\lambda x. \text{indicator } \{-b..-a\} x *_R f(-x))$   $i$ 
    using has_bochner_integral_lebesgue_real_affine_iff [of  $-1$   $(\lambda x. \text{indicator } \{a..b\} x *_R f x)$   $i$   $0$ ]
    by (auto simp: eq)
  then show ?thesis
    by (auto simp: has_bochner_integral_restrict_space)
qed

lemma has_bochner_integral_reflect_real[simp]:
  fixes  $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
  shows has_bochner_integral (lebesgue_on  $\{-b..-a\}$ )  $(\lambda x. f(-x))$   $i \longleftrightarrow \text{has\_bochner\_integral}$ 
  (lebesgue_on  $\{a..b\}$ )  $f$   $i$ 
  by (auto simp: dest: has_bochner_integral_reflect_real_lemma)

lemma integrable_reflect_real[simp]:
  fixes  $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
  shows integrable (lebesgue_on  $\{-b..-a\}$ )  $(\lambda x. f(-x)) \longleftrightarrow \text{integrable}$  (lebesgue_on
   $\{a..b\}$ )  $f$ 
  by (metis has_bochner_integral_iff has_bochner_integral_reflect_real)

lemma integral_reflect_real[simp]:
  fixes  $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
  shows  $\text{integral}^L$  (lebesgue_on  $\{-b..-a\}$ )  $(\lambda x. f(-x)) = \text{integral}^L$  (lebesgue_on
   $\{a..b::\text{real}\}$ )  $f$ 
  using has_bochner_integral_reflect_real [of  $b$   $a$   $f$ ]
  by (metis has_bochner_integral_iff not_integrable_integral_eq)

```

6.19.25 More results on integrability

```

lemma integrable_on_all_intervals_UNIV:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{banach}$ 
  assumes intf:  $\bigwedge a b. f \text{ integrable\_on } \text{cbox } a b$ 
    and normf:  $\bigwedge x. \text{norm}(f x) \leq g x$  and  $g: g \text{ integrable\_on } \text{UNIV}$ 
  shows  $f \text{ integrable\_on } \text{UNIV}$ 
proof -
  have intg:  $(\forall a b. g \text{ integrable\_on } \text{cbox } a b)$ 
    and gle_e:  $\forall e > 0. \exists B > 0. \forall a b c d.$ 

```

```

ball 0 B ⊆ cbox a b ∧ cbox a b ⊆ cbox c d ⟶
|integral (cbox a b) g - integral (cbox c d) g|
< e

using g
by (auto simp: integrable_alt_subset [of UNIV] intf)
have le: norm (integral (cbox a b) f - integral (cbox c d) f) ≤ |integral (cbox a
b) g - integral (cbox c d) g|
  if cbox a b ⊆ cbox c d for a b c d
proof -
  have norm (integral (cbox a b) f - integral (cbox c d) f) = norm (integral
(cbox c d - cbox a b) f)
    using intf that by (simp add: norm_minus_commute integral_setdiff)
  also have ... ≤ integral (cbox c d - cbox a b) g
  proof (rule integral_norm_bound_integral [OF _ normf])
    show f integrable_on cbox c d - cbox a b g integrable_on cbox c d - cbox a b
      by (meson integrable_integral integrable_setdiff intf intg negligible_setdiff
that)+
  qed
  also have ... = integral (cbox c d) g - integral (cbox a b) g
    using intg that by (simp add: integral_setdiff)
  also have ... ≤ |integral (cbox a b) g - integral (cbox c d) g|
    by simp
  finally show ?thesis .
qed
show ?thesis
  using gle_e
  apply (simp add: integrable_alt_subset [of UNIV] intf)
  apply (erule imp_forward all_forward ex_forward asm_rl)+
  by (meson not_less order_trans le)
qed

lemma integrable_on_all_intervals_integrable_bound:
fixes f :: 'a::euclidean_space ⇒ 'b::banach
assumes intf: ⋀a b. (λx. if x ∈ S then f x else 0) integrable_on cbox a b
  and normf: ⋀x. x ∈ S ⟹ norm(f x) ≤ g x and g: g integrable_on S
shows f integrable_on S
  using integrable_on_all_intervals_UNIV [OF intf, of (λx. if x ∈ S then g x
else 0)]
  by (simp add: g integrable_restrict_UNIV normf)

lemma measurable_bounded_lemma:
fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
assumes f: f ∈ borel_measurable_lebesgue and g: g integrable_on cbox a b
  and normf: ⋀x. x ∈ cbox a b ⟹ norm(f x) ≤ g x
shows f integrable_on cbox a b
proof -
  have g absolutely_integrable_on cbox a b
    by (metis (full_types) add_increasing g le_add_same_cancel1 nonnegative_absolutely_integrable_1
norm_ge_zero normf)

```

```

then have integrable (lebesgue_on (cbox a b)) g
  by (simp add: integrable_restrict_space set_integrable_def)
then have integrable (lebesgue_on (cbox a b)) f
proof (rule Bochner_Integration.integrable_bound)
  show AE x in lebesgue_on (cbox a b). norm (f x) ≤ norm (g x)
    by (rule AE_I2) (auto intro: normf_order_trans)
qed (simp add: f measurable_restrict_space1)
then show ?thesis
  by (simp add: integrable_on_lebesgue_on)
qed

```

```

proposition measurable_bounded_by_integrable_imp_integrable:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes f: f ∈ borel_measurable (lebesgue_on S) and g: g integrable_on S
    and normf:  $\bigwedge x. x \in S \implies \text{norm}(f x) \leq g x$  and S: S ∈ sets lebesgue
  shows f integrable_on S
proof (rule integrable_on_all_intervals_integrable_bound [OF _ normf g])
  show (λx. if x ∈ S then f x else 0) integrable_on cbox a b for a b
  proof (rule measurable_bounded_lemma)
    show (λx. if x ∈ S then f x else 0) ∈ borel_measurable lebesgue
      by (simp add: S borel_measurable_if f)
    show (λx. if x ∈ S then g x else 0) integrable_on cbox a b
      by (simp add: g integrable_altD(1))
    show norm (if x ∈ S then f x else 0) ≤ (if x ∈ S then g x else 0) for x
      using normf by simp
  qed
qed

```

```

lemma measurable_bounded_by_integrable_imp_lebesgue_integrable:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes f: f ∈ borel_measurable (lebesgue_on S) and g: integrable (lebesgue_on S) g
    and normf:  $\bigwedge x. x \in S \implies \text{norm}(f x) \leq g x$  and S: S ∈ sets lebesgue
  shows integrable (lebesgue_on S) f
proof -
  have f absolutely_integrable_on S
    by (metis (no_types) S absolutely_integrable_integrable_bound f g integrable_on_lebesgue_on
    measurable_bounded_by_integrable_imp_integrable normf)
  then show ?thesis
    by (simp add: S integrable_restrict_space set_integrable_def)
qed

```

```

lemma measurable_bounded_by_integrable_imp_integrable_real:
  fixes f :: 'a::euclidean_space ⇒ real
  assumes f ∈ borel_measurable (lebesgue_on S) g integrable_on S  $\bigwedge x. x \in S$ 
 $\implies \text{abs}(f x) \leq g x$  S ∈ sets lebesgue
  shows f integrable_on S
  using measurable_bounded_by_integrable_imp_integrable [of f S g] assms by
  simp

```


6.19.26 Relation between Borel measurability and integrability.

lemma *integrable_imp_measurable_weak*:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$

assumes $S \in \text{sets lebesgue } f \text{ integrable\_on } S$

shows $f \in \text{borel\_measurable } (\text{lebesgue\_on } S)$

by (*metis* (*mono_tags*, *lifting*) *assms has_integral_implies_lebesgue_measurable borel_measurable_restrict_space_iff integrable_on_def sets.Int_space_eq2*)

lemma *integrable_imp_measurable*:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$

assumes $f \text{ integrable\_on } S$

shows $f \in \text{borel\_measurable } (\text{lebesgue\_on } S)$

proof –

have $(\text{UNIV}::'a \text{ set}) \in \text{sets lborel}$

by *simp*

then show *?thesis*

by (*metis* (*mono_tags*, *lifting*) *assms borel_measurable_if_D integrable_imp_measurable_weak integrable_restrict_UNIV lebesgue_on_UNIV_eq sets_lebesgue_on_refl*)

qed

lemma *integrable_iff_integrable_on*:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$

assumes $S \in \text{sets lebesgue } (\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial \text{lebesgue\_on } S) < \infty$

shows $\text{integrable } (\text{lebesgue\_on } S) f \longleftrightarrow f \text{ integrable\_on } S$

using *assms integrable_iff_bounded integrable_imp_measurable integrable_on_lebesgue_on*

by *blast*

lemma *absolutely_integrable_measurable*:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$

assumes $S \in \text{sets lebesgue}$

shows $f \text{ absolutely\_integrable\_on } S \longleftrightarrow f \in \text{borel\_measurable } (\text{lebesgue\_on } S)$

$\wedge \text{integrable } (\text{lebesgue\_on } S) (\text{norm } \circ f)$

(*is ?lhs = ?rhs*)

proof

assume $L: ?lhs$

then have $f \in \text{borel\_measurable } (\text{lebesgue\_on } S)$

by (*simp add: absolutely_integrable_on_def integrable_imp_measurable*)

then show *?rhs*

using *assms set_integrable_norm [of lebesgue S f] L*

by (*simp add: integrable_restrict_space set_integrable_def*)

next

assume *?rhs* **then show** *?lhs*

using *assms integrable_on_lebesgue_on*

by (*metis absolutely_integrable_integrable_bound comp_def eq_iff measurable_bounded_by_integrable_imp_integrable*)

qed

lemma *absolutely_integrable_measurable_real*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  real
assumes S  $\in$  sets lebesgue
shows f absolutely_integrable_on S  $\longleftrightarrow$ 
      f  $\in$  borel_measurable (lebesgue_on S)  $\wedge$  integrable (lebesgue_on S) ( $\lambda x. |f$ 
x|)
by (simp add: absolutely_integrable_measurable assms o_def)

```

```

lemma absolutely_integrable_measurable_real':
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes S  $\in$  sets lebesgue
  shows f absolutely_integrable_on S  $\longleftrightarrow$  f  $\in$  borel_measurable (lebesgue_on S)
 $\wedge$  ( $\lambda x. |f x|$ ) integrable_on S
  by (metis abs_absolutely_integrableI_1 absolutely_integrable_measurable_real
assms
      measurable_bounded_by_integrable_imp_integrable order_refl real_norm_def
set_integrable_abs set_lebesgue_integral_eq_integral(1))

```

```

lemma absolutely_integrable_imp_borel_measurable:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes f absolutely_integrable_on S S  $\in$  sets lebesgue
  shows f  $\in$  borel_measurable (lebesgue_on S)
  using absolutely_integrable_measurable assms by blast

```

```

lemma measurable_bounded_by_integrable_imp_absolutely_integrable:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes f  $\in$  borel_measurable (lebesgue_on S) S  $\in$  sets lebesgue
  and g integrable_on S and  $\bigwedge x. x \in S \implies \text{norm}(f x) \leq (g x)$ 
  shows f absolutely_integrable_on S
  using assms absolutely_integrable_integrable_bound measurable_bounded_by_integrable_imp_integral
by blast

```

```

proposition negligible_differentiable_vimage:
  fixes f :: 'a  $\Rightarrow$  'a::euclidean_space
  assumes negligible T
  and f':  $\bigwedge x. x \in S \implies \text{inj}(f' x)$ 
  and derf:  $\bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } S)$ 
  shows negligible {x  $\in$  S. f x  $\in$  T}
proof –
  define U where
    U  $\equiv \lambda n::\text{nat}. \{x \in S. \forall y. y \in S \wedge \text{norm}(y - x) < 1/n$ 
       $\longrightarrow \text{norm}(y - x) \leq n * \text{norm}(f y - f x)\}$ 
  have negligible {x  $\in$  U n. f x  $\in$  T} if n > 0 for n
  proof (subst locally_negligible_alt, clarify)
    fix a
    assume a: a  $\in$  U n and fa: f a  $\in$  T
    define V where V  $\equiv \{x. x \in U n \wedge f x \in T\} \cap \text{ball } a (1 / n / 2)$ 
    show  $\exists V. \text{openin } (\text{top\_of\_set } \{x \in U n. f x \in T\}) V \wedge a \in V \wedge \text{negligible } V$ 
    proof (intro exI conjI)
      have noxy:  $\text{norm}(x - y) \leq n * \text{norm}(f x - f y)$  if x  $\in$  V y  $\in$  V for x y

```

```

    using that unfolding U_def V_def mem_Collect_eq Int_iff mem_ball
dist_norm
  by (meson norm_triangle_half_r)
then have inj_on f V
  by (force simp: inj_on_def)
then obtain g where g:  $\bigwedge x. x \in V \implies g(f x) = x$ 
  by (metis inv_into_f_f)
have  $\exists T' B. \text{open } T' \wedge f x \in T' \wedge$ 
   $(\forall y \in f^{-1} V \cap T \cap T'. \text{norm } (g y - g(f x)) \leq B * \text{norm } (y - f x))$ 
  if  $f x \in T \wedge x \in V$  for x
  using that noxy
  by (rule_tac x = ball (f x) 1 in exI) (force simp: g)
then have negligible (g ` (f ` V  $\cap$  T))
  by (force simp: <negligible T> negligible_Int intro!: negligible_locally_Lipschitz_image)
moreover have  $V \subseteq g^{-1} (f^{-1} V \cap T)$ 
  by (force simp: g image_iff V_def)
ultimately show negligible V
  by (rule negligible_subset)
qed (use a fa V_def that in auto)
qed
with negligible_countable_Union have negligible  $(\bigcup n \in \{0 < ..\}. \{x. x \in U n \wedge f x \in T\})$ 
  by auto
moreover have  $\{x \in S. f x \in T\} \subseteq (\bigcup n \in \{0 < ..\}. \{x. x \in U n \wedge f x \in T\})$ 
proof clarsimp
  fix x
  assume  $x \in S$  and  $f x \in T$ 
  then obtain inj: inj(f' x) and der: (f has_derivative f' x) (at x within S)
    using assms by metis
  moreover have linear(f' x)
  and eps:  $\bigwedge \varepsilon. \varepsilon > 0 \implies \exists \delta > 0. \forall y \in S. \text{norm } (y - x) < \delta \implies$ 
     $\text{norm } (f y - f x - f' x (y - x)) \leq \varepsilon * \text{norm } (y - x)$ 
    using der by (auto simp: has_derivative_within_alt linear_linear)
  ultimately obtain g where linear g and g:  $g \circ f' x = \text{id}$ 
    using linear_injective_left_inverse by metis
  then obtain B where  $B > 0$  and B:  $\bigwedge z. B * \text{norm } z \leq \text{norm } (f' x z)$ 
    using linear_invertible_bounded_below_pos <linear (f' x)> by blast
  then obtain i where  $i \neq 0$  and i:  $1 / \text{real } i < B$ 
    by (metis (full_types) inverse_eq_divide real_arch_invD)
  then obtain  $\delta$  where  $\delta > 0$ 
    and  $\delta: \bigwedge y. \llbracket y \in S; \text{norm } (y - x) < \delta \rrbracket \implies$ 
     $\text{norm } (f y - f x - f' x (y - x)) \leq (B - 1 / \text{real } i) * \text{norm } (y - x)$ 
    using eps [of B - 1/i] by auto
  then obtain j where  $j \neq 0$  and j:  $\text{inverse } (\text{real } j) < \delta$ 
    using real_arch_inverse by blast
  have  $\text{norm } (y - x) / (\max i j) \leq \text{norm } (f y - f x)$ 
    if  $y \in S$  and less:  $\text{norm } (y - x) < 1 / (\max i j)$  for y
  proof -
    have  $1 / \text{real } (\max i j) < \delta$ 

```

```

      using j ⟨j ≠ 0⟩ ⟨0 < δ⟩
      by (auto simp: field_split_simps max_mult_distrib_left of_nat_max)
    then have norm (y - x) < δ
      using less by linarith
    with δ ⟨y ∈ S⟩ have le: norm (f y - f x - f' x (y - x)) ≤ B * norm (y - x)
  - norm (y - x) / i
    by (auto simp: algebra_simps)
    have norm (y - x) / real (max i j) ≤ norm (y - x) / real i
      using ⟨i ≠ 0⟩ ⟨j ≠ 0⟩ by (simp add: field_split_simps max_mult_distrib_left
of_nat_max less_max_iff_disj)
    also have ... ≤ norm (f y - f x)
      using B [of y-x] le norm_triangle_ineq3 [of f y - f x f' x (y - x)]
      by linarith
    finally show ?thesis .
qed
with ⟨x ∈ S⟩ ⟨i ≠ 0⟩ ⟨j ≠ 0⟩ show ∃ n ∈ {0 < ..}. x ∈ U n
  by (rule_tac x=max i j in bexI) (auto simp: field_simps U_def less_max_iff_disj)
qed
ultimately show ?thesis
  by (rule negligible_subset)
qed

```

```

lemma absolutely_integrable_Un:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes S: f absolutely_integrable_on S and T: f absolutely_integrable_on T
  shows f absolutely_integrable_on (S ∪ T)
proof -
  have [simp]: {x. (if x ∈ A then f x else 0) ≠ 0} = {x ∈ A. f x ≠ 0} for A
    by auto
  let ?ST = {x ∈ S. f x ≠ 0} ∩ {x ∈ T. f x ≠ 0}
  have ?ST ∈ sets lebesgue
  proof (rule Sigma_Algebra.sets.Int)
    have f_integrable_on S
      using S absolutely_integrable_on_def by blast
    then have (λx. if x ∈ S then f x else 0) integrable_on UNIV
      by (simp add: integrable_restrict_UNIV)
    then have borel: (λx. if x ∈ S then f x else 0) ∈ borel_measurable (lebesgue_on
UNIV)
      using integrable_imp_measurable lebesgue_on_UNIV_eq by blast
    then show {x ∈ S. f x ≠ 0} ∈ sets lebesgue
      unfolding borel_measurable_vimage_open
      by (rule allE [where x = -{0}]) auto
  next
    have f_integrable_on T
      using T absolutely_integrable_on_def by blast
    then have (λx. if x ∈ T then f x else 0) integrable_on UNIV
      by (simp add: integrable_restrict_UNIV)
    then have borel: (λx. if x ∈ T then f x else 0) ∈ borel_measurable (lebesgue_on
UNIV)

```

```

    using integrable_imp_measurable lebesgue_on_UNIV_eq by blast
  then show  $\{x \in T. f\ x \neq 0\} \in \text{sets lebesgue}$ 
    unfolding borel_measurable_vimage_open
    by (rule allE [where  $x = -\{0\}$ ]) auto
qed
  then have  $f \text{ absolutely\_integrable\_on } ?ST$ 
    by (rule set_integrable_subset [OF S]) auto
  then have Int:  $(\lambda x. \text{if } x \in ?ST \text{ then } f\ x \text{ else } 0) \text{ absolutely\_integrable\_on UNIV}$ 
    using absolutely_integrable_restrict_UNIV by blast
  have  $(\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) \text{ absolutely\_integrable\_on UNIV}$ 
     $(\lambda x. \text{if } x \in T \text{ then } f\ x \text{ else } 0) \text{ absolutely\_integrable\_on UNIV}$ 
    using S T absolutely_integrable_restrict_UNIV by blast+
  then have  $(\lambda x. (\text{if } x \in S \text{ then } f\ x \text{ else } 0) + (\text{if } x \in T \text{ then } f\ x \text{ else } 0)) \text{ absolutely\_integrable\_on UNIV}$ 
    by (rule set_integral_add)
  then have  $(\lambda x. ((\text{if } x \in S \text{ then } f\ x \text{ else } 0) + (\text{if } x \in T \text{ then } f\ x \text{ else } 0)) - (\text{if } x \in ?ST \text{ then } f\ x \text{ else } 0)) \text{ absolutely\_integrable\_on UNIV}$ 
    using Int by (rule set_integral_diff)
  then have  $(\lambda x. \text{if } x \in S \cup T \text{ then } f\ x \text{ else } 0) \text{ absolutely\_integrable\_on UNIV}$ 
    by (rule absolutely_integrable_spike) (auto intro: empty_imp_negligible)
  then show ?thesis
    unfolding absolutely_integrable_restrict_UNIV .
qed

```

```

lemma absolutely_integrable_on_combine:
  fixes  $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
  assumes  $f \text{ absolutely\_integrable\_on } \{a..c\}$ 
    and  $f \text{ absolutely\_integrable\_on } \{c..b\}$ 
    and  $a \leq c$ 
    and  $c \leq b$ 
  shows  $f \text{ absolutely\_integrable\_on } \{a..b\}$ 
  by (metis absolutely_integrable_Un assms ivl_disj_un_two_touch(4))

```

```

lemma uniform_limit_set_lebesgue_integral_at_top:
  fixes  $f :: 'a \Rightarrow \text{real} \Rightarrow 'b::\{\text{banach, second\_countable\_topology}\}$ 
    and  $g :: \text{real} \Rightarrow \text{real}$ 
  assumes bound:  $\bigwedge x\ y. x \in A \implies y \geq a \implies \text{norm } (f\ x\ y) \leq g\ y$ 
  assumes integrable:  $\text{set\_integrable } M\ \{a..\} g$ 
  assumes measurable:  $\bigwedge x. x \in A \implies \text{set\_borel\_measurable } M\ \{a..\} (f\ x)$ 
  assumes sets borel  $\subseteq \text{sets } M$ 
  shows  $\text{uniform\_limit } A\ (\lambda b\ x. \text{LINT } y:\{a..b\}|M. f\ x\ y)\ (\lambda x. \text{LINT } y:\{a..\}|M. f\ x\ y) \text{ at\_top}$ 
proof (cases  $A = \{\}$ )
case False
  then obtain  $x$  where  $x: x \in A$  by auto
  have  $g\_nonneg: g\ y \geq 0$  if  $y \geq a$  for  $y$ 
  proof -
    have  $0 \leq \text{norm } (f\ x\ y)$  by simp
    also have  $\dots \leq g\ y$  using bound[OF  $x$  that] by simp
  
```

finally show ?thesis .
qed

have integrable': set_integrable M {a..} ($\lambda y. f x y$) if $x \in A$ for x
unfolding set_integrable_def
proof (rule Bochner_Integration.integrable_bound)
show integrable M ($\lambda x. \text{indicator } \{a..\} x * g x$)
using integrable by (simp add: set_integrable_def)
show ($\lambda y. \text{indicat\_real } \{a..\} y *_R f x y$) $\in \text{borel\_measurable } M$ using measurable[OF that]
by (simp add: set_borel_measurable_def)
show $\text{AE } y \text{ in } M. \text{norm } (\text{indicat\_real } \{a..\} y *_R f x y) \leq \text{norm } (\text{indicat\_real } \{a..\} y * g y)$
using bound[OF that] by (intro AE_I2) (auto simp: indicator_def g_nonneg)
qed

show ?thesis

proof (rule uniform_limitI)
fix $e :: \text{real}$ assume $e > 0$
have sets [intro]: $A \in \text{sets } M$ if $A \in \text{sets borel}$ for A
using that assms by blast

have $((\lambda b. \text{LINT } y:\{a..b\}|M. g y) \longrightarrow (\text{LINT } y:\{a..}|M. g y)) \text{ at\_top}$
by (intro tendsto_set_lebesgue_integral_at_top assms sets) auto
with e obtain $b0 :: \text{real}$ where $b0: \forall b \geq b0. |(\text{LINT } y:\{a..}|M. g y) - (\text{LINT } y:\{a..b\}|M. g y)| < e$
by (auto simp: tendsto_iff eventually_at_top_linorder dist_real_def abs_minus_commute)
define b where $b = \max a b0$
have $a \leq b$ by (simp add: b_def)
from $b0$ have $|(\text{LINT } y:\{a..}|M. g y) - (\text{LINT } y:\{a..b\}|M. g y)| < e$
by (auto simp: b_def)
also have $\{a..b\} = \{a..b\} \cup \{b<..\}$ by (auto simp: b_def)
also have $|(\text{LINT } y:..|M. g y) - (\text{LINT } y:\{a..b\}|M. g y)| = |(\text{LINT } y:\{b<..\}|M. g y)|$
using $\langle a \leq b \rangle$ by (subst set_integral_Un) (auto intro!: set_integrable_subset[OF integrable])
also have $(\text{LINT } y:\{b<..\}|M. g y) \geq 0$
using $g\_nonneg \langle a \leq b \rangle$ unfolding set_lebesgue_integral_def
by (intro Bochner_Integration.integral_nonneg) (auto simp: indicator_def)
hence $|(\text{LINT } y:\{b<..\}|M. g y)| = (\text{LINT } y:\{b<..\}|M. g y)$ by simp
finally have less: $(\text{LINT } y:\{b<..\}|M. g y) < e$.

have eventually $(\lambda b. b \geq b0) \text{ at\_top}$ by (rule eventually_ge_at_top)
moreover have eventually $(\lambda b. b \geq a) \text{ at\_top}$ by (rule eventually_ge_at_top)
ultimately show eventually $(\lambda b. \forall x \in A. \text{dist } (\text{LINT } y:\{a..b\}|M. f x y) (\text{LINT } y:\{a..}|M. f x y) < e)$

at_top

proof eventually_elim
case (elim b)

```

show ?case
proof
  fix x assume x: x ∈ A
  have dist (LINT y:{a..b}|M. f x y) (LINT y:{a..}|M. f x y) =
    norm ((LINT y:{a..}|M. f x y) - (LINT y:{a..b}|M. f x y))
    by (simp add: dist_norm norm_minus_commute)
  also have {a..} = {a..b} ∪ {b<..} using elim by auto
  also have (LINT y:...|M. f x y) - (LINT y:{a..b}|M. f x y) = (LINT
y:{b<..}|M. f x y)
    using elim x
    by (subst set_integral_Un) (auto intro!: set_integrable_subset[OF inte-
grable])
  also have norm ... ≤ (LINT y:{b<..}|M. norm (f x y)) using elim x
    by (intro set_integral_norm_bound set_integrable_subset[OF integrable])
auto
  also have ... ≤ (LINT y:{b<..}|M. g y) using elim x bound g_nonneg
  by (intro set_integral_mono set_integrable_norm set_integrable_subset[OF
integrable]
    set_integrable_subset[OF integrable]) auto
  also have (LINT y:{b<..}|M. g y) ≥ 0
    using g_nonneg ⟨a ≤ b⟩ unfolding set_lebesgue_integral_def
    by (intro Bochner_Integration.integral_nonneg) (auto simp: indicator_def)
  hence (LINT y:{b<..}|M. g y) = |(LINT y:{b<..}|M. g y)| by simp
  also have ... = |(LINT y:{a..b} ∪ {b<..}|M. g y) - (LINT y:{a..b}|M. g
y)|
    using elim by (subst set_integral_Un) (auto intro!: set_integrable_subset[OF
integrable])
  also have {a..b} ∪ {b<..} = {a..} using elim by auto
  also have |(LINT y:{a..}|M. g y) - (LINT y:{a..b}|M. g y)| < e
    using b0 elim by blast
  finally show dist (LINT y:{a..b}|M. f x y) (LINT y:{a..}|M. f x y) < e .
qed
qed
qed
qed auto

```

Differentiability of inverse function (most basic form)

```

proposition has_derivative_inverse_within:
  fixes f :: 'a::real_normed_vector ⇒ 'b::euclidean_space
  assumes der_f: (f has_derivative f') (at a within S)
    and cont_g: continuous (at (f a) within f ` S) g
    and a ∈ S linear g' and id: g' ∘ f' = id
    and gf: ∧x. x ∈ S ⇒ g(f x) = x
  shows (g has_derivative g') (at (f a) within f ` S)
proof -
  have [simp]: g' (f' x) = x for x
    by (simp add: local.id pointfree_idE)
  have bounded_linear f'

```

```

and f':  $\bigwedge e. e > 0 \implies \exists d > 0. \forall y \in S. \text{norm } (y - a) < d \implies$ 
 $\text{norm } (f y - f a - f' (y - a)) \leq e * \text{norm } (y - a)$ 
using der_f by (auto simp: has_derivative_within_alt)
obtain C where C > 0 and C:  $\bigwedge x. \text{norm } (g' x) \leq C * \text{norm } x$ 
using linear_bounded_pos [OF <linear g'>] by metis
obtain B k where B > 0 k > 0
and Bk:  $\bigwedge x. \llbracket x \in S; \text{norm}(f x - f a) < k \rrbracket \implies \text{norm}(x - a) \leq B * \text{norm}(f x$ 
- f a)
proof -
obtain B where B > 0 and B:  $\bigwedge x. B * \text{norm } x \leq \text{norm } (f' x)$ 
using linear_inj_bounded_below_pos [of f'] <linear g'> id der_f has_derivative_linear
linear_invertible_bounded_below_pos by blast
then obtain d where d > 0
and d:  $\bigwedge y. \llbracket y \in S; \text{norm } (y - a) < d \rrbracket \implies$ 
 $\text{norm } (f y - f a - f' (y - a)) \leq B / 2 * \text{norm } (y - a)$ 
using f' [of B/2] by auto
then obtain e where e > 0
and e:  $\bigwedge x. \llbracket x \in S; \text{norm } (f x - f a) < e \rrbracket \implies \text{norm } (g (f x) - g (f a)) < d$ 
using cont_g by (auto simp: continuous_within_eps_delta_dist_norm)
show thesis
proof
show 2/B > 0
using <B > 0> by simp
show  $\text{norm } (x - a) \leq 2 / B * \text{norm } (f x - f a)$ 
if  $x \in S$   $\text{norm } (f x - f a) < e$  for x
proof -
have xa:  $\text{norm } (x - a) < d$ 
using e [OF that] gf by (simp add: <a ∈ S> that)
have *:  $\llbracket \text{norm}(y - f') \leq B / 2 * \text{norm } x; B * \text{norm } x \leq \text{norm } f \rrbracket$ 
 $\implies \text{norm } y \geq B / 2 * \text{norm } x$  for  $y f'::'b$  and  $x::'a$ 
using norm_triangle_ineq3 [of y f'] by linarith
show ?thesis
using * [OF d [OF <x ∈ S> xa] B] <B > 0> by (simp add: field_simps)
qed
qed (use <e > 0> in auto)
qed
show ?thesis
unfolding has_derivative_within_alt
proof (intro conjI impI allI)
show bounded_linear g'
using <linear g'> by (simp add: linear_linear)
next
fix e :: real
assume e > 0
then obtain d where d > 0
and d:  $\bigwedge y. \llbracket y \in S; \text{norm } (y - a) < d \rrbracket \implies$ 
 $\text{norm } (f y - f a - f' (y - a)) \leq e / (B * C) * \text{norm } (y - a)$ 
using f' [of e / (B * C)] <B > 0> <C > 0> by auto
have  $\text{norm } (x - a - g' (f x - f a)) \leq e * \text{norm } (f x - f a)$ 

```



```

    if  $x \in S$  and  $lt\_k$ :  $norm (f x - f a) < k$  and  $lt\_dB$ :  $norm (f x - f a) < d/B$ 
  for  $x$ 
  proof -
    have  $norm (x - a) \leq B * norm (f x - f a)$ 
      using  $Bk\ lt\_k\ \langle x \in S \rangle$  by blast
    also have  $\dots < d$ 
      by (metis  $\langle 0 < B \rangle\ lt\_dB\ mult.commute\ pos\_less\_divide\_eq$ )
    finally have  $lt\_d$ :  $norm (x - a) < d$  .
    have  $norm (x - a - g' (f x - f a)) \leq norm (g' (f x - f a - (f' (x - a))))$ 
      by (simp add: linear_diff [OF  $\langle linear\ g' \rangle$ ] norm_minus_commute)
    also have  $\dots \leq C * norm (f x - f a - f' (x - a))$ 
      using  $C$  by blast
    also have  $\dots \leq e * norm (f x - f a)$ 
  proof -
    have  $norm (f x - f a - f' (x - a)) \leq e / (B * C) * norm (x - a)$ 
      using  $d\ [OF\ \langle x \in S \rangle\ lt\_d]$  .
    also have  $\dots \leq (norm (f x - f a) * e) / C$ 
      using  $\langle B > 0 \rangle\ \langle C > 0 \rangle\ \langle e > 0 \rangle$  by (simp add: field_simps  $Bk\ lt\_k\ \langle x \in S \rangle$ )
    finally show ?thesis
      using  $\langle C > 0 \rangle$  by (simp add: field_simps)
  qed
  finally show ?thesis .
qed
with  $\langle k > 0 \rangle\ \langle B > 0 \rangle\ \langle d > 0 \rangle\ \langle a \in S \rangle$ 
show  $\exists d > 0. \forall y \in f^{-1} S.$ 
 $norm (y - f a) < d \longrightarrow$ 
 $norm (g y - g (f a) - g' (y - f a)) \leq e * norm (y - f a)$ 
by (rule_tac  $x = \min k (d / B)$  in exI) (auto simp: gf)
qed
qed
end

```

6.20 Complex Analysis Basics

Definitions of analytic and holomorphic functions, limit theorems, complex differentiation

```

theory Complex_Analysis_Basics
  imports Derivative HOL-Library.Nonpos_Ints
begin

```

6.20.1 General lemmas

```

lemma nonneg_Reals_cmod_eq_Re:  $z \in \mathbb{R}_{\geq 0} \implies norm\ z = Re\ z$ 
  by (simp add: complex_nonneg_Reals_iff cmod_eq_Re)

```

```

lemma fact_cancel:

```

```

fixes  $c :: 'a::real\_field$ 
shows  $of\_nat (Suc\ n) * c / (fact (Suc\ n)) = c / (fact\ n)$ 
using  $of\_nat\_neq\_0$  by force

```

```

lemma vector_derivative_cnj_within:
  assumes  $at\ x\ within\ A \neq bot$  and  $f$  differentiable at x within A
  shows  $vector\_derivative\ (\lambda z. cnj\ (f\ z))\ (at\ x\ within\ A) =$ 
     $cnj\ (vector\_derivative\ f\ (at\ x\ within\ A))$  (is  $\_ = cnj\ ?D$ )
proof –
  let  $?D = vector\_derivative\ f\ (at\ x\ within\ A)$ 
  from assms have  $(f\ has\_vector\_derivative\ ?D)\ (at\ x\ within\ A)$ 
    by (subst (asm) vector_derivative_works)
  hence  $((\lambda x. cnj\ (f\ x))\ has\_vector\_derivative\ cnj\ ?D)\ (at\ x\ within\ A)$ 
    by (rule has_vector_derivative_cnj)
  thus  $?thesis$  using assms by (auto dest: vector_derivative_within)
qed

```

```

lemma vector_derivative_cnj:
  assumes  $f$  differentiable at x
  shows  $vector\_derivative\ (\lambda z. cnj\ (f\ z))\ (at\ x) = cnj\ (vector\_derivative\ f\ (at\ x))$ 
using assms by (intro vector_derivative_cnj_within) auto

```

```

lemma
  shows open_halfspace_Re_lt:  $open\ \{z. Re(z) < b\}$ 
    and open_halfspace_Re_gt:  $open\ \{z. Re(z) > b\}$ 
    and closed_halfspace_Re_ge:  $closed\ \{z. Re(z) \geq b\}$ 
    and closed_halfspace_Re_le:  $closed\ \{z. Re(z) \leq b\}$ 
    and closed_halfspace_Re_eq:  $closed\ \{z. Re(z) = b\}$ 
    and open_halfspace_Im_lt:  $open\ \{z. Im(z) < b\}$ 
    and open_halfspace_Im_gt:  $open\ \{z. Im(z) > b\}$ 
    and closed_halfspace_Im_ge:  $closed\ \{z. Im(z) \geq b\}$ 
    and closed_halfspace_Im_le:  $closed\ \{z. Im(z) \leq b\}$ 
    and closed_halfspace_Im_eq:  $closed\ \{z. Im(z) = b\}$ 
by (intro open_Collect_less closed_Collect_le closed_Collect_eq continuous_on_Re
    continuous_on_Im continuous_on_id continuous_on_const)+

```

```

lemma closed_complex_Reals:  $closed\ (\mathbb{R} :: complex\ set)$ 
proof –
  have  $(\mathbb{R} :: complex\ set) = \{z. Im\ z = 0\}$ 
    by (auto simp: complex_is_Real_iff)
  then show  $?thesis$ 
    by (metis closed_halfspace_Im_eq)
qed

```

```

lemma closed_Real_halfspace_Re_le:  $closed\ (\mathbb{R} \cap \{w. Re\ w \leq x\})$ 
by (simp add: closed_Int closed_complex_Reals closed_halfspace_Re_le)

```

```

lemma closed_nonpos_Reals_complex [simp]:  $closed\ (\mathbb{R}_{\leq 0} :: complex\ set)$ 

```

```

proof -
  have  $\mathbb{R}_{\leq 0} = \mathbb{R} \cap \{z. \operatorname{Re}(z) \leq 0\}$ 
    using complex_nonpos_Reals_iff_complex_is_Real_iff by auto
  then show ?thesis
    by (metis closed_Real_halfspace_Re_le)
qed

lemma closed_Real_halfspace_Re_ge: closed  $(\mathbb{R} \cap \{w. x \leq \operatorname{Re}(w)\})$ 
  using closed_halfspace_Re_ge
  by (simp add: closed_Int closed_complex_Reals)

lemma closed_nonneg_Reals_complex [simp]: closed  $(\mathbb{R}_{\geq 0} :: \text{complex set})$ 
proof -
  have  $\mathbb{R}_{\geq 0} = \mathbb{R} \cap \{z. \operatorname{Re}(z) \geq 0\}$ 
    using complex_nonneg_Reals_iff_complex_is_Real_iff by auto
  then show ?thesis
    by (metis closed_Real_halfspace_Re_ge)
qed

lemma closed_real_abs_le: closed  $\{w \in \mathbb{R}. |\operatorname{Re} w| \leq r\}$ 
proof -
  have  $\{w \in \mathbb{R}. |\operatorname{Re} w| \leq r\} = (\mathbb{R} \cap \{w. \operatorname{Re} w \leq r\}) \cap (\mathbb{R} \cap \{w. \operatorname{Re} w \geq -r\})$ 
    by auto
  then show closed  $\{w \in \mathbb{R}. |\operatorname{Re} w| \leq r\}$ 
    by (simp add: closed_Int closed_Real_halfspace_Re_ge closed_Real_halfspace_Re_le)
qed

lemma real_lim:
  fixes  $l::\text{complex}$ 
  assumes  $(f \longrightarrow l) F$  and  $\neg \text{trivial\_limit } F$  and eventually  $P F$  and  $\bigwedge a. P a$ 
 $\implies f a \in \mathbb{R}$ 
  shows  $l \in \mathbb{R}$ 
proof (rule Lim_in_closed_set[OF closed_complex_Reals _ assms(2,1)])
  show eventually  $(\lambda x. f x \in \mathbb{R}) F$ 
    using assms(3, 4) by (auto intro: eventually_mono)
qed

lemma real_lim_sequentially:
  fixes  $l::\text{complex}$ 
  shows  $(f \longrightarrow l) \text{ sequentially} \implies (\exists N. \forall n \geq N. f n \in \mathbb{R}) \implies l \in \mathbb{R}$ 
  by (rule real_lim [where  $F=\text{sequentially}$ ]) (auto simp: eventually_sequentially)

lemma real_series:
  fixes  $l::\text{complex}$ 
  shows  $f \text{ sums } l \implies (\bigwedge n. f n \in \mathbb{R}) \implies l \in \mathbb{R}$ 
  unfolding sums_def
  by (metis real_lim_sequentially sum_in_Reals)

lemma Lim_null_comparison_Re:

```

assumes *eventually* $(\lambda x. \text{norm}(f x) \leq \text{Re}(g x)) \ F \ (g \longrightarrow 0) \ F$ **shows** $(f \longrightarrow 0) \ F$
by (rule *Lim_null_comparison*[*OF assms(1)*] *tendsto_eq_intros assms(2)*) + *simp*

6.20.2 Holomorphic functions

definition *holomorphic_on* :: $[\text{complex} \Rightarrow \text{complex}, \text{complex set}] \Rightarrow \text{bool}$
 (infixl (*holomorphic'_on*) 50)
where $f \text{ holomorphic\_on } s \equiv \forall x \in s. f \text{ field\_differentiable } (\text{at } x \text{ within } s)$

named_theorems *holomorphic_intros* structural introduction rules for *holomorphic_on*

lemma *holomorphic_onI* [*intro?*]: $(\bigwedge x. x \in s \implies f \text{ field\_differentiable } (\text{at } x \text{ within } s)) \implies f \text{ holomorphic\_on } s$
by (*simp add: holomorphic_on_def*)

lemma *holomorphic_onD* [*dest?*]: $\llbracket f \text{ holomorphic\_on } s; x \in s \rrbracket \implies f \text{ field\_differentiable } (\text{at } x \text{ within } s)$
by (*simp add: holomorphic_on_def*)

lemma *holomorphic_on_imp_differentiable_on*:
 $f \text{ holomorphic\_on } s \implies f \text{ differentiable\_on } s$
unfolding *holomorphic_on_def differentiable_on_def*
by (*simp add: field_differentiable_imp_differentiable*)

lemma *holomorphic_on_imp_differentiable_at*:
 $\llbracket f \text{ holomorphic\_on } s; \text{open } s; x \in s \rrbracket \implies f \text{ field\_differentiable } (\text{at } x)$
using *at_within_open holomorphic_on_def* **by** *fastforce*

lemma *holomorphic_on_empty* [*holomorphic_intros*]: $f \text{ holomorphic\_on } \{\}$
by (*simp add: holomorphic_on_def*)

lemma *holomorphic_on_open*:
 $\text{open } s \implies f \text{ holomorphic\_on } s \longleftrightarrow (\forall x \in s. \exists f'. \text{DERIV } f x :> f')$
by (*auto simp: holomorphic_on_def field_differentiable_def has_field_derivative_def at_within_open [of _ s]*)

lemma *holomorphic_on_UN_open*:
assumes $\bigwedge n. n \in I \implies f \text{ holomorphic\_on } A \ n \ \bigwedge n. n \in I \implies \text{open } (A \ n)$
shows $f \text{ holomorphic\_on } (\bigcup_{n \in I}. A \ n)$
proof –
have $f \text{ field\_differentiable at } z \text{ within } (\bigcup_{n \in I}. A \ n)$ **if** $z \in (\bigcup_{n \in I}. A \ n)$ **for** z
proof –
from that obtain n **where** $n \in I \ z \in A \ n$
by *blast*
hence $f \text{ holomorphic\_on } A \ n$ **open** $(A \ n)$
by (*simp add: assms*) +

```

with ⟨ $z \in A$   $n$ ⟩ have  $f$  field_differentiable at  $z$ 
  by (auto simp: holomorphic_on_open field_differentiable_def)
thus ?thesis
  by (meson field_differentiable_at_within)
qed
thus ?thesis
  by (auto simp: holomorphic_on_def)
qed

```

```

lemma holomorphic_on_imp_continuous_on:
   $f$  holomorphic_on  $s \implies$  continuous_on  $s$   $f$ 
by (metis field_differentiable_imp_continuous_at continuous_on_eq_continuous_within
holomorphic_on_def)

```

```

lemma holomorphic_closedin_preimage_constant:
  assumes  $f$  holomorphic_on  $D$ 
  shows closedin (top_of_set  $D$ )  $\{z \in D. f\ z = a\}$ 
by (simp add: asms continuous_closedin_preimage_constant holomorphic_on_imp_continuous_on)

```

```

lemma holomorphic_closed_preimage_constant:
  assumes  $f$  holomorphic_on UNIV
  shows closed  $\{z. f\ z = a\}$ 
using holomorphic_closedin_preimage_constant [OF asms] by simp

```

```

lemma holomorphic_on_subset [elim]:
   $f$  holomorphic_on  $s \implies t \subseteq s \implies f$  holomorphic_on  $t$ 
unfolding holomorphic_on_def
by (metis field_differentiable_within_subset subsetD)

```

```

lemma holomorphic_transform:  $\llbracket f$  holomorphic_on  $s; \bigwedge x. x \in s \implies f\ x = g\ x \rrbracket$ 
 $\implies g$  holomorphic_on  $s$ 
by (metis field_differentiable_transform_within linordered_field_no_ub holomorphic_on_def)

```

```

lemma holomorphic_cong:  $s = t \implies (\bigwedge x. x \in s \implies f\ x = g\ x) \implies f$  holomor-
phic_on  $s \longleftrightarrow g$  holomorphic_on  $t$ 
by (metis holomorphic_transform)

```

```

lemma holomorphic_on_linear [simp, holomorphic_intros]:  $((*)\ c)$  holomorphic_on
 $s$ 
unfolding holomorphic_on_def by (metis field_differentiable_linear)

```

```

lemma holomorphic_on_const [simp, holomorphic_intros]:  $(\lambda z. c)$  holomorphic_on
 $s$ 
unfolding holomorphic_on_def by (metis field_differentiable_const)

```

```

lemma holomorphic_on_ident [simp, holomorphic_intros]:  $(\lambda x. x)$  holomorphic_on
 $s$ 
unfolding holomorphic_on_def by (metis field_differentiable_ident)

```

lemma *holomorphic_on_id* [*simp*, *holomorphic_intros*]: *id* *holomorphic_on* *s*
unfolding *id_def* **by** (*rule* *holomorphic_on_ident*)

lemma *constant_on_imp_holomorphic_on*:
assumes *f* *constant_on* *A*
shows *f* *holomorphic_on* *A*
proof –
from *assms* **obtain** *c* **where** *c*: $\forall x \in A. f\ x = c$
unfolding *constant_on_def* **by** *blast*
have *f* *holomorphic_on* *A* $\longleftrightarrow (\lambda_. c)$ *holomorphic_on* *A*
by (*intro* *holomorphic_cong*) (*use* *c* **in** *auto*)
thus *?thesis*
by *simp*
qed

lemma *holomorphic_on_compose*:
f *holomorphic_on* *s* $\implies$ *g* *holomorphic_on* (*f* ‘ *s*) $\implies$ (*g* *o* *f*) *holomorphic_on* *s*
using *field_differentiable_compose_within*[*of* *f* _ *s* *g*]
by (*auto* *simp*: *holomorphic_on_def*)

lemma *holomorphic_on_compose_gen*:
f *holomorphic_on* *s* $\implies$ *g* *holomorphic_on* *t* $\implies$ *f* ‘ *s* $\subseteq$ *t* $\implies$ (*g* *o* *f*) *holomorphic_on* *s*
by (*metis* *holomorphic_on_compose* *holomorphic_on_subset*)

lemma *holomorphic_on_balls_imp_entire*:
assumes $\neg \text{bdd\_above } A \wedge r. r \in A \implies f \text{ holomorphic\_on ball } c\ r$
shows *f* *holomorphic_on* *B*
proof (*rule* *holomorphic_on_subset*)
show *f* *holomorphic_on* *UNIV* **unfolding** *holomorphic_on_def*
proof
fix *z* :: *complex*
from $\langle \neg \text{bdd\_above } A \rangle$ **obtain** *r* **where** *r*: $r \in A \wedge r > \text{norm } (z - c)$
by (*meson* *bdd_aboveI* *not_le*)
with *assms*(2) **have** *f* *holomorphic_on* *ball* *c* *r* **by** *blast*
moreover **from** *r* **have** $z \in \text{ball } c\ r$ **by** (*auto* *simp*: *dist_norm* *norm_minus_commute*)
ultimately **show** *f* *field_differentiable* *at* *z*
by (*auto* *simp*: *holomorphic_on_def* *at_within_open*[*of* _ *ball* *c* *r*])
qed
qed *auto*

lemma *holomorphic_on_balls_imp_entire'*:
assumes $\bigwedge r. r > 0 \implies f \text{ holomorphic\_on ball } c\ r$
shows *f* *holomorphic_on* *B*
proof (*rule* *holomorphic_on_balls_imp_entire*)
{
fix *M* :: *real*
have $\exists x. x > \max M\ 0$ **by** (*intro* *gt_ex*)
}

```

    hence  $\exists x > 0. x > M$  by auto
  }
  thus  $\neg \text{bdd\_above } \{(0::\text{real}) < ..\}$  unfolding bdd_above_def
    by (auto simp: not_le)
qed (insert assms, auto)

```

```

lemma holomorphic_on_minus [holomorphic_intros]:  $f$  holomorphic_on  $s \implies$ 
 $(\lambda z. -(f z))$  holomorphic_on  $s$ 
  by (metis field_differentiable_minus holomorphic_on_def)

```

```

lemma holomorphic_on_add [holomorphic_intros]:
 $\llbracket f$  holomorphic_on  $s$ ;  $g$  holomorphic_on  $s \rrbracket \implies (\lambda z. f z + g z)$  holomorphic_on  $s$ 
  unfolding holomorphic_on_def by (metis field_differentiable_add)

```

```

lemma holomorphic_on_diff [holomorphic_intros]:
 $\llbracket f$  holomorphic_on  $s$ ;  $g$  holomorphic_on  $s \rrbracket \implies (\lambda z. f z - g z)$  holomorphic_on  $s$ 
  unfolding holomorphic_on_def by (metis field_differentiable_diff)

```

```

lemma holomorphic_on_mult [holomorphic_intros]:
 $\llbracket f$  holomorphic_on  $s$ ;  $g$  holomorphic_on  $s \rrbracket \implies (\lambda z. f z * g z)$  holomorphic_on  $s$ 
  unfolding holomorphic_on_def by (metis field_differentiable_mult)

```

```

lemma holomorphic_on_inverse [holomorphic_intros]:
 $\llbracket f$  holomorphic_on  $s$ ;  $\bigwedge z. z \in s \implies f z \neq 0 \rrbracket \implies (\lambda z. \text{inverse } (f z))$  holomor-
phic_on  $s$ 
  unfolding holomorphic_on_def by (metis field_differentiable_inverse)

```

```

lemma holomorphic_on_divide [holomorphic_intros]:
 $\llbracket f$  holomorphic_on  $s$ ;  $g$  holomorphic_on  $s$ ;  $\bigwedge z. z \in s \implies g z \neq 0 \rrbracket \implies (\lambda z. f z$ 
 $/ g z)$  holomorphic_on  $s$ 
  unfolding holomorphic_on_def by (metis field_differentiable_divide)

```

```

lemma holomorphic_on_power [holomorphic_intros]:
 $f$  holomorphic_on  $s \implies (\lambda z. (f z)^n)$  holomorphic_on  $s$ 
  unfolding holomorphic_on_def by (metis field_differentiable_power)

```

```

lemma holomorphic_on_sum [holomorphic_intros]:
 $(\bigwedge i. i \in I \implies (f i)$  holomorphic_on  $s) \implies (\lambda x. \text{sum } (\lambda i. f i x) I)$  holomorphic_on
 $s$ 
  unfolding holomorphic_on_def by (metis field_differentiable_sum)

```

```

lemma holomorphic_on_prod [holomorphic_intros]:
 $(\bigwedge i. i \in I \implies (f i)$  holomorphic_on  $s) \implies (\lambda x. \text{prod } (\lambda i. f i x) I)$  holomorphic_on
 $s$ 
  by (induction I rule: infinite_finite_induct) (auto intro: holomorphic_intros)

```

```

lemma holomorphic_pochhammer [holomorphic_intros]:
 $f$  holomorphic_on  $A \implies (\lambda s. \text{pochhammer } (f s) n)$  holomorphic_on  $A$ 
  by (induction n) (auto intro!: holomorphic_intros simp: pochhammer_Suc)

```

lemma *holomorphic_on_scaleR* [*holomorphic_intros*]:
 $f \text{ holomorphic\_on } A \implies (\lambda x. c *_{\mathbb{R}} f x) \text{ holomorphic\_on } A$
by (*auto simp: scaleR_conv_of_real intro!: holomorphic_intros*)

lemma *holomorphic_on_Un* [*holomorphic_intros*]:
assumes $f \text{ holomorphic\_on } A$ $f \text{ holomorphic\_on } B$ *open A open B*
shows $f \text{ holomorphic\_on } (A \cup B)$
using *assms* **by** (*auto simp: holomorphic_on_def at_within_open[of _ A]*
 $\text{at\_within\_open[of _ B] at\_within\_open[of _ } A \cup B]$
open_Un)

lemma *holomorphic_on_If_Un* [*holomorphic_intros*]:
assumes $f \text{ holomorphic\_on } A$ $g \text{ holomorphic\_on } B$ *open A open B*
assumes $\bigwedge z. z \in A \implies z \in B \implies f z = g z$
shows $(\lambda z. \text{if } z \in A \text{ then } f z \text{ else } g z) \text{ holomorphic\_on } (A \cup B)$ (*is ?h holomorphic_on _*)
proof (*intro holomorphic_on_Un*)
note $\langle f \text{ holomorphic\_on } A \rangle$
also have $f \text{ holomorphic\_on } A \longleftrightarrow ?h \text{ holomorphic\_on } A$
by (*intro holomorphic_cong*) *auto*
finally show
next
note $\langle g \text{ holomorphic\_on } B \rangle$
also have $g \text{ holomorphic\_on } B \longleftrightarrow ?h \text{ holomorphic\_on } B$
using *assms* **by** (*intro holomorphic_cong*) *auto*
finally show
qed (*insert assms, auto*)

lemma *holomorphic_derivI*:
 $\llbracket f \text{ holomorphic\_on } S; \text{ open } S; x \in S \rrbracket$
 $\implies (f \text{ has\_field\_derivative deriv } f x) \text{ (at } x \text{ within } T)$
by (*metis DERIV_deriv_iff_field_differentiable at_within_open holomorphic_on_def*
has_field_derivative_at_within)

lemma *complex_derivative_transform_within_open*:
 $\llbracket f \text{ holomorphic\_on } s; g \text{ holomorphic\_on } s; \text{ open } s; z \in s; \bigwedge w. w \in s \implies f w = g w \rrbracket$
 $\implies \text{deriv } f z = \text{deriv } g z$
unfolding *holomorphic_on_def*
by (*rule DERIV_imp_deriv*)
(metis DERIV_deriv_iff_field_differentiable has_field_derivative_transform_within_open at_within_open)

lemma *holomorphic_nonconstant*:
assumes *holf: f holomorphic_on S and open S* $\xi \in S$ $\text{deriv } f \xi \neq 0$
shows $\neg f \text{ constant\_on } S$
by (*rule nonzero_deriv_nonconstant [of f deriv f ξ ξ S]*)
(use assms in <auto simp: holomorphic_derivI>)

6.20.3 Analyticity on a set

definition *analytic_on* (**infixl** (*analytic'_on*) 50)

where $f \text{ analytic\_on } S \equiv \forall x \in S. \exists e. 0 < e \wedge f \text{ holomorphic\_on } (\text{ball } x \ e)$

named_theorems *analytic_intros* introduction rules for proving analyticity

lemma *analytic_imp_holomorphic*: $f \text{ analytic\_on } S \implies f \text{ holomorphic\_on } S$

by (*simp add: at_within_open [OF _ open_ball] analytic_on_def holomorphic_on_def*)

(*metis centre_in_ball field_differentiable_at_within*)

lemma *analytic_on_open*: $\text{open } S \implies f \text{ analytic\_on } S \longleftrightarrow f \text{ holomorphic\_on } S$

apply (*auto simp: analytic_imp_holomorphic*)

apply (*auto simp: analytic_on_def holomorphic_on_def*)

by (*metis holomorphic_on_def holomorphic_on_subset open_contains_ball*)

lemma *analytic_on_imp_differentiable_at*:

$f \text{ analytic\_on } S \implies x \in S \implies f \text{ field\_differentiable } (\text{at } x)$

apply (*auto simp: analytic_on_def holomorphic_on_def*)

by (*metis open_ball centre_in_ball field_differentiable_within_open*)

lemma *analytic_on_subset*: $f \text{ analytic\_on } S \implies T \subseteq S \implies f \text{ analytic\_on } T$

by (*auto simp: analytic_on_def*)

lemma *analytic_on_Un*: $f \text{ analytic\_on } (S \cup T) \longleftrightarrow f \text{ analytic\_on } S \wedge f \text{ analytic\_on } T$

by (*auto simp: analytic_on_def*)

lemma *analytic_on_Union*: $f \text{ analytic\_on } (\bigcup \mathcal{T}) \longleftrightarrow (\forall T \in \mathcal{T}. f \text{ analytic\_on } T)$

by (*auto simp: analytic_on_def*)

lemma *analytic_on_UN*: $f \text{ analytic\_on } (\bigcup_{i \in I} S \ i) \longleftrightarrow (\forall i \in I. f \text{ analytic\_on } (S \ i))$

by (*auto simp: analytic_on_def*)

lemma *analytic_on_holomorphic*:

$f \text{ analytic\_on } S \longleftrightarrow (\exists T. \text{open } T \wedge S \subseteq T \wedge f \text{ holomorphic\_on } T)$

(**is** ?lhs = ?rhs)

proof –

have ?lhs $\longleftrightarrow (\exists T. \text{open } T \wedge S \subseteq T \wedge f \text{ analytic\_on } T)$

proof *safe*

assume $f \text{ analytic\_on } S$

then show $\exists T. \text{open } T \wedge S \subseteq T \wedge f \text{ analytic\_on } T$

apply (*simp add: analytic_on_def*)

apply (*rule exI [where x = $\bigcup \{U. \text{open } U \wedge f \text{ analytic\_on } U\}$], auto*)

apply (*metis open_ball analytic_on_open centre_in_ball*)

by (*metis analytic_on_def*)

next

```

fix T
assume open T S  $\subseteq$  T f analytic_on T
then show f analytic_on S
  by (metis analytic_on_subset)
qed
also have ...  $\longleftrightarrow$  ?rhs
  by (auto simp: analytic_on_open)
finally show ?thesis .
qed

```

```

lemma analytic_on_linear [analytic_intros,simp]: ((*) c) analytic_on S
  by (auto simp add: analytic_on_holomorphic)

```

```

lemma analytic_on_const [analytic_intros,simp]: ( $\lambda z.$  c) analytic_on S
  by (metis analytic_on_def holomorphic_on_const zero_less_one)

```

```

lemma analytic_on_ident [analytic_intros,simp]: ( $\lambda x.$  x) analytic_on S
  by (simp add: analytic_on_def gt_ex)

```

```

lemma analytic_on_id [analytic_intros]: id analytic_on S
  unfolding id_def by (rule analytic_on_ident)

```

```

lemma analytic_on_compose:
  assumes f: f analytic_on S
  and g: g analytic_on (f ' S)
  shows (g o f) analytic_on S
unfolding analytic_on_def
proof (intro ballI)
  fix x
  assume x: x  $\in$  S
  then obtain e where e: 0 < e and fh: f holomorphic_on ball x e using f
  by (metis analytic_on_def)
  obtain e' where e': 0 < e' and gh: g holomorphic_on ball (f x) e' using g
  by (metis analytic_on_def g image_eqI x)
  have isCont f x
  by (metis analytic_on_imp_differentiable_at field_differentiable_imp_continuous_at
f x)
  with e' obtain d where d: 0 < d and fd: f ' ball x d  $\subseteq$  ball (f x) e'
  by (auto simp: continuous_at_ball)
  have g o f holomorphic_on ball x (min d e)
  apply (rule holomorphic_on_compose)
  apply (metis fh holomorphic_on_subset min.bounded_iff order_refl subset_ball)
  by (metis fd gh holomorphic_on_subset image_mono min.cobounded1 sub-
set_ball)
  then show  $\exists e > 0.$  g o f holomorphic_on ball x e
  by (metis d e min_less_iff_conj)
qed

```

```

lemma analytic_on_compose_gen:

```

$$f \text{ analytic\_on } S \implies g \text{ analytic\_on } T \implies (\bigwedge z. z \in S \implies f z \in T) \\ \implies g \circ f \text{ analytic\_on } S$$

by (metis analytic_on_compose analytic_on_subset image_subset_iff)

lemma analytic_on_neg [analytic_intros]:

$$f \text{ analytic\_on } S \implies (\lambda z. -(f z)) \text{ analytic\_on } S$$

by (metis analytic_on_holomorphic holomorphic_on_minus)

lemma analytic_on_add [analytic_intros]:
 assumes $f: f \text{ analytic\_on } S$
 and $g: g \text{ analytic\_on } S$
 shows $(\lambda z. f z + g z) \text{ analytic\_on } S$

unfolding analytic_on_def

proof (intro ballI)

fix z

assume $z: z \in S$

then obtain e where $e: 0 < e$ and $fh: f \text{ holomorphic\_on ball } z e$ using f

by (metis analytic_on_def)

obtain e' where $e': 0 < e'$ and $gh: g \text{ holomorphic\_on ball } z e'$ using g

by (metis analytic_on_def g z)

have $(\lambda z. f z + g z) \text{ holomorphic\_on ball } z (\min e e')$

apply (rule holomorphic_on_add)

apply (metis fh holomorphic_on_subset min.bounded_iff order_refl subset_ball)

by (metis gh holomorphic_on_subset min.bounded_iff order_refl subset_ball)

then show $\exists e > 0. (\lambda z. f z + g z) \text{ holomorphic\_on ball } z e$

by (metis e e' min_less_iff_conj)

qed

lemma analytic_on_diff [analytic_intros]:
 assumes $f: f \text{ analytic\_on } S$
 and $g: g \text{ analytic\_on } S$
 shows $(\lambda z. f z - g z) \text{ analytic\_on } S$

unfolding analytic_on_def

proof (intro ballI)

fix z

assume $z: z \in S$

then obtain e where $e: 0 < e$ and $fh: f \text{ holomorphic\_on ball } z e$ using f

by (metis analytic_on_def)

obtain e' where $e': 0 < e'$ and $gh: g \text{ holomorphic\_on ball } z e'$ using g

by (metis analytic_on_def g z)

have $(\lambda z. f z - g z) \text{ holomorphic\_on ball } z (\min e e')$

apply (rule holomorphic_on_diff)

apply (metis fh holomorphic_on_subset min.bounded_iff order_refl subset_ball)

by (metis gh holomorphic_on_subset min.bounded_iff order_refl subset_ball)

then show $\exists e > 0. (\lambda z. f z - g z) \text{ holomorphic\_on ball } z e$

by (metis e e' min_less_iff_conj)

qed

lemma analytic_on_mult [analytic_intros]:

```

    assumes f: f analytic_on S
    and g: g analytic_on S
    shows ( $\lambda z. f z * g z$ ) analytic_on S
  unfolding analytic_on_def
  proof (intro ballI)
    fix z
    assume z:  $z \in S$ 
    then obtain e where  $e: 0 < e$  and fh: f holomorphic_on ball z e using f
    by (metis analytic_on_def)
    obtain e' where  $e': 0 < e'$  and gh: g holomorphic_on ball z e' using g
    by (metis analytic_on_def)
    have ( $\lambda z. f z * g z$ ) holomorphic_on ball z (min e e')
    apply (rule holomorphic_on_mult)
    apply (metis fh holomorphic_on_subset min.bounded_iff order_refl subset_ball)
    by (metis gh holomorphic_on_subset min.bounded_iff order_refl subset_ball)
    then show  $\exists e > 0. (\lambda z. f z * g z)$  holomorphic_on ball z e
    by (metis e e' min_less_iff_conj)
  qed

```

```

lemma analytic_on_inverse [analytic_intros]:
  assumes f: f analytic_on S
  and nz: ( $\bigwedge z. z \in S \implies f z \neq 0$ )
  shows ( $\lambda z. \text{inverse } (f z)$ ) analytic_on S
  unfolding analytic_on_def
  proof (intro ballI)
    fix z
    assume z:  $z \in S$ 
    then obtain e where  $e: 0 < e$  and fh: f holomorphic_on ball z e using f
    by (metis analytic_on_def)
    have continuous_on (ball z e) f
    by (metis fh holomorphic_on_imp_continuous_on)
    then obtain e' where  $e': 0 < e'$  and nz': ( $\bigwedge y. \text{dist } z y < e' \implies f y \neq 0$ )
    by (metis open_ball centre_in_ball continuous_on_open_avoid e z nz)
    have ( $\lambda z. \text{inverse } (f z)$ ) holomorphic_on ball z (min e e')
    apply (rule holomorphic_on_inverse)
    apply (metis fh holomorphic_on_subset min.cobounded2 min.commute subset_ball)
    by (metis nz' mem_ball min_less_iff_conj)
    then show  $\exists e > 0. (\lambda z. \text{inverse } (f z))$  holomorphic_on ball z e
    by (metis e e' min_less_iff_conj)
  qed

```

```

lemma analytic_on_divide [analytic_intros]:
  assumes f: f analytic_on S
  and g: g analytic_on S
  and nz: ( $\bigwedge z. z \in S \implies g z \neq 0$ )
  shows ( $\lambda z. f z / g z$ ) analytic_on S
  unfolding divide_inverse
  by (metis analytic_on_inverse analytic_on_mult f g nz)

```

```

lemma analytic_on_power [analytic_intros]:
   $f \text{ analytic\_on } S \implies (\lambda z. (f z) ^ n) \text{ analytic\_on } S$ 
by (induct  $n$ ) (auto simp: analytic_on_mult)

lemma analytic_on_sum [analytic_intros]:
   $(\bigwedge i. i \in I \implies (f i) \text{ analytic\_on } S) \implies (\lambda x. \text{sum } (\lambda i. f i x) I) \text{ analytic\_on } S$ 
by (induct  $I$  rule: infinite_finite_induct) (auto simp: analytic_on_add)

lemma deriv_left_inverse:
  assumes  $f \text{ holomorphic\_on } S$  and  $g \text{ holomorphic\_on } T$ 
    and  $\text{open } S$  and  $\text{open } T$ 
    and  $f ' S \subseteq T$ 
    and [simp]:  $\bigwedge z. z \in S \implies g (f z) = z$ 
    and  $w \in S$ 
  shows  $\text{deriv } f w * \text{deriv } g (f w) = 1$ 
proof -
  have  $\text{deriv } f w * \text{deriv } g (f w) = \text{deriv } g (f w) * \text{deriv } f w$ 
    by (simp add: algebra_simps)
  also have  $\dots = \text{deriv } (g \circ f) w$ 
    using assms
    by (metis analytic_on_imp_differentiable_at analytic_on_open deriv_chain
image_subset_iff)
  also have  $\dots = \text{deriv id } w$ 
  proof (rule complex_derivative_transform_within_open [where s=S])
    show  $g \circ f \text{ holomorphic\_on } S$ 
    by (rule assms holomorphic_on_compose_gen holomorphic_intros) +
  qed (use assms in auto)
  also have  $\dots = 1$ 
    by simp
  finally show ?thesis .
qed

```

6.20.4 Analyticity at a point

```

lemma analytic_at_ball:
   $f \text{ analytic\_on } \{z\} \iff (\exists e. 0 < e \wedge f \text{ holomorphic\_on ball } z e)$ 
by (metis analytic_on_def singleton_iff)

lemma analytic_at:
   $f \text{ analytic\_on } \{z\} \iff (\exists s. \text{open } s \wedge z \in s \wedge f \text{ holomorphic\_on } s)$ 
by (metis analytic_on_holomorphic empty_subsetI insert_subset)

lemma analytic_on_analytic_at:
   $f \text{ analytic\_on } s \iff (\forall z \in s. f \text{ analytic\_on } \{z\})$ 
by (metis analytic_at_ball analytic_on_def)

lemma analytic_at_two:
   $f \text{ analytic\_on } \{z\} \wedge g \text{ analytic\_on } \{z\} \iff$ 

```

($\exists s. \text{open } s \wedge z \in s \wedge f \text{ holomorphic\_on } s \wedge g \text{ holomorphic\_on } s$)
 (is ?lhs = ?rhs)

proof

assume ?lhs

then obtain s t

where st: open s z $\in$ s f holomorphic_on s
 open t z $\in$ t g holomorphic_on t

by (auto simp: analytic_at)

show ?rhs

apply (rule_tac x=s $\cap$ t in exI)

using st

apply (auto simp: holomorphic_on_subset)

done

next

assume ?rhs

then show ?lhs

by (force simp add: analytic_at)

qed

6.20.5 Combining theorems for derivative with “analytic at” hypotheses

lemma

assumes f analytic_on {z} g analytic_on {z}

shows complex_derivative_add_at: deriv ($\lambda w. f w + g w$) z = deriv f z + deriv g z

and complex_derivative_diff_at: deriv ($\lambda w. f w - g w$) z = deriv f z - deriv g z

and complex_derivative_mult_at: deriv ($\lambda w. f w * g w$) z =

 f z * deriv g z + deriv f z * g z

proof –

obtain s **where** s: open s z $\in$ s f holomorphic_on s g holomorphic_on s

using assms **by** (metis analytic_at_two)

show deriv ($\lambda w. f w + g w$) z = deriv f z + deriv g z

apply (rule DERIV_imp_deriv [OF DERIV_add])

using s

apply (auto simp: holomorphic_on_open field_differentiable_def DERIV_deriv_iff_field_differentiable)

done

show deriv ($\lambda w. f w - g w$) z = deriv f z - deriv g z

apply (rule DERIV_imp_deriv [OF DERIV_diff])

using s

apply (auto simp: holomorphic_on_open field_differentiable_def DERIV_deriv_iff_field_differentiable)

done

show deriv ($\lambda w. f w * g w$) z = f z * deriv g z + deriv f z * g z

apply (rule DERIV_imp_deriv [OF DERIV_mult])

using s

apply (auto simp: holomorphic_on_open field_differentiable_def DERIV_deriv_iff_field_differentiable)

done

qed

lemma *deriv_cmult_at*:

*f analytic_on {z} $\implies$ deriv ($\lambda w. c * f w$) z = c * deriv f z*
by (auto simp: complex_derivative_mult_at)

lemma *deriv_cmult_right_at*:

*f analytic_on {z} $\implies$ deriv ($\lambda w. f w * c$) z = deriv f z * c*
by (auto simp: complex_derivative_mult_at)

6.20.6 Complex differentiation of sequences and series

lemma *has_complex_derivative_sequence*:

fixes *S* :: complex set
assumes *cvs*: convex *S*
and *df*: $\bigwedge n x. x \in S \implies (f n \text{ has\_field\_derivative } f' n x) \text{ (at } x \text{ within } S)$
and *conv*: $\bigwedge e. 0 < e \implies \exists N. \forall n x. n \geq N \longrightarrow x \in S \longrightarrow \text{norm } (f' n x - g' x) \leq e$
and $\exists x l. x \in S \wedge ((\lambda n. f n x) \longrightarrow l) \text{ sequentially}$
shows $\exists g. \forall x \in S. ((\lambda n. f n x) \longrightarrow g x) \text{ sequentially} \wedge$
 $(g \text{ has\_field\_derivative } (g' x)) \text{ (at } x \text{ within } S)$

proof –

from *assms* **obtain** *x l* **where** *x*: $x \in S$ **and** *tf*: $((\lambda n. f n x) \longrightarrow l) \text{ sequentially}$
by blast
{ **fix** *e* :: real **assume** *e*: $e > 0$
then obtain *N* **where** *N*: $\forall n \geq N. \forall x. x \in S \longrightarrow \text{cmod } (f' n x - g' x) \leq e$
by (metis *conv*)
have $\exists N. \forall n \geq N. \forall x \in S. \forall h. \text{cmod } (f' n x * h - g' x * h) \leq e * \text{cmod } h$
proof (rule *exI* [of *N*], *clarify*)
fix *n y h*
assume $N \leq n \wedge y \in S$
then have $\text{cmod } (f' n y - g' y) \leq e$
by (metis *N*)
then have $\text{cmod } h * \text{cmod } (f' n y - g' y) \leq \text{cmod } h * e$
by (auto simp: antisym_conv2 mult_le_cancel_left norm_triangle_ineq2)
then show $\text{cmod } (f' n y * h - g' y * h) \leq e * \text{cmod } h$
by (simp add: norm_mult [symmetric] field_simps)
qed
} **note** ** = *this*
show ?thesis
unfolding has_field_derivative_def
proof (rule has_derivative_sequence [OF *cvs* __ *x*])
show $(\lambda n. f n x) \longrightarrow l$
by (rule *tf*)
next show $\bigwedge e. e > 0 \implies \forall_F n \text{ in sequentially. } \forall x \in S. \forall h. \text{cmod } (f' n x * h - g' x * h) \leq e * \text{cmod } h$
unfolding eventually_sequentially **by** (blast intro: **)
qed (metis has_field_derivative_def *df*)
qed

```

lemma has_complex_derivative_series:
  fixes  $S :: \text{complex set}$ 
  assumes cvs: convex S
    and df:  $\bigwedge n x. x \in S \implies (f\ n \text{ has\_field\_derivative } f'\ n\ x) \text{ (at } x \text{ within } S)$ 
    and conv:  $\bigwedge e. 0 < e \implies \exists N. \forall n x. n \geq N \longrightarrow x \in S$ 
       $\longrightarrow \text{cmod } ((\sum i < n. f'\ i\ x) - g'\ x) \leq e$ 
    and  $\exists x\ l. x \in S \wedge ((\lambda n. f\ n\ x) \text{ sums } l)$ 
  shows  $\exists g. \forall x \in S. ((\lambda n. f\ n\ x) \text{ sums } g\ x) \wedge ((g \text{ has\_field\_derivative } g'\ x) \text{ (at } x \text{ within } S))$ 
proof –
  from assms obtain  $x\ l$  where  $x: x \in S$  and sf:  $((\lambda n. f\ n\ x) \text{ sums } l)$ 
  by blast
  { fix  $e :: \text{real}$  assume  $e: e > 0$ 
    then obtain  $N$  where  $N: \forall n x. n \geq N \longrightarrow x \in S$ 
       $\longrightarrow \text{cmod } ((\sum i < n. f'\ i\ x) - g'\ x) \leq e$ 
    by (metis conv)
    have  $\exists N. \forall n \geq N. \forall x \in S. \forall h. \text{cmod } ((\sum i < n. h * f'\ i\ x) - g'\ x * h) \leq e * \text{cmod } h$ 
proof (rule exI [of _ N], clarify)
    fix  $n\ y\ h$ 
    assume  $N \leq n\ y \in S$ 
    then have  $\text{cmod } ((\sum i < n. f'\ i\ y) - g'\ y) \leq e$ 
      by (metis N)
    then have  $\text{cmod } h * \text{cmod } ((\sum i < n. f'\ i\ y) - g'\ y) \leq \text{cmod } h * e$ 
      by (auto simp: antisym_conv2 mult_le_cancel_left norm_triangle_ineq2)
    then show  $\text{cmod } ((\sum i < n. h * f'\ i\ y) - g'\ y * h) \leq e * \text{cmod } h$ 
      by (simp add: norm_mult [symmetric] field_simps sum_distrib_left)
    qed
  } note  $** = \text{this}$ 
show ?thesis
unfolding has_field_derivative_def
proof (rule has_derivative_series [OF cvs _ _ x])
  fix  $n\ x$ 
  assume  $x \in S$ 
  then show  $((f\ n) \text{ has\_derivative } (\lambda z. z * f'\ n\ x)) \text{ (at } x \text{ within } S)$ 
    by (metis df has_field_derivative_def mult_commute_abs)
  next show  $((\lambda n. f\ n\ x) \text{ sums } l)$ 
    by (rule sf)
  next show  $\bigwedge e. e > 0 \implies \forall_F n \text{ in sequentially. } \forall x \in S. \forall h. \text{cmod } ((\sum i < n. h * f'\ i\ x) - g'\ x * h) \leq e * \text{cmod } h$ 
    unfolding eventually_sequentially by (blast intro: **)
  qed
qed

```

6.20.7 Taylor on Complex Numbers

```

lemma sum_Suc_reindex:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \text{ab\_group\_add}$ 
  shows  $\text{sum } f \ \{0..n\} = f\ 0 - f\ (Suc\ n) + \text{sum } (\lambda i. f\ (Suc\ i)) \ \{0..n\}$ 

```


by (induct n) auto

lemma field_Taylor:

assumes S : convex S

and f : $\bigwedge i x. x \in S \implies i \leq n \implies (f \text{ i has\_field\_derivative } f \text{ (Suc } i) \text{ } x) \text{ (at } x \text{ within } S)$

and B : $\bigwedge x. x \in S \implies \text{norm } (f \text{ (Suc } n) \text{ } x) \leq B$

and w : $w \in S$

and z : $z \in S$

shows $\text{norm}(f \text{ } 0 \text{ } z - (\sum_{i \leq n} f \text{ i } w * (z-w)^{\wedge i} / (\text{fact } i)))$
 $\leq B * \text{norm}(z - w)^{\wedge} (\text{Suc } n) / \text{fact } n$

proof -

have wzs : $\text{closed\_segment } w \text{ } z \subseteq S$ using $assms$

by (metis convex_contains_segment)

{ fix u

assume $u \in \text{closed\_segment } w \text{ } z$

then have $u \in S$

by (metis wzs subsetD)

have $(\sum_{i \leq n} f \text{ i } u * (- \text{of\_nat } i * (z-u)^{\wedge}(i-1)) / (\text{fact } i) +$
 $f \text{ (Suc } i) \text{ } u * (z-u)^{\wedge} i / (\text{fact } i)) =$
 $f \text{ (Suc } n) \text{ } u * (z-u)^{\wedge} n / (\text{fact } n)$

proof (induction n)

case 0 show ?case by simp

next

case (Suc n)

have $(\sum_{i \leq \text{Suc } n} f \text{ i } u * (- \text{of\_nat } i * (z-u)^{\wedge}(i-1)) / (\text{fact } i) +$
 $f \text{ (Suc } i) \text{ } u * (z-u)^{\wedge} i / (\text{fact } i)) =$

$f \text{ (Suc } n) \text{ } u * (z-u)^{\wedge} n / (\text{fact } n) +$
 $f \text{ (Suc (Suc } n)) \text{ } u * ((z-u) * (z-u)^{\wedge} n) / (\text{fact (Suc } n)) -$
 $f \text{ (Suc } n) \text{ } u * ((1 + \text{of\_nat } n) * (z-u)^{\wedge} n) / (\text{fact (Suc } n))$

using Suc by simp

also have $\dots = f \text{ (Suc (Suc } n)) \text{ } u * (z-u)^{\wedge} \text{Suc } n / (\text{fact (Suc } n))$

proof -

have $(\text{fact(Suc } n)) *$

$(f \text{ (Suc } n) \text{ } u * (z-u)^{\wedge} n / (\text{fact } n) +$
 $f \text{ (Suc(Suc } n)) \text{ } u * ((z-u) * (z-u)^{\wedge} n) / (\text{fact(Suc } n)) -$
 $f \text{ (Suc } n) \text{ } u * ((1 + \text{of\_nat } n) * (z-u)^{\wedge} n) / (\text{fact(Suc } n))) =$
 $((\text{fact(Suc } n)) * (f \text{ (Suc } n) \text{ } u * (z-u)^{\wedge} n) / (\text{fact } n) +$
 $((\text{fact(Suc } n)) * (f \text{ (Suc(Suc } n)) \text{ } u * ((z-u) * (z-u)^{\wedge} n) / (\text{fact(Suc } n))))$

-

$((\text{fact(Suc } n)) * (f \text{ (Suc } n) \text{ } u * (\text{of\_nat(Suc } n) * (z-u)^{\wedge} n))) / (\text{fact(Suc } n))$

by (simp add: algebra_simps del: fact_Suc)

also have $\dots = ((\text{fact (Suc } n)) * (f \text{ (Suc } n) \text{ } u * (z-u)^{\wedge} n) / (\text{fact } n) +$
 $(f \text{ (Suc (Suc } n)) \text{ } u * ((z-u) * (z-u)^{\wedge} n)) -$
 $(f \text{ (Suc } n) \text{ } u * ((1 + \text{of\_nat } n) * (z-u)^{\wedge} n)))$

by (simp del: fact_Suc)

also have $\dots = (\text{of\_nat (Suc } n) * (f \text{ (Suc } n) \text{ } u * (z-u)^{\wedge} n) +$
 $(f \text{ (Suc (Suc } n)) \text{ } u * ((z-u) * (z-u)^{\wedge} n)) -$

```

      (f (Suc n) u * ((1 + of_nat n) * (z-u) ^ n))
    by (simp only: fact_Suc of_nat_mult ac_simps) simp
  also have ... = f (Suc (Suc n)) u * ((z-u) * (z-u) ^ n)
    by (simp add: algebra_simps)
  finally show ?thesis
    by (simp add: mult_left_cancel [where c = (fact (Suc n)), THEN iffD1]
del: fact_Suc)
  qed
  finally show ?case .
  qed
  then have ((λv. (∑ i≤n. f i v * (z - v) ^ i / (fact i)))
    has_field_derivative f (Suc n) u * (z-u) ^ n / (fact n))
    (at u within S)
  apply (intro derivative_eq_intros)
  apply (blast intro: assms ⟨u ∈ S⟩)
  apply (rule refl)+
  apply (auto simp: field_simps)
  done
} note sum_deriv = this
{ fix u
  assume u: u ∈ closed_segment w z
  then have us: u ∈ S
    by (metis wzs_subsetD)
  have norm (f (Suc n) u) * norm (z - u) ^ n ≤ norm (f (Suc n) u) * norm
(u - z) ^ n
    by (metis norm_minus_commute order_refl)
  also have ... ≤ norm (f (Suc n) u) * norm (z - w) ^ n
    by (metis mult_left_mono norm_ge_zero power_mono segment_bound [OF
u])
  also have ... ≤ B * norm (z - w) ^ n
    by (metis norm_ge_zero zero_le_power mult_right_mono B [OF us])
  finally have norm (f (Suc n) u) * norm (z - u) ^ n ≤ B * norm (z - w) ^
n .
} note cmod_bound = this
have (∑ i≤n. f i z * (z - z) ^ i / (fact i)) = (∑ i≤n. (f i z / (fact i)) * 0 ^ i)
  by simp
also have ... = f 0 z / (fact 0)
  by (subst sum_zero_power) simp
finally have norm (f 0 z - (∑ i≤n. f i w * (z - w) ^ i / (fact i)))
  ≤ norm ((∑ i≤n. f i w * (z - w) ^ i / (fact i)) -
    (∑ i≤n. f i z * (z - z) ^ i / (fact i)))
  by (simp add: norm_minus_commute)
also have ... ≤ B * norm (z - w) ^ n / (fact n) * norm (w - z)
  apply (rule field_differentiable_bound
[where f' = λw. f (Suc n) w * (z - w) ^ n / (fact n)
and S = closed_segment w z, OF convex_closed_segment])
  apply (auto simp: DERIV_subset [OF sum_deriv wzs]
    norm_divide norm_mult norm_power divide_le_cancel cmod_bound)
  done

```

also have ... $\leq B * \text{norm } (z - w) \wedge \text{Suc } n / (\text{fact } n)$
 by (simp add: algebra_simps norm_minus_commute)
 finally show ?thesis .
 qed

lemma complex_Taylor:

assumes S : convex S
 and f : $\bigwedge i. x \in S \implies i \leq n \implies (f \text{ i has\_field\_derivative } f (\text{Suc } i) x) \text{ (at } x \text{ within } S)$
 and B : $\bigwedge x. x \in S \implies \text{cmod } (f (\text{Suc } n) x) \leq B$
 and w : $w \in S$
 and z : $z \in S$
 shows $\text{cmod}(f 0 z - (\sum_{i \leq n}. f i w * (z - w) \wedge i / (\text{fact } i)))$
 $\leq B * \text{cmod}(z - w) \wedge (\text{Suc } n) / \text{fact } n$
 using assms by (rule field_Taylor)

Something more like the traditional MVT for real components

lemma complex_mvt_line:

assumes $\bigwedge u. u \in \text{closed\_segment } w z \implies (f \text{ has\_field\_derivative } f'(u)) \text{ (at } u)$
 shows $\exists u. u \in \text{closed\_segment } w z \wedge \text{Re}(f z) - \text{Re}(f w) = \text{Re}(f'(u) * (z - w))$
proof –
 have twz: $\bigwedge t. (1 - t) *_{\mathbb{R}} w + t *_{\mathbb{R}} z = w + t *_{\mathbb{R}} (z - w)$
 by (simp add: real_vector.scale_left_diff_distrib real_vector.scale_right_diff_distrib)
 note assms[unfolded has_field_derivative_def, derivative_intros]
 show ?thesis
 apply (cut_tac mvt_simple
 $[of 0 1 \text{Re } o f o (\lambda t. (1 - t) *_{\mathbb{R}} w + t *_{\mathbb{R}} z)$
 $\lambda u. \text{Re } o (\lambda h. f'((1 - u) *_{\mathbb{R}} w + u *_{\mathbb{R}} z) * h) o (\lambda t. t *_{\mathbb{R}} (z - w))])$
 apply auto
 apply (rule_tac $x = (1 - x) *_{\mathbb{R}} w + x *_{\mathbb{R}} z$ in exI)
 apply (auto simp: closed_segment_def twz) []
 apply (intro derivative_eq_intros has_derivative_at_withinI, simp_all)
 apply (simp add: fun_eq_iff real_vector.scale_right_diff_distrib)
 apply (force simp: twz closed_segment_def)
 done
 qed

lemma complex_Taylor_mvt:

assumes $\bigwedge i x. \llbracket x \in \text{closed\_segment } w z; i \leq n \rrbracket \implies ((f i) \text{ has\_field\_derivative } f (\text{Suc } i) x) \text{ (at } x)$
 shows $\exists u. u \in \text{closed\_segment } w z \wedge$
 $\text{Re } (f 0 z) =$
 $\text{Re } ((\sum_{i = 0..n}. f i w * (z - w) \wedge i / (\text{fact } i)) +$
 $(f (\text{Suc } n) u * (z - u) \wedge n / (\text{fact } n)) * (z - w))$
proof –
 { fix u
 assume u : $u \in \text{closed\_segment } w z$
 have $(\sum_{i = 0..n}.$

```

      (f (Suc i) u * (z-u) ^ i - of_nat i * (f i u * (z-u) ^ (i - Suc 0))) /
      (fact i)) =
    f (Suc 0) u -
      (f (Suc (Suc n)) u * ((z-u) ^ Suc n - (of_nat (Suc n)) * (z-u) ^ n
    * f (Suc n) u) /
      (fact (Suc n)) +
      (∑ i = 0..n.
        (f (Suc (Suc i)) u * ((z-u) ^ Suc i - of_nat (Suc i) * (f (Suc i)
    u * (z-u) ^ i)) /
        (fact (Suc i)))
      by (subst sum_Suc_reindex) simp
    also have ... = f (Suc 0) u -
      (f (Suc (Suc n)) u * ((z-u) ^ Suc n - (of_nat (Suc n)) * (z-u) ^ n
    * f (Suc n) u) /
      (fact (Suc n)) +
      (∑ i = 0..n.
        f (Suc (Suc i)) u * ((z-u) ^ Suc i) / (fact (Suc i)) -
        f (Suc i) u * (z-u) ^ i / (fact i))
      by (simp only: diff_divide_distrib fact_cancel ac_simps)
    also have ... = f (Suc 0) u -
      (f (Suc (Suc n)) u * (z-u) ^ Suc n - of_nat (Suc n) * (z-u) ^ n * f
    (Suc n) u) /
      (fact (Suc n)) +
      f (Suc (Suc n)) u * (z-u) ^ Suc n / (fact (Suc n)) - f (Suc 0) u
      by (subst sum_Suc_diff) auto
    also have ... = f (Suc n) u * (z-u) ^ n / (fact n)
      by (simp only: algebra_simps diff_divide_distrib fact_cancel)
    finally have (∑ i = 0..n. (f (Suc i) u * (z-u) ^ i
      - of_nat i * (f i u * (z-u) ^ (i - Suc 0))) / (fact i)) =
      f (Suc n) u * (z-u) ^ n / (fact n) .
    then have ((λu. ∑ i = 0..n. f i u * (z-u) ^ i / (fact i)) has_field_derivative
      f (Suc n) u * (z-u) ^ n / (fact n)) (at u)
      apply (intro derivative_eq_intros)+
      apply (force intro: u assms)
      apply (rule refl)+
      apply (auto simp: ac_simps)
      done
  }
  then show ?thesis
    apply (cut_tac complex_mvt_line [of w z λu. ∑ i = 0..n. f i u * (z-u) ^ i /
    (fact i)
      λu. (f (Suc n) u * (z-u) ^ n / (fact n))])
      apply (auto simp add: intro: open_closed_segment)
      done
qed

end

```

6.21 Complex Transcendental Functions

By John Harrison et al. Ported from HOL Light by L C Paulson (2015)

theory *Complex_Transcendental*

imports

Complex_Analysis_Basics Summation_Tests HOL-Library.Periodic_Fun

begin

6.21.1 Möbius transformations

definition *moebius* $a\ b\ c\ d \equiv (\lambda z. (a*z+b) / (c*z+d :: 'a :: field))$

theorem *moebius_inverse*:

assumes $a * d \neq b * c$ $c * z + d \neq 0$

shows $\text{moebius } d\ (-b)\ (-c)\ a\ (\text{moebius } a\ b\ c\ d\ z) = z$

proof $-$

from *assms* **have** $(-c) * \text{moebius } a\ b\ c\ d\ z + a \neq 0$ **unfolding** *moebius_def*
by (*simp add: field_simps*)

with *assms* **show** *?thesis*

unfolding *moebius_def* **by** (*simp add: moebius_def divide_simps*) (*simp add: algebra_simps*)?

qed

lemma *moebius_inverse'*:

assumes $a * d \neq b * c$ $c * z - a \neq 0$

shows $\text{moebius } a\ b\ c\ d\ (\text{moebius } d\ (-b)\ (-c)\ a\ z) = z$

using *assms* *moebius_inverse* [*of d a -b -c z*]

by (*auto simp: algebra_simps*)

lemma *cmod_add_real_less*:

assumes $\text{Im } z \neq 0$ $r \neq 0$

shows $\text{cmod } (z + r) < \text{cmod } z + |r|$

proof (*cases z*)

case (*Complex x y*)

then have $0 < y * y$

using *assms* *mult_neg_neg* **by** *force*

with *assms* **have** $r * x / |r| < \text{sqrt } (x*x + y*y)$

by (*simp add: real_less_sqrt power2_eq_square*)

then show *?thesis* **using** *assms* *Complex*

apply (*simp add: cmod_def*)

apply (*rule power2_less_imp_less, auto*)

apply (*simp add: power2_eq_square field_simps*)

done

qed

lemma *cmod_diff_real_less*: $\text{Im } z \neq 0 \implies x \neq 0 \implies \text{cmod } (z - x) < \text{cmod } z + |x|$

using *cmod_add_real_less* [*of z -x*]

by *simp*

```

lemma cmod_square_less_1_plus:
  assumes  $\text{Im } z = 0 \implies |\text{Re } z| < 1$ 
  shows  $(\text{cmod } z)^2 < 1 + \text{cmod } (1 - z^2)$ 
proof (cases  $\text{Im } z = 0 \vee \text{Re } z = 0$ )
  case True
  with assms abs_square_less_1 show ?thesis
  by (force simp add: Re_power2 Im_power2 cmod_def)
next
  case False
  with cmod_diff_real_less [of  $1 - z^2$  1] show ?thesis
  by (simp add: norm_power Im_power2)
qed

```

6.21.2 The Exponential Function

```

lemma norm_exp_i_times [simp]:  $\text{norm } (\exp(i * \text{of\_real } y)) = 1$ 
  by simp

```

```

lemma norm_exp_imaginary:  $\text{norm}(\exp z) = 1 \implies \text{Re } z = 0$ 
  by simp

```

```

lemma field_differentiable_within_exp:  $\exp$  field_differentiable (at  $z$  within  $s$ )
  using DERIV_exp field_differentiable_at_within field_differentiable_def by
blast

```

```

lemma continuous_within_exp:
  fixes  $z::'a::\{\text{real\_normed\_field}, \text{banach}\}$ 
  shows continuous (at  $z$  within  $s$ )  $\exp$ 
by (simp add: continuous_at_imp_continuous_within)

```

```

lemma holomorphic_on_exp [holomorphic_intros]:  $\exp$  holomorphic_on  $s$ 
  by (simp add: field_differentiable_within_exp holomorphic_on_def)

```

```

lemma holomorphic_on_exp' [holomorphic_intros]:
   $f$  holomorphic_on  $s \implies (\lambda x. \exp(f x))$  holomorphic_on  $s$ 
  using holomorphic_on_compose[OF _ holomorphic_on_exp] by (simp add:
o_def)

```

6.21.3 Euler and de Moivre formulas

The sine series times i

```

lemma sin_i_eq:  $(\lambda n. (i * \text{sin\_coeff } n) * z^n)$  sums  $(i * \sin z)$ 
proof -
  have  $(\lambda n. i * \text{sin\_coeff } n * z^n)$  sums  $(i * \sin z)$ 
  using sin_converges sums_mult by blast
  then show ?thesis
  by (simp add: scaleR_conv_of_real field_simps)
qed

```

```

theorem exp_Euler:  $\exp(i * z) = \cos(z) + i * \sin(z)$ 
proof -
  have ( $\lambda n. (\cos\_coeff\ n + i * \sin\_coeff\ n) * z^n$ ) = ( $\lambda n. (i * z)^n /_R (fact\ n)$ )
  proof
    fix n
    show ( $\cos\_coeff\ n + i * \sin\_coeff\ n$ ) *  $z^n$  =  $(i * z)^n /_R (fact\ n)$ 
    by (auto simp: cos_coeff_def sin_coeff_def scaleR_conv_of_real field_simps
elim!: evenE oddE)
  qed
  also have ... sums ( $\exp(i * z)$ )
    by (rule exp_converges)
  finally have ( $\lambda n. (\cos\_coeff\ n + i * \sin\_coeff\ n) * z^n$ ) sums ( $\exp(i * z)$ ) .
  moreover have ( $\lambda n. (\cos\_coeff\ n + i * \sin\_coeff\ n) * z^n$ ) sums ( $\cos z + i * \sin z$ )
    using sums_add [OF cos_converges [of z] sin_i_eq [of z]]
    by (simp add: field_simps scaleR_conv_of_real)
  ultimately show ?thesis
    using sums_unique2 by blast
qed

corollary exp_minus_Euler:  $\exp(-(i * z)) = \cos(z) - i * \sin(z)$ 
  using exp_Euler [of -z]
  by simp

lemma sin_exp_eq:  $\sin z = (\exp(i * z) - \exp(-(i * z))) / (2*i)$ 
  by (simp add: exp_Euler exp_minus_Euler)

lemma sin_exp_eq':  $\sin z = i * (\exp(-(i * z)) - \exp(i * z)) / 2$ 
  by (simp add: exp_Euler exp_minus_Euler)

lemma cos_exp_eq:  $\cos z = (\exp(i * z) + \exp(-(i * z))) / 2$ 
  by (simp add: exp_Euler exp_minus_Euler)

theorem Euler:  $\exp(z) = of\_real(\exp(Re\ z)) * (of\_real(\cos(Im\ z)) + i * of\_real(\sin(Im\ z)))$ 
by (cases z) (simp add: exp_add exp_Euler cos_of_real exp_of_real sin_of_real Complex_eq)

lemma Re_sin:  $Re(\sin z) = \sin(Re\ z) * (\exp(Im\ z) + \exp(-(Im\ z))) / 2$ 
  by (simp add: sin_exp_eq field_simps Re_divide Im_exp)

lemma Im_sin:  $Im(\sin z) = \cos(Re\ z) * (\exp(Im\ z) - \exp(-(Im\ z))) / 2$ 
  by (simp add: sin_exp_eq field_simps Im_divide Re_exp)

lemma Re_cos:  $Re(\cos z) = \cos(Re\ z) * (\exp(Im\ z) + \exp(-(Im\ z))) / 2$ 
  by (simp add: cos_exp_eq field_simps Re_divide Re_exp)

lemma Im_cos:  $Im(\cos z) = \sin(Re\ z) * (\exp(-(Im\ z)) - \exp(Im\ z)) / 2$ 

```

by (simp add: cos_exp_eq field_simps Im_divide Im_exp)

lemma *Re_sin_pos*: $0 < \operatorname{Re} z \implies \operatorname{Re} z < \pi \implies \operatorname{Re} (\sin z) > 0$
 by (auto simp: Re_sin Im_sin add_pos_pos sin_gt_zero)

lemma *Im_sin_nonneg*: $\operatorname{Re} z = 0 \implies 0 \leq \operatorname{Im} z \implies 0 \leq \operatorname{Im} (\sin z)$
 by (simp add: Re_sin Im_sin algebra_simps)

lemma *Im_sin_nonneg2*: $\operatorname{Re} z = \pi \implies \operatorname{Im} z \leq 0 \implies 0 \leq \operatorname{Im} (\sin z)$
 by (simp add: Re_sin Im_sin algebra_simps)

6.21.4 Relationships between real and complex trigonometric and hyperbolic functions

lemma *real_sin_eq* [simp]: $\operatorname{Re}(\sin(\operatorname{of\_real} x)) = \sin x$
 by (simp add: sin_of_real)

lemma *real_cos_eq* [simp]: $\operatorname{Re}(\cos(\operatorname{of\_real} x)) = \cos x$
 by (simp add: cos_of_real)

lemma *DeMoivre*: $(\cos z + i * \sin z) ^ n = \cos(n * z) + i * \sin(n * z)$
 by (metis exp_Euler [symmetric] exp_of_nat_mult mult.left_commute)

lemma *exp_cnj*: $\operatorname{cnj} (\exp z) = \exp (\operatorname{cnj} z)$

proof –

have $(\lambda n. \operatorname{cnj} (z ^ n /_R (\operatorname{fact} n))) = (\lambda n. (\operatorname{cnj} z) ^ n /_R (\operatorname{fact} n))$

by auto

also have ... sums $(\exp (\operatorname{cnj} z))$

by (rule exp_converges)

finally have $(\lambda n. \operatorname{cnj} (z ^ n /_R (\operatorname{fact} n))) \text{ sums } (\exp (\operatorname{cnj} z))$.

moreover have $(\lambda n. \operatorname{cnj} (z ^ n /_R (\operatorname{fact} n))) \text{ sums } (\operatorname{cnj} (\exp z))$

by (metis exp_converges sums_cnj)

ultimately show ?thesis

using sums_unique2

by blast

qed

lemma *cnj_sin*: $\operatorname{cnj}(\sin z) = \sin(\operatorname{cnj} z)$
 by (simp add: sin_exp_eq exp_cnj field_simps)

lemma *cnj_cos*: $\operatorname{cnj}(\cos z) = \cos(\operatorname{cnj} z)$
 by (simp add: cos_exp_eq exp_cnj field_simps)

lemma *field_differentiable_at_sin*: $\sin$ field_differentiable at z
 using DERIV_sin field_differentiable_def by blast

lemma *field_differentiable_within_sin*: $\sin$ field_differentiable (at z within S)
 by (simp add: field_differentiable_at_sin field_differentiable_at_within)

lemma *field_differentiable_at_cos*: *cos* *field_differentiable* *at* *z*
using *DERIV_cos* *field_differentiable_def* **by** *blast*

lemma *field_differentiable_within_cos*: *cos* *field_differentiable* (*at* *z* *within* *S*)
by (*simp* *add*: *field_differentiable_at_cos* *field_differentiable_at_within*)

lemma *holomorphic_on_sin*: *sin* *holomorphic_on* *S*
by (*simp* *add*: *field_differentiable_within_sin* *holomorphic_on_def*)

lemma *holomorphic_on_cos*: *cos* *holomorphic_on* *S*
by (*simp* *add*: *field_differentiable_within_cos* *holomorphic_on_def*)

lemma *holomorphic_on_sin'* [*holomorphic_intros*]:
assumes *f* *holomorphic_on* *A*
shows $(\lambda x. \sin(f\ x))$ *holomorphic_on* *A*
using *holomorphic_on_compose*[*OF* *assms* *holomorphic_on_sin*] **by** (*simp* *add*:
o_def)

lemma *holomorphic_on_cos'* [*holomorphic_intros*]:
assumes *f* *holomorphic_on* *A*
shows $(\lambda x. \cos(f\ x))$ *holomorphic_on* *A*
using *holomorphic_on_compose*[*OF* *assms* *holomorphic_on_cos*] **by** (*simp* *add*:
o_def)

6.21.5 More on the Polar Representation of Complex Numbers

lemma *exp_Complex*: $\exp(\text{Complex } r\ t) = \text{of\_real}(\exp\ r) * \text{Complex } (\cos\ t)\ (\sin\ t)$
by (*simp* *add*: *Complex_eq* *exp_add* *exp_Euler* *exp_of_real* *sin_of_real* *cos_of_real*)

lemma *exp_eq_1*: $\exp\ z = 1 \longleftrightarrow \text{Re}(z) = 0 \wedge (\exists n::\text{int}. \text{Im}(z) = \text{of\_int } (2 * n) * \pi i)$
(is ?lhs = ?rhs)

proof
assume $\exp\ z = 1$
then have $\text{Re } z = 0$
by (*metis* *exp_eq_one_iff* *norm_exp_eq_Re* *norm_one*)
with $\langle ?lhs \rangle$ **show** *?rhs*
by (*metis* *Re_exp* *complex_Re_of_int* *cos_one_2pi_int* *exp_zero* *mult.commute* *mult_numeral_1* *numeral_One* *of_int_mult* *of_int_numeral*)
next
assume *?rhs* **then show** *?lhs*
using *Im_exp* *Re_exp* *complex_eq_iff*
by (*simp* *add*: *cos_one_2pi_int* *cos_one_sin_zero* *mult.commute*)
qed

lemma *exp_eq*: $\exp\ w = \exp\ z \longleftrightarrow (\exists n::\text{int}. w = z + (\text{of\_int } (2 * n) * \pi i) * i)$
(is ?lhs = ?rhs)

```

proof -
  have  $\exp w = \exp z \longleftrightarrow \exp (w - z) = 1$ 
    by (simp add: exp_diff)
  also have  $\dots \longleftrightarrow (Re\ w = Re\ z \wedge (\exists n::int. Im\ w - Im\ z = of\_int\ (2 * n) * \pi))$ 
    by (simp add: exp_eq_1)
  also have  $\dots \longleftrightarrow ?rhs$ 
    by (auto simp: algebra_simps intro!: complex_eqI)
  finally show ?thesis .
qed

```

```

lemma exp_complex_eqI:  $|Im\ w - Im\ z| < 2 * \pi \implies \exp w = \exp z \implies w = z$ 
  by (auto simp: exp_eq abs_mult)

```

```

lemma exp_integer_2pi:
  assumes  $n \in \mathbb{Z}$ 
  shows  $\exp((2 * n * \pi) * i) = 1$ 
proof -
  have  $\exp((2 * n * \pi) * i) = \exp 0$ 
    using assms unfolding Ints_def exp_eq by auto
  also have  $\dots = 1$ 
    by simp
  finally show ?thesis .
qed

```

```

lemma exp_plus_2pin [simp]:  $\exp (z + i * (of\_int\ n * (of\_real\ \pi * 2))) = \exp z$ 
  by (simp add: exp_eq)

```

```

lemma exp_integer_2pi_plus1:
  assumes  $n \in \mathbb{Z}$ 
  shows  $\exp(((2 * n + 1) * \pi) * i) = - 1$ 
proof -
  from assms obtain n' where [simp]:  $n = of\_int\ n'$ 
    by (auto simp: Ints_def)
  have  $\exp(((2 * n + 1) * \pi) * i) = \exp (\pi * i)$ 
    using assms by (subst exp_eq) (auto intro!: exI[of _ n'] simp: algebra_simps)
  also have  $\dots = - 1$ 
    by simp
  finally show ?thesis .
qed

```

```

lemma inj_on_exp_pi:
  fixes z::complex shows inj_on exp (ball z pi)
proof (clarsimp simp: inj_on_def exp_eq)
  fix y n
  assume  $dist\ z\ (y + 2 * of\_int\ n * of\_real\ \pi * i) < \pi$ 
     $dist\ z\ y < \pi$ 
  then have  $dist\ y\ (y + 2 * of\_int\ n * of\_real\ \pi * i) < \pi + \pi$ 
    using dist_commute_lessI dist_triangle_less_add by blast

```

```

    then have norm (2 * of_int n * of_real pi * i) < 2*pi
      by (simp add: dist_norm)
    then show n = 0
      by (auto simp: norm_mult)
  qed

```

lemma cmod_add_squared:

```

  fixes r1 r2::real
  assumes r1 ≥ 0 r2 ≥ 0
  shows (cmod (r1 * exp (i * ϑ1) + r2 * exp (i * ϑ2)))2 = r12 + r22 + 2 * r1
    * r2 * cos (ϑ1 - ϑ2) (is (cmod (?z1 + ?z2))2 = ?rhs)
proof -
  have (cmod (?z1 + ?z2))2 = (?z1 + ?z2) * cnj (?z1 + ?z2)
    by (rule complex_norm_square)
  also have ... = (?z1 * cnj ?z1 + ?z2 * cnj ?z2) + (?z1 * cnj ?z2 + cnj ?z1 *
    ?z2)
    by (simp add: algebra_simps)
  also have ... = (norm ?z1)2 + (norm ?z2)2 + 2 * Re (?z1 * cnj ?z2)
    unfolding complex_norm_square [symmetric] cnj_add_mult_eq_Re by simp
  also have ... = ?rhs
    by (simp add: norm_mult) (simp add: exp_Euler complex_is_Real_iff [THEN
    iffD1] cos_diff algebra_simps)
  finally show ?thesis
    using of_real_eq_iff by blast
qed

```

lemma cmod_diff_squared:

```

  fixes r1 r2::real
  assumes r1 ≥ 0 r2 ≥ 0
  shows (cmod (r1 * exp (i * ϑ1) - r2 * exp (i * ϑ2)))2 = r12 + r22 -
    2*r1*r2*cos (ϑ1 - ϑ2) (is (cmod (?z1 - ?z2))2 = ?rhs)
proof -
  have exp (i * (ϑ2 + pi)) = - exp (i * ϑ2)
    by (simp add: exp_Euler cos_plus_pi sin_plus_pi)
  then have (cmod (?z1 - ?z2))2 = cmod (?z1 + r2 * exp (i * (ϑ2 + pi)))2
    by simp
  also have ... = r12 + r22 + 2*r1*r2*cos (ϑ1 - (ϑ2 + pi))
    using assms cmod_add_squared by blast
  also have ... = ?rhs
    by (simp add: add_commute diff_add_eq_diff_diff_swap)
  finally show ?thesis .
qed

```

lemma polar_convergence:

```

  fixes R::real
  assumes ⋀j. r j > 0 R > 0
  shows ((λj. r j * exp (i * ϑ j)) ⟶ (R * exp (i * Θ))) ⟷
    (r ⟶ R) ∧ (∃k. ⋀j. ϑ j - of_int (k j) * (2 * pi) ⟶ Θ) (is
    (?z ⟶ ?Z) = ?rhs)

```

```

proof
  assume  $L: ?z \longrightarrow ?Z$ 
  have  $rR: r \longrightarrow R$ 
  using tendsto_norm [OF L] assms by (auto simp: norm_mult abs_of_pos)
  moreover obtain  $k$  where  $(\lambda j. \vartheta j - \text{of\_int } (k j) * (2 * \pi i)) \longrightarrow \Theta$ 
  proof –
    have  $\cos (\vartheta j - \Theta) = ((r j)^2 + R^2 - (\text{norm} (?z j - ?Z))^2) / (2 * R * r j)$  for
     $j$ 
    apply (subst cmod_diff_squared)
    using assms by (auto simp: field_split_simps less_le)
    moreover have  $(\lambda j. ((r j)^2 + R^2 - (\text{norm} (?z j - ?Z))^2) / (2 * R * r j))$ 
     $\longrightarrow ((R^2 + R^2 - (\text{norm} (?Z - ?Z))^2) / (2 * R * R))$ 
    by (intro L rR tendsto_intros) (use  $\langle R > 0 \rangle$  in force)
    moreover have  $((R^2 + R^2 - (\text{norm} (?Z - ?Z))^2) / (2 * R * R)) = 1$ 
    using  $\langle R > 0 \rangle$  by (simp add: power2_eq_square field_split_simps)
    ultimately have  $(\lambda j. \cos (\vartheta j - \Theta)) \longrightarrow 1$ 
    by auto
    then show ?thesis
    using that cos_diff_limit_1 by blast
  qed
  ultimately show ?rhs
  by metis
next
  assume  $R: ?rhs$ 
  show  $?z \longrightarrow ?Z$ 
  proof (rule tendsto_mult)
    show  $(\lambda x. \text{complex\_of\_real } (r x)) \longrightarrow \text{of\_real } R$ 
    using  $R$  by (auto simp: tendsto_of_real_iff)
    obtain  $k$  where  $(\lambda j. \vartheta j - \text{of\_int } (k j) * (2 * \pi i)) \longrightarrow \Theta$ 
    using  $R$  by metis
    then have  $(\lambda j. \text{complex\_of\_real } (\vartheta j - \text{of\_int } (k j) * (2 * \pi i))) \longrightarrow \text{of\_real } \Theta$ 
    using tendsto_of_real_iff by force
    then have  $(\lambda j. \exp (i * \text{of\_real } (\vartheta j - \text{of\_int } (k j) * (2 * \pi i)))) \longrightarrow \exp (i * \Theta)$ 
    using tendsto_mult [OF tendsto_const] isCont_exp isCont_tendsto_compose
by blast
    moreover have  $\exp (i * \text{of\_real } (\vartheta j - \text{of\_int } (k j) * (2 * \pi i))) = \exp (i * \vartheta j)$  for  $j$ 
    unfolding exp_eq
    by (rule_tac  $x = - k j$  in exI) (auto simp: algebra_simps)
    ultimately show  $(\lambda j. \exp (i * \vartheta j)) \longrightarrow \exp (i * \Theta)$ 
    by auto
  qed
qed

lemma sin_cos_eq_iff:  $\sin y = \sin x \wedge \cos y = \cos x \longleftrightarrow (\exists n::\text{int}. y = x + 2 * \pi i * n)$ 
proof –

```

```

{ assume  $\sin y = \sin x \cos y = \cos x$ 
  then have  $\cos (y-x) = 1$ 
    using  $\cos\_add$  [of  $y - x$ ] by simp
  then have  $\exists n::int. y-x = 2 * \pi * n$ 
    using  $\cos\_one\_2\pi\_int$  by auto }
then show ?thesis
apply (auto simp:  $\sin\_add \cos\_add$ )
apply (metis  $add.commute \diff\_add\_cancel$ )
done
qed

lemma  $\exp\_i\_ne\_1$ :
  assumes  $0 < x < 2*\pi$ 
  shows  $\exp(i * of\_real\ x) \neq 1$ 
proof
  assume  $\exp(i * of\_real\ x) = 1$ 
  then have  $\exp(i * of\_real\ x) = \exp\ 0$ 
    by simp
  then obtain  $n$  where  $i * of\_real\ x = (of\_int\ (2 * n) * \pi) * i$ 
    by (simp only:  $Ints\_def \exp\_eq$ ) auto
  then have  $of\_real\ x = (of\_int\ (2 * n) * \pi)$ 
    by (metis  $\text{complex\_i\_not\_zero mult.commute mult\_cancel\_left of\_real\_eq\_iff}$ 
 $real\_scaleR\_def \text{scaleR\_conv\_of\_real}$ )
  then have  $x = (of\_int\ (2 * n) * \pi)$ 
    by simp
  then show False using assms
    by (cases  $n$ ) (auto simp:  $\text{zero\_less\_mult\_iff mult\_less\_0\_iff}$ )
qed

lemma  $\sin\_eq\_0$ :
  fixes  $z::\text{complex}$ 
  shows  $\sin z = 0 \iff (\exists n::int. z = of\_real(n * \pi))$ 
  by (simp add:  $\sin\_exp\_eq \exp\_eq$ )

lemma  $\cos\_eq\_0$ :
  fixes  $z::\text{complex}$ 
  shows  $\cos z = 0 \iff (\exists n::int. z = of\_real(n * \pi) + of\_real\ \pi/2)$ 
  using  $\sin\_eq\_0$  [of  $z - of\_real\ \pi/2$ ]
  by (simp add:  $\sin\_diff \text{algebra\_simps}$ )

lemma  $\cos\_eq\_1$ :
  fixes  $z::\text{complex}$ 
  shows  $\cos z = 1 \iff (\exists n::int. z = of\_real(2 * n * \pi))$ 
proof -
  have  $\cos z = \cos (2*(z/2))$ 
    by simp
  also have  $\dots = 1 - 2 * \sin (z/2) ^ 2$ 
    by (simp only:  $\cos\_double\_sin$ )
  finally have [simp]:  $\cos z = 1 \iff \sin (z/2) = 0$ 

```

```

    by simp
  show ?thesis
    by (auto simp: sin_eq_0)
qed

```

```

lemma csin_eq_1:
  fixes z::complex
  shows  $\sin z = 1 \iff (\exists n::int. z = \text{of\_real}(2 * n * \pi) + \text{of\_real } \pi/2)$ 
  using cos_eq_1 [of z - of_real pi/2]
  by (simp add: cos_diff algebra_simps)

```

```

lemma csin_eq_minus1:
  fixes z::complex
  shows  $\sin z = -1 \iff (\exists n::int. z = \text{of\_real}(2 * n * \pi) + 3/2 * \pi)$ 
    (is _ = ?rhs)
proof -
  have  $\sin z = -1 \iff \sin (-z) = 1$ 
    by (simp add: equation_minus_iff)
  also have  $\dots \iff (\exists n::int. -z = \text{of\_real}(2 * n * \pi) + \text{of\_real } \pi/2)$ 
    by (simp only: csin_eq_1)
  also have  $\dots \iff (\exists n::int. z = -\text{of\_real}(2 * n * \pi) - \text{of\_real } \pi/2)$ 
    by (rule iff_exI) (metis add.inverse_inverse add_uminus_conv_diff minus_add_distrib)
  also have  $\dots = ?rhs$ 
    apply safe
    apply (rule_tac [2]  $x = -(x+1)$  in exI)
    apply (rule_tac  $x = -(x+1)$  in exI)
    apply (simp_all add: algebra_simps)
  done
  finally show ?thesis .
qed

```

```

lemma ccos_eq_minus1:
  fixes z::complex
  shows  $\cos z = -1 \iff (\exists n::int. z = \text{of\_real}(2 * n * \pi) + \pi)$ 
  using csin_eq_1 [of z - of_real pi/2]
  by (simp add: sin_diff algebra_simps equation_minus_iff)

```

```

lemma sin_eq_1:  $\sin x = 1 \iff (\exists n::int. x = (2 * n + 1 / 2) * \pi)$ 
  (is _ = ?rhs)
proof -
  have  $\sin x = 1 \iff \sin (\text{complex\_of\_real } x) = 1$ 
    by (metis of_real_1 one_complex.simps(1) real_sin_eq sin_of_real)
  also have  $\dots \iff (\exists n::int. \text{complex\_of\_real } x = \text{of\_real}(2 * n * \pi) + \text{of\_real } \pi/2)$ 
    by (simp only: csin_eq_1)
  also have  $\dots \iff (\exists n::int. x = \text{of\_real}(2 * n * \pi) + \text{of\_real } \pi/2)$ 
    by (rule iff_exI) (auto simp: algebra_simps intro: injD [OF inj_of_real [where
'a = complex]])
  also have  $\dots = ?rhs$ 

```

```

    by (auto simp: algebra_simps)
    finally show ?thesis .
qed

```

```

lemma sin_eq_minus1:  $\sin x = -1 \longleftrightarrow (\exists n::\text{int}. x = (2*n + 3/2) * \pi)$  (is _
= ?rhs)
proof -
  have  $\sin x = -1 \longleftrightarrow \sin (\text{complex\_of\_real } x) = -1$ 
    by (metis Re_complex_of_real of_real_def scaleR_minus1_left sin_of_real)
  also have  $\dots \longleftrightarrow (\exists n::\text{int}. \text{complex\_of\_real } x = \text{of\_real}(2 * n * \pi) + 3/2 * \pi)$ 
    by (simp only: csin_eq_minus1)
  also have  $\dots \longleftrightarrow (\exists n::\text{int}. x = \text{of\_real}(2 * n * \pi) + 3/2 * \pi)$ 
    by (rule iff_exI) (auto simp: algebra_simps intro: injD [OF inj_of_real [where
'a = complex]])
  also have  $\dots = ?\text{rhs}$ 
    by (auto simp: algebra_simps)
  finally show ?thesis .
qed

```

```

lemma cos_eq_minus1:  $\cos x = -1 \longleftrightarrow (\exists n::\text{int}. x = (2*n + 1) * \pi)$ 
(is _ = ?rhs)
proof -
  have  $\cos x = -1 \longleftrightarrow \cos (\text{complex\_of\_real } x) = -1$ 
    by (metis Re_complex_of_real of_real_def scaleR_minus1_left cos_of_real)
  also have  $\dots \longleftrightarrow (\exists n::\text{int}. \text{complex\_of\_real } x = \text{of\_real}(2 * n * \pi) + \pi)$ 
    by (simp only: ccos_eq_minus1)
  also have  $\dots \longleftrightarrow (\exists n::\text{int}. x = \text{of\_real}(2 * n * \pi) + \pi)$ 
    by (rule iff_exI) (auto simp: algebra_simps intro: injD [OF inj_of_real [where
'a = complex]])
  also have  $\dots = ?\text{rhs}$ 
    by (auto simp: algebra_simps)
  finally show ?thesis .
qed

```

```

lemma dist_exp_i_1:  $\text{norm}(\exp(i * \text{of\_real } t) - 1) = 2 * |\sin(t / 2)|$ 
proof -
  have  $\text{sqrt}(2 - \cos t * 2) = 2 * |\sin(t / 2)|$ 
    using cos_double_sin [of t/2] by (simp add: real_sqrt_mult)
  then show ?thesis
    by (simp add: exp_Euler cmod_def power2_diff sin_of_real cos_of_real alge-
bra_simps)
qed

```

```

lemma sin_cx_2pi [simp]:  $\llbracket z = \text{of\_int } m; \text{ even } m \rrbracket \implies \sin(z * \text{complex\_of\_real } \pi) = 0$ 
  by (simp add: sin_eq_0)

```

```

lemma cos_cx_2pi [simp]:  $\llbracket z = \text{of\_int } m; \text{ even } m \rrbracket \implies \cos(z * \text{complex\_of\_real } \pi) = 1$ 

```

using *cos_eq_1* by *auto*

lemma *complex_sin_eq*:

fixes *w* :: *complex*

shows $\sin w = \sin z \iff (\exists n \in \mathbb{Z}. w = z + \text{of\_real}(2*n*\pi) \vee w = -z + \text{of\_real}((2*n + 1)*\pi))$
 (is ?lhs = ?rhs)

proof

assume ?lhs

then have $\sin w - \sin z = 0$

by (auto simp: algebra_simps)

then have $\sin ((w - z) / 2) * \cos ((w + z) / 2) = 0$

by (auto simp: sin_diff_sin)

then consider $\sin ((w - z) / 2) = 0 \mid \cos ((w + z) / 2) = 0$

using *mult_eq_0_iff* by *blast*

then show ?rhs

proof cases

case 1

then show ?thesis

by (simp add: sin_eq_0 algebra_simps) (metis Ints_of_int of_real_of_int_eq)

next

case 2

then show ?thesis

by (simp add: cos_eq_0 algebra_simps) (metis Ints_of_int of_real_of_int_eq)

qed

next

assume ?rhs

then consider $n::\text{int}$ where $w = z + \text{of\_real}(2 * \text{of\_int } n * \pi)$

$\mid n::\text{int}$ where $w = -z + \text{of\_real}((2 * \text{of\_int } n + 1) * \pi)$

using *Ints_cases* by *blast*

then show ?lhs

proof cases

case 1

then show ?thesis

using *Periodic_Fun.sin.plus_of_int* [of *z n*]

by (auto simp: algebra_simps)

next

case 2

then show ?thesis

using *Periodic_Fun.sin.plus_of_int* [of $-z$ *n*]

apply (simp add: algebra_simps)

by (metis *add.commute* *add.inverse_inverse* *add_diff_cancel_left* *diff_add_cancel* *sin_plus_pi*)

qed

qed

lemma *complex_cos_eq*:

fixes *w* :: *complex*

shows $\cos w = \cos z \iff (\exists n \in \mathbb{Z}. w = z + \text{of\_real}(2*n*\pi) \vee w = -z +$


```

of_real(2*n*pi))
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then have cos w - cos z = 0
    by (auto simp: algebra_simps)
  then have sin ((w + z) / 2) * sin ((z - w) / 2) = 0
    by (auto simp: cos_diff_cos)
  then consider sin ((w + z) / 2) = 0 | sin ((z - w) / 2) = 0
    using mult_eq_0_iff by blast
  then show ?rhs
  proof cases
    case 1
    then obtain n where w + z = of_int n * (complex_of_real pi * 2)
      by (auto simp: sin_eq_0 algebra_simps)
    then have w = -z + of_real(2 * of_int n * pi)
      by (auto simp: algebra_simps)
    then show ?thesis
      using Ints_of_int by blast
  next
    case 2
    then obtain n where z = w + of_int n * (complex_of_real pi * 2)
      by (auto simp: sin_eq_0 algebra_simps)
    then have w = z + complex_of_real (2 * of_int(-n) * pi)
      by (auto simp: algebra_simps)
    then show ?thesis
      using Ints_of_int by blast
  qed
next
  assume ?rhs
  then obtain n::int where w: w = z + of_real (2* of_int n*pi) ∨
    w = -z + of_real(2*n*pi)
    using Ints_cases by (metis of_int_mult of_int_numeral)
  then show ?lhs
    using Periodic_Fun.cos.plus_of_int [of z n]
    apply (simp add: algebra_simps)
    by (metis cos.plus_of_int cos_minus minus_add_cancel mult.commute)
qed

lemma sin_eq:
  sin x = sin y ⟷ (∃ n ∈ ℤ. x = y + 2*n*pi ∨ x = -y + (2*n + 1)*pi)
  using complex_sin_eq [of x y]
  by (simp only: sin_of_real Re_complex_of_real of_real_add [symmetric] of_real_minus
[symmetric] of_real_mult [symmetric] of_real_eq_iff)

lemma cos_eq:
  cos x = cos y ⟷ (∃ n ∈ ℤ. x = y + 2*n*pi ∨ x = -y + 2*n*pi)
  using complex_cos_eq [of x y]
  by (simp only: cos_of_real Re_complex_of_real of_real_add [symmetric] of_real_minus

```

[symmetric] of_real_mult [symmetric] of_real_eq_iff)

lemma *sinh_complex*:
fixes $z :: \text{complex}$
shows $(\exp z - \text{inverse}(\exp z)) / 2 = -i * \sin(i * z)$
by (simp add: sin_exp_eq field_split_simps exp_minus)

lemma *sin_i_times*:
fixes $z :: \text{complex}$
shows $\sin(i * z) = i * ((\exp z - \text{inverse}(\exp z)) / 2)$
using *sinh_complex* **by** auto

lemma *sinh_real*:
fixes $x :: \text{real}$
shows $\text{of\_real}((\exp x - \text{inverse}(\exp x)) / 2) = -i * \sin(i * \text{of\_real } x)$
by (simp add: exp_of_real sin_i_times)

lemma *cosh_complex*:
fixes $z :: \text{complex}$
shows $(\exp z + \text{inverse}(\exp z)) / 2 = \cos(i * z)$
by (simp add: cos_exp_eq field_split_simps exp_minus exp_of_real)

lemma *cosh_real*:
fixes $x :: \text{real}$
shows $\text{of\_real}((\exp x + \text{inverse}(\exp x)) / 2) = \cos(i * \text{of\_real } x)$
by (simp add: cos_exp_eq field_split_simps exp_minus exp_of_real)

lemmas *cos_i_times* = *cosh_complex* [symmetric]

lemma *norm_cos_squared*:
 $\text{norm}(\cos z) ^ 2 = \cos(\text{Re } z) ^ 2 + (\exp(\text{Im } z) - \text{inverse}(\exp(\text{Im } z))) ^ 2 / 4$
proof (cases z)
case (Complex $x1\ x2$)
then show ?thesis
apply (simp only: cos_add cmod_power2 cos_of_real sin_of_real Complex_eq)
apply (simp add: cos_exp_eq sin_exp_eq exp_minus exp_of_real Re_divide
Im_divide power_divide)
apply (simp only: left_diff_distrib [symmetric] power_mult_distrib sin_squared_eq)
apply (simp add: power2_eq_square field_split_simps)
done
qed

lemma *norm_sin_squared*:
 $\text{norm}(\sin z) ^ 2 = (\exp(2 * \text{Im } z) + \text{inverse}(\exp(2 * \text{Im } z)) - 2 * \cos(2 * \text{Re } z)) / 4$
proof (cases z)
case (Complex $x1\ x2$)
then show ?thesis
apply (simp only: sin_add cmod_power2 cos_of_real sin_of_real cos_double_cos)

```

exp_double Complex_eq)
  apply (simp add: cos_exp_eq sin_exp_eq exp_minus exp_of_real Re_divide
Im_divide power_divide)
  apply (simp only: left_diff_distrib [symmetric] power_mult_distrib cos_squared_eq)
  apply (simp add: power2_eq_square field_split_simps)
  done
qed

lemma exp_uminus_Im: exp (- Im z) ≤ exp (cmod z)
  using abs_Im_le_cmod linear_order_trans by fastforce

lemma norm_cos_le:
  fixes z::complex
  shows norm(cos z) ≤ exp(norm z)
proof -
  have Im z ≤ cmod z
    using abs_Im_le_cmod abs_le_D1 by auto
  then have exp (- Im z) + exp (Im z) ≤ exp (cmod z) * 2
    by (metis exp_uminus_Im add_mono exp_le_cancel_iff mult_2_right)
  then show ?thesis
    by (force simp add: cos_exp_eq norm_divide intro: order_trans [OF norm_triangle_ineq])
qed

lemma norm_cos_plus1_le:
  fixes z::complex
  shows norm(1 + cos z) ≤ 2 * exp(norm z)
proof -
  have mono:  $\bigwedge u w z::\text{real}. (1 \leq w \mid 1 \leq z) \implies (w \leq u \ \& \ z \leq u) \implies 2 + w + z \leq 4 * u$ 
    by arith
  have *: Im z ≤ cmod z
    using abs_Im_le_cmod abs_le_D1 by auto
  have triangle3:  $\bigwedge x y z. \text{norm}(x + y + z) \leq \text{norm}(x) + \text{norm}(y) + \text{norm}(z)$ 
    by (simp add: norm_add_rule_thm)
  have norm(1 + cos z) = cmod (1 + (exp (i * z) + exp (- (i * z))) / 2)
    by (simp add: cos_exp_eq)
  also have ... = cmod ((2 + exp (i * z) + exp (- (i * z))) / 2)
    by (simp add: field_simps)
  also have ... = cmod (2 + exp (i * z) + exp (- (i * z))) / 2
    by (simp add: norm_divide)
  finally show ?thesis
    by (metis exp_eq_one_iff exp_le_cancel_iff mult_2 norm_cos_le norm_ge_zero
norm_one norm_triangle_mono)
qed

lemma sinh_conv_sin: sinh z = -i * sin (i*z)
  by (simp add: sinh_field_def sin_i_times exp_minus)

lemma cosh_conv_cos: cosh z = cos (i*z)

```

2320

by (simp add: cosh_field_def cos_i_times_exp_minus)

lemma tanh_conv_tan: $\tanh z = -i * \tan (i*z)$
 by (simp add: tanh_def sinh_conv_sin cosh_conv_cos tan_def)

lemma sin_conv_sinh: $\sin z = -i * \sinh (i*z)$
 by (simp add: sinh_conv_sin)

lemma cos_conv_cosh: $\cos z = \cosh (i*z)$
 by (simp add: cosh_conv_cos)

lemma tan_conv_tanh: $\tan z = -i * \tanh (i*z)$
 by (simp add: tan_def sin_conv_sinh cos_conv_cosh tanh_def)

lemma sinh_complex_eq_iff:
 $\sinh (z :: \text{complex}) = \sinh w \longleftrightarrow$
 $(\exists n \in \mathbb{Z}. z = w - 2 * i * \text{of\_real } n * \text{of\_real } \pi \vee$
 $z = -(2 * \text{complex\_of\_real } n + 1) * i * \text{complex\_of\_real } \pi - w) \text{ (is$
 $\_ = ?rhs)$
proof –
 have $\sinh z = \sinh w \longleftrightarrow \sin (i * z) = \sin (i * w)$
 by (simp add: sinh_conv_sin)
 also have $\dots \longleftrightarrow ?rhs$
 by (subst complex_sin_eq) (force simp: field_simps complex_eq_iff)
 finally show ?thesis .
qed

6.21.6 Taylor series for complex exponential, sine and cosine

declare power_Suc [simp del]

lemma Taylor_exp_field:
 fixes $z :: 'a :: \{\text{banach}, \text{real\_normed\_field}\}$
 shows $\text{norm} (\exp z - (\sum i \leq n. z^i / \text{fact } i)) \leq \exp (\text{norm } z) * (\text{norm } z \wedge \text{Suc } n) / \text{fact } n$
proof (rule field_Taylor[of _ n $\lambda k. \exp \exp (\text{norm } z) 0 z$, simplified])
 show convex (closed_segment 0 z)
 by (rule convex_closed_segment [of 0 z])
next
 fix $k \ x$
 assume $x \in \text{closed\_segment } 0 \ z \ k \leq n$
 show $(\exp \text{ has\_field\_derivative } \exp \ x) \text{ (at } x \text{ within closed\_segment } 0 \ z)$
 using DERIV_exp DERIV_subset by blast
next
 fix x
 assume $x: x \in \text{closed\_segment } 0 \ z$
 have $\text{norm} (\exp x) \leq \exp (\text{norm } x)$
 by (rule norm_exp)
 also have $\text{norm } x \leq \text{norm } z$

```

    using x by (auto simp: closed_segment_def intro!: mult_left_le_one_le)
  finally show norm (exp x) ≤ exp (norm z)
    by simp
qed auto

lemma Taylor_exp:
  norm(exp z - (∑ k≤n. z ^ k / (fact k))) ≤ exp|Re z| * (norm z) ^ (Suc n) /
  (fact n)
proof (rule complex_Taylor [of _ n λk. exp exp|Re z| 0 z, simplified])
  show convex (closed_segment 0 z)
    by (rule convex_closed_segment [of 0 z])
next
fix k x
assume x ∈ closed_segment 0 z k ≤ n
show (exp has_field_derivative exp x) (at x within closed_segment 0 z)
  using DERIV_exp DERIV_subset by blast
next
fix x
assume x ∈ closed_segment 0 z
then obtain u where u: x = complex_of_real u * z 0 ≤ u u ≤ 1
  by (auto simp: closed_segment_def scaleR_conv_of_real)
then have u * Re z ≤ |Re z|
  by (metis abs_ge_self abs_ge_zero mult.commute mult.right_neutral mult_mono)
then show Re x ≤ |Re z|
  by (simp add: u)
qed auto

lemma
  assumes 0 ≤ u u ≤ 1
  shows cmod_sin_le_exp: cmod (sin (u *R z)) ≤ exp |Im z|
    and cmod_cos_le_exp: cmod (cos (u *R z)) ≤ exp |Im z|
proof -
  have mono: ∧u w z::real. w ≤ u ⇒ z ≤ u ⇒ (w + z)/2 ≤ u
    by simp
  have *: (cmod (exp (i * (u * z))) + cmod (exp (- (i * (u * z)))) ) / 2 ≤ exp
  |Im z|
  proof (rule mono)
    show cmod (exp (i * (u * z))) ≤ exp |Im z|
      using assms
      by (auto simp: abs_if mult_left_le_one_le not_less intro: order_trans [of _
0])
    show cmod (exp (- (i * (u * z)))) ≤ exp |Im z|
      using assms
      by (auto simp: abs_if mult_left_le_one_le mult_nonneg_nonpos intro: or-
der_trans [of _ 0])
  qed
  have cmod (sin (u *R z)) = cmod (exp (i * (u * z)) - exp (- (i * (u * z)))) / 2
    by (auto simp: scaleR_conv_of_real norm_mult norm_power sin_exp_eq
norm_divide)

```

also have ... $\leq (cmod (exp (i * (u * z))) + cmod (exp (- (i * (u * z))))) / 2$
by (intro divide_right_mono norm_triangle_ineq4) simp
also have ... $\leq exp |Im z|$
by (rule *)
finally show $cmod (sin (u *_R z)) \leq exp |Im z|$.
have $cmod (cos (u *_R z)) = cmod (exp (i * (u * z)) + exp (- (i * (u * z)))) / 2$
by (auto simp: scaleR_conv_of_real norm_mult norm_power cos_exp_eq norm_divide)
also have ... $\leq (cmod (exp (i * (u * z))) + cmod (exp (- (i * (u * z))))) / 2$
by (intro divide_right_mono norm_triangle_ineq) simp
also have ... $\leq exp |Im z|$
by (rule *)
finally show $cmod (cos (u *_R z)) \leq exp |Im z|$.
qed

lemma Taylor_sin:

$norm(sin z - (\sum k \leq n. complex\_of\_real (sin\_coeff k) * z ^ k))$
 $\leq exp |Im z| * (norm z) ^ (Suc n) / (fact n)$

proof -

have mono: $\bigwedge u w z :: real. w \leq u \implies z \leq u \implies w + z \leq u * 2$

by arith

have *: $cmod (sin z - (\sum i \leq n. (-1) ^ (i div 2) * (if even i then sin 0 else cos 0) * z ^ i / (fact i)))$
 $\leq exp |Im z| * cmod z ^ Suc n / (fact n)$

proof (rule complex_Taylor [of closed_segment 0 z n

$\lambda k x. (-1) ^ (k div 2) * (if even k then sin x else cos x)$
 $exp |Im z| 0 z, simplified])$

fix k x

show $((\lambda x. (-1) ^ (k div 2) * (if even k then sin x else cos x)) has\_field\_derivative$
 $(-1) ^ (Suc k div 2) * (if odd k then sin x else cos x))$
 $(at x within closed\_segment 0 z)$

apply (auto simp: power_Suc)

apply (intro derivative_eq_intros | simp)+

done

next

fix x

assume $x \in closed\_segment 0 z$

then show $cmod ((-1) ^ (Suc n div 2) * (if odd n then sin x else cos x)) \leq$
 $exp |Im z|$

by (auto simp: closed_segment_def norm_mult norm_power cmod_sin_le_exp cmod_cos_le_exp)

qed

have **: $\bigwedge k. complex\_of\_real (sin\_coeff k) * z ^ k$

$= (-1) ^ (k div 2) * (if even k then sin 0 else cos 0) * z ^ k / of\_nat (fact k)$

by (auto simp: sin_coeff_def elim!: oddE)

show ?thesis

by (simp add: ** order_trans [OF _ *])

qed

lemma *Taylor_cos*:

$\text{norm}(\cos z - (\sum_{k \leq n. \text{complex\_of\_real } (\cos\_coeff\ k) * z^k}))$
 $\leq \exp|Im\ z| * (\text{norm } z)^\wedge \text{Suc } n / (\text{fact } n)$

proof –

have *mono*: $\bigwedge u\ w\ z::\text{real}. w \leq u \implies z \leq u \implies w + z \leq u * 2$

by *arith*

have *: $\text{cmod } (\cos z -$

$(\sum_{i \leq n. (-1)^\wedge (\text{Suc } i \text{ div } 2) * (\text{if even } i \text{ then } \cos 0 \text{ else } \sin 0) * z^\wedge i / (\text{fact } i)))$
 $\leq \exp |Im\ z| * \text{cmod } z^\wedge \text{Suc } n / (\text{fact } n)$

proof (*rule complex_Taylor [of closed_segment 0 z n $\lambda k\ x. (-1)^\wedge (\text{Suc } k \text{ div } 2) * (\text{if even } k \text{ then } \cos x \text{ else } \sin x) \exp|Im\ z| 0 z,$*
simplified])

fix *k x*

assume $x \in \text{closed\_segment } 0\ z\ k \leq n$

show $((\lambda x. (-1)^\wedge (\text{Suc } k \text{ div } 2) * (\text{if even } k \text{ then } \cos x \text{ else } \sin x))$

has_field_derivative

$(-1)^\wedge \text{Suc } (k \text{ div } 2) * (\text{if odd } k \text{ then } \cos x \text{ else } \sin x))$
 $(\text{at } x \text{ within closed\_segment } 0\ z)$

apply (*auto simp: power_Suc*)

apply (*intro derivative_eq_intros | simp*) +

done

next

fix *x*

assume $x \in \text{closed\_segment } 0\ z$

then show $\text{cmod } ((-1)^\wedge \text{Suc } (n \text{ div } 2) * (\text{if odd } n \text{ then } \cos x \text{ else } \sin x)) \leq$

$\exp |Im\ z|$

by (*auto simp: closed_segment_def norm_mult norm_power cmod_sin_le_exp cmod_cos_le_exp*)

qed

have **: $\bigwedge k. \text{complex\_of\_real } (\cos\_coeff\ k) * z^\wedge k$

$= (-1)^\wedge (\text{Suc } k \text{ div } 2) * (\text{if even } k \text{ then } \cos 0 \text{ else } \sin 0) * z^\wedge k / \text{of\_nat}$
 $(\text{fact } k)$

by (*auto simp: cos_coeff_def elim!: evenE*)

show *?thesis*

by (*simp add: ** order_trans [OF _ *]*)

qed

declare *power_Suc* [*simp*]

32-bit Approximation to e

lemma *e_approx_32*: $|\exp(1) - 5837465777 / 2147483648| \leq (\text{inverse}(2^\wedge 32)::\text{real})$

using *Taylor_exp [of 1 14] exp_le*

apply (*simp add: sum_distrib_right in_Reals_norm Re_exp atMost_nat_numeral fact_numeral*)

apply (*simp only: pos_le_divide_eq [symmetric]*)

done

```

lemma e_less_272: exp 1 < (272/100::real)
  using e_approx_32
  by (simp add: abs_if_split: if_split_asm)

```

```

lemma ln_272_gt_1: ln (272/100) > (1::real)
  by (metis e_less_272 exp_less_cancel_iff exp_ln_iff less_trans ln_exp)

```

Apparently redundant. But many arguments involve integers.

```

lemma ln3_gt_1: ln 3 > (1::real)
  by (simp add: less_trans [OF ln_272_gt_1])

```

6.21.7 The argument of a complex number (HOL Light version)

```

definition is_Arg :: [complex, real]  $\Rightarrow$  bool
  where is_Arg z r  $\equiv z = \text{of\_real}(\text{norm } z) * \text{exp}(i * \text{of\_real } r)$ 

```

```

definition Arg2pi :: complex  $\Rightarrow$  real
  where Arg2pi z  $\equiv$  if z = 0 then 0 else THE t. 0  $\leq$  t  $\wedge$  t < 2*pi  $\wedge$  is_Arg z t

```

```

lemma is_Arg_2pi_iff: is_Arg z (r + of_int k * (2 * pi))  $\longleftrightarrow$  is_Arg z r
  by (simp add: algebra_simps is_Arg_def)

```

```

lemma is_Arg_eqI:
  assumes r: is_Arg z r and s: is_Arg z s and rs: abs (r-s) < 2*pi and z  $\neq$  0
  shows r = s

```

proof –

```

  have zr: z = (cmod z) * exp (i * r) and zs: z = (cmod z) * exp (i * s)
  using r s by (auto simp: is_Arg_def)
  with  $\langle z \neq 0 \rangle$  have eq: exp (i * r) = exp (i * s)
  by (metis mult_eq_0_iff mult_left_cancel)
  have i * r = i * s
  by (rule exp_complex_eqI) (use rs in <auto simp: eq exp_complex_eqI>)
  then show ?thesis
  by simp

```

qed

This function returns the angle of a complex number from its representation in polar coordinates. Due to periodicity, its range is arbitrary. *Arg2pi* follows HOL Light in adopting the interval $[0, 2\pi)$. But we have the same periodicity issue with logarithms, and it is usual to adopt the same interval for the complex logarithm and argument functions. Further on down, we shall define both functions for the interval $(-\pi, \pi]$. The present version is provided for compatibility.

```

lemma Arg2pi_0 [simp]: Arg2pi(0) = 0
  by (simp add: Arg2pi_def)

```


lemma *Arg2pi_unique_lemma*:

```

  assumes z: is_Arg z t
    and z': is_Arg z t'
    and t: 0 ≤ t & t < 2*pi
    and t': 0 ≤ t' & t' < 2*pi
    and nz: z ≠ 0
  shows t' = t
proof -
  have [dest]: ∧x y z::real. x ≥ 0 ⇒ x + y < z ⇒ y < z
    by arith
  have of_real (cmod z) * exp (i * of_real t') = of_real (cmod z) * exp (i * of_real
t)
    by (metis z z' is_Arg_def)
  then have exp (i * of_real t') = exp (i * of_real t)
    by (metis nz mult_left_cancel mult_zero_left z is_Arg_def)
  then have sin t' = sin t ∧ cos t' = cos t
    by (metis cis.simps cis_conv_exp)
  then obtain n::int where n: t' = t + 2 * n * pi
    by (auto simp: sin_cos_eq_iff)
  then have n=0
    by (cases n) (use t t' in ⟨auto simp: mult_less_0_iff algebra_simps⟩)
  then show t' = t
    by (simp add: n)
qed

```

lemma *Arg2pi*: $0 \leq \text{Arg}2\pi z \wedge \text{Arg}2\pi z < 2\pi \wedge \text{is\_Arg } z (\text{Arg}2\pi z)$

```

proof (cases z=0)
  case True then show ?thesis
    by (simp add: Arg2pi_def is_Arg_def)
next
  case False
  obtain t where t: 0 ≤ t & t < 2*pi
    and ReIm: Re z / cmod z = cos t & Im z / cmod z = sin t
    using sincos_total_2pi [OF complex_unit_circle [OF False]]
    by blast
  have z: is_Arg z t
    unfolding is_Arg_def
    using t False ReIm
    by (intro complex_eqI) (auto simp: exp_Euler sin_of_real cos_of_real field_split_simps)
  show ?thesis
    apply (simp add: Arg2pi_def False)
    apply (rule theI [where a=t])
    using t z False
    apply (auto intro: Arg2pi_unique_lemma)
    done
qed

```

corollary

shows *Arg2pi_ge_0*: $0 \leq \text{Arg}2\pi z$

```

    and Arg2pi_lt_2pi: Arg2pi z < 2*pi
    and Arg2pi_eq: z = of_real(norm z) * exp(i * of_real(Arg2pi z))
    using Arg2pi_is_Arg_def by auto

```

```

lemma complex_norm_eq_1_exp: norm z = 1  $\longleftrightarrow$  exp(i * of_real (Arg2pi z))
= z
by (metis Arg2pi_eq cis_conv_exp mult.left_neutral norm_cis of_real_1)

```

```

lemma Arg2pi_unique:  $\llbracket$ of_real r * exp(i * of_real a) = z; 0 < r; 0  $\leq$  a; a <
2*pi $\rrbracket \implies$  Arg2pi z = a
by (rule Arg2pi_unique_lemma [unfolded is_Arg_def, OF Arg2pi_eq]) (use
Arg2pi [of z] in <auto simp: norm_mult>)

```

```

lemma cos_Arg2pi: cmod z * cos (Arg2pi z) = Re z and sin_Arg2pi: cmod z *
sin (Arg2pi z) = Im z
using Arg2pi_eq [of z] cis_conv_exp Re_rcis Im_rcis unfolding rcis_def by
metis+

```

```

lemma Arg2pi_minus:
  assumes z  $\neq$  0 shows Arg2pi (-z) = (if Arg2pi z < pi then Arg2pi z + pi else
Arg2pi z - pi)
  apply (rule Arg2pi_unique [of norm z, OF complex_eqI])
  using cos_Arg2pi sin_Arg2pi Arg2pi_ge_0 Arg2pi_lt_2pi [of z] assms
  by (auto simp: Re_exp Im_exp)

```

```

lemma Arg2pi_times_of_real [simp]:
  assumes 0 < r shows Arg2pi (of_real r * z) = Arg2pi z
proof (cases z=0)
  case False
  show ?thesis
    by (rule Arg2pi_unique [of r * norm z]) (use Arg2pi False assms is_Arg_def
in auto)
qed auto

```

```

lemma Arg2pi_times_of_real2 [simp]: 0 < r  $\implies$  Arg2pi (z * of_real r) = Arg2pi
z
by (metis Arg2pi_times_of_real mult.commute)

```

```

lemma Arg2pi_divide_of_real [simp]: 0 < r  $\implies$  Arg2pi (z / of_real r) = Arg2pi
z
by (metis Arg2pi_times_of_real2 less_numeral_extra(3) nonzero_eq_divide_eq
of_real_eq_0_iff)

```

```

lemma Arg2pi_le_pi: Arg2pi z  $\leq$  pi  $\longleftrightarrow$  0  $\leq$  Im z
proof (cases z=0)
  case False
  have 0  $\leq$  Im z  $\longleftrightarrow$  0  $\leq$  Im (of_real (cmod z) * exp (i * complex_of_real (Arg2pi
z)))
  by (metis Arg2pi_eq)

```

```

also have ... = (0 ≤ Im (exp (i * complex_of_real (Arg2pi z))))
  using False by (simp add: zero_le_mult_iff)
also have ... ⟷ Arg2pi z ≤ pi
  by (simp add: Im_exp) (metis Arg2pi_ge_0 Arg2pi_lt_2pi sin_lt_zero sin_ge_zero
not_le)
finally show ?thesis
  by blast
qed auto

```

```

lemma Arg2pi_lt_pi: 0 < Arg2pi z ∧ Arg2pi z < pi ⟷ 0 < Im z
proof (cases z=0)
case False
have 0 < Im z ⟷ 0 < Im (of_real (cmod z) * exp (i * complex_of_real (Arg2pi
z)))
  by (metis Arg2pi_eq)
also have ... = (0 < Im (exp (i * complex_of_real (Arg2pi z))))
  using False by (simp add: zero_less_mult_iff)
also have ... ⟷ 0 < Arg2pi z ∧ Arg2pi z < pi (is _ = ?rhs)
proof -
have 0 < sin (Arg2pi z) ⟹ ?rhs
  by (meson Arg2pi_ge_0 Arg2pi_lt_2pi less_le_trans not_le sin_le_zero
sin_x_le_x)
then show ?thesis
  by (auto simp: Im_exp sin_gt_zero)
qed
finally show ?thesis
  by blast
qed auto

```

```

lemma Arg2pi_eq_0: Arg2pi z = 0 ⟷ z ∈ ℝ ∧ 0 ≤ Re z
proof (cases z=0)
case False
have z ∈ ℝ ∧ 0 ≤ Re z ⟷ z ∈ ℝ ∧ 0 ≤ Re (of_real (cmod z) * exp (i *
complex_of_real (Arg2pi z)))
  by (metis Arg2pi_eq)
also have ... ⟷ z ∈ ℝ ∧ 0 ≤ Re (exp (i * complex_of_real (Arg2pi z)))
  using False by (simp add: zero_le_mult_iff)
also have ... ⟷ Arg2pi z = 0
proof -
have [simp]: Arg2pi z = 0 ⟹ z ∈ ℝ
  using Arg2pi_eq [of z] by (auto simp: Reals_def)
moreover have [z ∈ ℝ; 0 ≤ cos (Arg2pi z)] ⟹ Arg2pi z = 0
  by (metis Arg2pi_lt_pi Arg2pi_ge_0 Arg2pi_le_pi cos_pi complex_is_Real_iff
leD less_linear less_minus_one_simps(2) minus_minus neg_less_eq_nonneg or-
der_refl)
ultimately show ?thesis
  by (auto simp: Re_exp)
qed
finally show ?thesis

```

by blast
qed auto

corollary *Arg2pi_gt_0*:
assumes $z \notin \mathbb{R}_{\geq 0}$
shows $\text{Arg2pi } z > 0$
using *Arg2pi_eq_0* *Arg2pi_ge_0* *assms dual_order.strict_iff_order*
unfolding *nonneg_Reals_def* by fastforce

lemma *Arg2pi_eq_pi*: $\text{Arg2pi } z = \pi \longleftrightarrow z \in \mathbb{R} \wedge \text{Re } z < 0$
using *Arg2pi_le_pi* [of *z*] *Arg2pi_lt_pi* [of *z*] *Arg2pi_eq_0* [of *z*] *Arg2pi_ge_0*
[of *z*]
by (fastforce simp: *complex_is_Real_iff*)

lemma *Arg2pi_eq_0_pi*: $\text{Arg2pi } z = 0 \vee \text{Arg2pi } z = \pi \longleftrightarrow z \in \mathbb{R}$
using *Arg2pi_eq_0* *Arg2pi_eq_pi* *not_le* by auto

lemma *Arg2pi_of_real*: $\text{Arg2pi } (\text{of\_real } r) = (\text{if } r < 0 \text{ then } \pi \text{ else } 0)$
using *Arg2pi_eq_0_pi* *Arg2pi_eq_pi* by fastforce

lemma *Arg2pi_real*: $z \in \mathbb{R} \implies \text{Arg2pi } z = (\text{if } 0 \leq \text{Re } z \text{ then } 0 \text{ else } \pi)$
using *Arg2pi_eq_0* *Arg2pi_eq_0_pi* by auto

lemma *Arg2pi_inverse*: $\text{Arg2pi}(\text{inverse } z) = (\text{if } z \in \mathbb{R} \text{ then } \text{Arg2pi } z \text{ else } 2\pi - \text{Arg2pi } z)$
proof (cases $z=0$)
case False
show ?thesis
apply (rule *Arg2pi_unique* [of *inverse (norm z)*])
using *Arg2pi_eq* False *Arg2pi_ge_0* [of *z*] *Arg2pi_lt_2pi* [of *z*] *Arg2pi_eq_0*
[of *z*]
by (auto simp: *Arg2pi_real in_Reals_norm exp_diff field_simps*)
qed auto

lemma *Arg2pi_eq_iff*:
assumes $w \neq 0$ $z \neq 0$
shows $\text{Arg2pi } w = \text{Arg2pi } z \longleftrightarrow (\exists x. 0 < x \ \& \ w = \text{of\_real } x * z)$
using *assms* *Arg2pi_eq* [of *z*] *Arg2pi_eq* [of *w*]
apply auto
apply (rule_tac $x = \text{norm } w / \text{norm } z$ in *exI*)
apply (simp add: *field_split_simps*)
by (metis *mult.commute mult.left_commute*)

lemma *Arg2pi_inverse_eq_0*: $\text{Arg2pi}(\text{inverse } z) = 0 \longleftrightarrow \text{Arg2pi } z = 0$
by (metis *Arg2pi_eq_0* *Arg2pi_inverse inverse_inverse_eq*)

lemma *Arg2pi_divide*:
assumes $w \neq 0$ $z \neq 0$ $\text{Arg2pi } w \leq \text{Arg2pi } z$
shows $\text{Arg2pi}(z / w) = \text{Arg2pi } z - \text{Arg2pi } w$

```

apply (rule Arg2pi_unique [of norm(z / w)])
using assms Arg2pi_eq Arg2pi_ge_0 [of w] Arg2pi_lt_2pi [of z]
apply (auto simp: exp_diff norm_divide field_simps)
done

```

```

lemma Arg2pi_le_div_sum:
  assumes  $w \neq 0$   $z \neq 0$   $\text{Arg2pi } w \leq \text{Arg2pi } z$ 
  shows  $\text{Arg2pi } z = \text{Arg2pi } w + \text{Arg2pi}(z / w)$ 
by (simp add: Arg2pi_divide assms)

```

```

lemma Arg2pi_le_div_sum_eq:
  assumes  $w \neq 0$   $z \neq 0$ 
  shows  $\text{Arg2pi } w \leq \text{Arg2pi } z \longleftrightarrow \text{Arg2pi } z = \text{Arg2pi } w + \text{Arg2pi}(z / w)$ 
using assms by (auto simp: Arg2pi_ge_0 intro: Arg2pi_le_div_sum)

```

```

lemma Arg2pi_diff:
  assumes  $w \neq 0$   $z \neq 0$ 
  shows  $\text{Arg2pi } w - \text{Arg2pi } z = (\text{if } \text{Arg2pi } z \leq \text{Arg2pi } w \text{ then } \text{Arg2pi}(w / z) \text{ else } \text{Arg2pi}(w/z) - 2*\pi)$ 
using assms Arg2pi_divide Arg2pi_inverse [of w/z] Arg2pi_eq_0_pi
by (force simp add: Arg2pi_ge_0 Arg2pi_divide not_le split: if_split_asm)

```

```

lemma Arg2pi_add:
  assumes  $w \neq 0$   $z \neq 0$ 
  shows  $\text{Arg2pi } w + \text{Arg2pi } z = (\text{if } \text{Arg2pi } w + \text{Arg2pi } z < 2*\pi \text{ then } \text{Arg2pi}(w * z) \text{ else } \text{Arg2pi}(w * z) + 2*\pi)$ 
using assms Arg2pi_diff [of w*z z] Arg2pi_le_div_sum_eq [of z w*z]
apply (auto simp: Arg2pi_ge_0 Arg2pi_divide not_le)
apply (metis Arg2pi_lt_2pi add.commute)
apply (metis (no_types) Arg2pi add.commute diff_0 diff_add_cancel diff_less_eq diff_minus_eq_add not_less)
done

```

```

lemma Arg2pi_times:
  assumes  $w \neq 0$   $z \neq 0$ 
  shows  $\text{Arg2pi } (w * z) = (\text{if } \text{Arg2pi } w + \text{Arg2pi } z < 2*\pi \text{ then } \text{Arg2pi } w + \text{Arg2pi } z$ 
     $\text{else } (\text{Arg2pi } w + \text{Arg2pi } z) - 2*\pi)$ 
using Arg2pi_add [OF assms]
by auto

```

```

lemma Arg2pi_cnj_eq_inverse:  $z \neq 0 \implies \text{Arg2pi } (\text{cnj } z) = \text{Arg2pi } (\text{inverse } z)$ 
apply (simp add: Arg2pi_eq_iff field_split_simps complex_norm_square [symmetric])
by (metis of_real_power zero_less_norm_iff zero_less_power)

```

```

lemma Arg2pi_cnj:  $\text{Arg2pi}(\text{cnj } z) = (\text{if } z \in \mathbb{R} \text{ then } \text{Arg2pi } z \text{ else } 2*\pi - \text{Arg2pi } z)$ 
proof (cases  $z=0$ )
  case False

```

2330

```

    then show ?thesis
      by (simp add: Arg2pi_cnj_eq_inverse Arg2pi_inverse)
qed auto

```

```

lemma Arg2pi_exp:  $0 \leq \text{Im } z \implies \text{Im } z < 2\pi \implies \text{Arg2pi}(\exp z) = \text{Im } z$ 
  by (rule Arg2pi_unique [of exp(Re z)]) (auto simp: exp_eq_polar)

```

```

lemma complex_split_polar:
  obtains  $r a :: \text{real}$  where  $z = \text{complex\_of\_real } r * (\cos a + i * \sin a)$   $0 \leq r$   $0 \leq a < 2\pi$ 
  using Arg2pi_cis.ctr cis_conv_exp unfolding Complex_eq is_Arg_def by fast-
  force

```

```

lemma Re_Im_le_cmod:  $\text{Im } w * \sin \varphi + \text{Re } w * \cos \varphi \leq \text{cmod } w$ 
proof (cases w rule: complex_split_polar)
  case (1 r a) with sin_cos_le1 [of a  $\varphi$ ] show ?thesis
    apply (simp add: norm_mult cmod_unit_one)
    by (metis (no_types, opaque_lifting) abs_le_D1 distrib_left mult.commute
        mult.left_commute mult_left_le)
qed

```

6.21.8 Analytic properties of tangent function

```

lemma cnj_tan:  $\text{cnj}(\tan z) = \tan(\text{cnj } z)$ 
  by (simp add: cnj_cos cnj_sin tan_def)

```

```

lemma field_differentiable_at_tan:  $\cos z \neq 0 \implies \tan \text{ field\_differentiable at } z$ 
  unfolding field_differentiable_def
  using DERIV_tan by blast

```

```

lemma field_differentiable_within_tan:  $\cos z \neq 0 \implies \tan \text{ field\_differentiable (at } z \text{ within } s)$ 
  using field_differentiable_at_tan field_differentiable_at_within by blast

```

```

lemma continuous_within_tan:  $\cos z \neq 0 \implies \text{continuous (at } z \text{ within } s) \tan$ 
  using continuous_at_imp_continuous_within isCont_tan by blast

```

```

lemma continuous_on_tan [continuous_intros]:  $(\bigwedge z. z \in s \implies \cos z \neq 0) \implies \text{continuous\_on } s \tan$ 
  by (simp add: continuous_at_imp_continuous_on)

```

```

lemma holomorphic_on_tan:  $(\bigwedge z. z \in s \implies \cos z \neq 0) \implies \tan \text{ holomorphic\_on } s$ 
  by (simp add: field_differentiable_within_tan holomorphic_on_def)

```

6.21.9 The principal branch of the Complex logarithm

```

instantiation complex :: ln
begin

```

definition $\ln\_complex :: complex \Rightarrow complex$
where $\ln\_complex \equiv \lambda z. \text{THE } w. \exp w = z \ \& \ -\pi < \text{Im}(w) \ \& \ \text{Im}(w) \leq \pi$

NOTE: within this scope, the constant Ln is not yet available!

lemma

assumes $z \neq 0$
shows $\exp\_Ln \ [simp]: \exp(\ln z) = z$
and $\text{mpi\_less\_Im\_Ln}: -\pi < \text{Im}(\ln z)$
and $\text{Im\_Ln\_le\_pi}: \text{Im}(\ln z) \leq \pi$

proof –

obtain ψ **where** $z / (cmod\ z) = \text{Complex}(\cos \psi) (\sin \psi)$
using $\text{complex\_unimodular\_polar} \ [of\ z / (norm\ z)] \ \text{assms}$
by $(\text{auto simp: norm\_divide field\_split\_sims})$
obtain φ **where** $-\pi < \varphi \leq \pi \ \sin \varphi = \sin \psi \ \cos \varphi = \cos \psi$
using $\text{sincos\_principal\_value} \ [of\ \psi] \ \text{assms}$
by $(\text{auto simp: norm\_divide field\_split\_sims})$
have $\exp(\ln z) = z \ \& \ -\pi < \text{Im}(\ln z) \ \& \ \text{Im}(\ln z) \leq \pi$ **unfolding** $\ln\_complex\_def$
apply $(\text{rule theI} \ [where\ a = \text{Complex}(\ln(norm\ z))\ \varphi])$
using $z \ \text{assms}\ \varphi$
apply $(\text{auto simp: field\_sims exp\_complex\_eqI exp\_eq\_polar cis.code})$
done
then show $\exp(\ln z) = z \ -\pi < \text{Im}(\ln z) \ \text{Im}(\ln z) \leq \pi$
by auto

qed

lemma $\text{Ln\_exp} \ [simp]:$

assumes $-\pi < \text{Im}(z) \ \text{Im}(z) \leq \pi$
shows $\ln(\exp z) = z$

proof $(\text{rule exp\_complex\_eqI})$

show $|\text{Im}(\ln(\exp z)) - \text{Im}\ z| < 2 * \pi$
using $\text{assms mpi\_less\_Im\_Ln} \ [of\ \exp\ z] \ \text{Im\_Ln\_le\_pi} \ [of\ \exp\ z] \ \text{by auto}$

qed auto

6.21.10 Relation to Real Logarithm

lemma $\text{Ln\_of\_real}:$

assumes $0 < z$
shows $\ln(\text{of\_real } z::\text{complex}) = \text{of\_real}(\ln z)$

proof –

have $\ln(\text{of\_real}(\exp(\ln z)::\text{complex})) = \ln(\exp(\text{of\_real}(\ln z)))$
by $(\text{simp add: exp\_of\_real})$
also have $\dots = \text{of\_real}(\ln z)$
using $\text{assms by (subst Ln\_exp) auto}$
finally show $?thesis$
using assms by simp

qed

corollary $\text{Ln\_in\_Reals} \ [simp]: z \in \mathbb{R} \implies \text{Re } z > 0 \implies \ln z \in \mathbb{R}$

by $(\text{auto simp: Ln\_of\_real elim: Reals\_cases})$

corollary *Im Ln_of_real* [simp]: $r > 0 \implies \text{Im} (\ln (\text{of\_real } r)) = 0$
by (*simp add: Ln_of_real*)

lemma *cmod Ln_Reals* [simp]: $z \in \mathbb{R} \implies 0 < \text{Re } z \implies \text{cmod} (\ln z) = \text{norm} (\ln (\text{Re } z))$
using *Ln_of_real* **by** *force*

lemma *Ln_Reals_eq*: $\llbracket x \in \mathbb{R}; \text{Re } x > 0 \rrbracket \implies \ln x = \text{of\_real} (\ln (\text{Re } x))$
using *Ln_of_real* **by** *force*

lemma *Ln_1* [simp]: $\ln 1 = (0::\text{complex})$

proof –

have $\ln (\exp 0) = (0::\text{complex})$

by (*simp add: del: exp_zero*)

then show *?thesis*

by *simp*

qed

lemma *Ln_eq_zero_iff* [simp]: $x \notin \mathbb{R}_{\leq 0} \implies \ln x = 0 \iff x = 1$ **for** $x::\text{complex}$
by *auto (metis exp_Ln exp_zero nonpos_Reals_zero_I)*

instance

by *intro_classes (rule ln_complex_def Ln_1)*

end

abbreviation $\text{Ln} :: \text{complex} \Rightarrow \text{complex}$

where $\text{Ln} \equiv \ln$

lemma *Ln_eq_iff*: $w \neq 0 \implies z \neq 0 \implies (\text{Ln } w = \text{Ln } z \iff w = z)$
by (*metis exp_Ln*)

lemma *Ln_unique*: $\exp(z) = w \implies -\pi < \text{Im}(z) \implies \text{Im}(z) \leq \pi \implies \text{Ln } w = z$
using *Ln_exp* **by** *blast*

lemma *Re_Ln* [simp]: $z \neq 0 \implies \text{Re}(\text{Ln } z) = \ln(\text{norm } z)$
by (*metis exp_Ln ln_exp norm_exp_eq_Re*)

corollary *ln_cmod_le*:

assumes $z: z \neq 0$

shows $\ln (\text{cmod } z) \leq \text{cmod} (\text{Ln } z)$

using *norm_exp [of Ln z, simplified exp_Ln [OF z]]*

by (*metis Re_Ln complex_Re_le_cmod z*)

proposition *exists_complex_root*:

fixes $z :: \text{complex}$

assumes $n \neq 0$ **obtains** w **where** $z = w \wedge^n$


```

proof (cases z=0)
  case False
  then show ?thesis
    by (rule_tac w = exp(Ln z / n) in that) (simp add: assms exp_of_nat_mult
[symmetric])
qed (use assms in auto)

```

corollary *exists_complex_root_nonzero*:

```

fixes z::complex
assumes z ≠ 0 n ≠ 0
obtains w where w ≠ 0 z = w ^ n
by (metis exists_complex_root [of n z] assms power_0_left)

```

6.21.11 Derivative of Ln away from the branch cut

lemma

```

assumes z ∉ ℝ≤0
shows has_field_derivative Ln: (Ln has_field_derivative inverse(z)) (at z)
and Im_Ln_less_pi: Im (Ln z) < pi

```

proof –

```

have znz [simp]: z ≠ 0
using assms by auto
then have Im (Ln z) ≠ pi
by (metis (no_types) Im_exp Ln_in_Reals assms complex_nonpos_Reals_iff
complex_is_Real_iff exp_Ln_mult_zero_right not_less pi_neq_zero sin_pi znz)

```

```

then show *: Im (Ln z) < pi using assms Im_Ln_le_pi
by (simp add: le_neq_trans)

```

```

let ?U = {w. -pi < Im(w) ∧ Im(w) < pi}
have 1: open ?U

```

```

by (simp add: open_Collect_conj open_halfspace_Im_gt open_halfspace_Im_lt)

```

```

have 2: ∧x. x ∈ ?U ⇒ (exp has_derivative blinfun_apply(Blinfun ((*)) (exp
x)))) (at x)

```

```

by (simp add: bounded_linear_Blinfun_apply bounded_linear_mult_right has_field_derivative_imp_has_derivative)

```

```

have 3: continuous_on ?U (λx. Blinfun ((*)) (exp x))

```

```

unfolding blinfun_mult_right.abs_eq [symmetric] by (intro continuous_intros)

```

```

have 4: Ln z ∈ ?U

```

```

by (auto simp: mpi_less_Im_Ln *)

```

```

have 5: Blinfun ((*)) (inverse z) o_L Blinfun ((*)) (exp (Ln z)) = id_blinfun

```

```

by (rule blinfun_eqI) (simp add: bounded_linear_mult_right bounded_linear_Blinfun_apply)

```

```

obtain U' V g g' where open U' and sub: U' ⊆ ?U

```

```

and Ln z ∈ U' open V z ∈ V

```

```

and hom: homeomorphism U' V exp g

```

```

and g: ∧y. y ∈ V ⇒ (g has_derivative (g' y)) (at y)

```

```

and g': ∧y. y ∈ V ⇒ g' y = inv ((*)) (exp (g y))

```

```

and bij: ∧y. y ∈ V ⇒ bij ((*)) (exp (g y))

```

```

using inverse_function_theorem [OF 1 2 3 4 5]

```

```

by (simp add: bounded_linear_Blinfun_apply bounded_linear_mult_right)

```

blast

```

show (Ln has_field_derivative inverse(z)) (at z)
  unfolding has_field_derivative_def
proof (rule has_derivative_transform_within_open)
  show g_eq_Ln: g y = Ln y if y ∈ V for y
  proof -
    obtain x where y = exp x x ∈ U'
    using hom.homeomorphism_image1 that ⟨y ∈ V⟩ by blast
    then show ?thesis
    using sub.hom.homeomorphism_apply1 by fastforce
  qed
have 0 ∉ V
  by (meson exp_not_eq_zero hom.homeomorphism_def)
then have ∧y. y ∈ V ⇒ g' y = inv ((* ) y)
  by (metis exp_Ln g' g_eq_Ln)
then have g': g' z = (λx. x/z)
  by (metis (no_types, opaque_lifting) bij ⟨z ∈ V⟩ bij_inv_eq_iff exp_Ln
g_eq_Ln nonzero_mult_div_cancel_left znz)
  show (g has_derivative (* ) (inverse z)) (at z)
  using g [OF ⟨z ∈ V⟩] g'
  by (simp add: ⟨z ∈ V⟩ field_class.field_divide_inverse has_derivative_imp_has_field_derivative
has_field_derivative_imp_has_derivative)
  qed (auto simp: ⟨z ∈ V⟩ ⟨open V⟩)
qed

declare has_field_derivative_Ln [derivative_intros]
declare has_field_derivative_Ln [THEN DERIV_chain2, derivative_intros]

lemma field_differentiable_at_Ln: z ∉ ℝ≤0 ⇒ Ln field_differentiable at z
  using field_differentiable_def has_field_derivative_Ln by blast

lemma field_differentiable_within_Ln: z ∉ ℝ≤0
  ⇒ Ln field_differentiable (at z within S)
  using field_differentiable_at_Ln field_differentiable_within_subset by blast

lemma continuous_at_Ln: z ∉ ℝ≤0 ⇒ continuous (at z) Ln
  by (simp add: field_differentiable_imp_continuous_at field_differentiable_within_Ln)

lemma isCont_Ln' [simp, continuous_intros]:
  ⟦isCont f z; f z ∉ ℝ≤0⟧ ⇒ isCont (λx. Ln (f x)) z
  by (blast intro: isCont_o2 [OF continuous_at_Ln])

lemma continuous_within_Ln [continuous_intros]: z ∉ ℝ≤0 ⇒ continuous (at
z within S) Ln
  using continuous_at_Ln continuous_at_imp_continuous_within by blast

lemma continuous_on_Ln [continuous_intros]: (∧z. z ∈ S ⇒ z ∉ ℝ≤0) ⇒
continuous_on S Ln
  by (simp add: continuous_at_imp_continuous_on continuous_within_Ln)

```

lemma *continuous_on_Ln'* [continuous_intros]:
 $\text{continuous\_on } S \ f \implies (\bigwedge z. z \in S \implies f \ z \notin \mathbb{R}_{\leq 0}) \implies \text{continuous\_on } S \ (\lambda x. \text{Ln } (f \ x))$
by (rule continuous_on_compose2[OF continuous_on_Ln, of UNIV - non-pos_Reals S f]) auto

lemma *holomorphic_on_Ln* [holomorphic_intros]: $S \cap \mathbb{R}_{\leq 0} = \{\}$ $\implies \text{Ln holomorphic\_on } S$
by (simp add: disjoint_iff field_differentiable_within_Ln holomorphic_on_def)

lemma *holomorphic_on_Ln'* [holomorphic_intros]:
 $(\bigwedge z. z \in A \implies f \ z \notin \mathbb{R}_{\leq 0}) \implies f \text{ holomorphic\_on } A \implies (\lambda z. \text{Ln } (f \ z)) \text{ holomorphic\_on } A$
using holomorphic_on_compose_gen[OF holomorphic_on_Ln, of f A - $\mathbb{R}_{\leq 0}$]
by (auto simp: o_def)

lemma *tendsto_Ln* [tendsto_intros]:
fixes $L \ F$
assumes $(f \longrightarrow L) \ F \ L \notin \mathbb{R}_{\leq 0}$
shows $((\lambda x. \text{Ln } (f \ x)) \longrightarrow \text{Ln } L) \ F$
proof -
have $\text{nhds } L \geq \text{filtermap } f \ F$
using assms(1) **by** (simp add: filterlim_def)
moreover have $\forall_F y \text{ in nhds } L. y \in -\mathbb{R}_{\leq 0}$
using eventually_nhds_in_open[of $-\mathbb{R}_{\leq 0}$ L] assms **by** (auto simp: open_Compl)
ultimately have $\forall_F y \text{ in filtermap } f \ F. y \in -\mathbb{R}_{\leq 0}$ **by** (rule filter_leD)
moreover have continuous_on $(-\mathbb{R}_{\leq 0}) \ \text{Ln}$ **by** (rule continuous_on_Ln) auto
ultimately show ?thesis **using** continuous_on_tendsto_compose[of $-\mathbb{R}_{\leq 0}$ Ln f L F] assms
by (simp add: eventually_filtermap)
qed

lemma *divide_ln_mono*:
fixes $x \ y :: \text{real}$
assumes $3 \leq x \leq y$
shows $x / \ln x \leq y / \ln y$
proof (rule exE [OF complex_mvt_line [of x y $\lambda z. z / \text{Ln } z \ \lambda z. 1 / (\text{Ln } z) - 1 / (\text{Ln } z)^2$]]);
clarisimp simp add: closed_segment_Reals closed_segment_eq_real_ivl assms)
show $\bigwedge u. [x \leq u; u \leq y] \implies ((\lambda z. z / \text{Ln } z) \text{ has\_field\_derivative } 1 / \text{Ln } u - 1 / (\text{Ln } u)^2) \text{ (at } u)$
using $\langle 3 \leq x \rangle$ **by** (force intro!: derivative_eq_intros simp: field_simps power_eq_if)
show $x / \ln x \leq y / \ln y$
if $\text{Re } (y / \text{Ln } y) - \text{Re } (x / \text{Ln } x) = (\text{Re } (1 / \text{Ln } u) - \text{Re } (1 / (\text{Ln } u)^2)) * (y - x)$
and $x: x \leq u \leq y$ **for** u
proof -
have $\text{eq}: y / \ln y = (1 / \ln u - 1 / (\ln u)^2) * (y - x) + x / \ln x$
using that $\langle 3 \leq x \rangle$ **by** (auto simp: Ln_Reals_eq_in_Reals_norm_group_add_class.diff_eq_eq)

```

show ?thesis
  using exp_le ⟨3 ≤ x⟩ x by (simp add: eq) (simp add: power_eq_if di-
vide_simps ln_ge_iff)
qed
qed

```

theorem *Ln_series*:

```

fixes z :: complex
assumes norm z < 1
shows (λn. (-1)^Suc n / of_nat n * z^n) sums ln (1 + z) (is (λn. ?f n *
z^n) sums _)
proof -
  let ?F = λz. ∑ n. ?f n * z^n and ?F' = λz. ∑ n. diffs ?f n * z^n
  have r: conv_radius ?f = 1
    by (intro conv_radius_ratio_limit_nonzero[of _ 1])
    (simp_all add: norm_divide LIMSEQ_Suc_n_over_n del: of_nat_Suc)

  have ∃ c. ∀ z ∈ ball 0 1. ln (1 + z) - ?F z = c
  proof (rule has_field_derivative_zero_constant)
    fix z :: complex assume z': z ∈ ball 0 1
    hence z: norm z < 1 by simp
    define t :: complex where t = of_real (1 + norm z) / 2
    from z have t: norm z < norm t norm t < 1 unfolding t_def
      by (simp_all add: field_simps norm_divide del: of_real_add)

    have Re (-z) ≤ norm (-z) by (rule complex_Re_le_cmod)
    also from z have ... < 1 by simp
    finally have ((λz. ln (1 + z)) has_field_derivative inverse (1+z)) (at z)
      by (auto intro!: derivative_eq_intros simp: complex_nonpos_Reals_iff)
    moreover have (?F has_field_derivative ?F' z) (at z) using t r
      by (intro termdiffs_strong[of _ t] summable_in_conv_radius) simp_all
    ultimately have ((λz. ln (1 + z) - ?F z) has_field_derivative (inverse (1 +
z) - ?F' z))
      (at z within ball 0 1)
      by (intro derivative_intros) (simp_all add: at_within_open[OF z'])
    also have (λn. of_nat n * ?f n * z^(n - Suc 0)) sums ?F' z using t r
      by (intro diffs_equiv termdiff_converges[OF t(1)] summable_in_conv_radius)
    simp_all
    from sums_split_initial_segment[OF this, of 1]
      have (λi. (-z) ^ i) sums ?F' z by (simp add: power_minus[of z] del:
of_nat_Suc)
    hence ?F' z = inverse (1 + z) using z by (simp add: sums_iff sum-
inf_geometric_divide_inverse)
    also have inverse (1 + z) - inverse (1 + z) = 0 by simp
    finally show ((λz. ln (1 + z) - ?F z) has_field_derivative 0) (at z within ball
0 1) .
  qed simp_all
  then obtain c where c: ∧z. z ∈ ball 0 1 ⇒ ln (1 + z) - ?F z = c by blast
  from c[of 0] have c = 0 by (simp only: powser_zero) simp

```

with $c[of\ z]$ *assms* have $\ln (1 + z) = ?F\ z$ **by** *simp*
 moreover have *summable* $(\lambda n. ?f\ n * z^{\wedge}n)$ **using** *assms r*
by (*intro summable_in_conv_radius*) *simp_all*
 ultimately show *?thesis* **by** (*simp add: sums_iff*)
qed

lemma *Ln_series'*: $cmod\ z < 1 \implies (\lambda n. -((-z)^{\wedge}n) / of\_nat\ n)\ sums\ \ln (1 + z)$
by (*drule Ln_series*) (*simp add: power_minus'*)

lemma *ln_series'*:
 assumes $abs\ (x::real) < 1$
 shows $(\lambda n. -((-x)^{\wedge}n) / of\_nat\ n)\ sums\ \ln (1 + x)$
proof -
 from *assms* have $(\lambda n. -((-of\_real\ x)^{\wedge}n) / of\_nat\ n)\ sums\ \ln (1 + complex\_of\_real\ x)$
by (*intro Ln_series'*) *simp_all*
 also have $(\lambda n. -((-of\_real\ x)^{\wedge}n) / of\_nat\ n) = (\lambda n. complex\_of\_real\ (-((-x)^{\wedge}n) / of\_nat\ n))$
by (*rule ext*) *simp*
 also from *assms* have $\ln (1 + complex\_of\_real\ x) = of\_real\ (\ln (1 + x))$
by (*subst Ln_of_real [symmetric]*) *simp_all*
 finally show *?thesis* **by** (*subst (asm) sums_of_real_iff*)
qed

lemma *Ln_approx_linear*:
 fixes $z :: complex$
 assumes $norm\ z < 1$
 shows $norm\ (\ln (1 + z) - z) \leq norm\ z^{\wedge}2 / (1 - norm\ z)$
proof -
 let $?f = \lambda n. (-1)^{\wedge}Suc\ n / of\_nat\ n$
 from *assms* have $(\lambda n. ?f\ n * z^{\wedge}n)\ sums\ \ln (1 + z)$ **using** *Ln_series* **by** *simp*
 moreover have $(\lambda n. (if\ n = 1\ then\ 1\ else\ 0) * z^{\wedge}n)\ sums\ z$ **using** *powser_sums_if* [*of 1*] **by** *simp*
 ultimately have $(\lambda n. (?f\ n - (if\ n = 1\ then\ 1\ else\ 0)) * z^{\wedge}n)\ sums\ (\ln (1 + z) - z)$
by (*subst left_diff_distrib, intro sums_diff*) *simp_all*
 from *sums_split_initial_segment* [*OF this, of Suc 1*]
 have $(\lambda i. -(z^{\wedge}2)) * inverse\ (2 + of\_nat\ i) * (-z)^{\wedge}i\ sums\ (\ln (1 + z) - z)$
by (*simp add: power2_eq_square mult_ac power_minus [of z] divide_inverse*)
 hence $(\ln (1 + z) - z) = (\sum i. -(z^{\wedge}2)) * inverse\ (of\_nat\ (i+2)) * (-z)^{\wedge}i$
by (*simp add: sums_iff*)
 also have $A: summable\ (\lambda n. norm\ z^{\wedge}2 * (inverse\ (real\_of\_nat\ (Suc\ (Suc\ n)))) * cmod\ z^{\wedge}n)$
by (*rule summable_mult, rule summable_comparison_test_ev* [*OF summable_geometric [of norm z]*])
 (*auto simp: assms field_simps intro!: always_eventually*)
 hence $norm\ (\sum i. -(z^{\wedge}2)) * inverse\ (of\_nat\ (i+2)) * (-z)^{\wedge}i$

```

    ≤ (∑ i. norm (-(z^2) * inverse (of_nat (i+2))) * (-z)^i)
  by (intro summable_norm)
    (auto simp: norm_power norm_inverse norm_mult mult_ac simp del:
of_nat_add of_nat_Suc)
  also have norm ((-z)^2 * (-z)^i * inverse (of_nat (i+2))) ≤ norm ((-z)^2
* (-z)^i) * 1 for i
  by (intro mult_left_mono) (simp_all add: field_split_simps)
  hence (∑ i. norm (-(z^2) * inverse (of_nat (i+2))) * (-z)^i)
    ≤ (∑ i. norm (-(z^2) * (-z)^i))
  using A assms
  unfolding norm_power norm_inverse norm_divide norm_mult
  apply (intro suminf_le summable_mult summable_geometric)
  apply (auto simp: norm_power field_simps simp del: of_nat_add of_nat_Suc)
  done
  also have ... = norm z^2 * (∑ i. norm z^i) using assms
  by (subst suminf_mult [symmetric]) (auto intro!: summable_geometric simp:
norm_mult norm_power)
  also have (∑ i. norm z^i) = inverse (1 - norm z) using assms
  by (subst suminf_geometric) (simp_all add: divide_inverse)
  also have norm z^2 * ... = norm z^2 / (1 - norm z) by (simp add: di-
vide_inverse)
  finally show ?thesis .
qed

```

6.21.12 Quadrant-type results for Ln

```

lemma cos_lt_zero_pi: pi/2 < x ⟹ x < 3*pi/2 ⟹ cos x < 0
  using cos_minus_pi cos_gt_zero_pi [of x-pi]
  by simp

```

```

lemma Re_Ln_pos_lt:
  assumes z ≠ 0
  shows |Im(Ln z)| < pi/2 ⟷ 0 < Re(z)
proof -
  { fix w
    assume w = Ln z
    then have w: Im w ≤ pi - pi < Im w
      using Im_Ln_le_pi [of z] mpi_less_Im_Ln [of z] assms
      by auto
    have |Im w| < pi/2 ⟷ 0 < Re(exp w)
  proof
    assume |Im w| < pi/2 then show 0 < Re(exp w)
      by (auto simp: Re_exp cos_gt_zero_pi split: if_split_asm)
    next
      assume R: 0 < Re(exp w) then
        have |Im w| ≠ pi/2
        by (metis cos_minus_pi_half mult_eq_0_iff Re_exp abs_if_order_less_irrefl)
        then show |Im w| < pi/2
          using cos_lt_zero_pi [of -(Im w)] cos_lt_zero_pi [of (Im w)] w R

```

```

      by (force simp: Re_exp zero_less_mult_iff abs_if not_less_iff_gr_or_eq)
    qed
  }
  then show ?thesis using assms
    by auto
  qed

```

```

lemma Re_Ln_pos_le:
  assumes  $z \neq 0$ 
  shows  $|Im(Ln\ z)| \leq \pi/2 \longleftrightarrow 0 \leq Re(z)$ 
proof -
  { fix w
    assume  $w = Ln\ z$ 
    then have  $w: Im\ w \leq \pi - \pi < Im\ w$ 
      using Im_Ln_le_pi [of z] mpi_less_Im_Ln [of z] assms
      by auto
    then have  $|Im\ w| \leq \pi/2 \longleftrightarrow 0 \leq Re(exp\ w)$ 
      using cos_lt_zero_pi [of - (Im w)] cos_lt_zero_pi [of (Im w)] not_le
      by (auto simp: Re_exp zero_le_mult_iff abs_if intro: cos_ge_zero)
  }
  then show ?thesis using assms
    by auto
  qed

```

```

lemma Im_Ln_pos_lt:
  assumes  $z \neq 0$ 
  shows  $0 < Im(Ln\ z) \wedge Im(Ln\ z) < \pi \longleftrightarrow 0 < Im(z)$ 
proof -
  { fix w
    assume  $w = Ln\ z$ 
    then have  $w: Im\ w \leq \pi - \pi < Im\ w$ 
      using Im_Ln_le_pi [of z] mpi_less_Im_Ln [of z] assms
      by auto
    then have  $0 < Im\ w \wedge Im\ w < \pi \longleftrightarrow 0 < Im(exp\ w)$ 
      using sin_gt_zero [of - (Im w)] sin_gt_zero [of (Im w)] less_linear
      by (fastforce simp add: Im_exp zero_less_mult_iff)
  }
  then show ?thesis using assms
    by auto
  qed

```

```

lemma Im_Ln_pos_le:
  assumes  $z \neq 0$ 
  shows  $0 \leq Im(Ln\ z) \wedge Im(Ln\ z) \leq \pi \longleftrightarrow 0 \leq Im(z)$ 
proof -
  { fix w
    assume  $w = Ln\ z$ 
    then have  $w: Im\ w \leq \pi - \pi < Im\ w$ 

```

```

    using Im_Ln_le_pi [of z] mpi_less_Im_Ln [of z] assms
  by auto
  then have  $0 \leq \text{Im } w \wedge \text{Im } w \leq \pi \iff 0 \leq \text{Im}(\exp w)$ 
    using sin_ge_zero [of  $-(\text{Im } w)$ ] sin_ge_zero [of  $\text{abs}(\text{Im } w)$ ] sin_zero_pi_iff
  [of  $\text{Im } w$ ]
    by (force simp: Im_exp_zero_le_mult_iff sin_ge_zero) }
  then show ?thesis using assms
    by auto
qed

```

```

lemma Re_Ln_pos_lt_imp:  $0 < \text{Re}(z) \implies |\text{Im}(\text{Ln } z)| < \pi/2$ 
  by (metis Re_Ln_pos_lt less_irrefl zero_complex.simps(1))

```

```

lemma Im_Ln_pos_lt_imp:  $0 < \text{Im}(z) \implies 0 < \text{Im}(\text{Ln } z) \wedge \text{Im}(\text{Ln } z) < \pi$ 
  by (metis Im_Ln_pos_lt not_le order_refl zero_complex.simps(2))

```

A reference to the set of positive real numbers

```

lemma Im_Ln_eq_0:  $z \neq 0 \implies (\text{Im}(\text{Ln } z) = 0 \iff 0 < \text{Re}(z) \wedge \text{Im}(z) = 0)$ 
  by (metis Im_complex_of_real Im_exp Ln_in_Reals Re_Ln_pos_lt Re_Ln_pos_lt_imp
    Re_complex_of_real complex_is_Real_iff exp_Ln_exp_of_real pi_gt_zero)

```

```

lemma Im_Ln_eq_pi:  $z \neq 0 \implies (\text{Im}(\text{Ln } z) = \pi \iff \text{Re}(z) < 0 \wedge \text{Im}(z) = 0)$ 
  by (metis Im_Ln_eq_0 Im_Ln_pos_le Im_Ln_pos_lt add.left_neutral complex_eq
    less_eq_real_def
    mult_zero_right not_less_iff_gr_or_eq pi_ge_zero pi_neq_zero rcis_zero_arg
    rcis_zero_mod)

```

6.21.13 More Properties of Ln

```

lemma cnj_Ln: assumes  $z \notin \mathbb{R}_{\leq 0}$  shows  $\text{cnj}(\text{Ln } z) = \text{Ln}(\text{cnj } z)$ 
proof (cases  $z=0$ )
  case False
  show ?thesis
  proof (rule exp_complex_eqI)
    have  $|\text{Im}(\text{cnj}(\text{Ln } z)) - \text{Im}(\text{Ln}(\text{cnj } z))| \leq |\text{Im}(\text{cnj}(\text{Ln } z))| + |\text{Im}(\text{Ln}(\text{cnj } z))|$ 
  z))|
    by (rule abs_triangle_ineq4)
  also have  $\dots < \pi + \pi$ 
  proof -
    have  $|\text{Im}(\text{cnj}(\text{Ln } z))| < \pi$ 
    by (simp add: False Im_Ln_less_pi abs_if assms minus_less_iff mpi_less_Im_Ln)
    moreover have  $|\text{Im}(\text{Ln}(\text{cnj } z))| \leq \pi$ 
    by (meson abs_le_iff complex_cnj_zero_iff less_eq_real_def minus_less_iff
      False Im_Ln_le_pi mpi_less_Im_Ln)
    ultimately show ?thesis
    by simp
  qed
  finally show  $|\text{Im}(\text{cnj}(\text{Ln } z)) - \text{Im}(\text{Ln}(\text{cnj } z))| < 2 * \pi$ 
    by simp

```



```

  show  $\exp(\text{cnj } (Ln\ z)) = \exp(Ln(\text{cnj } z))$ 
  by (metis False complex_cnj_zero_iff exp_Ln exp_cnj)
qed
qed (use assms in auto)

```

```

lemma Ln_inverse: assumes  $z \notin \mathbb{R}_{\leq 0}$  shows  $Ln(\text{inverse } z) = -(Ln\ z)$ 
proof (cases  $z=0$ )
  case False
  show ?thesis
  proof (rule exp_complex_eqI)
    have  $|Im(Ln(\text{inverse } z)) - Im(-Ln\ z)| \leq |Im(Ln(\text{inverse } z))| + |Im(-Ln\ z)|$ 
    by (rule abs_triangle_ineq4)
    also have  $\dots < \pi + \pi$ 
    proof -
      have  $|Im(Ln(\text{inverse } z))| < \pi$ 
      by (simp add: False Im_Ln_less_pi abs_if assms minus_less_iff mpi_less_Im_Ln)
      moreover have  $|Im(-Ln\ z)| \leq \pi$ 
      using False Im_Ln_le_pi mpi_less_Im_Ln by fastforce
      ultimately show ?thesis
      by simp
    qed
  qed
  finally show  $|Im(Ln(\text{inverse } z)) - Im(-Ln\ z)| < 2 * \pi$ 
  by simp
  show  $\exp(Ln(\text{inverse } z)) = \exp(-Ln\ z)$ 
  by (simp add: False exp_minus)
qed
qed (use assms in auto)

```

```

lemma Ln_minus1 [simp]:  $Ln(-1) = i * \pi$ 
proof (rule exp_complex_eqI)
  show  $|Im(Ln(-1)) - Im(i * \pi)| < 2 * \pi$ 
  using Im_Ln_le_pi [of -1] mpi_less_Im_Ln [of -1] by auto
qed auto

```

```

lemma Ln_ii [simp]:  $Ln\ i = i * \text{of\_real } \pi/2$ 
  using Ln_exp [of  $i * (\text{of\_real } \pi/2)$ ]
  unfolding exp_Euler
  by simp

```

```

lemma Ln_minus_ii [simp]:  $Ln(-i) = -(i * \pi/2)$ 
proof -
  have  $Ln(-i) = Ln(\text{inverse } i)$  by simp
  also have  $\dots = -(Ln\ i)$  using Ln_inverse by blast
  also have  $\dots = -(i * \pi/2)$  by simp
  finally show ?thesis .
qed

```

lemma *Ln_times*:

assumes $w \neq 0$ $z \neq 0$

shows $\text{Ln}(w * z) =$

(if $\text{Im}(\text{Ln } w + \text{Ln } z) \leq -\pi$ then $(\text{Ln}(w) + \text{Ln}(z)) + i * \text{of\_real}(2 * \pi)$
 else if $\text{Im}(\text{Ln } w + \text{Ln } z) > \pi$ then $(\text{Ln}(w) + \text{Ln}(z)) - i * \text{of\_real}(2 * \pi)$
 else $\text{Ln}(w) + \text{Ln}(z)$)

using *pi_ge_zero Im_Ln_le_pi [of w] Im_Ln_le_pi [of z]*

using *assms mpi_less_Im_Ln [of w] mpi_less_Im_Ln [of z]*

by (*auto simp: exp_add exp_diff sin_double cos_double exp_Euler intro!: Ln_unique*)

corollary *Ln_times_simple*:

$\llbracket w \neq 0; z \neq 0; -\pi < \text{Im}(\text{Ln } w) + \text{Im}(\text{Ln } z); \text{Im}(\text{Ln } w) + \text{Im}(\text{Ln } z) \leq \pi \rrbracket$
 $\implies \text{Ln}(w * z) = \text{Ln}(w) + \text{Ln}(z)$

by (*simp add: Ln_times*)

corollary *Ln_times_of_real*:

$\llbracket r > 0; z \neq 0 \rrbracket \implies \text{Ln}(\text{of\_real } r * z) = \ln r + \text{Ln}(z)$

using *mpi_less_Im_Ln Im_Ln_le_pi*

by (*force simp: Ln_times*)

corollary *Ln_times_of_nat*:

$\llbracket r > 0; z \neq 0 \rrbracket \implies \text{Ln}(\text{of\_nat } r * z :: \text{complex}) = \ln(\text{of\_nat } r) + \text{Ln}(z)$

using *Ln_times_of_real[of of_nat r z]* **by** *simp*

corollary *Ln_times_Reals*:

$\llbracket r \in \text{Reals}; \text{Re } r > 0; z \neq 0 \rrbracket \implies \text{Ln}(r * z) = \ln(\text{Re } r) + \text{Ln}(z)$

using *Ln_Reals_eq Ln_times_of_real* **by** *fastforce*

corollary *Ln_divide_of_real*:

$\llbracket r > 0; z \neq 0 \rrbracket \implies \text{Ln}(z / \text{of\_real } r) = \text{Ln}(z) - \ln r$

using *Ln_times_of_real [of inverse r z]*

by (*simp add: ln_inverse Ln_of_real mult.commute divide_inverse of_real_inverse [symmetric]*)

del: of_real_inverse)

corollary *Ln_prod*:

fixes $f :: 'a \Rightarrow \text{complex}$

assumes *finite A* $\bigwedge x. x \in A \implies f x \neq 0$

shows $\exists n. \text{Ln}(\text{prod } f A) = (\sum x \in A. \text{Ln}(f x) + (\text{of\_int } (n x) * (2 * \pi)) * i)$

using *assms*

proof (*induction A*)

case (*insert x A*)

then obtain n **where** $n: \text{Ln}(\text{prod } f A) = (\sum x \in A. \text{Ln}(f x) + \text{of\_real}(\text{of\_int}(n x) * (2 * \pi)) * i)$

by *auto*

define D **where** $D \equiv \text{Im}(\text{Ln}(f x)) + \text{Im}(\text{Ln}(\text{prod } f A))$

define $q::\text{int}$ **where** $q \equiv (\text{if } D \leq -\pi \text{ then } 1 \text{ else if } D > \pi \text{ then } -1 \text{ else } 0)$

have $\text{prod } f A \neq 0$ $f x \neq 0$

by (*auto simp: insert.hyps insert.premis*)

```

with insert.hyps pi_ge_zero show ?case
  by (rule_tac x=n(x:=q) in exI) (force simp: Ln_times q_def D_def n intro!:
sum.cong)
qed auto

```

```

lemma Ln_minus:
  assumes  $z \neq 0$ 
  shows  $\text{Ln}(-z) = (\text{if } \text{Im}(z) \leq 0 \wedge \neg(\text{Re}(z) < 0 \wedge \text{Im}(z) = 0)$ 
    then  $\text{Ln}(z) + i * \pi$ 
    else  $\text{Ln}(z) - i * \pi$ ) (is  $\_ = ?rhs$ )
  using Im_Ln_le_pi [of z] mpi_less_Im_Ln [of z] assms
    Im_Ln_eq_pi [of z] Im_Ln_pos_lt [of z]
  by (fastforce simp: exp_add exp_diff exp_Euler intro!: Ln_unique)

```

```

lemma Ln_inverse_if:
  assumes  $z \neq 0$ 
  shows  $\text{Ln}(\text{inverse } z) = (\text{if } z \in \mathbb{R}_{\leq 0} \text{ then } -(\text{Ln } z) + i * 2 * \text{complex\_of\_real}$ 
     $\pi$  else  $-(\text{Ln } z)$ )
  proof (cases  $z \in \mathbb{R}_{\leq 0}$ )
    case False then show ?thesis
      by (simp add: Ln_inverse)
  next
    case True
    then have  $z: \text{Im } z = 0 \wedge \text{Re } z < 0 \rightarrow z \notin \mathbb{R}_{\leq 0}$ 
      using assms complex_eq_iff complex_nonpos_Reals_iff by auto
    have  $\text{Ln}(\text{inverse } z) = \text{Ln}(-(\text{inverse } (-z)))$ 
      by simp
    also have  $\dots = \text{Ln}(\text{inverse } (-z)) + i * \text{complex\_of\_real } \pi$ 
      using assms z by (simp add: Ln_minus divide_less_0_iff)
    also have  $\dots = -\text{Ln}(-z) + i * \text{complex\_of\_real } \pi$ 
      using z Ln_inverse by presburger
    also have  $\dots = -(\text{Ln } z) + i * 2 * \text{complex\_of\_real } \pi$ 
      using Ln_minus assms z by auto
    finally show ?thesis by (simp add: True)
  qed

```

```

lemma Ln_times_ii:
  assumes  $z \neq 0$ 
  shows  $\text{Ln}(i * z) = (\text{if } 0 \leq \text{Re}(z) \mid \text{Im}(z) < 0$ 
    then  $\text{Ln}(z) + i * \text{of\_real } \pi/2$ 
    else  $\text{Ln}(z) - i * \text{of\_real } (3 * \pi/2)$ )
  using Im_Ln_le_pi [of z] mpi_less_Im_Ln [of z] assms
    Im_Ln_eq_pi [of z] Im_Ln_pos_lt [of z] Re_Ln_pos_le [of z]
  by (simp add: Ln_times) auto

```

```

lemma Ln_of_nat [simp]:  $0 < n \implies \text{Ln}(\text{of\_nat } n) = \text{of\_real}(\ln(\text{of\_nat } n))$ 
  by (subst of_real_of_nat_eq[symmetric], subst Ln_of_real[symmetric]) simp_all

```

```

lemma Ln_of_nat_over_of_nat:

```

```

assumes  $m > 0 \ n > 0$ 
shows  $\text{Ln} (\text{of\_nat } m / \text{of\_nat } n) = \text{of\_real} (\ln (\text{of\_nat } m) - \ln (\text{of\_nat } n))$ 
proof -
  have  $\text{of\_nat } m / \text{of\_nat } n = (\text{of\_real} (\text{of\_nat } m / \text{of\_nat } n) :: \text{complex})$  by
    simp
  also from assms have  $\text{Ln} \dots = \text{of\_real} (\ln (\text{of\_nat } m / \text{of\_nat } n))$ 
    by (simp add: Ln_of_real[symmetric])
  also from assms have  $\dots = \text{of\_real} (\ln (\text{of\_nat } m) - \ln (\text{of\_nat } n))$ 
    by (simp add: ln_div)
  finally show ?thesis .
qed

```

6.21.14 The Argument of a Complex Number

Unlike in HOL Light, it's defined for the same interval as the complex logarithm: $(-\pi, \pi]$.

```

lemma Arg_eq_Im_Ln:
  assumes  $z \neq 0$  shows  $\text{Arg } z = \text{Im} (\text{Ln } z)$ 
proof (rule cis_Arg_unique)
  show  $\text{sgn } z = \text{cis} (\text{Im} (\text{Ln } z))$ 
    by (metis assms exp_Ln exp_eq_polar nonzero_mult_div_cancel_left norm_eq_zero

      norm_exp_eq_Re of_real_eq_0_iff sgn_eq)
  show  $-\pi < \text{Im} (\text{Ln } z)$ 
    by (simp add: assms mpi_less_Im_Ln)
  show  $\text{Im} (\text{Ln } z) \leq \pi$ 
    by (simp add: Im_Ln_le_pi assms)
qed

```

The 1990s definition of argument coincides with the more recent one

```

lemma Arg_def:
  shows  $\text{Arg } z = (\text{if } z = 0 \text{ then } 0 \text{ else } \text{Im} (\text{Ln } z))$ 
  by (simp add: Arg_eq_Im_Ln Arg_zero)

```

```

lemma Arg_of_real [simp]:  $\text{Arg} (\text{of\_real } r) = (\text{if } r < 0 \text{ then } \pi \text{ else } 0)$ 
  by (simp add: Im_Ln_eq_pi Arg_def)

```

```

lemma mpi_less_Arg:  $-\pi < \text{Arg } z$ 
  and Arg_le_pi:  $\text{Arg } z \leq \pi$ 
  by (auto simp: Arg_def mpi_less_Im_Ln Im_Ln_le_pi)

```

```

lemma
  assumes  $z \neq 0$ 
  shows  $\text{Arg\_eq}: z = \text{of\_real}(\text{norm } z) * \exp(i * \text{Arg } z)$ 
  using assms exp_Ln exp_eq_polar
  by (auto simp: Arg_def)

```

```

lemma is_Arg_Arg:  $z \neq 0 \implies \text{is\_Arg } z (\text{Arg } z)$ 

```

by (simp add: Arg_eq is_Arg_def)

lemma Argument_exists:

assumes $z \neq 0$ and $R: R = \{r - \pi < \dots r + \pi\}$

obtains s where $\text{is\_Arg } z \ s \ s \in R$

proof –

let $?rp = r - \text{Arg } z + \pi$

define k where $k \equiv \lfloor ?rp / (2 * \pi) \rfloor$

have $(\text{Arg } z + \text{of\_int } k * (2 * \pi)) \in R$

using floor_divide_lower [of $2 * \pi$ $?rp$] floor_divide_upper [of $2 * \pi$ $?rp$]

by (auto simp: k_def algebra_simps R)

then show ?thesis

using Arg_eq $\langle z \neq 0 \rangle$ is_Arg_2pi_iff is_Arg_def that by blast

qed

lemma Argument_exists_unique:

assumes $z \neq 0$ and $R: R = \{r - \pi < \dots r + \pi\}$

obtains s where $\text{is\_Arg } z \ s \ s \in R \wedge t. \llbracket \text{is\_Arg } z \ t; t \in R \rrbracket \implies s = t$

proof –

obtain s where $s: \text{is\_Arg } z \ s \ s \in R$

using Argument_exists [OF assms] .

moreover have $\wedge t. \llbracket \text{is\_Arg } z \ t; t \in R \rrbracket \implies s = t$

using assms s by (auto simp: is_Arg_eqI)

ultimately show thesis

using that by blast

qed

lemma Argument_Ex1:

assumes $z \neq 0$ and $R: R = \{r - \pi < \dots r + \pi\}$

shows $\exists! s. \text{is\_Arg } z \ s \wedge s \in R$

using Argument_exists_unique [OF assms] by metis

lemma Arg_divide:

assumes $w \neq 0 \ z \neq 0$

shows $\text{is\_Arg } (z / w) (\text{Arg } z - \text{Arg } w)$

using Arg_eq [of z] Arg_eq [of w] Arg_eq [of $\text{norm}(z / w)$] assms

by (auto simp: is_Arg_def norm_divide field_simps exp_diff Arg_of_real)

lemma Arg_unique_lemma:

assumes $z: \text{is\_Arg } z \ t$

and $z': \text{is\_Arg } z \ t'$

and $t: -\pi < t \leq \pi$

and $t': -\pi < t' \leq \pi$

and $nz: z \neq 0$

shows $t' = t$

using Arg2pi_unique_lemma [of $-(\text{inverse } z)$]

proof –

have $\pi - t' = \pi - t$

proof (rule Arg2pi_unique_lemma [of $-(\text{inverse } z)$])

```

have - (inverse z) = - (inverse (of_real(norm z) * exp(i * t)))
  by (metis is_Arg_def z)
also have ... = (cmod (- inverse z)) * exp (i * (pi - t))
  by (auto simp: field_simps exp_diff norm_divide)
finally show is_Arg (- inverse z) (pi - t)
  unfolding is_Arg_def .
have - (inverse z) = - (inverse (of_real(norm z) * exp(i * t')))
  by (metis is_Arg_def z')
also have ... = (cmod (- inverse z)) * exp (i * (pi - t'))
  by (auto simp: field_simps exp_diff norm_divide)
finally show is_Arg (- inverse z) (pi - t')
  unfolding is_Arg_def .
qed (use assms in auto)
then show ?thesis
  by simp
qed

lemma complex_norm_eq_1_exp_eq: norm z = 1  $\longleftrightarrow$  exp(i * (Arg z)) = z
  by (metis Arg_eq exp_not_eq_zero exp_zero mult.left_neutral norm_zero of_real_1
    norm_exp_i_times)

lemma Arg_unique: [of_real r * exp(i * a) = z; 0 < r; -pi < a; a  $\leq$  pi]  $\implies$ 
  Arg z = a
  by (rule Arg_unique_lemma [unfolded is_Arg_def, OF Arg_eq])
  (use mpi_less_Arg Arg_le_pi in <auto simp: norm_mult>)

lemma Arg_minus:
  assumes z  $\neq$  0
  shows Arg (-z) = (if Arg z  $\leq$  0 then Arg z + pi else Arg z - pi)
proof -
  have [simp]: cmod z * cos (Arg z) = Re z
    using assms Arg_eq [of z] by (metis Re_exp exp_Ln norm_exp_eq Re
    Arg_def)
  have [simp]: cmod z * sin (Arg z) = Im z
    using assms Arg_eq [of z] by (metis Im_exp exp_Ln norm_exp_eq Re
    Arg_def)
  show ?thesis
    apply (rule Arg_unique [of norm z, OF complex_eqI])
    using mpi_less_Arg [of z] Arg_le_pi [of z] assms
    by (auto simp: Re_exp Im_exp)
qed

lemma Arg_times_of_real [simp]:
  assumes 0 < r shows Arg (of_real r * z) = Arg z
proof (cases z=0)
case True
  then show ?thesis
    by (simp add: Arg_def)
next

```

```

case False
with Arg_eq assms show ?thesis
by (auto simp: mpi_less_Arg Arg_le_pi intro!: Arg_unique [of r * norm z])
qed

```

```

lemma Arg_times_of_real2 [simp]:  $0 < r \implies \text{Arg } (z * \text{of\_real } r) = \text{Arg } z$ 
by (metis Arg_times_of_real mult.commute)

```

```

lemma Arg_divide_of_real [simp]:  $0 < r \implies \text{Arg } (z / \text{of\_real } r) = \text{Arg } z$ 
by (metis Arg_times_of_real2 less_numeral_extra(3) nonzero_eq_divide_eq
of_real_eq_0_iff)

```

```

lemma Arg_less_0:  $0 \leq \text{Arg } z \longleftrightarrow 0 \leq \text{Im } z$ 
using Im_Ln_le_pi Im_Ln_pos_le
by (simp add: Arg_def)

```

```

lemma Arg_eq_pi:  $\text{Arg } z = \pi \longleftrightarrow \text{Re } z < 0 \wedge \text{Im } z = 0$ 
by (auto simp: Arg_def Im_Ln_eq_pi)

```

```

lemma Arg_lt_pi:  $0 < \text{Arg } z \wedge \text{Arg } z < \pi \longleftrightarrow 0 < \text{Im } z$ 
using Arg_less_0 [of z] Im_Ln_pos_lt
by (auto simp: order.order_iff_strict Arg_def)

```

```

lemma Arg_eq_0:  $\text{Arg } z = 0 \longleftrightarrow z \in \mathbb{R} \wedge 0 \leq \text{Re } z$ 
using complex_is_Real_iff
by (simp add: Arg_def Im_Ln_eq_0) (metis less_eq_real_def of_real_Re of_real_def
scale_zero_left)

```

```

corollary Arg_ne_0: assumes  $z \notin \mathbb{R}_{\geq 0}$  shows  $\text{Arg } z \neq 0$ 
using assms by (auto simp: nonneg_Reals_def Arg_eq_0)

```

```

lemma Arg_eq_pi_iff:  $\text{Arg } z = \pi \longleftrightarrow z \in \mathbb{R} \wedge \text{Re } z < 0$ 
proof (cases z=0)
case False
then show ?thesis
using Arg_eq_0 [of -z] Arg_eq_pi complex_is_Real_iff by blast
qed (simp add: Arg_def)

```

```

lemma Arg_eq_0_pi:  $\text{Arg } z = 0 \vee \text{Arg } z = \pi \longleftrightarrow z \in \mathbb{R}$ 
using Arg_eq_pi_iff Arg_eq_0 by force

```

```

lemma Arg_real:  $z \in \mathbb{R} \implies \text{Arg } z = (\text{if } 0 \leq \text{Re } z \text{ then } 0 \text{ else } \pi)$ 
using Arg_eq_0 Arg_eq_0_pi by auto

```

```

lemma Arg_inverse:  $\text{Arg}(\text{inverse } z) = (\text{if } z \in \mathbb{R} \text{ then } \text{Arg } z \text{ else } -\text{Arg } z)$ 
proof (cases z ∈ ℝ)
case True
then show ?thesis
by simp (metis Arg2pi_inverse Arg2pi_real Arg_real Reals_inverse)

```

```

next
  case False
  then have z: Arg z < pi z ≠ 0
    using Arg_eq_0_pi Arg_le_pi by (auto simp: less_eq_real_def)
  show ?thesis
    apply (rule Arg_unique [of inverse (norm z)])
    using False z mpi_less_Arg [of z] Arg_eq [of z]
    by (auto simp: exp_minus field_simps)
qed

lemma Arg_eq_iff:
  assumes w ≠ 0 z ≠ 0
  shows Arg w = Arg z ⟷ (∃ x. 0 < x ∧ w = of_real x * z) (is ?lhs = ?rhs)
proof
  assume ?lhs
  then have w = complex_of_real (cmod w / cmod z) * z
    by (metis Arg_eq assms divide_divide_eq_right eq_divide_eq exp_not_eq_zero
of_real_divide)
  then show ?rhs
    using assms divide_pos_pos zero_less_norm_iff by blast
qed auto

lemma Arg_inverse_eq_0: Arg(inverse z) = 0 ⟷ Arg z = 0
  by (metis Arg_eq_0 Arg_inverse inverse_inverse_eq)

lemma Arg_cnj_eq_inverse: z ≠ 0 ⟹ Arg (cnj z) = Arg (inverse z)
  using Arg2pi_cnj_eq_inverse Arg2pi_eq_iff Arg_eq_iff by auto

lemma Arg_cnj: Arg(cnj z) = (if z ∈ ℝ then Arg z else - Arg z)
  by (metis Arg_cnj_eq_inverse Arg_inverse Reals_0 complex_cnj_zero)

lemma Arg_exp: -pi < Im z ⟹ Im z ≤ pi ⟹ Arg(exp z) = Im z
  by (rule Arg_unique [of exp(Re z)]) (auto simp: exp_eq_polar)

lemma Ln_Arg: z ≠ 0 ⟹ Ln(z) = ln(norm z) + i * Arg(z)
  by (metis Arg_def Re_Ln complex_eq)

lemma continuous_at_Arg:
  assumes z ∉ ℝ≤0
  shows continuous (at z) Arg
proof -
  have [simp]: (λz. Im (Ln z)) -z→ Arg z
    using Arg_def assms continuous_at by fastforce
  show ?thesis
    unfolding continuous_at
  proof (rule Lim_transform_within_open)
    show ∧w. [w ∈ - ℝ≤0; w ≠ z] ⟹ Im (Ln w) = Arg w
      by (metis Arg_def Compl_iff nonpos_Reals_zero_I)
    qed (use assms in auto)

```


qed

lemma *continuous_within_Arg*: $z \notin \mathbb{R}_{\leq 0} \implies \text{continuous (at } z \text{ within } S) \text{ Arg}$
using *continuous_at_Arg continuous_at_imp_continuous_within* **by** *blast*

6.21.15 The Unwinding Number and the Ln product Formula

Note that in this special case the unwinding number is -1, 0 or 1. But it's always an integer.

lemma *is_Arg_exp_Im*: $\text{is\_Arg (exp } z) (\text{Im } z)$
using *exp_eq_polar is_Arg_def norm_exp_eq_Re* **by** *auto*

lemma *is_Arg_exp_diff_2pi*:
assumes *is_Arg (exp z) ϑ*
shows $\exists k. \text{Im } z - \text{of\_int } k * (2 * \pi) = \vartheta$
proof (*intro exI is_Arg_eqI*)
let $?k = \lfloor (\text{Im } z - \vartheta) / (2 * \pi) \rfloor$
show $\text{is\_Arg (exp } z) (\text{Im } z - \text{real\_of\_int } ?k * (2 * \pi))$
by (*metis diff_add_cancel is_Arg_2pi_iff is_Arg_exp_Im*)
show $|\text{Im } z - \text{real\_of\_int } ?k * (2 * \pi) - \vartheta| < 2 * \pi$
using *floor_divide_upper [of 2*pi Im z - ϑ] floor_divide_lower [of 2*pi Im z - ϑ]*
by (*auto simp: algebra_simps abs_if*)
qed (*auto simp: is_Arg_exp_Im assms*)

lemma *Arg_exp_diff_2pi*: $\exists k. \text{Im } z - \text{of\_int } k * (2 * \pi) = \text{Arg (exp } z)$
using *is_Arg_exp_diff_2pi [OF is_Arg_Arg]* **by** *auto*

lemma *unwinding_in_Ints*: $(z - \text{Ln}(\text{exp } z)) / (\text{of\_real}(2 * \pi) * i) \in \mathbb{Z}$
using *Arg_exp_diff_2pi [of z]*
by (*force simp: Ints_def image_def field_simps Arg_def intro!: complex_eqI*)

definition *unwinding* :: $\text{complex} \Rightarrow \text{int}$ **where**
 $\text{unwinding } z \equiv \text{THE } k. \text{of\_int } k = (z - \text{Ln}(\text{exp } z)) / (\text{of\_real}(2 * \pi) * i)$

lemma *unwinding*: $\text{of\_int } (\text{unwinding } z) = (z - \text{Ln}(\text{exp } z)) / (\text{of\_real}(2 * \pi) * i)$
using *unwinding_in_Ints [of z]*
unfolding *unwinding_def Ints_def* **by** *force*

lemma *unwinding_2pi*: $(2 * \pi) * i * \text{unwinding}(z) = z - \text{Ln}(\text{exp } z)$
by (*simp add: unwinding*)

lemma *Ln_times_unwinding*:
 $w \neq 0 \implies z \neq 0 \implies \text{Ln}(w * z) = \text{Ln}(w) + \text{Ln}(z) - (2 * \pi) * i * \text{unwinding}(\text{Ln } w + \text{Ln } z)$
using *unwinding_2pi* **by** (*simp add: exp_add*)

```

lemma arg_conv_arctan:
  assumes  $\operatorname{Re} z > 0$ 
  shows  $\operatorname{Arg} z = \arctan (\operatorname{Im} z / \operatorname{Re} z)$ 
proof (rule cis_Arg_unique)
  show  $\operatorname{sgn} z = \operatorname{cis} (\arctan (\operatorname{Im} z / \operatorname{Re} z))$ 
  proof (rule complex_eqI)
    have  $\operatorname{Re} (\operatorname{cis} (\arctan (\operatorname{Im} z / \operatorname{Re} z))) = 1 / \sqrt{1 + (\operatorname{Im} z)^2 / (\operatorname{Re} z)^2}$ 
      by (simp add: cos_arctan power_divide)
    also have  $1 + \operatorname{Im} z^2 / \operatorname{Re} z^2 = \operatorname{norm} z^2 / \operatorname{Re} z^2$ 
      using assms by (simp add: cmod_def field_simps)
    also have  $1 / \sqrt{\dots} = \operatorname{Re} z / \operatorname{norm} z$ 
      using assms by (simp add: real_sqrt_divide)
    finally show  $\operatorname{Re} (\operatorname{sgn} z) = \operatorname{Re} (\operatorname{cis} (\arctan (\operatorname{Im} z / \operatorname{Re} z)))$ 
      by simp
  next
    have  $\operatorname{Im} (\operatorname{cis} (\arctan (\operatorname{Im} z / \operatorname{Re} z))) = \operatorname{Im} z / (\operatorname{Re} z * \sqrt{1 + (\operatorname{Im} z)^2 / (\operatorname{Re} z)^2})$ 
      by (simp add: sin_arctan field_simps)
    also have  $1 + \operatorname{Im} z^2 / \operatorname{Re} z^2 = \operatorname{norm} z^2 / \operatorname{Re} z^2$ 
      using assms by (simp add: cmod_def field_simps)
    also have  $\operatorname{Im} z / (\operatorname{Re} z * \sqrt{\dots}) = \operatorname{Im} z / \operatorname{norm} z$ 
      using assms by (simp add: real_sqrt_divide)
    finally show  $\operatorname{Im} (\operatorname{sgn} z) = \operatorname{Im} (\operatorname{cis} (\arctan (\operatorname{Im} z / \operatorname{Re} z)))$ 
      by simp
  qed
next
  show  $\arctan (\operatorname{Im} z / \operatorname{Re} z) > -\pi$ 
    by (rule le_less_trans[OF _ arctan_lbound]) auto
next
  have  $\arctan (\operatorname{Im} z / \operatorname{Re} z) < \pi / 2$ 
    by (rule arctan_ubound)
  also have  $\dots \leq \pi$  by simp
  finally show  $\arctan (\operatorname{Im} z / \operatorname{Re} z) \leq \pi$ 
    by simp
qed

```

6.21.16 Relation between Ln and $\operatorname{Arg}2\pi$, and hence continuity of $\operatorname{Arg}2\pi$

```

lemma Arg2pi_Ln:
  assumes  $0 < \operatorname{Arg}2\pi z$  shows  $\operatorname{Arg}2\pi z = \operatorname{Im}(\operatorname{Ln}(-z)) + \pi$ 
proof (cases  $z = 0$ )
  case True
    with assms show ?thesis
      by simp
  next
    case False
    then have  $z / \operatorname{of\_real}(\operatorname{norm} z) = \exp(i * \operatorname{of\_real}(\operatorname{Arg}2\pi z))$ 

```

```

    using Arg2pi [of z]
  by (metis is_Arg_def abs_norm_cancel nonzero_mult_div_cancel_left norm_of_real
zero_less_norm_iff)
  then have  $-z / \text{of\_real}(\text{norm } z) = \exp(i * (\text{of\_real}(\text{Arg2pi } z) - \pi))$ 
    using cis_conv_exp cis_pi
  by (auto simp: exp_diff algebra_simps)
  then have  $\ln(-z / \text{of\_real}(\text{norm } z)) = \ln(\exp(i * (\text{of\_real}(\text{Arg2pi } z) - \pi)))$ 
    by simp
  also have  $\dots = i * (\text{of\_real}(\text{Arg2pi } z) - \pi)$ 
    using Arg2pi [of z] assms pi_not_less_zero
  by auto
  finally have  $\text{Arg2pi } z = \text{Im}(\ln(-z / \text{of\_real}(\text{cmod } z))) + \pi$ 
    by simp
  also have  $\dots = \text{Im}(\ln(-z) - \ln(\text{cmod } z)) + \pi$ 
    by (metis diff_0_right minus_diff_eq zero_less_norm_iff Ln_divide_of_real
False)
  also have  $\dots = \text{Im}(\ln(-z)) + \pi$ 
    by simp
  finally show ?thesis .
qed

```

```

lemma continuous_at_Arg2pi:
  assumes  $z \notin \mathbb{R}_{\geq 0}$ 
  shows continuous (at z) Arg2pi
proof -
  have *:  $\text{isCont}(\lambda z. \text{Im}(\ln(-z)) + \pi) z$ 
    by (rule Complex.isCont_Im isCont_Ln' continuous_intros | simp add: assms
complex_is_Real_iff)+
  have [simp]:  $\text{Im } x \neq 0 \implies \text{Im}(\ln(-x)) + \pi = \text{Arg2pi } x$  for  $x$ 
    using Arg2pi_Ln by (simp add: Arg2pi_gt_0 complex_nonneg_Reals_iff)
  consider  $\text{Re } z < 0 \mid \text{Im } z \neq 0$  using assms
  using complex_nonneg_Reals_iff not_le by blast
  then have [simp]:  $(\lambda z. \text{Im}(\ln(-z)) + \pi) - z \rightarrow \text{Arg2pi } z$ 
    using * by (simp add: Arg2pi_Ln Arg2pi_gt_0 assms continuous_within)
  show ?thesis
    unfolding continuous_at
  proof (rule Lim_transform_within_open)
    show  $\bigwedge x. [x \in -\mathbb{R}_{\geq 0}; x \neq z] \implies \text{Im}(\ln(-x)) + \pi = \text{Arg2pi } x$ 
      by (auto simp add: Arg2pi_Ln [OF Arg2pi_gt_0] complex_nonneg_Reals_iff)
    qed (use assms in auto)
  qed

```

Relation between Arg2pi and arctangent in upper halfplane

```

lemma Arg2pi_arctan_upperhalf:
  assumes  $0 < \text{Im } z$ 
  shows  $\text{Arg2pi } z = \pi/2 - \arctan(\text{Re } z / \text{Im } z)$ 
proof (cases  $z = 0$ )
  case False
  show ?thesis

```

```

proof (rule Arg2pi_unique [of norm z])
  show (cmod z) * exp (i * (pi / 2 - arctan (Re z / Im z))) = z
  apply (rule complex_eqI)
  using assms norm_complex_def [of z, symmetric]
  unfolding exp_Euler cos_diff sin_diff sin_of_real cos_of_real
  by (simp_all add: field_simps real_sqrt_divide sin_arctan cos_arctan)
qed (use False arctan [of Re z / Im z] in auto)
qed (use assms in auto)

lemma Arg2pi_eq_Im_Ln:
  assumes  $0 \leq \text{Im } z$   $0 < \text{Re } z$ 
  shows  $\text{Arg2pi } z = \text{Im } (\text{Ln } z)$ 
proof (cases  $\text{Im } z = 0$ )
  case True then show ?thesis
    using Arg2pi_eq_0 Ln_in_Reals assms(2) complex_is_Real_iff by auto
  next
  case False
  then have *:  $\text{Arg2pi } z > 0$ 
    using Arg2pi_gt_0 complex_is_Real_iff by blast
  then have  $z \neq 0$ 
    by auto
  with * assms False show ?thesis
    by (subst Arg2pi_Ln) (auto simp: Ln_minus)
qed

lemma continuous_within_upperhalf_Arg2pi:
  assumes  $z \neq 0$ 
  shows continuous (at z within {z.  $0 \leq \text{Im } z$ }) Arg2pi
proof (cases  $z \in \mathbb{R}_{\geq 0}$ )
  case False then show ?thesis
    using continuous_at_Arg2pi continuous_at_imp_continuous_within by auto
  next
  case True
  then have z:  $z \in \mathbb{R}$   $0 < \text{Re } z$ 
    using assms by (auto simp: complex_nonneg_Reals_iff complex_is_Real_iff
    complex_neq_0)
  then have [simp]:  $\text{Arg2pi } z = 0$   $\text{Im } (\text{Ln } z) = 0$ 
    by (auto simp: Arg2pi_eq_0 Im_Ln_eq_0 assms complex_is_Real_iff)
  show ?thesis
proof (clarsimp simp add: continuous_within Lim_within dist_norm)
  fix e::real
  assume  $0 < e$ 
  moreover have continuous (at z) ( $\lambda x. \text{Im } (\text{Ln } x)$ )
    using z by (simp add: continuous_at_Ln complex_nonpos_Reals_iff)
  ultimately
  obtain d where  $d > 0 \wedge x \neq z \implies \text{cmod } (x - z) < d \implies |\text{Im } (\text{Ln } x)| < e$ 
    by (auto simp: continuous_within Lim_within dist_norm)
  { fix x

```

```

    assume  $cmod (x - z) < Re\ z / 2$ 
    then have  $|Re\ x - Re\ z| < Re\ z / 2$ 
      by (metis le_less_trans abs_Re_le_cmod minus_complex.simps(1))
    then have  $0 < Re\ x$ 
      using  $z$  by linarith
  }
  then show  $\exists d > 0. \forall x. 0 \leq Im\ x \longrightarrow x \neq z \wedge cmod (x - z) < d \longrightarrow |Arg2pi\ x| < e$ 
    apply (rule_tac  $x = \min\ d\ (Re\ z / 2)$  in exI)
    using  $z\ d$  by (auto simp: Arg2pi_eq_Im_Ln)
qed
qed

```

```

lemma continuous_on_upperhalf_Arg2pi: continuous_on ( $\{z. 0 \leq Im\ z\} - \{0\}$ )
  Arg2pi
  unfolding continuous_on_eq_continuous_within
  by (metis DiffE Diff_subset continuous_within_subset continuous_within_upperhalf_Arg2pi
    insertCI)

```

```

lemma open_Arg2pi2pi_less_Int:
  assumes  $0 \leq s\ t \leq 2 * pi$ 
  shows  $open\ (\{y. s < Arg2pi\ y\} \cap \{y. Arg2pi\ y < t\})$ 
proof -
  have 1: continuous_on (UNIV -  $\mathbb{R}_{\geq 0}$ ) Arg2pi
    using continuous_at_Arg2pi continuous_at_imp_continuous_within
    by (auto simp: continuous_on_eq_continuous_within)
  have 2: open (UNIV -  $\mathbb{R}_{\geq 0} :: complex\ set$ ) by (simp add: open_Diff)
  have open ( $\{z. s < z\} \cap \{z. z < t\}$ )
    using open_lessThan [of  $t$ ] open_greaterThan [of  $s$ ]
    by (metis greaterThan_def lessThan_def open_Int)
  moreover have  $\{y. s < Arg2pi\ y\} \cap \{y. Arg2pi\ y < t\} \subseteq -\mathbb{R}_{\geq 0}$ 
    using assms by (auto simp: Arg2pi_real complex_nonneg_Reals_iff complex_is_Real_iff)
  ultimately show ?thesis
    using continuous_imp_open_vimage [OF 1 2, of  $\{z. Re\ z > s\} \cap \{z. Re\ z < t\}$ ]
    by auto
qed

```

```

lemma open_Arg2pi2pi_gt: open  $\{z. t < Arg2pi\ z\}$ 
proof (cases  $t < 0$ )
  case True then have  $\{z. t < Arg2pi\ z\} = UNIV$ 
    using Arg2pi_ge_0 less_le_trans by auto
  then show ?thesis
    by simp
next
  case False then show ?thesis
    using open_Arg2pi2pi_less_Int [of  $t\ 2 * pi$ ] Arg2pi_lt_2pi
    by auto

```

2354

qed

```
lemma closed_Arg2pi2pi_le: closed {z. Arg2pi z ≤ t}
  using open_Arg2pi2pi_gt [of t]
  by (simp add: closed_def Set.Collect_neg_eq [symmetric] not_le)
```

6.21.17 Complex Powers

```
lemma powr_to_1 [simp]: z powr 1 = (z::complex)
  by (simp add: powr_def)
```

```
lemma powr_nat:
  fixes n::nat and z::complex shows z powr n = (if z = 0 then 0 else z^n)
  by (simp add: exp_of_nat_mult powr_def)
```

```
lemma norm_powr_real: w ∈ ℝ ⇒ 0 < Re w ⇒ norm(w powr z) = exp(Re z
* ln(Re w))
  using Ln_Reals_eq norm_exp_eq_Re by (auto simp: Im_Ln_eq_0 powr_def
norm_complex_def)
```

```
lemma powr_complexpow [simp]:
  fixes x::complex shows x ≠ 0 ⇒ x powr (of_nat n) = x^n
  by (induct n) (auto simp: ac_simps powr_add)
```

```
lemma powr_complexnumeral [simp]:
  fixes x::complex shows x powr (numeral n) = x ^ (numeral n)
  by (metis of_nat_numeral power_zero_numeral powr_nat)
```

```
lemma cnj_powr:
  assumes Im a = 0 ⇒ Re a ≥ 0
  shows cnj (a powr b) = cnj a powr cnj b
proof (cases a = 0)
  case False
  with assms have a ∉ ℝ≤0 by (auto simp: complex_eq_iff complex_nonpos_Reals_iff)
  with False show ?thesis by (simp add: powr_def exp_cnj cnj_Ln)
qed simp
```

```
lemma powr_real_real:
  assumes w ∈ ℝ z ∈ ℝ 0 < Re w
  shows w powr z = exp(Re z * ln(Re w))
proof -
  have w ≠ 0
    using assms by auto
  with assms show ?thesis
    by (simp add: powr_def Ln_Reals_eq of_real_exp)
qed
```

```
lemma powr_of_real:
  fixes x::real and y::real
```

shows $0 \leq x \implies \text{of\_real } x \text{ powr } (\text{of\_real } y :: \text{complex}) = \text{of\_real } (x \text{ powr } y)$
by (simp_all add: powr_def exp_eq_polar)

lemma powr_of_int:
fixes $z :: \text{complex}$ **and** $n :: \text{int}$
assumes $z \neq (0 :: \text{complex})$
shows $z \text{ powr of\_int } n = (\text{if } n \geq 0 \text{ then } z^{\text{nat } n} \text{ else } \text{inverse } (z^{\text{nat } (-n)}))$
by (metis assms not_le of_int_of_nat powr_complexpow powr_minus)

lemma powr_Reals_eq: $\llbracket x \in \mathbb{R}; y \in \mathbb{R}; \text{Re } x \geq 0 \rrbracket \implies x \text{ powr } y = \text{of\_real } (\text{Re } x \text{ powr } \text{Re } y)$
by (metis of_real_Re powr_of_real)

lemma norm_powr_real_mono:
 $\llbracket w \in \mathbb{R}; 1 < \text{Re } w \rrbracket$
 $\implies \text{cmod}(w \text{ powr } z1) \leq \text{cmod}(w \text{ powr } z2) \longleftrightarrow \text{Re } z1 \leq \text{Re } z2$
by (auto simp: powr_def algebra_simps Reals_def Ln_of_real)

lemma powr_times_real:
 $\llbracket x \in \mathbb{R}; y \in \mathbb{R}; 0 \leq \text{Re } x; 0 \leq \text{Re } y \rrbracket$
 $\implies (x * y) \text{ powr } z = x \text{ powr } z * y \text{ powr } z$
by (auto simp: Reals_def powr_def Ln_times exp_add algebra_simps less_eq_real_def Ln_of_real)

lemma Re_powr_le: $r \in \mathbb{R}_{\geq 0} \implies \text{Re } (r \text{ powr } z) \leq \text{Re } r \text{ powr } \text{Re } z$
by (auto simp: powr_def nonneg_Reals_def order_trans [OF complex_Re_le_cmod])

lemma
fixes $w :: \text{complex}$
shows $\text{Reals\_powr } [\text{simp}]: \llbracket w \in \mathbb{R}_{\geq 0}; z \in \mathbb{R} \rrbracket \implies w \text{ powr } z \in \mathbb{R}$
and $\text{nonneg\_Reals\_powr } [\text{simp}]: \llbracket w \in \mathbb{R}_{\geq 0}; z \in \mathbb{R} \rrbracket \implies w \text{ powr } z \in \mathbb{R}_{\geq 0}$
by (auto simp: nonneg_Reals_def Reals_def powr_of_real)

lemma powr_neg_real_complex:
 $(-\text{of\_real } x) \text{ powr } a = (-1) \text{ powr } (\text{of\_real } (\text{sgn } x) * a) * \text{of\_real } x \text{ powr } (a :: \text{complex})$
proof (cases $x = 0$)
assume $x: x \neq 0$
hence $(-x) \text{ powr } a = \exp(a * \ln(-\text{of\_real } x))$ **by** (simp add: powr_def)
also from x **have** $\ln(-\text{of\_real } x) = \text{Ln}(\text{of\_real } x) + \text{of\_real } (\text{sgn } x) * \pi * i$
by (simp add: Ln_minus Ln_of_real)
also from x **have** $\exp(a * \dots) = \text{cis } \pi \text{ powr } (\text{of\_real } (\text{sgn } x) * a) * \text{of\_real } x \text{ powr } a$
by (simp add: powr_def exp_add algebra_simps Ln_of_real cis_conv_exp)
also note cis_pi
finally show ?thesis **by** simp
qed simp_all

lemma has_field_derivative_powr:

```

fixes  $z :: \text{complex}$ 
assumes  $z \notin \mathbb{R}_{\leq 0}$ 
shows  $((\lambda z. z \text{ powr } s) \text{ has\_field\_derivative } (s * z \text{ powr } (s - 1))) \text{ (at } z)$ 
proof (cases  $z=0$ )
  case False
    then have  $\S: \exp (s * \text{Ln } z) * \text{inverse } z = \exp ((s - 1) * \text{Ln } z)$ 
      by (simp add: divide_complex_def exp_diff left_diff_distrib')
    show ?thesis
      unfolding powr_def
    proof (rule has_field_derivative_transform_within)
      show  $((\lambda z. \exp (s * \text{Ln } z)) \text{ has\_field\_derivative } s * (\text{if } z = 0 \text{ then } 0 \text{ else } \exp ((s - 1) * \text{Ln } z)))$ 
        (at  $z$ )
      by (intro derivative_eq_intros | simp add: assms False  $\S$ ) +
    qed (use False in auto)
qed (use assms in auto)

declare has_field_derivative_powr[THEN DERIV_chain2, derivative_intros]

lemma has_field_derivative_powr_of_int:
  fixes  $z :: \text{complex}$ 
  assumes gderiv:  $(g \text{ has\_field\_derivative } gd) \text{ (at } z \text{ within } S) \text{ and } g \text{ z} \neq 0$ 
  shows  $((\lambda z. g \text{ z powr of\_int } n) \text{ has\_field\_derivative } (n * g \text{ z powr (of\_int } n - 1) * gd)) \text{ (at } z \text{ within } S)$ 
proof -
  define  $dd$  where  $dd = \text{of\_int } n * g \text{ z powr (of\_int } (n - 1)) * gd$ 
  obtain  $e$  where  $e > 0$  and  $e\_dist: \forall y \in S. \text{dist } z \ y < e \longrightarrow g \ y \neq 0$ 
  using DERIV_continuous[OF gderiv, THEN continuous_within_avoid]  $\langle g \text{ z} \neq 0 \rangle$ 
by auto
  have ?thesis when  $n \geq 0$ 
  proof -
    define  $dd'$  where  $dd' = \text{of\_int } n * g \text{ z}^{\wedge} (\text{nat } n - 1) * gd$ 
    have  $dd = dd'$ 
    proof (cases  $n=0$ )
      case False
        then have  $n - 1 \geq 0$  using  $\langle n \geq 0 \rangle$  by auto
        then have  $g \text{ z powr (of\_int } (n - 1)) = g \text{ z}^{\wedge} (\text{nat } n - 1)$ 
          using powr_of_int[OF  $\langle g \text{ z} \neq 0 \rangle$ , of  $n - 1$ ] by (simp add: nat_diff_distrib')
        then show ?thesis unfolding dd_def dd'_def by simp
    qed (simp add: dd_def dd'_def)
    then have  $((\lambda z. g \text{ z powr of\_int } n) \text{ has\_field\_derivative } dd) \text{ (at } z \text{ within } S)$ 
       $\longleftrightarrow ((\lambda z. g \text{ z powr of\_int } n) \text{ has\_field\_derivative } dd') \text{ (at } z \text{ within } S)$ 
    by simp
    also have  $\dots \longleftrightarrow ((\lambda z. g \text{ z}^{\wedge} \text{nat } n) \text{ has\_field\_derivative } dd') \text{ (at } z \text{ within } S)$ 
proof (rule has_field_derivative_cong_eventually)
  show  $\forall_F x \text{ in at } z \text{ within } S. g \ x \text{ powr of\_int } n = g \ x^{\wedge} \text{nat } n$ 
    unfolding eventually_at
    apply (rule exI[where  $x=e$ ])
    using powr_of_int that  $\langle e > 0 \rangle$   $e\_dist$  by (simp add: dist_commute)

```



```

qed (use powr_of_int ⟨g z ≠ 0⟩ that in simp)
also have ... unfolding dd'_def using gderiv that
  by (auto intro!: derivative_eq_intros)
finally have ((λz. g z powr_of_int n) has_field_derivative dd) (at z within S)
.
  then show ?thesis unfolding dd_def by simp
qed
moreover have ?thesis when n < 0
proof -
  define dd' where dd' = of_int n / g z ^ (nat (1 - n)) * gd
  have dd = dd'
  proof -
    have g z powr_of_int (n - 1) = inverse (g z ^ nat (1 - n))
      using powr_of_int[OF ⟨g z ≠ 0⟩, of n-1] that by auto
    then show ?thesis
      unfolding dd_def dd'_def by (simp add: divide_inverse)
  qed
  then have ((λz. g z powr_of_int n) has_field_derivative dd) (at z within S)
    ↔ ((λz. g z powr_of_int n) has_field_derivative dd') (at z within S)
  by simp
  also have ... ↔ ((λz. inverse (g z ^ nat (-n))) has_field_derivative dd') (at
z within S)
  proof (rule has_field_derivative_cong_eventually)
    show ∀ x in at z within S. g x powr_of_int n = inverse (g x ^ nat (- n))
      unfolding eventually_at
      apply (rule exI[where x=e])
      using powr_of_int that ⟨e > 0⟩ e_dist by (simp add: dist_commute)
  qed (use powr_of_int ⟨g z ≠ 0⟩ that in simp)
  also have ...
  proof -
    have nat (- n) + nat (1 - n) - Suc 0 = nat (- n) + nat (- n)
      by auto
    then show ?thesis
      unfolding dd'_def using gderiv that ⟨g z ≠ 0⟩
      by (auto intro!: derivative_eq_intros simp add: field_split_simps power_add[symmetric])
  qed
  finally have ((λz. g z powr_of_int n) has_field_derivative dd) (at z within S)
.
  then show ?thesis unfolding dd_def by simp
qed
ultimately show ?thesis by force
qed

lemma field_differentiable_powr_of_int:
  fixes z :: complex
  assumes gderiv: g field_differentiable (at z within S) and g z ≠ 0
  shows (λz. g z powr_of_int n) field_differentiable (at z within S)
using has_field_derivative_powr_of_int assms(2) field_differentiable_def gderiv
by blast

```

```

lemma holomorphic_on_pour_of_int [holomorphic_intros]:
  assumes holf:  $f$  holomorphic_on  $S$  and 0:  $\bigwedge z. z \in S \implies f z \neq 0$ 
  shows  $(\lambda z. (f z)^{\text{pour\_of\_int } n})$  holomorphic_on  $S$ 
proof (cases  $n \geq 0$ )
  case True
  then have ?thesis  $\longleftrightarrow (\lambda z. (f z)^{\wedge \text{nat } n})$  holomorphic_on  $S$ 
    by (metis (no_types, lifting) 0 holomorphic_cong power_of_int)
  moreover have  $(\lambda z. (f z)^{\wedge \text{nat } n})$  holomorphic_on  $S$ 
    using holf by (auto intro: holomorphic_intros)
  ultimately show ?thesis by auto
next
  case False
  then have ?thesis  $\longleftrightarrow (\lambda z. \text{inverse } (f z)^{\wedge \text{nat } (-n)})$  holomorphic_on  $S$ 
    by (metis (no_types, lifting) 0 holomorphic_cong power_inverse power_of_int)
  moreover have  $(\lambda z. \text{inverse } (f z)^{\wedge \text{nat } (-n)})$  holomorphic_on  $S$ 
    using assms by (auto intro!: holomorphic_intros)
  ultimately show ?thesis by auto
qed

lemma has_field_derivative_pour_right [derivative_intros]:
   $w \neq 0 \implies ((\lambda z. w^{\text{pour } z}) \text{ has\_field\_derivative } L n \ w * w^{\text{pour } z}) \ (at \ z)$ 
  unfolding pour_def by (intro derivative_eq_intros | simp)+

lemma field_differentiable_pour_right [derivative_intros]:
  fixes  $w :: \text{complex}$ 
  shows  $w \neq 0 \implies (\lambda z. w^{\text{pour } z})$  field_differentiable  $(at \ z)$ 
using field_differentiable_def has_field_derivative_pour_right by blast

lemma holomorphic_on_pour_right [holomorphic_intros]:
  assumes  $f$  holomorphic_on  $s$ 
  shows  $(\lambda z. w^{\text{pour } (f z)})$  holomorphic_on  $s$ 
proof (cases  $w = 0$ )
  case False
  with assms show ?thesis
    unfolding holomorphic_on_def field_differentiable_def
    by (metis (full_types) DERIV_chain' has_field_derivative_pour_right)
qed simp

lemma holomorphic_on_divide_gen [holomorphic_intros]:
  assumes  $f: f$  holomorphic_on  $s$  and  $g: g$  holomorphic_on  $s$  and 0:  $\bigwedge z \ z'. [z \in s; z' \in s] \implies g z = 0 \longleftrightarrow g z' = 0$ 
  shows  $(\lambda z. f z / g z)$  holomorphic_on  $s$ 
proof (cases  $\exists z \in s. g z = 0$ )
  case True
  with 0 have  $g z = 0$  if  $z \in s$  for  $z$ 
    using that by blast
  then show ?thesis
    using g holomorphic_transform by auto

```

```

next
  case False
  with 0 have  $g\ z \neq 0$  if  $z \in s$  for  $z$ 
    using that by blast
  with holomorphic_on_divide show ?thesis
    using f g by blast
qed

```

```

lemma norm_powr_real_powr:
   $w \in \mathbb{R} \implies 0 \leq \text{Re } w \implies \text{cmod } (w \text{ powr } z) = \text{Re } w \text{ powr } \text{Re } z$ 
  by (metis dual_order.order_iff_strict norm_powr_real norm_zero of_real_0
    of_real_Re powr_def)

```

```

lemma tendsto_powr_complex:
  fixes f g ::  $\_ \Rightarrow \text{complex}$ 
  assumes a:  $a \notin \mathbb{R}_{\leq 0}$ 
  assumes f:  $(f \longrightarrow a) F$  and g:  $(g \longrightarrow b) F$ 
  shows  $((\lambda z. f\ z \text{ powr } g\ z) \longrightarrow a \text{ powr } b) F$ 
proof -
  from a have [simp]:  $a \neq 0$  by auto
  from f g a have  $((\lambda z. \exp (g\ z * \ln (f\ z))) \longrightarrow a \text{ powr } b) F$  (is ?P)
    by (auto intro!: tendsto_intros simp: powr_def)
  also {
    have eventually  $(\lambda z. z \neq 0)$  (nhds a)
      by (intro t1_space_nhds) simp_all
    with f have eventually  $(\lambda z. f\ z \neq 0) F$  using filterlim_iff by blast
  }
  hence ?P  $\longleftrightarrow ((\lambda z. f\ z \text{ powr } g\ z) \longrightarrow a \text{ powr } b) F$ 
    by (intro tendsto_cong refl) (simp_all add: powr_def mult_ac)
  finally show ?thesis .
qed

```

```

lemma tendsto_powr_complex_0:
  fixes f g ::  $'a \Rightarrow \text{complex}$ 
  assumes f:  $(f \longrightarrow 0) F$  and g:  $(g \longrightarrow b) F$  and b:  $\text{Re } b > 0$ 
  shows  $((\lambda z. f\ z \text{ powr } g\ z) \longrightarrow 0) F$ 
proof (rule tendsto_norm_zero_cancel)
  define h where
     $h = (\lambda z. \text{if } f\ z = 0 \text{ then } 0 \text{ else } \exp (\text{Re } (g\ z) * \ln (\text{cmod } (f\ z)) + \text{abs } (\text{Im } (g\ z)) * \pi))$ 
  {
    fix z :: 'a assume z:  $f\ z \neq 0$ 
    define c where  $c = \text{abs } (\text{Im } (g\ z)) * \pi$ 
    from mpi_less_Im_Ln[OF z] Im_Ln_le_pi[OF z]
      have  $\text{abs } (\text{Im } (\ln (f\ z))) \leq \pi$  by simp
    from mult_left_mono[OF this, of  $\text{abs } (\text{Im } (g\ z))$ ]
      have  $\text{abs } (\text{Im } (g\ z) * \text{Im } (\ln (f\ z))) \leq c$  by (simp add: abs_mult c_def)
    hence  $-\text{Im } (g\ z) * \text{Im } (\ln (f\ z)) \leq c$  by simp
    hence  $\text{norm } (f\ z \text{ powr } g\ z) \leq h\ z$  by (simp add: powr_def field_simps h_def)
  }

```

```

c_def)
}
hence le: norm (f z powr g z) ≤ h z for z by (cases f x = 0) (simp_all add:
h_def)

have g': (g → b) (inf F (principal {z. f z ≠ 0}))
by (rule tendsto_mono[OF _ g]) simp_all
have ((λx. norm (f x)) → 0) (inf F (principal {z. f z ≠ 0}))
by (subst tendsto_norm_zero_iff, rule tendsto_mono[OF _ f]) simp_all
moreover {
  have filterlim (λx. norm (f x)) (principal {0<..}) (principal {z. f z ≠ 0})
  by (auto simp: filterlim_def)
  hence filterlim (λx. norm (f x)) (principal {0<..})
    (inf F (principal {z. f z ≠ 0}))
  by (rule filterlim_mono) simp_all
}
ultimately have norm: filterlim (λx. norm (f x)) (at_right 0) (inf F (principal
{z. f z ≠ 0}))
by (simp add: filterlim_inf at_within_def)

have A: LIM x inf F (principal {z. f z ≠ 0}). Re (g x) * -ln (cmod (f x)) :>
at_top
by (rule filterlim_tendsto_pos_mult_at_top tendsto_intros g' b
  filterlim_compose[OF filterlim_uminus_at_top_at_bot] filterlim_compose[OF
ln_at_0] norm)+
have B: LIM x inf F (principal {z. f z ≠ 0}).
  -|Im (g x)| * pi + -(Re (g x) * ln (cmod (f x))) :> at_top
by (rule filterlim_tendsto_add_at_top tendsto_intros g') + (insert A, simp_all)
have C: (h → 0) F unfolding h_def
by (intro filterlim_If tendsto_const filterlim_compose[OF exp_at_bot])
(insert B, auto simp: filterlim_uminus_at_bot algebra_simps)
show ((λx. norm (f x powr g x)) → 0) F
by (rule Lim_null_comparison[OF always_eventually C]) (insert le, auto)
qed

```

```

lemma tendsto_powr_complex' [tendsto_intros]:
  fixes f g :: _ ⇒ complex
  assumes a ∉ ℝ≤0 ∨ (a = 0 ∧ Re b > 0) and (f → a) F (g → b) F
  shows ((λz. f z powr g z) → a powr b) F
  using assms tendsto_powr_complex tendsto_powr_complex_0 by fastforce

```

```

lemma tendsto_neg_powr_complex_of_real:
  assumes filterlim f at_top F and Re s < 0
  shows ((λx. complex_of_real (f x) powr s) → 0) F
proof -
  have ((λx. norm (complex_of_real (f x) powr s)) → 0) F
proof (rule Lim_transform_eventually)
  from assms(1) have eventually (λx. f x ≥ 0) F
  by (auto simp: filterlim_at_top)

```

```

    thus eventually ( $\lambda x. f\ x\ \text{powr}\ Re\ s = \text{norm}\ (\text{of\_real}\ (f\ x)\ \text{powr}\ s)$ )  $F$ 
      by eventually_elim (simp add: norm_powr_real_powr)
    from assms show  $((\lambda x. f\ x\ \text{powr}\ Re\ s) \longrightarrow 0)\ F$ 
      by (intro tendsto_neg_powr)
  qed
  thus ?thesis by (simp add: tendsto_norm_zero_iff)
qed

lemma tendsto_neg_powr_complex_of_nat:
  assumes filterlim  $f$  at_top  $F$  and  $Re\ s < 0$ 
  shows  $((\lambda x. \text{of\_nat}\ (f\ x)\ \text{powr}\ s) \longrightarrow 0)\ F$ 
proof -
  have  $((\lambda x. \text{of\_real}\ (\text{real}\ (f\ x))\ \text{powr}\ s) \longrightarrow 0)\ F$  using assms(2)
    by (intro filterlim_compose[OF tendsto_neg_powr_complex_of_real]
        filterlim_compose[OF assms(1)] filterlim_real_sequentially filter-
        lim_ident) auto
  thus ?thesis by simp
qed

lemma continuous_powr_complex:
  assumes  $f\ (\text{netlimit}\ F) \notin \mathbb{R}_{\leq 0}$  continuous  $F\ f$  continuous  $F\ g$ 
  shows continuous  $F\ (\lambda z. f\ z\ \text{powr}\ g\ z :: \text{complex})$ 
  using assms unfolding continuous_def by (intro tendsto_powr_complex) simp_all

lemma isCont_powr_complex [continuous_intros]:
  assumes  $f\ z \notin \mathbb{R}_{\leq 0}$  isCont  $f\ z$  isCont  $g\ z$ 
  shows isCont  $(\lambda z. f\ z\ \text{powr}\ g\ z :: \text{complex})\ z$ 
  using assms unfolding isCont_def by (intro tendsto_powr_complex) simp_all

lemma continuous_on_powr_complex [continuous_intros]:
  assumes  $A \subseteq \{z. Re\ (f\ z) \geq 0 \vee Im\ (f\ z) \neq 0\}$ 
  assumes  $\bigwedge z. z \in A \implies f\ z = 0 \implies Re\ (g\ z) > 0$ 
  assumes continuous_on  $A\ f$  continuous_on  $A\ g$ 
  shows continuous_on  $A\ (\lambda z. f\ z\ \text{powr}\ g\ z)$ 
  unfolding continuous_on_def
proof
  fix  $z$  assume  $z: z \in A$ 
  show  $((\lambda z. f\ z\ \text{powr}\ g\ z) \longrightarrow f\ z\ \text{powr}\ g\ z)\ (\text{at}\ z\ \text{within}\ A)$ 
  proof (cases  $f\ z = 0$ )
    case False
    from assms(1,2)  $z$  have  $Re\ (f\ z) \geq 0 \vee Im\ (f\ z) \neq 0\ f\ z = 0 \longrightarrow Re\ (g\ z) > 0$ 
    0 by auto
    with assms(3,4)  $z$  show ?thesis
      by (intro tendsto_powr_complex')
      (auto elim!: nonpos_Reals_cases simp: complex_eq_iff continuous_on_def)
  next
    case True
    with assms  $z$  show ?thesis
      by (auto intro!: tendsto_powr_complex_0 simp: continuous_on_def)
  qed

```

qed
qed

6.21.18 Some Limits involving Logarithms

```

lemma lim_Ln_over_power:
  fixes s::complex
  assumes  $0 < \operatorname{Re} s$ 
  shows  $(\lambda n. \operatorname{Ln} (\operatorname{of\_nat} n) / \operatorname{of\_nat} n \operatorname{powr} s) \longrightarrow 0$ 
proof (simp add: lim_sequentially dist_norm, clarify)
  fix e::real
  assume e:  $0 < e$ 
  have  $\exists x_0 > 0. \forall x \geq x_0. 0 < e * 2 + (e * \operatorname{Re} s * 2 - 2) * x + e * (\operatorname{Re} s)^2 * x^2$ 
  proof (rule_tac x=2/(e * (Re s)2) in exI, safe)
    show  $0 < 2 / (e * (\operatorname{Re} s)^2)$ 
    using e assms by (simp add: field_simps)
  next
    fix x::real
    assume s:  $2 / (e * (\operatorname{Re} s)^2) \leq x$ 
    have  $2 / (e * (\operatorname{Re} s)^2) > 0$ 
    using e assms by simp
    with x have x > 0
    by linarith
    then have  $x * 2 \leq e * (x^2 * (\operatorname{Re} s)^2)$ 
    using e assms x by (auto simp: power2_eq_square field_simps)
    also have  $\dots < e * (2 + (x * (\operatorname{Re} s * 2) + x^2 * (\operatorname{Re} s)^2))$ 
    using e assms  $\langle x > 0 \rangle$ 
    by (auto simp: power2_eq_square field_simps add_pos_pos)
    finally show  $0 < e * 2 + (e * \operatorname{Re} s * 2 - 2) * x + e * (\operatorname{Re} s)^2 * x^2$ 
    by (auto simp: algebra_simps)
  qed
  then have  $\exists x_0 > 0. \forall x \geq x_0. x / e < 1 + (\operatorname{Re} s * x) + (1/2) * (\operatorname{Re} s * x)^2$ 
  using e by (simp add: field_simps)
  then have  $\exists x_0 > 0. \forall x \geq x_0. x / e < \exp (\operatorname{Re} s * x)$ 
  using assms
  by (force intro: less_le_trans [OF exp_lower_Taylor_quadratic])
  then obtain xo where xo > 0 and xo:  $\bigwedge x. x \geq x_0 \implies x < e * \exp (\operatorname{Re} s * x)$ 
  using e by (auto simp: field_simps)
  have norm (Ln (of_nat n) / of_nat n powr s) < e if n ≥ nat [exp xo] for n
  proof -
    have ln (real n) ≥ xo
    using that exp_gt_zero ln_ge_iff [of n] nat_ceiling_le_eq by fastforce
    then show ?thesis
    using e xo [of ln n] by (auto simp: norm_divide norm_powr_real field_split_simps)
  qed
  then show  $\exists n_0. \forall n \geq n_0. \operatorname{norm} (\operatorname{Ln} (\operatorname{of\_nat} n) / \operatorname{of\_nat} n \operatorname{powr} s) < e$ 
  by blast
qed

```

lemma *lim_Ln_over_n*: $((\lambda n. \text{Ln}(\text{of\_nat } n) / \text{of\_nat } n) \longrightarrow 0)$ *sequentially*
using *lim_Ln_over_power* [of 1] **by** *simp*

lemma *lim_ln_over_power*:
fixes $s :: \text{real}$
assumes $0 < s$
shows $((\lambda n. \ln n / (n \text{ powr } s)) \longrightarrow 0)$ *sequentially*
proof –
have $(\lambda n. \ln (\text{Suc } n) / (\text{Suc } n \text{ powr } s) \longrightarrow 0$
using *lim_Ln_over_power* [of of_real s, THEN *filterlim_sequentially_Suc*
[*THEN iffD2*]] *assms*
by (*simp add: lim_sequentially_dist_norm Ln_Reals_eq norm_powr_real_powr*
norm_divide)
then show ?thesis
using *filterlim_sequentially_Suc* [of $\lambda n::\text{nat}. \ln n / n \text{ powr } s$] **by** *auto*
qed

lemma *lim_ln_over_n [tendsto_intros]*: $((\lambda n. \ln(\text{real\_of\_nat } n) / \text{of\_nat } n) \longrightarrow 0)$ *sequentially*
using *lim_ln_over_power* [of 1] **by** *auto*

lemma *lim_log_over_n [tendsto_intros]*:
 $(\lambda n. \log k \, n / n) \longrightarrow 0$
proof –
have $*$: $\log k \, n / n = (1 / \ln k) * (\ln n / n)$ **for** n
unfolding *log_def* **by** *auto*
have $(\lambda n. (1 / \ln k) * (\ln n / n)) \longrightarrow (1 / \ln k) * 0$
by (*intro tendsto_intros*)
then show ?thesis
unfolding $*$ **by** *auto*
qed

lemma *lim_1_over_complex_power*:
assumes $0 < \text{Re } s$
shows $(\lambda n. 1 / \text{of\_nat } n \text{ powr } s) \longrightarrow 0$
proof (*rule Lim_null_comparison*)
have $\forall n > 0. \exists \leq n \longrightarrow 1 \leq \ln (\text{real\_of\_nat } n)$
using *ln_272_gt_1*
by (*force intro: order_trans [of _ $\ln (272/100)$]*)
then show $\forall_F x \text{ in sequentially. } \text{cmod } (1 / \text{of\_nat } x \text{ powr } s) \leq \text{cmod } (\text{Ln } (\text{of\_nat } x) / \text{of\_nat } x \text{ powr } s)$
by (*auto simp: norm_divide field_split_simps eventually_sequentially*)
show $(\lambda n. \text{cmod } (\text{Ln } (\text{of\_nat } n) / \text{of\_nat } n \text{ powr } s)) \longrightarrow 0$
using *lim_Ln_over_power* [*OF assms*] **by** (*metis tendsto_norm_zero_iff*)
qed

lemma *lim_1_over_real_power*:
fixes $s :: \text{real}$
assumes $0 < s$

```

shows ((λn. 1 / (of_nat n powr s)) → 0) sequentially
using lim_1_over_complex_power [of_of_real s, THEN filterlim_sequentially_Suc
[THEN iffD2]] assms
apply (subst filterlim_sequentially_Suc [symmetric])
by (simp add: lim_sequentially dist_norm Ln_Reals_eq norm_powr_real_powr
norm_divide)

```

```

lemma lim_1_over_Ln: ((λn. 1 / Ln(of_nat n)) → 0) sequentially
proof (clarsimp simp add: lim_sequentially dist_norm norm_divide field_split_simps)
fix r::real
assume 0 < r
have ir: inverse (exp (inverse r)) > 0
by simp
obtain n where n: 1 < of_nat n * inverse (exp (inverse r))
using ex_less_of_nat_mult [of_ 1, OF ir]
by auto
then have exp (inverse r) < of_nat n
by (simp add: field_split_simps)
then have ln (exp (inverse r)) < ln (of_nat n)
by (metis exp_gt_zero less_trans ln_exp ln_less_cancel_iff)
with ‹0 < r› have 1 < r * ln (real_of_nat n)
by (simp add: field_simps)
moreover have n > 0 using n
using neq0_conv by fastforce
ultimately show ∃ no. ∀ k. Ln (of_nat k) ≠ 0 → no ≤ k → 1 < r * cmod
(Ln (of_nat k))
using n ‹0 < r›
by (rule_tac x=n in exI) (force simp: field_split_simps intro: less_le_trans)
qed

```

```

lemma lim_1_over_ln: ((λn. 1 / ln(real_of_nat n)) → 0) sequentially
using lim_1_over_Ln [THEN filterlim_sequentially_Suc [THEN iffD2]]
apply (subst filterlim_sequentially_Suc [symmetric])
by (simp add: lim_sequentially dist_norm Ln_Reals_eq norm_powr_real_powr
norm_divide)

```

```

lemma lim_ln1_over_ln: (λn. ln(Suc n) / ln n) → 1
proof (rule Lim_transform_eventually)
have (λn. ln(1 + 1/n) / ln n) → 0
proof (rule Lim_transform_bound)
show (inverse o real) → 0
by (metis comp_def lim_inverse_n lim_explicit)
show ∀_F n in sequentially. norm (ln (1 + 1 / n) / ln n) ≤ norm ((inverse o
real) n)
proof
fix n::nat
assume n: 3 ≤ n
then have ln 3 ≤ ln n and ln0: 0 ≤ ln n
by auto

```



```

with ln3_gt_1 have 1 / ln n ≤ 1
  by (simp add: field_split_simps)
moreover have ln (1 + 1 / real n) ≤ 1/n
  by (simp add: ln_add_one_self_le_self)
ultimately have ln (1 + 1 / real n) * (1 / ln n) ≤ (1/n) * 1
  by (intro mult_mono) (use n in auto)
then show norm (ln (1 + 1 / n) / ln n) ≤ norm ((inverse ∘ real) n)
  by (simp add: field_simps ln0)
qed
qed
then show (λn. 1 + ln(1 + 1/n) / ln n) → 1
  by (metis (full_types) add.right_neutral tendsto_add_const_iff)
show ∀_F k in sequentially. 1 + ln (1 + 1 / k) / ln k = ln(Suc k) / ln k
  by (simp add: field_split_simps ln_div eventually_sequentiallyI [of 2])
qed

```

```

lemma lim_ln_over_ln1: (λn. ln n / ln(Suc n)) → 1
proof -
  have (λn. inverse (ln(Suc n) / ln n)) → inverse 1
    by (rule tendsto_inverse [OF lim_ln1_over_ln]) auto
  then show ?thesis
    by simp
qed

```

6.21.19 Relation between Square Root and exp/ln, hence its derivative

```

lemma csqrt_exp_Ln:
  assumes z ≠ 0
  shows csqrt z = exp(Ln(z) / 2)
proof -
  have (exp (Ln z / 2))^2 = (exp (Ln z))
    by (metis exp_double nonzero_mult_div_cancel_left times_divide_eq_right
      zero_neq_numeral)
  also have ... = z
    using assms exp_Ln by blast
  finally have csqrt z = csqrt ((exp (Ln z / 2))^2)
    by simp
  also have ... = exp (Ln z / 2)
    apply (rule csqrt_square)
    using cos_gt_zero_pi [of (Im (Ln z) / 2)] Im_Ln_le_pi mpi_less_Im_Ln
    assms
    by (fastforce simp: Re_exp Im_exp )
  finally show ?thesis using assms csqrt_square
    by simp
qed

```

```

lemma csqrt_inverse:
  assumes z ∉ ℝ≤0

```

```

    shows  $\text{csqrt } (\text{inverse } z) = \text{inverse } (\text{csqrt } z)$ 
  proof (cases  $z=0$ )
    case False
    then show ?thesis
      using assms  $\text{csqrt\_exp\_Ln } \text{Ln\_inverse } \text{exp\_minus}$ 
      by (simp add:  $\text{csqrt\_exp\_Ln } \text{Ln\_inverse } \text{exp\_minus}$ )
  qed auto

```

```

lemma  $\text{cnj\_csqrt}$ :
  assumes  $z \notin \mathbb{R}_{\leq 0}$ 
  shows  $\text{cnj}(\text{csqrt } z) = \text{csqrt}(\text{cnj } z)$ 
  proof (cases  $z=0$ )
    case False
    then show ?thesis
      by (simp add: assms  $\text{cnj\_Ln } \text{csqrt\_exp\_Ln } \text{exp\_cnj}$ )
  qed auto

```

```

lemma  $\text{has\_field\_derivative\_csqrt}$ :
  assumes  $z \notin \mathbb{R}_{\leq 0}$ 
  shows  $(\text{csqrt } \text{has\_field\_derivative } \text{inverse}(2 * \text{csqrt } z)) \text{ (at } z)$ 
  proof -
    have  $z: z \neq 0$ 
    using assms by auto
    then have *:  $\text{inverse } z = \text{inverse } (2 * z) * 2$ 
    by (simp add:  $\text{field\_split\_simps}$ )
    have [simp]:  $\text{exp } (\text{Ln } z / 2) * \text{inverse } z = \text{inverse } (\text{csqrt } z)$ 
    by (simp add:  $z \text{ field\_simps } \text{csqrt\_exp\_Ln } [\text{symmetric}]$ ) (metis  $\text{power2\_csqrt}$ 
     $\text{power2\_eq\_square}$ )
    have  $\text{Im } z = 0 \implies 0 < \text{Re } z$ 
    using assms  $\text{complex\_nonpos\_Reals\_iff\_not\_less}$  by blast
    with  $z$  have  $((\lambda z. \text{exp } (\text{Ln } z / 2)) \text{ has\_field\_derivative } \text{inverse } (2 * \text{csqrt } z))$ 
    (at  $z$ )
    by (force intro:  $\text{derivative\_eq\_intros} * \text{simp add: assms}$ )
    then show ?thesis
    proof (rule  $\text{has\_field\_derivative\_transform\_within}$ )
      show  $\bigwedge x. \text{dist } x \ z < cmod \ z \implies \text{exp } (\text{Ln } x / 2) = \text{csqrt } x$ 
      by (metis  $\text{csqrt\_exp\_Ln } \text{dist\_0\_norm } \text{less\_irrefl}$ )
    qed (use  $z$  in auto)
  qed

```

```

lemma  $\text{field\_differentiable\_at\_csqrt}$ :
   $z \notin \mathbb{R}_{\leq 0} \implies \text{csqrt } \text{field\_differentiable at } z$ 
  using  $\text{field\_differentiable\_def } \text{has\_field\_derivative\_csqrt}$  by blast

```

```

lemma  $\text{field\_differentiable\_within\_csqrt}$ :
   $z \notin \mathbb{R}_{\leq 0} \implies \text{csqrt } \text{field\_differentiable (at } z \text{ within } s)$ 
  using  $\text{field\_differentiable\_at\_csqrt } \text{field\_differentiable\_within\_subset}$  by blast

```

```

lemma  $\text{continuous\_at\_csqrt}$ :

```

```

  z  $\notin \mathbb{R}_{\leq 0} \implies \text{continuous (at } z) \text{ csqrt}$ 
  by (simp add: field_differentiable_within_csqrt field_differentiable_imp_continuous_at)

corollary isCont_csqrt' [simp]:
   $\llbracket \text{isCont } f \text{ } z; f \text{ } z \notin \mathbb{R}_{\leq 0} \rrbracket \implies \text{isCont } (\lambda x. \text{csqrt } (f \text{ } x)) \text{ } z$ 
  by (blast intro: isCont_o2 [OF continuous_at_csqrt])

lemma continuous_within_csqrt:
  z  $\notin \mathbb{R}_{\leq 0} \implies \text{continuous (at } z \text{ within } s) \text{ csqrt}$ 
  by (simp add: field_differentiable_imp_continuous_at field_differentiable_within_csqrt)

lemma continuous_on_csqrt [continuous_intros]:
   $(\bigwedge z. z \in s \implies z \notin \mathbb{R}_{\leq 0}) \implies \text{continuous\_on } s \text{ csqrt}$ 
  by (simp add: continuous_at_imp_continuous_on continuous_within_csqrt)

lemma holomorphic_on_csqrt:
   $(\bigwedge z. z \in s \implies z \notin \mathbb{R}_{\leq 0}) \implies \text{csqrt holomorphic\_on } s$ 
  by (simp add: field_differentiable_within_csqrt holomorphic_on_def)

lemma continuous_within_closed_nontrivial:
  closed s  $\implies a \notin s \implies \text{continuous (at } a \text{ within } s) f$ 
  using open_Compl
  by (force simp add: continuous_def eventually_at_topological filterlim_iff open_Collect_neg)

lemma continuous_within_csqrt_posreal:
  continuous (at z within  $(\mathbb{R} \cap \{w. 0 \leq \text{Re}(w)\})$ ) csqrt
proof (cases z  $\in \mathbb{R}_{\leq 0}$ )
  case True
  have [simp]:  $\text{Im } z = 0$  and  $0: \text{Re } z < 0 \vee z = 0$ 
  using True cnj.code complex_cnj_zero_iff by (auto simp: Complex_eq complex_nonpos_Reals_iff) fastforce
  show ?thesis
  using 0
  proof
  assume  $\text{Re } z < 0$ 
  then show ?thesis
  by (auto simp: continuous_within_closed_nontrivial [OF closed_Real_halfspace_Re_ge])
  next
  assume  $z = 0$ 
  moreover
  have  $\bigwedge e. 0 < e$ 
   $\implies \forall x' \in \mathbb{R} \cap \{w. 0 \leq \text{Re } w\}. \text{cmod } x' < e^2 \longrightarrow \text{cmod } (\text{csqrt } x') < e$ 
  by (auto simp: Reals_def real_less_sqrt)
  ultimately show ?thesis
  using zero_less_power by (fastforce simp: continuous_within_eps_delta)
  qed
qed (blast intro: continuous_within_csqrt)

```

6.21.20 Complex arctangent

The branch cut gives standard bounds in the real case.

definition *Arctan* :: *complex* $\Rightarrow$ *complex* **where**

$$\text{Arctan} \equiv \lambda z. (i/2) * \text{Ln}((1 - i*z) / (1 + i*z))$$

lemma *Arctan_def_moebius*: *Arctan* *z* = $i/2 * \text{Ln}(\text{moebius } (-i) \ 1 \ i \ 1 \ z)$
by (*simp add: Arctan_def moebius_def add_ac*)

lemma *Ln_conv_Arctan*:

assumes $z \neq -1$

shows $\text{Ln } z = -2*i * \text{Arctan}(\text{moebius } 1 \ (-1) \ (-i) \ (-i) \ z)$

proof –

have $\text{Arctan}(\text{moebius } 1 \ (-1) \ (-i) \ (-i) \ z) =$

$$i/2 * \text{Ln}(\text{moebius } (-i) \ 1 \ i \ 1 \ (\text{moebius } 1 \ (-1) \ (-i) \ (-i) \ z))$$

by (*simp add: Arctan_def moebius*)

also from *assms* **have** $i * z \neq i * (-1)$ **by** (*subst mult_left_cancel*) *simp*

hence $i * z - -i \neq 0$ **by** (*simp add: eq_neg_iff_add_eq_0*)

from *moebius_inverse'[OF_ this, of 1 1]*

have $\text{moebius } (-i) \ 1 \ i \ 1 \ (\text{moebius } 1 \ (-1) \ (-i) \ (-i) \ z) = z$ **by** *simp*

finally show *?thesis* **by** (*simp add: field_simps*)

qed

lemma *Arctan_0* [*simp*]: *Arctan* 0 = 0

by (*simp add: Arctan_def*)

lemma *Im_complex_div_lemma*: $\text{Im}((1 - i*z) / (1 + i*z)) = 0 \longleftrightarrow \text{Re } z = 0$

by (*auto simp: Im_complex_div_eq_0 algebra_simps*)

lemma *Re_complex_div_lemma*: $0 < \text{Re}((1 - i*z) / (1 + i*z)) \longleftrightarrow \text{norm } z < 1$

by (*simp add: Re_complex_div_gt_0 algebra_simps cmod_def power2_eq_square*)

lemma *tan_Arctan*:

assumes $z^2 \neq -1$

shows [*simp*]: $\tan(\text{Arctan } z) = z$

proof –

have $1 + i*z \neq 0$

by (*metis assms complex_i_mult_minus_i_squared minus_unique power2_eq_square power2_minus*)

moreover

have $1 - i*z \neq 0$

by (*metis assms complex_i_mult_minus_i_squared power2_eq_square power2_minus right_minus_eq*)

ultimately

show *?thesis*

by (*simp add: Arctan_def tan_def sin_exp_eq cos_exp_eq exp_minus csqrt_exp_Ln [symmetric]*)

divide_simps power2_eq_square [symmetric])

qed

lemma *Arctan_tan [simp]*:

assumes $|Re\ z| < \pi/2$

shows $Arctan(\tan z) = z$

proof -

have $Ln((1 - i * \tan z) / (1 + i * \tan z)) = 2 * z / i$

proof (rule *Ln_unique*)

have $ge\_pi2: \bigwedge n::int. |of\_int(2*n + 1) * \pi/2| \geq \pi/2$

by (case_tac *n* rule: *int_cases*) (auto simp: *abs_mult*)

have $\exp(i*z)*\exp(i*z) = -1 \longleftrightarrow \exp(2*i*z) = -1$

by (metis *distrib_right exp_add mult_2*)

also have $\dots \longleftrightarrow \exp(2*i*z) = \exp(i*\pi)$

using *cis_conv_exp cis_pi* by auto

also have $\dots \longleftrightarrow \exp(2*i*z - i*\pi) = 1$

by (metis (no_types) *diff_add_cancel diff_minus_eq_add exp_add exp_minus_inverse mult.commute*)

also have $\dots \longleftrightarrow Re(i*2*z - i*\pi) = 0 \wedge (\exists n::int. Im(i*2*z - i*\pi) = of\_int(2 * n) * \pi)$

by (simp add: *exp_eq_1*)

also have $\dots \longleftrightarrow Im\ z = 0 \wedge (\exists n::int. 2 * Re\ z = of\_int(2*n + 1) * \pi)$

by (simp add: *algebra_simps*)

also have $\dots \longleftrightarrow False$

using *assms ge_pi2*

apply (auto simp: *algebra_simps*)

by (metis *abs_mult_pos not_less of_nat_less_0_iff of_nat_numeral*)

finally have $\exp(i*z)*\exp(i*z) + 1 \neq 0$

by (auto simp: *add.commute minus_unique*)

then show $\exp(2 * z / i) = (1 - i * \tan z) / (1 + i * \tan z)$

apply (simp add: *tan_def sin_exp_eq cos_exp_eq exp_minus_divide_simps*)

by (simp add: *algebra_simps flip: power2_eq_square exp_double*)

qed (use *assms* in auto)

then show ?thesis

by (auto simp: *Arctan_def*)

qed

lemma

assumes $Re\ z = 0 \implies |Im\ z| < 1$

shows $Re\_Arctan\_bounds: |Re(Arctan\ z)| < \pi/2$

and $has\_field\_derivative\_Arctan: (Arctan\ has\_field\_derivative\ inverse(1 + z^2))\ (at\ z)$

proof -

have $nz0: 1 + i*z \neq 0$

using *assms*

by (metis *abs_one add_diff_cancel_left' complex_i_mult_minus diff_0 i_squared imaginary_unit_simps*)

less_asym neg_equal_iff_equal)

have $z \neq -i$ using *assms*

by auto

```

then have zz:  $1 + z * z \neq 0$ 
  by (metis abs_one assms i_squared imaginary_unit.simps less_irrefl minus_unique square_eq_iff)
have nz1:  $1 - i*z \neq 0$ 
  using assms by (force simp add: i_times_eq_iff)
have nz2:  $\text{inverse}(1 + i*z) \neq 0$ 
  using assms
  by (metis Im_complex_div_lemma Re_complex_div_lemma cmod_eq_Im divide_complex_def
    less_irrefl mult_zero_right zero_complex.simps(1) zero_complex.simps(2))
have nzi:  $((1 - i*z) * \text{inverse}(1 + i*z)) \neq 0$ 
  using nz1 nz2 by auto
have  $\text{Im}((1 - i*z) / (1 + i*z)) = 0 \implies 0 < \text{Re}((1 - i*z) / (1 + i*z))$ 
  apply (simp add: divide_complex_def)
  apply (simp add: divide_simps split: if_split_asm)
  using assms
  apply (auto simp: algebra_simps abs_square_less_1 [unfolded power2_eq_square])
  done
then have *:  $((1 - i*z) / (1 + i*z)) \notin \mathbb{R}_{\leq 0}$ 
  by (auto simp add: complex_nonpos_Reals_iff)
show  $|\text{Re}(\text{Arctan } z)| < \pi/2$ 
  unfolding Arctan_def divide_complex_def
  using mpi_less_Im_Ln [OF nzi]
  by (auto simp: abs_if intro!: Im_Ln_less_pi * [unfolded divide_complex_def])
show  $(\text{Arctan } \text{has\_field\_derivative } \text{inverse}(1 + z^2)) \text{ (at } z)$ 
  unfolding Arctan_def scaleR_conv_of_real
  apply (intro derivative_eq_intros | simp add: nz0 *)+
  using nz1 zz
  apply (simp add: field_split_simps power2_eq_square)
  apply algebra
  done
qed

```

```

lemma field_differentiable_at_Arctan:  $(\text{Re } z = 0 \implies |\text{Im } z| < 1) \implies \text{Arctan}$ 
  field_differentiable at  $z$ 
  using has_field_derivative_Arctan
  by (auto simp: field_differentiable_def)

```

```

lemma field_differentiable_within_Arctan:
   $(\text{Re } z = 0 \implies |\text{Im } z| < 1) \implies \text{Arctan field\_differentiable (at } z \text{ within } s)$ 
  using field_differentiable_at_Arctan field_differentiable_at_within by blast

```

```

declare has_field_derivative_Arctan [derivative_intros]
declare has_field_derivative_Arctan [THEN DERIV_chain2, derivative_intros]

```

```

lemma continuous_at_Arctan:
   $(\text{Re } z = 0 \implies |\text{Im } z| < 1) \implies \text{continuous (at } z) \text{ Arctan}$ 
  by (simp add: field_differentiable_imp_continuous_at field_differentiable_within_Arctan)

```

lemma *continuous_within_Arctan*:

($\text{Re } z = 0 \implies |\text{Im } z| < 1$) $\implies$ *continuous* (at z within s) *Arctan*
using *continuous_at_Arctan continuous_at_imp_continuous_within* **by** *blast*

lemma *continuous_on_Arctan* [*continuous_intros*]:

($\bigwedge z. z \in s \implies \text{Re } z = 0 \implies |\text{Im } z| < 1$) $\implies$ *continuous_on* s *Arctan*
by (*auto simp: continuous_at_imp_continuous_on continuous_within_Arctan*)

lemma *holomorphic_on_Arctan*:

($\bigwedge z. z \in s \implies \text{Re } z = 0 \implies |\text{Im } z| < 1$) $\implies$ *Arctan* *holomorphic_on* s
by (*simp add: field_differentiable_within_Arctan holomorphic_on_def*)

theorem *Arctan_series*:

assumes $z: \text{norm } (z :: \text{complex}) < 1$
defines $g \equiv \lambda n. \text{if odd } n \text{ then } -i * i^n / n \text{ else } 0$
defines $h \equiv \lambda z n. (-1)^n / \text{of\_nat } (2*n+1) * (z :: \text{complex})^{2*n+1}$
shows $(\lambda n. g \ n * z^n)$ *sums* *Arctan* z
and $h \ z$ *sums* *Arctan* z

proof –

define G **where** [*abs_def*]: $G \ z = (\sum n. g \ n * z^n)$ **for** z

have *summable*: *summable* $(\lambda n. g \ n * u^n)$ **if** $\text{norm } u < 1$ **for** u

proof (*cases* $u = 0$)

assume $u: u \neq 0$

have $(\lambda n. \text{ereal } (\text{norm } (h \ u \ n) / \text{norm } (h \ u \ (\text{Suc } n)))) = (\lambda n. \text{ereal } (\text{inverse } (\text{norm } u)^2) * \text{ereal } ((2 + \text{inverse } (\text{real } (\text{Suc } n))) / (2 - \text{inverse } (\text{real } (\text{Suc } n)))))$

proof

fix n

have $\text{ereal } (\text{norm } (h \ u \ n) / \text{norm } (h \ u \ (\text{Suc } n))) = \text{ereal } (\text{inverse } (\text{norm } u)^2) * \text{ereal } (((2 * \text{Suc } n + 1) / (\text{Suc } n)) / ((2 * \text{Suc } n - 1) / (\text{Suc } n)))$

by (*simp add: h_def norm_mult norm_power norm_divide field_split_simps power2_eq_square eval_nat_numeral del: of_nat_add of_nat_Suc*)

also have $\text{of\_nat } (2 * \text{Suc } n + 1) / \text{of\_nat } (\text{Suc } n) = (2 :: \text{real}) + \text{inverse } (\text{real } (\text{Suc } n))$

by (*auto simp: field_split_simps simp del: of_nat_Suc simp_all?*)

also have $\text{of\_nat } (2 * \text{Suc } n - 1) / \text{of\_nat } (\text{Suc } n) = (2 :: \text{real}) - \text{inverse } (\text{real } (\text{Suc } n))$

by (*auto simp: field_split_simps simp del: of_nat_Suc simp_all?*)

finally show $\text{ereal } (\text{norm } (h \ u \ n) / \text{norm } (h \ u \ (\text{Suc } n))) = \text{ereal } (\text{inverse } (\text{norm } u)^2) * \text{ereal } ((2 + \text{inverse } (\text{real } (\text{Suc } n))) / (2 - \text{inverse } (\text{real } (\text{Suc } n))))$

qed

also have $\dots \longrightarrow \text{ereal } (\text{inverse } (\text{norm } u)^2) * \text{ereal } ((2 + 0) / (2 - 0))$

by (*intro tendsto_intros LIMSEQ_inverse_real_of_nat simp_all*)

finally have $\liminf (\lambda n. \text{ereal } (\text{cmod } (h \ u \ n) / \text{cmod } (h \ u \ (\text{Suc } n)))) = \text{inverse } (\text{norm } u)^2$

by (*intro lim_imp_Liminf simp_all*)

moreover from *power_strict_mono* [*OF that, of 2*] u **have** $\text{inverse } (\text{norm } u)^2$

```

> 1
  by (simp add: field_split_simps)
  ultimately have A: liminf (λn. ereal (cmod (h u n) / cmod (h u (Suc n))))
> 1 by simp
  from u have summable (h u)
  by (intro summable_norm_cancel[OF ratio_test_convergence[OF _ A]])
  (auto simp: h_def norm_divide norm_mult norm_power simp del: of_nat_Suc
    intro!: mult_pos_pos divide_pos_pos always_eventually)
  thus summable (λn. g n * u^n)
  by (subst summable_mono_reindex[of λn. 2*n+1, symmetric])
  (auto simp: power_mult strict_mono_def g_def h_def elim!: oddE)
qed (simp add: h_def)

have ∃ c. ∀ u ∈ ball 0 1. Arctan u - G u = c
proof (rule has_field_derivative_zero_constant)
  fix u :: complex assume u ∈ ball 0 1
  hence u: norm u < 1 by (simp)
  define K where K = (norm u + 1) / 2
  from u and abs_Im_le_cmod[of u] have Im_u: |Im u| < 1 by linarith
  from u have K: 0 ≤ K norm u < K K < 1 by (simp_all add: K_def)
  hence (G has_field_derivative (∑ n. diffs g n * u^n)) (at u) unfolding
G_def
  by (intro termdiffs_strong[of _ of_real K] summable) simp_all
  also have (λn. diffs g n * u^n) = (λn. if even n then (i*u)^n else 0)
  by (intro ext) (simp_all del: of_nat_Suc add: g_def diffs_def power_mult_distrib)
  also have suminf ... = (∑ n. -(u^2)^n)
  by (subst suminf_mono_reindex[of λn. 2*n, symmetric])
  (auto elim!: evenE simp: strict_mono_def power_mult power_mult_distrib)
  also from u have norm u^2 < 1^2 by (intro power_strict_mono) simp_all
  hence (∑ n. -(u^2)^n) = inverse (1 + u^2)
  by (subst suminf_geometric) (simp_all add: norm_power inverse_eq_divide)
  finally have (G has_field_derivative inverse (1 + u^2)) (at u) .
  from DERIV_diff[OF has_field_derivative_Arctan this] Im_u u
  show ((λu. Arctan u - G u) has_field_derivative 0) (at u within ball 0 1)
  by (simp_all add: at_within_open[OF _ open_ball])
qed simp_all
then obtain c where c: ∧ u. norm u < 1 ⇒ Arctan u - G u = c by auto
from this[of 0] have c = 0 by (simp add: G_def g_def)
with c z have Arctan z = G z by simp
with summable[OF z] show (λn. g n * z^n) sums Arctan z unfolding G_def
by (simp add: sums_iff)
  thus h z sums Arctan z by (subst (asm) sums_mono_reindex[of λn. 2*n+1,
symmetric])
  (auto elim!: oddE simp: strict_mono_def power_mult
g_def h_def)
qed

```

A quickly-converging series for the logarithm, based on the arctangent.

theorem *ln_series_quadratic*:


```

  assumes  $x: x > (0::real)$ 
  shows  $(\lambda n. (2*((x - 1) / (x + 1))) ^ (2*n+1) / of\_nat (2*n+1))) \text{ sums } \ln x$ 
proof -
  define  $y :: complex$  where  $y = of\_real ((x-1)/(x+1))$ 
  from  $x$  have  $x': complex\_of\_real\ x \neq of\_real\ (-1)$  by (subst of\_real\_eq\_iff)
  auto
  from  $x$  have  $|x - 1| < |x + 1|$  by linarith
  hence  $norm\ (complex\_of\_real\ (x - 1) / complex\_of\_real\ (x + 1)) < 1$ 
    by (simp add: norm\_divide del: of\_real\_add of\_real\_diff)
  hence  $norm\ (i * y) < 1$  unfolding  $y\_def$  by (subst norm\_mult) simp
  hence  $(\lambda n. (-2*i) * ((-1)^n / of\_nat (2*n+1) * (i*y)^(2*n+1))) \text{ sums }$ 
 $((-2*i) * Arctan\ (i*y))$ 
    by (intro Arctan\_series sums\_mult) simp\_all
  also have  $(\lambda n. (-2*i) * ((-1)^n / of\_nat (2*n+1) * (i*y)^(2*n+1))) =$ 
 $(\lambda n. (-2*i) * ((-1)^n * (i*y*(-y^2)^n) / of\_nat (2*n+1)))$ 
    by (intro ext) (simp\_all add: power\_mult power\_mult\_distrib)
  also have  $\dots = (\lambda n. 2*y * ((-1) * (-y^2))^n / of\_nat (2*n+1))$ 
    by (intro ext, subst power\_mult\_distrib) (simp add: algebra\_simps power\_mult)
  also have  $\dots = (\lambda n. 2*y^(2*n+1) / of\_nat (2*n+1))$ 
    by (subst power\_add, subst power\_mult) (simp add: mult\_ac)
  also have  $\dots = (\lambda n. of\_real\ (2*((x-1)/(x+1)))^(2*n+1) / of\_nat (2*n+1))$ 
    by (intro ext) (simp add:  $y\_def$ )
  also have  $i * y = (of\_real\ x - 1) / (-i * (of\_real\ x + 1))$ 
    by (subst divide\_divide\_eq\_left [symmetric]) (simp add:  $y\_def$ )
  also have  $\dots = moebius\ 1\ (-1)\ (-i)\ (-i)\ (of\_real\ x)$  by (simp add: moebius\_def
algebra\_simps)
  also from  $x'$  have  $-2*i*Arctan\ \dots = Ln\ (of\_real\ x)$  by (intro Ln\_conv\_Arctan
[symmetric]) simp\_all
  also from  $x$  have  $\dots = \ln x$  by (rule Ln\_of\_real)
  finally show ?thesis by (subst (asm) sums\_of\_real\_iff)
qed

```

6.21.21 Real arctangent

lemma $Im\_Arctan\_of\_real$ [simp]: $Im\ (Arctan\ (of\_real\ x)) = 0$

proof -

```

  have  $ne: 1 + x^2 \neq 0$ 
    by (metis power\_one sum\_power2\_eq\_zero\_iff zero\_neq\_one)
  have  $ne1: 1 + i * complex\_of\_real\ x \neq 0$ 
    using Complex\_eq complex\_eq\_cancel\_iff2 by fastforce
  have  $Re\ (Ln\ ((1 - i * x) * inverse\ (1 + i * x))) = 0$ 
    apply (rule norm\_exp\_imaginary)
    using ne
    apply (simp add: ne1 cmod\_def)
    apply (auto simp: field\_split\_simps)
    apply algebra
  done
  then show ?thesis
    unfolding Arctan\_def divide\_complex\_def by (simp add: complex\_eq\_iff)

```

2374

qed

lemma *arctan_eq_Re_Arctan*: $\arctan x = \text{Re} (\text{Arctan} (\text{of\_real } x))$

proof (*rule arctan_unique*)

have $(1 - i * x) / (1 + i * x) \notin \mathbb{R}_{\leq 0}$

by (*auto simp: Im_complex_div lemma complex_nonpos_Reals_iff*)

then show $-(\pi / 2) < \text{Re} (\text{Arctan} (\text{complex\_of\_real } x))$

by (*simp add: Arctan_def Im_Ln_less_pi*)

next

have $*$: $(1 - i*x) / (1 + i*x) \neq 0$

by (*simp add: field_split_simps*) (*simp add: complex_eq_iff*)

show $\text{Re} (\text{Arctan} (\text{complex\_of\_real } x)) < \pi / 2$

using *mpi_less_Im_Ln [OF *]*

by (*simp add: Arctan_def*)

next

have $\tan (\text{Re} (\text{Arctan} (\text{of\_real } x))) = \text{Re} (\tan (\text{Arctan} (\text{of\_real } x)))$

by (*auto simp: tan_def Complex.Re_divide Re_sin Re_cos Im_sin Im_cos field_simps power2_eq_square*)

also have $\dots = x$

proof $-$

have $(\text{complex\_of\_real } x)^2 \neq -1$

by (*metis diff_0_right minus_diff_eq mult_zero_left not_le of_real_1 of_real_eq_iff of_real_minus of_real_power power2_eq_square real_minus_mult_self_le zero_less_one*)

then show *?thesis*

by *simp*

qed

finally show $\tan (\text{Re} (\text{Arctan} (\text{complex\_of\_real } x))) = x$.

qed

lemma *Arctan_of_real*: $\text{Arctan} (\text{of\_real } x) = \text{of\_real} (\arctan x)$

unfolding *arctan_eq_Re_Arctan divide_complex_def*

by (*simp add: complex_eq_iff*)

lemma *Arctan_in_Reals [simp]*: $z \in \mathbb{R} \implies \text{Arctan } z \in \mathbb{R}$

by (*metis Reals_cases Reals_of_real Arctan_of_real*)

declare *arctan_one [simp]*

lemma *arctan_less_pi4_pos*: $x < 1 \implies \arctan x < \pi/4$

by (*metis arctan_less_iff arctan_one*)

lemma *arctan_less_pi4_neg*: $-1 < x \implies -(\pi/4) < \arctan x$

by (*metis arctan_less_iff arctan_minus arctan_one*)

lemma *arctan_less_pi4*: $|x| < 1 \implies |\arctan x| < \pi/4$

by (*metis abs_less_iff arctan_less_pi4_pos arctan_minus*)

lemma *arctan_le_pi4*: $|x| \leq 1 \implies |\arctan x| \leq \pi/4$

by (metis abs_le_iff arctan_le_iff arctan_minus arctan_one)

lemma abs_arctan: $|\arctan x| = \arctan |x|$
 by (simp add: abs_if arctan_minus)

lemma arctan_add_raw:
 assumes $|\arctan x + \arctan y| < \pi/2$
 shows $\arctan x + \arctan y = \arctan((x + y) / (1 - x * y))$
proof (rule arctan_unique [symmetric])
 show $12: -(\pi / 2) < \arctan x + \arctan y < \pi / 2$
 using assms by linarith+
 show $\tan(\arctan x + \arctan y) = (x + y) / (1 - x * y)$
 using cos_gt_zero_pi [OF 12]
 by (simp add: arctan_tan_add)
qed

lemma arctan_inverse:
 assumes $0 < x$
 shows $\arctan(\text{inverse } x) = \pi/2 - \arctan x$
proof -
 have $\arctan(\text{inverse } x) = \arctan(\text{inverse}(\tan(\arctan x)))$
 by (simp add: arctan)
 also have $\dots = \arctan(\tan(\pi / 2 - \arctan x))$
 by (simp add: tan_cot)
 also have $\dots = \pi/2 - \arctan x$
proof -
 have $0 < \pi - \arctan x$
 using arctan_ubound [of x] pi_gt_zero by linarith
 with assms show ?thesis
 by (simp add: Transcendental.arctan_tan)
qed
 finally show ?thesis .
qed

lemma arctan_add_small:
 assumes $|x * y| < 1$
 shows $\arctan x + \arctan y = \arctan((x + y) / (1 - x * y))$
proof (cases $x = 0 \vee y = 0$)
 case False
 with assms have $|x| < \text{inverse } |y|$
 by (simp add: field_split_simps abs_mult)
 with False have $|\arctan x| < \pi / 2 - |\arctan y|$ using assms
 by (auto simp add: abs_arctan arctan_inverse [symmetric] arctan_less_iff)
 then show ?thesis
 by (intro arctan_add_raw) linarith
qed auto

lemma abs_arctan_le:
 fixes $x::\text{real}$ shows $|\arctan x| \leq |x|$

proof –

have $1: \bigwedge x. x \in \mathbb{R} \implies \text{cmod}(\text{inverse}(1 + x^2)) \leq 1$
by (*simp add: norm_divide divide_simps in_Reals_norm complex_is_Real_iff power2_eq_square*)
have $\text{cmod}(\text{Arctan } w - \text{Arctan } z) \leq 1 * \text{cmod}(w - z)$ **if** $w \in \mathbb{R} \ z \in \mathbb{R}$ **for** $w \ z$
apply (*rule field_differentiable_bound [OF convex_Reals, of Arctan _ 1]*)
apply (*rule has_field_derivative_at_within [OF has_field_derivative_Arctan]*)
using 1 **that** **by** (*auto simp: Reals_def*)
then **have** $\text{cmod}(\text{Arctan}(\text{of\_real } x) - \text{Arctan } 0) \leq 1 * \text{cmod}(\text{of\_real } x - 0)$
using *Reals_0 Reals_of_real* **by** *blast*
then **show** *?thesis*
by (*simp add: Arctan_of_real*)
qed

lemma *arctan_le_self*: $0 \leq x \implies \arctan x \leq x$
by (*metis abs_arctan_le abs_of_nonneg zero_le_arctan_iff*)

lemma *abs_tan_ge*: $|x| < \pi/2 \implies |x| \leq |\tan x|$
by (*metis abs_arctan_le abs_less_iff arctan_tan_minus_less_iff*)

lemma *arctan_bounds*:
assumes $0 \leq x < 1$
shows *arctan_lower_bound*:
 $(\sum_{k < 2 * n. (-1)^k * (1 / \text{real}(k * 2 + 1)) * x^{(k * 2 + 1)}}) \leq \arctan x$
(is $(\sum_{k < \_. (-1)^k * ?a k} \leq \_)$
and *arctan_upper_bound*:
 $\arctan x \leq (\sum_{k < 2 * n + 1. (-1)^k * (1 / \text{real}(k * 2 + 1)) * x^{(k * 2 + 1)}})$

proof –

have *tendsto_zero*: $?a \longrightarrow 0$
proof (*rule tendsto_eq_rhs*)
show $(\lambda k. 1 / \text{real}(k * 2 + 1) * x^{(k * 2 + 1)}) \longrightarrow 0 * 0$
using *assms*
by (*intro tendsto_mult real_tendsto_divide_at_top*)
(auto simp: filterlim_real_sequentially filterlim_sequentially_iff filterlim_real intro!: real_tendsto_divide_at_top tendsto_power_zero filterlim_real_sequentially tendsto_eq_intros filterlim_at_top_mult tendsto_pos filterlim_tendsto_add_at_top)
qed *simp*
have *nonneg*: $0 \leq ?a \ n \text{ for } n$
by (*force intro!: divide_nonneg_nonneg mult_nonneg_nonneg zero_le_power assms*)
have *le*: $?a (Suc \ n) \leq ?a \ n \text{ for } n$
by (*rule mult_mono [OF _ power_decreasing]*) (*auto simp: field_split_simps assms less_imp_le*)
from *summable_Leibniz' (4) [of ?a, OF tendsto_zero nonneg le, of n]*
 $\text{summable\_Leibniz' (2) [of ?a, OF tendsto_zero nonneg le, of n]}$
assms
show $(\sum_{k < 2 * n. (-1)^k * ?a k} \leq \arctan x \ \arctan x \leq (\sum_{k < 2 * n + 1. (-1)^k * ?a k})$

by (auto simp: arctan_series)
qed

6.21.22 Bounds on pi using real arctangent

lemma pi_machin: $\pi = 16 * \arctan (1 / 5) - 4 * \arctan (1 / 239)$
using machin by simp

lemma pi_approx: $3.141592653588 \leq \pi \leq 3.141592653589$
unfolding pi_machin
using arctan_bounds[of 1/5 4]
arctan_bounds[of 1/239 4]
by (simp_all add: eval_nat_numeral)

lemma pi_gt3: $\pi > 3$
using pi_approx by simp

6.21.23 Inverse Sine

definition Arcsin :: complex $\Rightarrow$ complex where
 $Arcsin \equiv \lambda z. -i * Ln(i * z + \sqrt{1 - z^2})$

lemma Arcsin_body_lemma: $i * z + \sqrt{1 - z^2} \neq 0$
using power2_sqrt [of 1 - z²]
by (metis add.inverse_inverse complex_i_mult_minus diff_0 diff_add_cancel
diff_minus_eq_add mult.assoc mult.commute numeral_One power2_eq_square zero_neq_numeral)

lemma Arcsin_range_lemma: $|Re\ z| < 1 \implies 0 < Re(i * z + \sqrt{1 - z^2})$
using Complex.cmod_power2 [of z, symmetric]
by (simp add: real_less_sqrt algebra_simps Re_power2 cmod_square_less_1_plus)

lemma Re_Arcsin: $Re(Arcsin\ z) = Im\ (Ln\ (i * z + \sqrt{1 - z^2}))$
by (simp add: Arcsin_def)

lemma Im_Arcsin: $Im(Arcsin\ z) = -\ln\ (cmod\ (i * z + \sqrt{1 - z^2}))$
by (simp add: Arcsin_def Arcsin_body_lemma)

lemma one_minus_z2_notin_nonpos_Reals:
assumes $Im\ z = 0 \implies |Re\ z| < 1$
shows $1 - z^2 \notin \mathbb{R}_{\leq 0}$
proof (cases $Im\ z = 0$)
case True
with assms show ?thesis
by (simp add: complex_nonpos_Reals_iff flip: abs_square_less_1)
next
case False
have $\neg (Im\ z)^2 \leq -1$
using False power2_less_eq_zero_iff by fastforce
with False show ?thesis
by (auto simp add: complex_nonpos_Reals_iff Re_power2 Im_power2)

qed

lemma *isCont_Arcsin_lemma*:

assumes *le0*: $\text{Re } (i * z + \text{csqrt } (1 - z^2)) \leq 0$ **and** $(\text{Im } z = 0 \implies |\text{Re } z| < 1)$
shows *False*

proof (*cases* $\text{Im } z = 0$)

case *True*

then show *?thesis*

using *assms* **by** (*fastforce simp: cmod_def abs_square_less_1 [symmetric]*)

next

case *False*

have *leim*: $(\text{cmod } (1 - z^2) + (1 - \text{Re } (z^2))) / 2 \leq (\text{Im } z)^2$

using *le0 sqrt_le_D* **by** *fastforce*

have *neq*: $(\text{cmod } z)^2 \neq 1 + \text{cmod } (1 - z^2)$

proof (*clarsimp simp add: cmod_def*)

assume $(\text{Re } z)^2 + (\text{Im } z)^2 = 1 + \text{sqrt } ((1 - \text{Re } (z^2))^2 + (\text{Im } (z^2))^2)$

then have $((\text{Re } z)^2 + (\text{Im } z)^2 - 1)^2 = ((1 - \text{Re } (z^2))^2 + (\text{Im } (z^2))^2)$

by *simp*

then show *False* **using** *False*

by (*simp add: power2_eq_square algebra_simps*)

qed

moreover have *2*: $(\text{Im } z)^2 = (1 + ((\text{Im } z)^2 + \text{cmod } (1 - z^2)) - (\text{Re } z)^2) / 2$

using *leim cmod_power2 [of z] norm_triangle_ineq2 [of z^2 1]*

by (*simp add: norm_power Re_power2 norm_minus_commute [of 1]*)

ultimately show *False*

by (*simp add: Re_power2 Im_power2 cmod_power2*)

qed

lemma *isCont_Arcsin*:

assumes $(\text{Im } z = 0 \implies |\text{Re } z| < 1)$

shows *isCont Arcsin z*

proof –

have *1*: $i * z + \text{csqrt } (1 - z^2) \notin \mathbb{R}_{\leq 0}$

by (*metis isCont_Arcsin_lemma assms complex_nonpos_Reals_iff*)

have *2*: $1 - z^2 \notin \mathbb{R}_{\leq 0}$

by (*simp add: one_minus_z2_notin_nonpos_Reals assms*)

show *?thesis*

using *assms* **unfolding** *Arcsin_def* **by** (*intro isCont_Ln' isCont_csqrt' continuous_intros 1 2*)

qed

lemma *isCont_Arcsin' [simp]*:

shows $\text{isCont } f \ z \implies (\text{Im } (f \ z) = 0 \implies |\text{Re } (f \ z)| < 1) \implies \text{isCont } (\lambda x. \text{Arcsin } (f \ x)) \ z$

by (*blast intro: isCont_o2 [OF _ isCont_Arcsin]*)

lemma *sin_Arcsin [simp]*: $\text{sin}(\text{Arcsin } z) = z$

proof –

have $i * z * 2 + \text{csqrt } (1 - z^2) * 2 = 0 \longleftrightarrow (i * z) * 2 + \text{csqrt } (1 - z^2) * 2 = 0$

```

    by (simp add: algebra_simps) — Cancelling a factor of 2
  moreover have ...  $\longleftrightarrow (i*z) + \text{csqrt } (1 - z^2) = 0$ 
  by (metis Arcsin_body_lemma distrib_right no_zero_divisors zero_neq_numeral)
  ultimately show ?thesis
    apply (simp add: sin_exp_eq Arcsin_def Arcsin_body_lemma exp_minus divide_simps)
    apply (simp add: algebra_simps)
    apply (simp add: power2_eq_square [symmetric] algebra_simps)
    done
qed

```

lemma *Re_eq_pihalf_lemma*:

```

  |Re z| = pi/2  $\implies$  Im z = 0  $\implies$ 
    Re ((exp (i*z) + inverse (exp (i*z))) / 2) = 0  $\wedge$  0  $\leq$  Im ((exp (i*z) + inverse
(exp (i*z))) / 2)
  apply (simp add: cos_i_times [symmetric] Re_cos Im_cos abs_if del: eq_divide_eq_numeral1)
  by (metis cos_minus cos_pi_half)

```

lemma *Re_less_pihalf_lemma*:

```

  assumes |Re z| < pi / 2
  shows 0 < Re ((exp (i*z) + inverse (exp (i*z))) / 2)
proof —
  have 0 < cos (Re z) using assms
    using cos_gt_zero_pi by auto
  then show ?thesis
    by (simp add: cos_i_times [symmetric] Re_cos Im_cos add_pos_pos)
qed

```

lemma *Arcsin_sin*:

```

  assumes |Re z| < pi/2  $\vee$  (|Re z| = pi/2  $\wedge$  Im z = 0)
  shows Arcsin(sin z) = z
proof —
  have Arcsin(sin z) = - (i * Ln (csqrt (1 - (i * (exp (i*z) - inverse (exp (i*z))))^2
/ 4) - (inverse (exp (i*z)) - exp (i*z)) / 2))
    by (simp add: sin_exp_eq Arcsin_def exp_minus power_divide)
  also have ... = - (i * Ln (csqrt (((exp (i*z) + inverse (exp (i*z))) / 2)^2) -
(inverse (exp (i*z)) - exp (i*z)) / 2))
    by (simp add: field_simps power2_eq_square)
  also have ... = - (i * Ln (((exp (i*z) + inverse (exp (i*z))) / 2) - (inverse (exp
(i*z)) - exp (i*z)) / 2))
    apply (subst csqrt_square)
    using assms Re_eq_pihalf_lemma Re_less_pihalf_lemma by auto
  also have ... = - (i * Ln (exp (i*z)))
    by (simp add: field_simps power2_eq_square)
  also have ... = z
    using assms by (auto simp: abs_if simp del: eq_divide_eq_numeral1 split:
if_split_asm)
  finally show ?thesis .
qed

```

lemma *Arcsin_unique*:

$\llbracket \sin z = w; |Re\ z| < \pi/2 \vee (|Re\ z| = \pi/2 \wedge Im\ z = 0) \rrbracket \implies Arcsin\ w = z$
by (*metis Arcsin_sin*)

lemma *Arcsin_0* [*simp*]: $Arcsin\ 0 = 0$

by (*metis Arcsin_sin norm_zero pi_half_gt_zero real_norm_def sin_zero zero_complex.simps(1)*)

lemma *Arcsin_1* [*simp*]: $Arcsin\ 1 = \pi/2$

by (*metis Arcsin_sin Im_complex_of_real Re_complex_of_real numeral_One of_real_numeral pi_half_ge_zero real_sqrt_abs real_sqrt_pow2 real_sqrt_power sin_of_real sin_pi_half*)

lemma *Arcsin_minus_1* [*simp*]: $Arcsin(-1) = -(\pi/2)$

by (*metis Arcsin_1 Arcsin_sin Im_complex_of_real Re_complex_of_real abs_of_nonneg of_real_minus pi_half_ge_zero power2_minus real_sqrt_abs sin_Arcsin sin_minus*)

lemma *has_field_derivative_Arcsin*:

assumes $Im\ z = 0 \implies |Re\ z| < 1$

shows (*Arcsin* *has_field_derivative* *inverse(cos(Arcsin z))*) (at *z*)

proof –

have $(\sin(Arcsin\ z))^2 \neq 1$

using *assms one_minus_z2_notin_nonpos_Reals* **by** *force*

then have $\cos(Arcsin\ z) \neq 0$

by (*metis diff_0_right power_zero_numeral sin_squared_eq*)

then show *?thesis*

by (*rule has_field_derivative_inverse_basic [OF DERIV_sin _ _ open_ball [of z 1]]*) (*auto intro: isCont_Arcsin assms*)

qed

declare *has_field_derivative_Arcsin* [*derivative_intros*]

declare *has_field_derivative_Arcsin* [*THEN DERIV_chain2, derivative_intros*]

lemma *field_differentiable_at_Arcsin*:

$(Im\ z = 0 \implies |Re\ z| < 1) \implies Arcsin\ \text{field\_differentiable at } z$

using *field_differentiable_def has_field_derivative_Arcsin* **by** *blast*

lemma *field_differentiable_within_Arcsin*:

$(Im\ z = 0 \implies |Re\ z| < 1) \implies Arcsin\ \text{field\_differentiable (at } z \text{ within } s)$

using *field_differentiable_at_Arcsin field_differentiable_within_subset* **by** *blast*

lemma *continuous_within_Arcsin*:

$(Im\ z = 0 \implies |Re\ z| < 1) \implies \text{continuous (at } z \text{ within } s)\ Arcsin$

using *continuous_at_imp_continuous_within isCont_Arcsin* **by** *blast*

lemma *continuous_on_Arcsin* [*continuous_intros*]:

$(\bigwedge z. z \in s \implies Im\ z = 0 \implies |Re\ z| < 1) \implies \text{continuous\_on } s\ Arcsin$

by (*simp add: continuous_at_imp_continuous_on*)

lemma *holomorphic_on_Arcsin*: $(\bigwedge z. z \in s \implies \text{Im } z = 0 \implies |\text{Re } z| < 1) \implies$
Arcsin holomorphic_on s
by (*simp add: field_differentiable_within_Arcsin holomorphic_on_def*)

6.21.24 Inverse Cosine

definition *Arccos* :: *complex* $\Rightarrow$ *complex* **where**
Arccos $\equiv \lambda z. -i * \text{Ln}(z + i * \text{csqrt}(1 - z^2))$

lemma *Arccos_range_lemma*: $|\text{Re } z| < 1 \implies 0 < \text{Im}(z + i * \text{csqrt}(1 - z^2))$
using *Arcsin_range_lemma* [*of -z*] **by** *simp*

lemma *Arccos_body_lemma*: $z + i * \text{csqrt}(1 - z^2) \neq 0$
using *Arcsin_body_lemma* [*of z*]
by (*metis Arcsin_body_lemma complex_i_mult_minus_diff_minus_eq_add_power2_minus_right_minus_eq*)

lemma *Re_Arccos*: $\text{Re}(\text{Arccos } z) = \text{Im}(\text{Ln}(z + i * \text{csqrt}(1 - z^2)))$
by (*simp add: Arccos_def*)

lemma *Im_Arccos*: $\text{Im}(\text{Arccos } z) = -\text{ln}(\text{cmod}(z + i * \text{csqrt}(1 - z^2)))$
by (*simp add: Arccos_def Arccos_body_lemma*)

A very tricky argument to find!

lemma *isCont_Arccos_lemma*:
assumes *eq0*: $\text{Im}(z + i * \text{csqrt}(1 - z^2)) = 0$ **and** $(\text{Im } z = 0 \implies |\text{Re } z| < 1)$
shows *False*
proof (*cases Im z = 0*)
case *True*
then show *?thesis*
using *assms* **by** (*fastforce simp add: cmod_def abs_square_less_1 [symmetric]*)
next
case *False*
have *Imz*: $\text{Im } z = -\text{sqrt}((1 + ((\text{Im } z)^2 + \text{cmod}(1 - z^2)) - (\text{Re } z)^2) / 2)$
using *eq0 abs_Re_le_cmod* [*of 1-z^2*]
by (*simp add: Re_power2 algebra_simps*)
have $(\text{cmod } z)^2 - 1 \neq \text{cmod}(1 - z^2)$
proof (*clarsimp simp add: cmod_def*)
assume $(\text{Re } z)^2 + (\text{Im } z)^2 - 1 = \text{sqrt}((1 - \text{Re } z^2))^2 + (\text{Im } z^2))^2$
then have $((\text{Re } z)^2 + (\text{Im } z)^2 - 1)^2 = ((1 - \text{Re } z^2))^2 + (\text{Im } z^2))^2$
by *simp*
then show *False* **using** *False*
by (*simp add: power2_eq_square algebra_simps*)
qed
moreover have $(\text{Im } z)^2 = (1 + ((\text{Im } z)^2 + \text{cmod}(1 - z^2)) - (\text{Re } z)^2) / 2$
using *abs_Re_le_cmod* [*of 1-z^2*] **by** (*subst Imz*) (*simp add: Re_power2*)
ultimately show *False*
by (*simp add: cmod_power2*)
qed

```

lemma isCont_Arccos:
  assumes (Im z = 0  $\implies$  |Re z| < 1)
  shows isCont Arccos z
proof -
  have z + i * csqrt (1 - z2)  $\notin \mathbb{R}_{\leq 0}$ 
  by (metis complex_nonpos_Reals_iff isCont_Arccos_lemma assms)
  with assms show ?thesis
  unfolding Arccos_def
  by (simp_all add: one_minus_z2_notin_nonpos_Reals assms)
qed

lemma isCont_Arccos' [simp]:
  isCont f z  $\implies$  (Im (f z) = 0  $\implies$  |Re (f z)| < 1)  $\implies$  isCont ( $\lambda x$ . Arccos (f x)) z
  by (blast intro: isCont_o2 [OF_ isCont_Arccos])

lemma cos_Arccos [simp]: cos(Arccos z) = z
proof -
  have z*2 + i * (2 * csqrt (1 - z2)) = 0  $\longleftrightarrow$  z*2 + i * csqrt (1 - z2)*2 = 0
  by (simp add: algebra_simps) — Cancelling a factor of 2
  moreover have ...  $\longleftrightarrow$  z + i * csqrt (1 - z2) = 0
  by (metis distrib_right_mult_eq_0_iff zero_neq_numeral)
  ultimately show ?thesis
  by (simp add: cos_exp_eq Arccos_def Arccos_body_lemma exp_minus_field_simps
    flip: power2_eq_square)
qed

lemma Arccos_cos:
  assumes 0 < Re z  $\wedge$  Re z < pi  $\vee$ 
    Re z = 0  $\wedge$  0  $\leq$  Im z  $\vee$ 
    Re z = pi  $\wedge$  Im z  $\leq$  0
  shows Arccos(cos z) = z
proof -
  have *: ((i - (exp (i * z))2 * i) / (2 * exp (i * z))) = sin z
  by (simp add: sin_exp_eq exp_minus_field_simps power2_eq_square)
  have 1 - (exp (i * z) + inverse (exp (i * z)))2 / 4 = ((i - (exp (i * z))2 * i) /
    (2 * exp (i * z)))2
  by (simp add: field_simps power2_eq_square)
  then have Arccos(cos z) = - (i * Ln ((exp (i * z) + inverse (exp (i * z))) / 2
    +
    i * csqrt (((i - (exp (i * z))2 * i) / (2 * exp (i * z)))2)))
  by (simp add: cos_exp_eq Arccos_def exp_minus_power_divide)
  also have ... = - (i * Ln ((exp (i * z) + inverse (exp (i * z))) / 2 +
    i * ((i - (exp (i * z))2 * i) / (2 * exp (i * z)))))
  apply (subst csqrt_square)
  using assms Re_sin_pos [of z] Im_sin_nonneg [of z] Im_sin_nonneg2 [of z]
  by (auto simp: * Re_sin Im_sin)
  also have ... = - (i * Ln (exp (i*z)))
  by (simp add: field_simps power2_eq_square)

```

```

also have ... = z
  using assms
  by (subst Complex_Transcendental.Ln_exp, auto)
finally show ?thesis .
qed

```

```

lemma Arccos_unique:
   $\llbracket \cos z = w;$ 
 $0 < \operatorname{Re} z \wedge \operatorname{Re} z < \pi \vee$ 
 $\operatorname{Re} z = 0 \wedge 0 \leq \operatorname{Im} z \vee$ 
 $\operatorname{Re} z = \pi \wedge \operatorname{Im} z \leq 0 \rrbracket \implies \operatorname{Arccos} w = z$ 
  using Arccos_cos by blast

```

```

lemma Arccos_0 [simp]:  $\operatorname{Arccos} 0 = \pi/2$ 
  by (rule Arccos_unique) auto

```

```

lemma Arccos_1 [simp]:  $\operatorname{Arccos} 1 = 0$ 
  by (rule Arccos_unique) auto

```

```

lemma Arccos_minus1:  $\operatorname{Arccos}(-1) = \pi$ 
  by (rule Arccos_unique) auto

```

```

lemma has_field_derivative_Arccos:
  assumes  $(\operatorname{Im} z = 0 \implies |\operatorname{Re} z| < 1)$ 
  shows  $(\operatorname{Arccos} \text{ has\_field\_derivative } - \operatorname{inverse}(\sin(\operatorname{Arccos} z))) \text{ (at } z)$ 
proof -
  have  $x^2 \neq -1$  for  $x::\text{real}$ 
  by (sos  $((R < 1 + ((\sim 1) * A = 0) + (R < 1 * (R < 1 * [x]^2))))$ )
  with assms have  $(\cos(\operatorname{Arccos} z))^2 \neq 1$ 
  by (auto simp: complex_eq_iff Re_power2 Im_power2 abs_square_eq_1)
  then have  $-\sin(\operatorname{Arccos} z) \neq 0$ 
  by (metis cos_squared_eq_diff_0_right mult_zero_left neg_0_equal_iff_equal
    power2_eq_square)
  then have  $(\operatorname{Arccos} \text{ has\_field\_derivative } \operatorname{inverse}(-\sin(\operatorname{Arccos} z))) \text{ (at } z)$ 
  by (rule has_field_derivative_inverse_basic [OF DERIV_cos _ _ open_ball
    [of z 1]])
  (auto intro: isCont_Arccos assms)
  then show ?thesis
  by simp
qed

```

```

declare has_field_derivative_Arcsin [derivative_intros]
declare has_field_derivative_Arcsin [THEN DERIV_chain2, derivative_intros]

```

```

lemma field_differentiable_at_Arccos:
   $(\operatorname{Im} z = 0 \implies |\operatorname{Re} z| < 1) \implies \operatorname{Arccos} \text{ field\_differentiable at } z$ 
  using field_differentiable_def has_field_derivative_Arccos by blast

```

```

lemma field_differentiable_within_Arccos:

```

$(\text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{Arccos field\_differentiable (at } z \text{ within } s)$
using *field_differentiable_at_Arccos field_differentiable_within_subset* **by** *blast*

lemma *continuous_within_Arccos*:

$(\text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{continuous (at } z \text{ within } s) \text{ Arccos}$
using *continuous_at_imp_continuous_within isCont_Arccos* **by** *blast*

lemma *continuous_on_Arccos* [*continuous_intros*]:

$(\bigwedge z. z \in s \implies \text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{continuous\_on } s \text{ Arccos}$
by (*simp add: continuous_at_imp_continuous_on*)

lemma *holomorphic_on_Arccos*: $(\bigwedge z. z \in s \implies \text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{Arccos holomorphic\_on } s$

by (*simp add: field_differentiable_within_Arccos holomorphic_on_def*)

6.21.25 Upper and Lower Bounds for Inverse Sine and Cosine

lemma *Arcsin_bounds*: $|\text{Re } z| < 1 \implies |\text{Re}(\text{Arcsin } z)| < \pi/2$

unfolding *Re_Arcsin*

by (*blast intro: Re_Ln_pos_lt_imp Arcsin_range_lemma*)

lemma *Arccos_bounds*: $|\text{Re } z| < 1 \implies 0 < \text{Re}(\text{Arccos } z) \wedge \text{Re}(\text{Arccos } z) < \pi$

unfolding *Re_Arccos*

by (*blast intro!: Im_Ln_pos_lt_imp Arccos_range_lemma*)

lemma *Re_Arccos_bounds*: $-\pi < \text{Re}(\text{Arccos } z) \wedge \text{Re}(\text{Arccos } z) \leq \pi$

unfolding *Re_Arccos*

by (*blast intro!: mpi_less_Im_Ln Im_Ln_le_pi Arccos_body_lemma*)

lemma *Re_Arccos_bound*: $|\text{Re}(\text{Arccos } z)| \leq \pi$

by (*meson Re_Arccos_bounds abs_le_iff less_eq_real_def minus_less_iff*)

lemma *Im_Arccos_bound*: $|\text{Im}(\text{Arccos } w)| \leq \text{cmod } w$

proof –

have $(\text{Im}(\text{Arccos } w))^2 \leq (\text{cmod}(\cos(\text{Arccos } w)))^2 - (\cos(\text{Re}(\text{Arccos } w)))^2$

using *norm_cos_squared* [of *Arccos w*] *real_le_abs_sinh* [of *Im(Arccos w)*]

by (*simp only: abs_le_square_iff*) (*simp add: field_split_simps*)

also have $\dots \leq (\text{cmod } w)^2$

by (*auto simp: cmod_power2*)

finally show *?thesis*

using *abs_le_square_iff* **by** *force*

qed

lemma *Re_Arcsin_bounds*: $-\pi < \text{Re}(\text{Arcsin } z) \wedge \text{Re}(\text{Arcsin } z) \leq \pi$

unfolding *Re_Arcsin*

by (*blast intro!: mpi_less_Im_Ln Im_Ln_le_pi Arcsin_body_lemma*)

lemma *Re_Arcsin_bound*: $|\text{Re}(\text{Arcsin } z)| \leq \pi$

by (meson Re_Arcsin_bounds abs_le_iff less_eq_real_def minus_less_iff)

lemma norm_Arccos_bounded:

fixes $w :: \text{complex}$

shows $\text{norm} (\text{Arccos } w) \leq \pi + \text{norm } w$

proof -

have Re: $(\text{Re} (\text{Arccos } w))^2 \leq \pi^2$ $(\text{Im} (\text{Arccos } w))^2 \leq (\text{cmod } w)^2$

using Re_Arccos_bound [of w] Im_Arccos_bound [of w] abs_le_square_iff by force+

have $\text{Arccos } w \cdot \text{Arccos } w \leq \pi^2 + (\text{cmod } w)^2$

using Re by (simp add: dot_square_norm cmod_power2 [of Arccos w])

then have $\text{cmod} (\text{Arccos } w) \leq \pi + \text{cmod} (\cos (\text{Arccos } w))$

apply (simp add: norm_le_square)

by (metis dot_square_norm norm_ge_zero norm_le_square pi_ge_zero triangle_lemma)

then show $\text{cmod} (\text{Arccos } w) \leq \pi + \text{cmod } w$

by auto

qed

6.21.26 Interrelations between Arcsin and Arccos

lemma cos_Arcsin_nonzero:

assumes $z^2 \neq 1$ shows $\cos(\text{Arcsin } z) \neq 0$

proof -

have eq: $(i * z * (\text{csqrt } (1 - z^2)))^2 = z^2 * (z^2 - 1)$

by (simp add: algebra_simps)

have $i * z * (\text{csqrt } (1 - z^2)) \neq z^2 - 1$

proof

assume $i * z * (\text{csqrt } (1 - z^2)) = z^2 - 1$

then have $(i * z * (\text{csqrt } (1 - z^2)))^2 = (z^2 - 1)^2$

by simp

then have $z^2 * (z^2 - 1) = (z^2 - 1) * (z^2 - 1)$

using eq power2_eq_square by auto

then show False

using assms by simp

qed

then have $1 + i * z * (\text{csqrt } (1 - z * z)) \neq z^2$

by (metis add_minus_cancel power2_eq_square uminus_add_conv_diff)

then have $2 * (1 + i * z * (\text{csqrt } (1 - z * z))) \neq 2 * z^2$

by (metis mult_cancel_left zero_neq_numeral)

then have $(i * z + \text{csqrt } (1 - z^2))^2 \neq -1$

using assms

apply (simp add: power2_sum)

apply (simp add: power2_eq_square algebra_simps)

done

then show ?thesis

apply (simp add: cos_exp_eq Arcsin_def exp_minus)

apply (simp add: divide_simps Arcsin_body_lemma)

apply (metis add.commute minus_unique power2_eq_square)

```

done
qed

lemma sin_Arccos_nonzero:
  assumes  $z^2 \neq 1$  shows  $\sin(\text{Arccos } z) \neq 0$ 
proof -
  have eq:  $(i * z * (\text{csqrt } (1 - z^2)))^2 = -(z^2) * (1 - z^2)$ 
    by (simp add: algebra_simps)
  have  $i * z * (\text{csqrt } (1 - z^2)) \neq 1 - z^2$ 
  proof
    assume  $i * z * (\text{csqrt } (1 - z^2)) = 1 - z^2$ 
    then have  $(i * z * (\text{csqrt } (1 - z^2)))^2 = (1 - z^2)^2$ 
      by simp
    then have  $-(z^2) * (1 - z^2) = (1 - z^2) * (1 - z^2)$ 
      using eq power2_eq_square by auto
    then have  $-(z^2) = (1 - z^2)$ 
      using assms
    by (metis add.commute add.right_neutral diff_add_cancel mult_right_cancel)
    then show False
      using assms by simp
  qed
  then have  $z^2 + i * z * (\text{csqrt } (1 - z^2)) \neq 1$ 
    by (simp add: algebra_simps)
  then have  $2 * (z^2 + i * z * (\text{csqrt } (1 - z^2))) \neq 2 * 1$ 
    by (metis mult_cancel_left2 zero_neq_numeral)
  then have  $(z + i * \text{csqrt } (1 - z^2))^2 \neq 1$ 
    using assms
  by (metis Arccos_def add.commute add.left_neutral cancel_comm_monoid_add_class.diff_cancel
cos_Arccos csqrt_0 mult_zero_right)
  then show ?thesis
    apply (simp add: sin_exp_eq Arccos_def exp_minus)
    apply (simp add: divide_simps Arccos_body_lemma)
    apply (simp add: power2_eq_square)
  done
qed

lemma cos_sin_csqrt:
  assumes  $0 < \cos(\text{Re } z) \vee \cos(\text{Re } z) = 0 \wedge \text{Im } z * \sin(\text{Re } z) \leq 0$ 
  shows  $\cos z = \text{csqrt}(1 - (\sin z)^2)$ 
proof (rule csqrt_unique [THEN sym])
  show  $(\cos z)^2 = 1 - (\sin z)^2$ 
    by (simp add: cos_squared_eq)
qed (use assms in ⟨auto simp: Re_cos Im_cos add_pos_pos mult_le_0_iff zero_le_mult_iff⟩)

lemma sin_cos_csqrt:
  assumes  $0 < \sin(\text{Re } z) \vee \sin(\text{Re } z) = 0 \wedge 0 \leq \text{Im } z * \cos(\text{Re } z)$ 
  shows  $\sin z = \text{csqrt}(1 - (\cos z)^2)$ 
proof (rule csqrt_unique [THEN sym])
  show  $(\sin z)^2 = 1 - (\cos z)^2$ 

```

by (simp add: sin_squared_eq)
 qed (use assms in ⟨auto simp: Re_sin Im_sin add_pos_pos mult_le_0_iff zero_le_mult_iff⟩)

lemma Arcsin_Arccos_csqrt_pos:
 $(0 < \operatorname{Re} z \mid \operatorname{Re} z = 0 \ \& \ 0 \leq \operatorname{Im} z) \implies \operatorname{Arcsin} z = \operatorname{Arccos}(\operatorname{csqrt}(1 - z^2))$
 by (simp add: Arcsin_def Arccos_def Complex.csqrt_square add commute)

lemma Arccos_Arcsin_csqrt_pos:
 $(0 < \operatorname{Re} z \mid \operatorname{Re} z = 0 \ \& \ 0 \leq \operatorname{Im} z) \implies \operatorname{Arccos} z = \operatorname{Arcsin}(\operatorname{csqrt}(1 - z^2))$
 by (simp add: Arcsin_def Arccos_def Complex.csqrt_square add commute)

lemma sin_Arccos:
 $0 < \operatorname{Re} z \mid \operatorname{Re} z = 0 \ \& \ 0 \leq \operatorname{Im} z \implies \sin(\operatorname{Arccos} z) = \operatorname{csqrt}(1 - z^2)$
 by (simp add: Arccos_Arcsin_csqrt_pos)

lemma cos_Arcsin:
 $0 < \operatorname{Re} z \mid \operatorname{Re} z = 0 \ \& \ 0 \leq \operatorname{Im} z \implies \cos(\operatorname{Arcsin} z) = \operatorname{csqrt}(1 - z^2)$
 by (simp add: Arcsin_Arccos_csqrt_pos)

6.21.27 Relationship with Arcsin on the Real Numbers

lemma Im_Arcsin_of_real:
 assumes $|x| \leq 1$
 shows $\operatorname{Im}(\operatorname{Arcsin}(\operatorname{of\_real} x)) = 0$
proof –
 have $\operatorname{csqrt}(1 - (\operatorname{of\_real} x)^2) = (\text{if } x^2 \leq 1 \text{ then } \operatorname{sqrt}(1 - x^2) \text{ else } i * \operatorname{sqrt}(x^2 - 1))$
 by (simp add: of_real_sqrt del: csqrt_of_real_nonneg)
 then have $\operatorname{cmod}(i * \operatorname{of\_real} x + \operatorname{csqrt}(1 - (\operatorname{of\_real} x)^2))^2 = 1$
 using assms abs_square_le_1
 by (force simp add: Complex.cmod_power2)
 then have $\operatorname{cmod}(i * \operatorname{of\_real} x + \operatorname{csqrt}(1 - (\operatorname{of\_real} x)^2)) = 1$
 by (simp add: norm_complex_def)
 then show ?thesis
 by (simp add: Im_Arcsin_exp_minus)
 qed

corollary Arcsin_in_Reals [simp]: $z \in \mathbb{R} \implies |\operatorname{Re} z| \leq 1 \implies \operatorname{Arcsin} z \in \mathbb{R}$
 by (metis Im_Arcsin_of_real Re_complex_of_real Reals_cases complex_is_Real_iff)

lemma arcsin_eq_Re_Arcsin:
 assumes $|x| \leq 1$
 shows $\operatorname{arcsin} x = \operatorname{Re}(\operatorname{Arcsin}(\operatorname{of\_real} x))$
unfolding arcsin_def
proof (rule the_equality, safe)
 show $-(\pi / 2) \leq \operatorname{Re}(\operatorname{Arcsin}(\operatorname{complex\_of\_real} x))$
 using Re_Ln_pos_le [OF Arcsin_body_lemma, of of_real x]
 by (auto simp: Complex.in_Reals_norm Re_Arcsin)
next

```

show Re (Arcsin (complex_of_real x)) ≤ pi / 2
  using Re_Ln_pos_le [OF Arcsin_body_lemma, of_of_real x]
  by (auto simp: Complex.in_Reals_norm Re_Arcsin)
next
show sin (Re (Arcsin (complex_of_real x))) = x
  using Re_sin [of Arcsin (of_real x)] Arcsin_body_lemma [of_of_real x]
  by (simp add: Im_Arcsin_of_real assms)
next
fix x'
assume - (pi / 2) ≤ x' x' ≤ pi / 2 x = sin x'
then show x' = Re (Arcsin (complex_of_real (sin x')))
  unfolding sin_of_real [symmetric]
  by (subst Arcsin_sin) auto
qed

lemma of_real_arcsin: |x| ≤ 1 ⇒ of_real(arcsin x) = Arcsin(of_real x)
  by (metis Im_Arcsin_of_real add.right_neutral arcsin_eq_Re_Arcsin complex_eq
mult_zero_right of_real_0)

```

6.21.28 Relationship with Arccos on the Real Numbers

```

lemma Im_Arccos_of_real:
  assumes |x| ≤ 1
  shows Im (Arccos (of_real x)) = 0
proof -
  have csqrt (1 - (of_real x)^2) = (if x^2 ≤ 1 then sqrt (1 - x^2) else i * sqrt
(x^2 - 1))
    by (simp add: of_real_sqrt del: csqrt_of_real_nonneg)
  then have cmod (of_real x + i * csqrt (1 - (of_real x)^2))^2 = 1
    using assms abs_square_le_1
    by (force simp add: Complex.cmod_power2)
  then have cmod (of_real x + i * csqrt (1 - (of_real x)^2)) = 1
    by (simp add: norm_complex_def)
  then show ?thesis
    by (simp add: Im_Arccos exp_minus)
qed

corollary Arccos_in_Reals [simp]: z ∈ ℝ ⇒ |Re z| ≤ 1 ⇒ Arccos z ∈ ℝ
  by (metis Im_Arccos_of_real Re_complex_of_real Reals_cases complex_is_Real_iff)

lemma arccos_eq_Re_Arccos:
  assumes |x| ≤ 1
  shows arccos x = Re (Arccos (of_real x))
unfolding arccos_def
proof (rule the_equality, safe)
  show 0 ≤ Re (Arccos (complex_of_real x))
    using Im_Ln_pos_le [OF Arccos_body_lemma, of_of_real x]
    by (auto simp: Complex.in_Reals_norm Re_Arccos)
next

```



```

show Re (Arccos (complex_of_real x)) ≤ pi
  using Im_Ln_pos_le [OF Arccos_body_lemma, of of_real x]
  by (auto simp: Complex.in_Reals_norm Re_Arccos)
next
show cos (Re (Arccos (complex_of_real x))) = x
  using Re_cos [of Arccos (of_real x)] Arccos_body_lemma [of of_real x]
  by (simp add: Im_Arccos_of_real assms)
next
fix x'
assume 0 ≤ x' x' ≤ pi x = cos x'
then show x' = Re (Arccos (complex_of_real (cos x')))
  unfolding cos_of_real [symmetric]
  by (subst Arccos_cos) auto
qed

lemma of_real_arccos: |x| ≤ 1 ⇒ of_real(arccos x) = Arccos(of_real x)
  by (metis Im_Arccos_of_real add.right_neutral arccos_eq_Re_Arccos complex_eq
mult_zero_right of_real_0)

```

6.21.29 Some interrelationships among the real inverse trig functions

```

lemma arccos_arctan:
  assumes -1 < x x < 1
  shows arccos x = pi/2 - arctan(x / sqrt(1 - x2))
proof -
  have arctan(x / sqrt(1 - x2)) - (pi/2 - arccos x) = 0
  proof (rule sin_eq_0_pi)
    show - pi < arctan (x / sqrt (1 - x2)) - (pi / 2 - arccos x)
    using arctan_lbound [of x / sqrt(1 - x2)] arccos_bounded [of x] assms
    by (simp add: algebra_simps)
  next
    show arctan (x / sqrt (1 - x2)) - (pi / 2 - arccos x) < pi
    using arctan_ubound [of x / sqrt(1 - x2)] arccos_bounded [of x] assms
    by (simp add: algebra_simps)
  next
    show sin (arctan (x / sqrt (1 - x2)) - (pi / 2 - arccos x)) = 0
    using assms
    by (simp add: algebra_simps sin_diff cos_add sin_arccos sin_arctan cos_arctan
power2_eq_square square_eq_1_iff)
  qed
then show ?thesis
  by simp
qed

```

```

lemma arcsin_plus_arccos:
  assumes -1 ≤ x x ≤ 1
  shows arcsin x + arccos x = pi/2
proof -

```

```

have arcsin x = pi/2 - arccos x
  apply (rule sin_inj_pi)
  using assms arcsin [OF assms] arccos [OF assms]
  by (auto simp: algebra_simps sin_diff)
then show ?thesis
  by (simp add: algebra_simps)
qed

```

```

lemma arcsin_arccos_eq:  $-1 \leq x \implies x \leq 1 \implies \arcsin x = \pi/2 - \arccos x$ 
  using arcsin_plus_arccos by force

```

```

lemma arccos_arcsin_eq:  $-1 \leq x \implies x \leq 1 \implies \arccos x = \pi/2 - \arcsin x$ 
  using arcsin_plus_arccos by force

```

```

lemma arcsin_arctan:  $-1 < x \implies x < 1 \implies \arcsin x = \arctan(x / \sqrt{1 - x^2})$ 
  by (simp add: arccos_arctan arcsin_arccos_eq)

```

```

lemma csqrt_1_diff_eq:  $\text{csqrt } (1 - (\text{of\_real } x)^2) = (\text{if } x^2 \leq 1 \text{ then } \sqrt{1 - x^2} \text{ else } i * \sqrt{x^2 - 1})$ 
  by (simp add: of_real_sqrt del: csqrt_of_real_nonneg)

```

```

lemma arcsin_arccos_sqrt_pos:  $0 \leq x \implies x \leq 1 \implies \arcsin x = \arccos(\sqrt{1 - x^2})$ 
  apply (simp add: abs_square_le_1 arcsin_eq_Re_Arcsin arccos_eq_Re_Arccos)
  apply (subst Arccos_Arcsin_csqrt_pos)
  apply (auto simp: power_le_one csqrt_1_diff_eq)
  done

```

```

lemma arcsin_arccos_sqrt_neg:  $-1 \leq x \implies x \leq 0 \implies \arcsin x = -\arccos(\sqrt{1 - x^2})$ 
  using arcsin_arccos_sqrt_pos [of -x]
  by (simp add: arcsin_minus)

```

```

lemma arccos_arcsin_sqrt_pos:  $0 \leq x \implies x \leq 1 \implies \arccos x = \arcsin(\sqrt{1 - x^2})$ 
  apply (simp add: abs_square_le_1 arcsin_eq_Re_Arcsin arccos_eq_Re_Arccos)
  apply (subst Arccos_Arcsin_csqrt_pos)
  apply (auto simp: power_le_one csqrt_1_diff_eq)
  done

```

```

lemma arccos_arcsin_sqrt_neg:  $-1 \leq x \implies x \leq 0 \implies \arccos x = \pi - \arcsin(\sqrt{1 - x^2})$ 
  using arccos_arcsin_sqrt_pos [of -x]
  by (simp add: arccos_minus)

```

6.21.30 Continuity results for arcsin and arccos

```

lemma continuous_on_Arcsin_real [continuous_intros]:

```

```

    continuous_on {w ∈ ℝ. |Re w| ≤ 1} Arcsin
  proof -
    have continuous_on {w ∈ ℝ. |Re w| ≤ 1} (λx. complex_of_real (arcsin (Re
x))) =
      continuous_on {w ∈ ℝ. |Re w| ≤ 1} (λx. complex_of_real (Re (Arcsin
(of_real (Re x)))))
    by (rule continuous_on_cong [OF refl]) (simp add: arcsin_eq_Re_Arcsin)
    also have ... = ?thesis
    by (rule continuous_on_cong [OF refl]) simp
    finally show ?thesis
    using continuous_on_arcsin [OF continuous_on_Re [OF continuous_on_id],
of {w ∈ ℝ. |Re w| ≤ 1}]
      continuous_on_of_real
    by fastforce
  qed

```

```

lemma continuous_within_Arcsin_real:
  continuous (at z within {w ∈ ℝ. |Re w| ≤ 1}) Arcsin
proof (cases z ∈ {w ∈ ℝ. |Re w| ≤ 1})
  case True then show ?thesis
    using continuous_on_Arcsin_real continuous_on_eq_continuous_within
    by blast
  next
  case False
    with closed_real_abs_le [of 1] show ?thesis
    by (rule continuous_within_closed_nontrivial)
  qed

```

```

lemma continuous_on_Arccos_real:
  continuous_on {w ∈ ℝ. |Re w| ≤ 1} Arccos
proof -
  have continuous_on {w ∈ ℝ. |Re w| ≤ 1} (λx. complex_of_real (arccos (Re
x))) =
    continuous_on {w ∈ ℝ. |Re w| ≤ 1} (λx. complex_of_real (Re (Arccos
(of_real (Re x)))))
  by (rule continuous_on_cong [OF refl]) (simp add: arccos_eq_Re_Arccos)
  also have ... = ?thesis
  by (rule continuous_on_cong [OF refl]) simp
  finally show ?thesis
  using continuous_on_arccos [OF continuous_on_Re [OF continuous_on_id],
of {w ∈ ℝ. |Re w| ≤ 1}]
    continuous_on_of_real
  by fastforce
  qed

```

```

lemma continuous_within_Arccos_real:
  continuous (at z within {w ∈ ℝ. |Re w| ≤ 1}) Arccos
proof (cases z ∈ {w ∈ ℝ. |Re w| ≤ 1})
  case True then show ?thesis

```

```

    using continuous_on_Arccos_real continuous_on_eq_continuous_within
    by blast
next
case False
with closed_real_abs_le [of 1] show ?thesis
by (rule continuous_within_closed_nontrivial)
qed

```

```

lemma sinh_ln_complex:  $x \neq 0 \implies \sinh (\ln x :: \text{complex}) = (x - \text{inverse } x) / 2$ 
by (simp add: sinh_def exp_minus scaleR_conv_of_real exp_of_real)

```

```

lemma cosh_ln_complex:  $x \neq 0 \implies \cosh (\ln x :: \text{complex}) = (x + \text{inverse } x) / 2$ 
by (simp add: cosh_def exp_minus scaleR_conv_of_real)

```

```

lemma tanh_ln_complex:  $x \neq 0 \implies \tanh (\ln x :: \text{complex}) = (x^2 - 1) / (x^2 + 1)$ 
by (simp add: tanh_def sinh_ln_complex cosh_ln_complex divide_simps power2_eq_square)

```

6.21.31 Roots of unity

```

theorem complex_root_unity:
  fixes j::nat
  assumes  $n \neq 0$ 
  shows  $\exp(2 * \text{of\_real } \pi * i * \text{of\_nat } j / \text{of\_nat } n)^n = 1$ 
proof -
  have *:  $\text{of\_nat } j * (\text{complex\_of\_real } \pi * 2) = \text{complex\_of\_real } (2 * \text{real } j * \pi i)$ 
  by (simp)
  then show ?thesis
  apply (simp add: exp_of_nat_mult [symmetric] mult_ac exp_Euler)
  apply (simp only: * cos_of_real sin_of_real)
  apply simp
  done
qed

```

```

lemma complex_root_unity_eq:
  fixes j::nat and k::nat
  assumes  $1 \leq n$ 
  shows  $(\exp(2 * \text{of\_real } \pi * i * \text{of\_nat } j / \text{of\_nat } n) = \exp(2 * \text{of\_real } \pi * i * \text{of\_nat } k / \text{of\_nat } n) \iff j \bmod n = k \bmod n)$ 
proof -
  have  $(\exists z::\text{int}. i * (\text{of\_nat } j * (\text{of\_real } \pi * 2)) = i * (\text{of\_nat } k * (\text{of\_real } \pi * 2)) + i * (\text{of\_int } z * (\text{of\_nat } n * (\text{of\_real } \pi * 2)))) \iff$ 
     $(\exists z::\text{int}. \text{of\_nat } j * (i * (\text{of\_real } \pi * 2)) = (\text{of\_nat } k + \text{of\_nat } n * \text{of\_int } z) * (i * (\text{of\_real } \pi * 2)))$ 
  by (simp add: algebra_simps)
  also have ...  $\iff (\exists z::\text{int}. \text{of\_nat } j = \text{of\_nat } k + \text{of\_nat } n * (\text{of\_int } z ::$ 

```

```

complex))
  by simp
  also have ...  $\longleftrightarrow (\exists z::int. \text{of\_nat } j = \text{of\_nat } k + \text{of\_nat } n * z)$ 
  by (metis (mono_tags, opaque_lifting) of_int_add of_int_eq_iff of_int_mult
of_int_of_nat_eq)
  also have ...  $\longleftrightarrow \text{int } j \bmod \text{int } n = \text{int } k \bmod \text{int } n$ 
  by (auto simp: mod_eq_dvd_iff dvd_def algebra_simps)
  also have ...  $\longleftrightarrow j \bmod n = k \bmod n$ 
  by (metis of_nat_eq_iff zmod_int)
  finally have  $(\exists z. i * (\text{of\_nat } j * (\text{of\_real } \pi * 2)) =$ 
 $i * (\text{of\_nat } k * (\text{of\_real } \pi * 2)) + i * (\text{of\_int } z * (\text{of\_nat } n * (\text{of\_real}$ 
 $\pi * 2)))) \longleftrightarrow j \bmod n = k \bmod n .$ 
  note * = this
  show ?thesis
  using assms
  by (simp add: exp_eq_field_split_simps *)
qed

```

corollary *bij_betw_roots_unity:*

```

bij_betw ( $\lambda j. \exp(2 * \text{of\_real } \pi * i * \text{of\_nat } j / \text{of\_nat } n)$ )
  {.. $n$ } { $\exp(2 * \text{of\_real } \pi * i * \text{of\_nat } j / \text{of\_nat } n) \mid j. j < n$ }
by (auto simp: bij_betw_def inj_on_def complex_root_unity_eq)

```

lemma *complex_root_unity_eq_1:*

```

fixes j::nat and k::nat
assumes  $1 \leq n$ 
shows  $\exp(2 * \text{of\_real } \pi * i * \text{of\_nat } j / \text{of\_nat } n) = 1 \longleftrightarrow n \text{ dvd } j$ 
proof -
  have  $1 = \exp(2 * \text{of\_real } \pi * i * (\text{of\_nat } n / \text{of\_nat } n))$ 
  using assms by simp
  then have  $\exp(2 * \text{of\_real } \pi * i * (\text{of\_nat } j / \text{of\_nat } n)) = 1 \longleftrightarrow j \bmod n =$ 
 $n \bmod n$ 
  using complex_root_unity_eq [of  $n$   $j$   $n$ ] assms
  by simp
  then show ?thesis
  by auto
qed

```

lemma *finite_complex_roots_unity_explicit:*

```

finite { $\exp(2 * \text{of\_real } \pi * i * \text{of\_nat } j / \text{of\_nat } n) \mid j::nat. j < n$ }
by simp

```

lemma *card_complex_roots_unity_explicit:*

```

card { $\exp(2 * \text{of\_real } \pi * i * \text{of\_nat } j / \text{of\_nat } n) \mid j::nat. j < n$ } = n
by (simp add: Finite_Set.bij_betw_same_card [OF bij_betw_roots_unity, sym-
metric])

```

lemma *complex_roots_unity:*

```

assumes  $1 \leq n$ 

```

```

    shows {z::complex. z^n = 1} = {exp(2 * of_real pi * i * of_nat j / of_nat
n) | j. j < n}
    apply (rule Finite_Set.card_seteq [symmetric])
    using assms
    apply (auto simp: card_complex_roots_unity_explicit finite_roots_unity com-
plex_root_unity card_roots_unity)
    done

```

```

lemma card_complex_roots_unity: 1 ≤ n ⇒ card {z::complex. z^n = 1} = n
  by (simp add: card_complex_roots_unity_explicit complex_roots_unity)

```

```

lemma complex_not_root_unity:
  1 ≤ n ⇒ ∃ u::complex. norm u = 1 ∧ u^n ≠ 1
  apply (rule_tac x=exp (of_real pi * i * of_real (1 / n)) in exI)
  apply (auto simp: Re_complex_div_eq_0 exp_of_nat_mult [symmetric] mult_ac
exp_Euler)
  done

```

```
end
```

6.22 Harmonic Numbers

```

theory Harmonic_Numbers
imports
  Complex_Transcendental
  Summation_Tests
begin

```

The definition of the Harmonic Numbers and the Euler-Mascheroni constant. Also provides a reasonably accurate approximation of $\ln 2$ and the Euler-Mascheroni constant.

6.22.1 The Harmonic numbers

```

definition harm :: nat ⇒ 'a :: real_normed_field where
  harm n = (∑ k=1..n. inverse (of_nat k))

```

```

lemma harm_altdef: harm n = (∑ k<n. inverse (of_nat (Suc k)))
  unfolding harm_def by (induction n) simp_all

```

```

lemma harm_Suc: harm (Suc n) = harm n + inverse (of_nat (Suc n))
  by (simp add: harm_def)

```

```

lemma harm_nonneg: harm n ≥ (0 :: 'a :: {real_normed_field, linordered_field})
  unfolding harm_def by (intro sum_nonneg) simp_all

```

```

lemma harm_pos: n > 0 ⇒ harm n > (0 :: 'a :: {real_normed_field, linordered_field})
  unfolding harm_def by (intro sum_pos) simp_all

```

lemma *harm_mono*: $m \leq n \implies \text{harm } m \leq (\text{harm } n :: 'a :: \{\text{real\_normed\_field}, \text{linordered\_field}\})$
by (*simp add: harm_def sum_mono2*)

lemma *of_real_harm*: $\text{of\_real } (\text{harm } n) = \text{harm } n$
unfolding *harm_def* **by** *simp*

lemma *abs_harm* [*simp*]: $(\text{abs } (\text{harm } n) :: \text{real}) = \text{harm } n$
using *harm_nonneg[of n]* **by** (*rule abs_of_nonneg*)

lemma *norm_harm*: $\text{norm } (\text{harm } n) = \text{harm } n$
by (*subst of_real_harm [symmetric]*) (*simp add: harm_nonneg*)

lemma *harm_expand*:
 $\text{harm } 0 = 0$
 $\text{harm } (\text{Suc } 0) = 1$
 $\text{harm } (\text{numeral } n) = \text{harm } (\text{pred\_numeral } n) + \text{inverse } (\text{numeral } n)$
proof –
have $\text{numeral } n = \text{Suc } (\text{pred\_numeral } n)$ **by** *simp*
also have $\text{harm } \dots = \text{harm } (\text{pred\_numeral } n) + \text{inverse } (\text{numeral } n)$
by (*subst harm_Suc, subst numeral_eq_Suc[symmetric]*) *simp*
finally show $\text{harm } (\text{numeral } n) = \text{harm } (\text{pred\_numeral } n) + \text{inverse } (\text{numeral } n)$.
qed (*simp_all add: harm_def*)

theorem *not_convergent_harm*: $\neg \text{convergent } (\text{harm } :: \text{nat} \Rightarrow 'a :: \text{real\_normed\_field})$
proof –
have $\text{convergent } (\lambda n. \text{norm } (\text{harm } n :: 'a)) \longleftrightarrow$
 $\text{convergent } (\text{harm } :: \text{nat} \Rightarrow \text{real})$ **by** (*simp add: norm_harm*)
also have $\dots \longleftrightarrow \text{convergent } (\lambda n. \sum k=\text{Suc } 0.. \text{Suc } n. \text{inverse } (\text{of\_nat } k) :: \text{real})$
unfolding *harm_def[abs_def]* **by** (*subst convergent_Suc_iff*) *simp_all*
also have $\dots \longleftrightarrow \text{convergent } (\lambda n. \sum k \leq n. \text{inverse } (\text{of\_nat } (\text{Suc } k)) :: \text{real})$
by (*subst sum.shift_bounds_cl_Suc_ivl*) (*simp add: atLeast0AtMost*)
also have $\dots \longleftrightarrow \text{summable } (\lambda n. \text{inverse } (\text{of\_nat } n) :: \text{real})$
by (*subst summable_Suc_iff [symmetric]*) (*simp add: summable_iff_convergent'*)
also have $\neg \dots$ **by** (*rule not_summable_harmonic*)
finally show *?thesis* **by** (*blast dest: convergent_norm*)
qed

lemma *harm_pos_iff* [*simp*]: $\text{harm } n > (0 :: 'a :: \{\text{real\_normed\_field}, \text{linordered\_field}\})$
 $\longleftrightarrow n > 0$
by (*rule iffI, cases n, simp add: harm_expand, simp, rule harm_pos*)

lemma *ln_diff_le_inverse*:
assumes $x \geq (1 :: \text{real})$
shows $\ln (x + 1) - \ln x < 1 / x$
proof –
from *assms* **have** $\exists z > x. z < x + 1 \wedge \ln (x + 1) - \ln x = (x + 1 - x) * \text{inverse } z$

```

    by (intro MVT2) (auto intro!: derivative_eq_intros simp: field_simps)
  then obtain z where z:  $z > x$   $z < x + 1$   $\ln(x + 1) - \ln x = \text{inverse } z$  by auto
  have  $\ln(x + 1) - \ln x = \text{inverse } z$  by fact
  also from  $z(1,2)$  assms have  $\dots < 1 / x$  by (simp add: field_simps)
  finally show ?thesis .
qed

```

```

lemma ln_le_harm:  $\ln(\text{real } n + 1) \leq (\text{harm } n :: \text{real})$ 
proof (induction n)
  fix n assume IH:  $\ln(\text{real } n + 1) \leq \text{harm } n$ 
  have  $\ln(\text{real } (\text{Suc } n) + 1) = \ln(\text{real } n + 1) + (\ln(\text{real } n + 2) - \ln(\text{real } n + 1))$  by simp
  also have  $(\ln(\text{real } n + 2) - \ln(\text{real } n + 1)) \leq 1 / \text{real } (\text{Suc } n)$ 
    using ln_diff_le_inverse[of  $\text{real } n + 1$ ] by (simp add: add_ac)
  also note IH
  also have  $\text{harm } n + 1 / \text{real } (\text{Suc } n) = \text{harm } (\text{Suc } n)$  by (simp add: harm_Suc field_simps)
  finally show  $\ln(\text{real } (\text{Suc } n) + 1) \leq \text{harm } (\text{Suc } n)$  by - simp
qed (simp_all add: harm_def)

```

```

lemma harm_at_top:  $\text{filterlim } (\text{harm } :: \text{nat} \Rightarrow \text{real}) \text{ at\_top sequentially}$ 
proof (rule filterlim_at_top_mono)
  show eventually  $(\lambda n. \text{harm } n \geq \ln(\text{real } (\text{Suc } n))) \text{ at\_top}$ 
    using ln_le_harm by (intro always_eventually_allI) (simp_all add: add_ac)
  show filterlim  $(\lambda n. \ln(\text{real } (\text{Suc } n))) \text{ at\_top sequentially}$ 
    by (intro filterlim_compose[OF ln_at_top] filterlim_compose[OF filterlim_real_sequentially] filterlim_Suc)
qed

```

6.22.2 The Euler-Mascheroni constant

The limit of the difference between the partial harmonic sum and the natural logarithm (approximately 0.577216). This value occurs e.g. in the definition of the Gamma function.

definition *euler_mascheroni* :: 'a :: *real_normed_algebra_1* where
 $\text{euler\_mascheroni} = \text{of\_real } (\lim (\lambda n. \text{harm } n - \ln(\text{of\_nat } n)))$

```

lemma of_real_euler_mascheroni [simp]:  $\text{of\_real } \text{euler\_mascheroni} = \text{euler\_mascheroni}$ 
  by (simp add: euler_mascheroni_def)

```

```

lemma harm_ge_ln:  $\text{harm } n \geq \ln(\text{real } n + 1)$ 
proof -
  have  $\ln(n + 1) = (\sum j < n. \ln(\text{real } (\text{Suc } j + 1)) - \ln(\text{real } (j + 1)))$ 
    by (subst sum_lessThan_telescope) auto
  also have  $\dots \leq (\sum j < n. 1 / (\text{Suc } j))$ 
  proof (intro sum_mono, clarify)
    fix j assume j:  $j < n$ 
    have  $\exists \xi. \xi > \text{real } j + 1 \wedge \xi < \text{real } j + 2 \wedge$   

 $\ln(\text{real } j + 2) - \ln(\text{real } j + 1) = (\text{real } j + 2 - (\text{real } j + 1)) * (1 / \xi)$ 

```



```

    by (intro MVT2) (auto intro!: derivative_eq_intros)
  then obtain  $\xi :: \text{real}$ 
    where  $\xi: \xi \in \{\text{real } j + 1 .. \text{real } j + 2\} \text{ ln } (\text{real } j + 2) - \text{ln } (\text{real } j + 1) = 1$ 
  /  $\xi$ 
    by auto
  note  $\xi(2)$ 
  also have  $1 / \xi \leq 1 / (\text{Suc } j)$ 
    using  $\xi(1)$  by (auto simp: field_simps)
  finally show  $\text{ln } (\text{real } (\text{Suc } j + 1)) - \text{ln } (\text{real } (j + 1)) \leq 1 / (\text{Suc } j)$ 
    by (simp add: add_ac)
qed
also have  $\dots = \text{harm } n$ 
  by (simp add: harm_altdef field_simps)
finally show ?thesis by (simp add: add_ac)
qed

```

```

lemma decseq_harm_diff_ln: decseq ( $\lambda n. \text{harm } (\text{Suc } n) - \text{ln } (\text{Suc } n)$ )
proof (rule decseq_SucI)
  fix  $m :: \text{nat}$ 
  define  $n$  where  $n = \text{Suc } m$ 
  have  $n > 0$  by (simp add: n_def)
  have convex_on { $0 < ..$ } ( $\lambda x :: \text{real}. -\text{ln } x$ )
    by (rule convex_on_realI [where  $f' = \lambda x. -1/x$ ])
    (auto intro!: derivative_eq_intros simp: field_simps)
  hence  $(-1 / (n + 1)) * (\text{real } n - \text{real } (n + 1)) \leq (-\text{ln } (\text{real } n)) - (-\text{ln } (\text{real } (n + 1)))$ 
    using  $\langle n > 0 \rangle$  by (intro convex_on_imp_above_tangent [where  $A = \{0 < ..\}$ ])
    (auto intro!: derivative_eq_intros simp: interior_open)
  thus  $\text{harm } (\text{Suc } n) - \text{ln } (\text{Suc } n) \leq \text{harm } n - \text{ln } n$ 
    by (auto simp: harm_Suc field_simps)
qed

```

```

lemma euler_mascheroni_sequence_nonneg:
  assumes  $n > 0$ 
  shows  $\text{harm } n - \text{ln } (\text{real } n) \geq (0 :: \text{real})$ 
proof -
  have  $\text{ln } (\text{real } n) \leq \text{ln } (\text{real } n + 1)$ 
    using assms by simp
  also have  $\dots \leq \text{harm } n$ 
    by (rule harm_ge_ln)
  finally show ?thesis by simp
qed

```

```

lemma euler_mascheroni_convergent: convergent ( $\lambda n. \text{harm } n - \text{ln } n$ )
proof -
  have  $\text{harm } (\text{Suc } n) - \text{ln } (\text{real } (\text{Suc } n)) \geq 0$  for  $n :: \text{nat}$ 
    using euler_mascheroni_sequence_nonneg [of  $\text{Suc } n$ ] by simp
  hence convergent ( $\lambda n. \text{harm } (\text{Suc } n) - \text{ln } (\text{Suc } n)$ )
    by (intro Bseq_monoseq_convergent decseq_bounded [of  $0$ ] decseq_harm_diff_ln

```

```

decseq_imp_monoseq)
  auto
  thus ?thesis
    by (subst (asm) convergent_Suc_iff)
qed

```

```

lemma euler_mascheroni_sequence_decreasing:
   $m > 0 \implies m \leq n \implies \text{harm } n - \ln (\text{of\_nat } n) \leq \text{harm } m - \ln (\text{of\_nat } m :: \text{real})$ 
  using decseqD[OF decseq_harm_diff_ln, of m - 1 n - 1] by simp

```

```

lemma euler_mascheroni_LIMSEQ:
   $(\lambda n. \text{harm } n - \ln (\text{of\_nat } n) :: \text{real}) \longrightarrow \text{euler\_mascheroni}$ 
  unfolding euler_mascheroni_def
  by (simp add: convergent_LIMSEQ_iff [symmetric] euler_mascheroni_convergent)

```

```

lemma euler_mascheroni_LIMSEQ_of_real:
   $(\lambda n. \text{of\_real } (\text{harm } n - \ln (\text{of\_nat } n))) \longrightarrow$ 
   $(\text{euler\_mascheroni} :: 'a :: \{\text{real\_normed\_algebra\_1}, \text{topological\_space}\})$ 
  proof -
    have  $(\lambda n. \text{of\_real } (\text{harm } n - \ln (\text{of\_nat } n))) \longrightarrow (\text{of\_real } (\text{euler\_mascheroni}))$ 
      :: 'a
    by (intro tendsto_of_real euler_mascheroni_LIMSEQ)
    thus ?thesis by simp
  qed

```

```

lemma euler_mascheroni_sum_real:
   $(\lambda n. \text{inverse } (\text{of\_nat } (n+1)) + \ln (\text{of\_nat } (n+1)) - \ln (\text{of\_nat } (n+2)) :: \text{real})$ 
  sums euler_mascheroni
  using sums_add[OF telescope_sums[OF LIMSEQ_Suc[OF euler_mascheroni_LIMSEQ]]
    telescope_sums'[OF LIMSEQ_inverse_real_of_nat]]
  by (simp_all add: harm_def algebra_simps)

```

```

lemma euler_mascheroni_sum:
   $(\lambda n. \text{inverse } (\text{of\_nat } (n+1)) + \text{of\_real } (\ln (\text{of\_nat } (n+1)))) - \text{of\_real } (\ln (\text{of\_nat } (n+2)))$ 
  sums  $(\text{euler\_mascheroni} :: 'a :: \{\text{banach}, \text{real\_normed\_field}\})$ 
  proof -
    have  $(\lambda n. \text{of\_real } (\text{inverse } (\text{of\_nat } (n+1)) + \ln (\text{of\_nat } (n+1)) - \ln (\text{of\_nat } (n+2))))$ 
      sums  $(\text{of\_real } \text{euler\_mascheroni} :: 'a :: \{\text{banach}, \text{real\_normed\_field}\})$ 
    by (subst sums_of_real_iff) (rule euler_mascheroni_sum_real)
    thus ?thesis by simp
  qed

```

```

theorem alternating_harmonic_series_sums:  $(\lambda k. (-1)^k / \text{real\_of\_nat } (\text{Suc } k)) \text{ sums } \ln 2$ 
  proof -
    let ?f =  $\lambda n. \text{harm } n - \ln (\text{real\_of\_nat } n)$ 

```

```

let ?g =  $\lambda n.$  if even  $n$  then 0 else (2::real)
let ?em =  $\lambda n.$  harm  $n$  - ln (real_of_nat  $n$ )
have eventually ( $\lambda n.$  ?em (2*n) - ?em  $n$  + ln 2 = ( $\sum_{k < 2*n} (-1)^k /$ 
real_of_nat (Suc  $k$ ))) at_top
using eventually_gt_at_top[of 0::nat]
proof eventually_elim
  fix n :: nat assume n:  $n > 0$ 
  have ( $\sum_{k < 2*n} (-1)^k /$  real_of_nat (Suc  $k$ )) =
    ( $\sum_{k < 2*n} ((-1)^k + ?g k) /$  of_nat (Suc  $k$ )) - ( $\sum_{k < 2*n} ?g k /$ 
of_nat (Suc  $k$ ))
    by (simp add: sum.distrib algebra_simps divide_inverse)
  also have ( $\sum_{k < 2*n} ((-1)^k + ?g k) /$  real_of_nat (Suc  $k$ )) = harm (2*n)
    unfolding harm_altdef by (intro sum.cong) (auto simp: field_simps)
  also have ( $\sum_{k < 2*n} ?g k /$  real_of_nat (Suc  $k$ )) = ( $\sum_{k | k < 2*n \wedge \text{odd } k} ?g$ 
k / of_nat (Suc  $k$ ))
    by (intro sum.mono_neutral_right) auto
  also have ... = ( $\sum_{k | k < 2*n \wedge \text{odd } k} 2 /$  (real_of_nat (Suc  $k$ )))
    by (intro sum.cong) auto
  also have ( $\sum_{k | k < 2*n \wedge \text{odd } k} 2 /$  (real_of_nat (Suc  $k$ ))) = harm  $n$ 
    unfolding harm_altdef
  by (intro sum.reindex_cong[of  $\lambda n. 2*n+1$ ]) (auto simp: inj_on_def field_simps
elim!: oddE)
  also have harm (2*n) - harm  $n$  = ?em (2*n) - ?em  $n$  + ln 2 using n
    by (simp_all add: algebra_simps ln_mult)
  finally show ?em (2*n) - ?em  $n$  + ln 2 = ( $\sum_{k < 2*n} (-1)^k /$  real_of_nat
(Suc  $k$ )) ..
qed
moreover have ( $\lambda n.$  ?em (2*n) - ?em  $n$  + ln (2::real))
   $\longrightarrow$  euler_mascheroni - euler_mascheroni + ln 2
    by (intro tendsto_intros euler_mascheroni_LIMSEQ filterlim_compose[OF
euler_mascheroni_LIMSEQ]
filterlim_subseq) (auto simp: strict_mono_def)
hence ( $\lambda n.$  ?em (2*n) - ?em  $n$  + ln (2::real))  $\longrightarrow$  ln 2 by simp
ultimately have ( $\lambda n.$  ( $\sum_{k < 2*n} (-1)^k /$  real_of_nat (Suc  $k$ )))  $\longrightarrow$  ln 2
by (blast intro: Lim_transform_eventually)

moreover have summable ( $\lambda k. (-1)^k * \text{inverse} (\text{real\_of\_nat } (\text{Suc } k))$ )
using LIMSEQ_inverse_real_of_nat
by (intro summable_Leibniz(1) decseq_imp_monoseq decseq_SucI) simp_all
hence A: ( $\lambda n.$   $\sum_{k < n} (-1)^k /$  real_of_nat (Suc  $k$ ))  $\longrightarrow$  ( $\sum k. (-1)^k /$ 
real_of_nat (Suc  $k$ ))
by (simp add: summable_sums_iff divide_inverse sums_def)
from filterlim_compose[OF this filterlim_subseq[of (*) (2::nat)]]
have ( $\lambda n.$   $\sum_{k < 2*n} (-1)^k /$  real_of_nat (Suc  $k$ ))  $\longrightarrow$  ( $\sum k. (-1)^k /$ 
real_of_nat (Suc  $k$ ))
by (simp add: strict_mono_def)
ultimately have ( $\sum k. (-1)^k /$  real_of_nat (Suc  $k$ )) = ln 2 by (intro
LIMSEQ_unique)
with A show ?thesis by (simp add: sums_def)

```

2400

qed

lemma *alternating_harmonic_series_sums'*:

$(\lambda k. \text{inverse} (\text{real\_of\_nat} (2*k+1)) - \text{inverse} (\text{real\_of\_nat} (2*k+2))) \text{ sums } \ln 2$

unfolding *sums_def*

proof (*rule Lim_transform_eventually*)

show $(\lambda n. \sum k < 2*n. (-1)^k / (\text{real\_of\_nat} (\text{Suc } k))) \longrightarrow \ln 2$

using *alternating_harmonic_series_sums* **unfolding** *sums_def*

by (*rule filterlim_compose*) (*rule mult_nat_left_at_top, simp*)

show *eventually* $(\lambda n. (\sum k < 2*n. (-1)^k / (\text{real\_of\_nat} (\text{Suc } k))) =$

$(\sum k < n. \text{inverse} (\text{real\_of\_nat} (2*k+1)) - \text{inverse} (\text{real\_of\_nat} (2*k+2)))) \text{ sequentially}$

proof (*intro always_eventually_allI*)

fix $n :: \text{nat}$

show $(\sum k < 2*n. (-1)^k / (\text{real\_of\_nat} (\text{Suc } k))) =$

$(\sum k < n. \text{inverse} (\text{real\_of\_nat} (2*k+1)) - \text{inverse} (\text{real\_of\_nat} (2*k+2)))$

by (*induction n*) (*simp_all add: inverse_eq_divide*)

qed

qed

6.22.3 Bounds on the Euler-Mascheroni constant

lemma *ln_inverse_approx_le*:

assumes $(x :: \text{real}) > 0$ $a > 0$

shows $\ln (x + a) - \ln x \leq a * (\text{inverse } x + \text{inverse} (x + a)) / 2$ (*is _ ≤ ?A*)

proof –

define f' **where** $f' = (\text{inverse} (x + a) - \text{inverse } x) / a$

let $?f = \lambda t. (t - x) * f' + \text{inverse } x$

let $?F = \lambda t. (t - x)^2 * f' / 2 + t * \text{inverse } x$

have *deriv*: $\exists D. ((\lambda x. ?F x - \ln x) \text{ has\_field\_derivative } D) \text{ (at } \xi) \wedge D \geq 0$

if $\xi \geq x$ $\xi \leq x + a$ **for** ξ

proof –

from *that assms* **have** $t: 0 \leq (\xi - x) / a$ $(\xi - x) / a \leq 1$ **by** *simp_all*

have $\text{inverse } \xi = \text{inverse} ((1 - (\xi - x) / a) * x + ((\xi - x) / a) * (x + a))$ (*is _ = ?A*)

using *assms* **by** (*simp add: field_simps*)

also from *assms* **have** *convex_on* $\{x..x+a\}$ *inverse* **by** (*intro convex_on_inverse*) *auto*

from *convex_onD_Icc[OF this _ t]* *assms*

have $?A \leq (1 - (\xi - x) / a) * \text{inverse } x + (\xi - x) / a * \text{inverse} (x + a)$

by *simp*

also have $\dots = (\xi - x) * f' + \text{inverse } x$ **using** *assms*

by (*simp add: f'_def divide_simps*) (*simp add: field_simps*)

finally have $?f \xi - 1 / \xi \geq 0$ **by** (*simp add: field_simps*)

moreover have $((\lambda x. ?F x - \ln x) \text{ has\_field\_derivative } ?f \xi - 1 / \xi) \text{ (at } \xi)$

using *that assms* **by** (*auto intro!: derivative_eq_intros simp: field_simps*)

```

ultimately show ?thesis by blast
qed
have ?F x - ln x ≤ ?F (x + a) - ln (x + a)
  by (rule DERIV_nonneg_imp_nondecreasing[of x x + a, OF _ deriv]) (use
assms in auto)
thus ?thesis
  using assms by (simp add: f'_def divide_simps) (simp add: algebra_simps
power2_eq_square)?
qed

```

```

lemma ln_inverse_approx_ge:
  assumes (x::real) > 0 x < y
  shows ln y - ln x ≥ 2 * (y - x) / (x + y) (is _ ≥ ?A)
proof -
  define m where m = (x+y)/2
  define f' where f' = -inverse (m^2)
  from assms have m: m > 0 by (simp add: m_def)
  let ?F = λt. (t - m)^2 * f' / 2 + t / m
  let ?f = λt. (t - m) * f' + inverse m

  have deriv: ∃ D. ((λx. ln x - ?F x) has_field_derivative D) (at ξ) ∧ D ≥ 0
  if ξ ≥ x ξ ≤ y for ξ
proof -
  from that assms have inverse ξ - inverse m ≥ f' * (ξ - m)
  by (intro convex_on_imp_above_tangent[of {0<..}] convex_on_inverse)
  (auto simp: m_def interior_open f'_def power2_eq_square intro!: deriva-
tive_eq_intros)
  hence 1 / ξ - ?f ξ ≥ 0 by (simp add: field_simps f'_def)
  moreover have ((λx. ln x - ?F x) has_field_derivative 1 / ξ - ?f ξ) (at ξ)
  using that assms m by (auto intro!: derivative_eq_intros simp: field_simps)
  ultimately show ?thesis by blast
qed
have ln x - ?F x ≤ ln y - ?F y
  by (rule DERIV_nonneg_imp_nondecreasing[of x y, OF _ deriv]) (use assms
in auto)
hence ln y - ln x ≥ ?F y - ?F x
  by (simp add: algebra_simps)
also have ?F y - ?F x = ?A
  using assms by (simp add: f'_def m_def divide_simps) (simp add: alge-
bra_simps power2_eq_square)
finally show ?thesis .
qed

```

```

lemma euler_mascheroni_lower:
  euler_mascheroni ≥ harm (Suc n) - ln (real_of_nat (n + 2)) +
1/real_of_nat (2 * (n + 2))
and euler_mascheroni_upper:
  euler_mascheroni ≤ harm (Suc n) - ln (real_of_nat (n + 2)) +
1/real_of_nat (2 * (n + 1))

```

proof –

```

define  $D :: \_ \Rightarrow \text{real}$ 
  where  $D\ n = \text{inverse}\ (\text{of\_nat}\ (n+1)) + \ln\ (\text{of\_nat}\ (n+1)) - \ln\ (\text{of\_nat}\ (n+2))$  for  $n$ 
  let  $?g = \lambda n. \ln\ (\text{of\_nat}\ (n+2)) - \ln\ (\text{of\_nat}\ (n+1)) - \text{inverse}\ (\text{of\_nat}\ (n+1))$ 
 $:: \text{real}$ 
  define  $\text{inv}$  where  $[\text{abs\_def}]: \text{inv}\ n = \text{inverse}\ (\text{real\_of\_nat}\ n)$  for  $n$ 
  fix  $n :: \text{nat}$ 
  note  $\text{summable} = \text{sums\_summable}[OF\ \text{euler\_mascheroni\_sum\_real}, \text{folded}\ D\_def]$ 
  have  $\text{sums}: (\lambda k. (\text{inv}\ (\text{Suc}\ (k + (n+1)))) - \text{inv}\ (\text{Suc}\ (\text{Suc}\ k + (n+1))))/2$   $\text{sums}$ 
 $((\text{inv}\ (\text{Suc}\ (0 + (n+1)))) - 0)/2$ 
    unfolding  $\text{inv\_def}$ 
    by  $(\text{intro}\ \text{sums\_divide}\ \text{telescope\_sums}'\ \text{LIMSEQ\_ignore\_initial\_segment}\ \text{LIMSEQ\_inverse\_real\_of\_nat})$ 
    have  $\text{sums}': (\lambda k. (\text{inv}\ (\text{Suc}\ (k + n)) - \text{inv}\ (\text{Suc}\ (\text{Suc}\ k + n))))/2$   $\text{sums}\ ((\text{inv}\ (\text{Suc}\ (0 + n)) - 0)/2)$ 
      unfolding  $\text{inv\_def}$ 
      by  $(\text{intro}\ \text{sums\_divide}\ \text{telescope\_sums}'\ \text{LIMSEQ\_ignore\_initial\_segment}\ \text{LIMSEQ\_inverse\_real\_of\_nat})$ 
    from  $\text{euler\_mascheroni\_sum\_real}$  have  $\text{euler\_mascheroni} = (\sum k. D\ k)$ 
    by  $(\text{simp}\ \text{add:}\ \text{sums\_iff}\ D\_def)$ 
    also have  $\dots = (\sum k. D\ (k + \text{Suc}\ n)) + (\sum k \leq n. D\ k)$ 
    by  $(\text{subst}\ \text{suminf\_split\_initial\_segment}[OF\ \text{summable},\ \text{of}\ \text{Suc}\ n],$ 
 $\text{subst}\ \text{lessThan\_Suc\_atMost})\ \text{simp}$ 
    finally have  $\text{sum}: (\sum k \leq n. D\ k) - \text{euler\_mascheroni} = -(\sum k. D\ (k + \text{Suc}\ n))$ 
by  $\text{simp}$ 

```

```

  note  $\text{sum}$ 
  also have  $\dots \leq -(\sum k. (\text{inv}\ (k + \text{Suc}\ n + 1) - \text{inv}\ (k + \text{Suc}\ n + 2))) / 2$ 
  proof  $(\text{intro}\ \text{le\_imp\_neg\_le}\ \text{suminf\_le}\ \text{allI}\ \text{summable\_ignore\_initial\_segment}[OF\ \text{summable}])$ 
    fix  $k' :: \text{nat}$ 
    define  $k$  where  $k = k' + \text{Suc}\ n$ 
    hence  $k: k > 0$  by  $(\text{simp}\ \text{add:}\ k\_def)$ 
    have  $\text{real\_of\_nat}\ (k+1) > 0$  by  $(\text{simp}\ \text{add:}\ k\_def)$ 
    with  $\ln\_inverse\_approx\_le[OF\ \text{this}\ \text{zero\_less\_one}]$ 
      have  $\ln\ (\text{of\_nat}\ k + 2) - \ln\ (\text{of\_nat}\ k + 1) \leq (\text{inv}\ (k+1) + \text{inv}\ (k+2))/2$ 
      by  $(\text{simp}\ \text{add:}\ \text{inv\_def}\ \text{add\_ac})$ 
      hence  $(\text{inv}\ (k+1) - \text{inv}\ (k+2))/2 \leq \text{inv}\ (k+1) + \ln\ (\text{of\_nat}\ (k+1)) - \ln\ (\text{of\_nat}\ (k+2))$ 
      by  $(\text{simp}\ \text{add:}\ \text{field\_simps})$ 
      also have  $\dots = D\ k$  unfolding  $D\_def\ \text{inv\_def}\ ..$ 
      finally show  $D\ (k' + \text{Suc}\ n) \geq (\text{inv}\ (k' + \text{Suc}\ n + 1) - \text{inv}\ (k' + \text{Suc}\ n + 2)) / 2$ 
      by  $(\text{simp}\ \text{add:}\ k\_def)$ 
    from  $\text{sums\_summable}[OF\ \text{sums}]$ 
    show  $\text{summable}\ (\lambda k. (\text{inv}\ (k + \text{Suc}\ n + 1) - \text{inv}\ (k + \text{Suc}\ n + 2))/2)$  by
 $\text{simp}$ 
  qed

```

```

also from sums have ... = -inv (n+2) / 2 by (simp add: sums_iff)
finally have euler_mascheroni ≥ (∑ k≤n. D k) + 1 / (of_nat (2 * (n+2)))
  by (simp add: inv_def field_simps)
also have (∑ k≤n. D k) = harm (Suc n) - (∑ k≤n. ln (real_of_nat (Suc k+1))
- ln (of_nat (k+1)))
  unfolding harm_altdef D_def by (subst lessThan_Suc_atMost) (simp add:
sum.distrib sum_subtractf)
also have (∑ k≤n. ln (real_of_nat (Suc k+1)) - ln (of_nat (k+1))) = ln
(of_nat (n+2))
  by (subst atLeast0AtMost [symmetric], subst sum_Suc_diff) simp_all
finally show euler_mascheroni ≥ harm (Suc n) - ln (real_of_nat (n + 2)) +
1 / real_of_nat (2 * (n + 2))
  by simp

note sum
also have -(∑ k. D (k + Suc n)) ≥ -(∑ k. (inv (Suc (k + n)) - inv (Suc (Suc
k + n))))/2
proof (intro le_imp_neg_le suminf_le allI summable_ignore_initial_segment[OF
summable])
  fix k' :: nat
  define k where k = k' + Suc n
  hence k: k > 0 by (simp add: k_def)
  have real_of_nat (k+1) > 0 by (simp add: k_def)
  from ln_inverse_approx_ge[of of_nat k + 1 of_nat k + 2]
  have 2 / (2 * real_of_nat k + 3) ≤ ln (of_nat (k+2)) - ln (real_of_nat
(k+1))
  by (simp add: add_ac)
  hence D k ≤ 1 / real_of_nat (k+1) - 2 / (2 * real_of_nat k + 3)
  by (simp add: D_def inverse_eq_divide inv_def)
  also have ... = inv ((k+1)*(2*k+3)) unfolding inv_def by (simp add:
field_simps)
  also have ... ≤ inv (2*k*(k+1)) unfolding inv_def using k
  by (intro le_imp_inverse_le)
  (simp add: algebra_simps, simp del: of_nat_add)
  also have ... = (inv k - inv (k+1))/2 unfolding inv_def using k
  by (simp add: divide_simps del: of_nat_mult) (simp add: algebra_simps)
  finally show D k ≤ (inv (Suc (k' + n)) - inv (Suc (Suc k' + n)))/2 unfolding
k_def by simp
next
from sums_summable[OF sums']
  show summable (λk. (inv (Suc (k + n)) - inv (Suc (Suc k + n)))/2) by
simp
qed
also from sums' have (∑ k. (inv (Suc (k + n)) - inv (Suc (Suc k + n)))/2)
= inv (n+1)/2
  by (simp add: sums_iff)
finally have euler_mascheroni ≤ (∑ k≤n. D k) + 1 / of_nat (2 * (n+1))
  by (simp add: inv_def field_simps)
also have (∑ k≤n. D k) = harm (Suc n) - (∑ k≤n. ln (real_of_nat (Suc k+1))

```

```

- ln (of_nat (k+1)))
  unfolding harm_altdef D_def by (subst lessThan_Suc_atMost) (simp add:
sum.distrib sum_subtractf)
  also have  $(\sum k \leq n. \ln (\text{real\_of\_nat } (\text{Suc } k+1)) - \ln (\text{of\_nat } (k+1))) = \ln$ 
 $(\text{of\_nat } (n+2))$ 
  by (subst atLeast0AtMost [symmetric], subst sum_Suc_diff) simp_all
  finally show  $\text{euler\_mascheroni} \leq \text{harm } (\text{Suc } n) - \ln (\text{real\_of\_nat } (n + 2)) +$ 
 $1 / \text{real\_of\_nat } (2 * (n + 1))$ 
  by simp
qed

```

```

lemma euler_mascheroni_pos:  $\text{euler\_mascheroni} > (0::\text{real})$ 
  using euler_mascheroni_lower[of 0] ln_2_less_1 by (simp add: harm_def)

```

```

context
begin

```

```

private lemma ln_approx_aux:
  fixes n :: nat and x :: real
  defines  $y \equiv (x-1)/(x+1)$ 
  assumes x:  $x > 0 \ x \neq 1$ 
  shows  $\text{inverse } (2*y^{2*n+1}) * (\ln x - (\sum k < n. 2*y^{2*k+1} / \text{of\_nat } (2*k+1)))$ 
 $\in$ 
 $\{0..(1 / (1 - y^2) / \text{of\_nat } (2*n+1))\}$ 

```

```

proof -
  from x have norm_y:  $\text{norm } y < 1$  unfolding y_def by simp
  from power_strict_mono[OF this, of 2] have norm_y':  $\text{norm } y^2 < 1$  by simp

  let ?f =  $\lambda k. 2 * y^{2*k+1} / \text{of\_nat } (2*k+1)$ 
  note sums = ln_series_quadratic[OF x(1)]
  define c where  $c = \text{inverse } (2*y^{2*n+1})$ 
  let ?d =  $c * (\ln x - (\sum k < n. ?f k))$ 
  have  $\bigwedge k. y^{2*k} / \text{of\_nat } (2*(k+n)+1) \leq y^{2*k} / \text{of\_nat } (2*n+1)$ 
  by (intro divide_left_mono mult_right_mono mult_pos_pos zero_le_power[of
 $y^2$ ]) simp_all
  moreover {
    have  $(\lambda k. ?f (k + n)) \text{ sums } (\ln x - (\sum k < n. ?f k))$ 
    using sums_split_initial_segment[OF sums] by (simp add: y_def)
    hence  $(\lambda k. c * ?f (k + n)) \text{ sums } ?d$  by (rule sums_mult)
    also have  $(\lambda k. c * (2*y^{2*(k+n)+1} / \text{of\_nat } (2*(k+n)+1))) =$ 
 $(\lambda k. (c * (2*y^{2*n+1})) * ((y^2)^k / \text{of\_nat } (2*(k+n)+1)))$ 
    by (simp only: ring_distrib power_add power_mult) (simp add: mult_ac)
    also from x have  $c * (2*y^{2*n+1}) = 1$  by (simp add: c_def y_def)
    finally have  $(\lambda k. (y^2)^k / \text{of\_nat } (2*(k+n)+1)) \text{ sums } ?d$  by simp
  } note sums' = this
  moreover from norm_y' have  $(\lambda k. (y^2)^k / \text{of\_nat } (2*n+1)) \text{ sums } (1 / (1$ 
 $- y^2) / \text{of\_nat } (2*n+1))$ 
  by (intro sums_divide geometric_sums) (simp_all add: norm_power)
  ultimately have  $?d \leq (1 / (1 - y^2) / \text{of\_nat } (2*n+1))$  by (rule sums_le)

```



```

    moreover have  $c * (\ln x - (\sum_{k < n}. 2 * y^{(2 * k + 1)} / \text{real\_of\_nat } (2 * k + 1))) \geq 0$ 
    by (intro sums_le[OF sums_zero sums]) simp_all
    ultimately show ?thesis unfolding c_def by simp
qed

```

lemma

```

    fixes  $n :: \text{nat}$  and  $x :: \text{real}$ 
    defines  $y \equiv (x-1)/(x+1)$ 
    defines  $\text{approx} \equiv (\sum_{k < n}. 2 * y^{(2 * k + 1)} / \text{of\_nat } (2 * k + 1))$ 
    defines  $d \equiv y^{(2 * n + 1)} / (1 - y^2) / \text{of\_nat } (2 * n + 1)$ 
    assumes  $x: x > 1$ 
    shows  $\text{ln\_approx\_bounds}: \ln x \in \{\text{approx}.. \text{approx} + 2 * d\}$ 
    and  $\text{ln\_approx\_abs}: \text{abs } (\ln x - (\text{approx} + d)) \leq d$ 
  proof -
    define  $c$  where  $c = 2 * y^{(2 * n + 1)}$ 
    from  $x$  have  $c\_pos: c > 0$  unfolding c_def y_def
    by (intro mult_pos_pos zero_less_power) simp_all
    have  $A: \text{inverse } c * (\ln x - (\sum_{k < n}. 2 * y^{(2 * k + 1)} / \text{of\_nat } (2 * k + 1))) \in$ 
       $\{0.. (1 / (1 - y^2) / \text{of\_nat } (2 * n + 1))\}$  using assms unfolding
    y_def c_def
    by (intro ln_approx_aux) simp_all
    hence  $\text{inverse } c * (\ln x - (\sum_{k < n}. 2 * y^{(2 * k + 1)} / \text{of\_nat } (2 * k + 1))) \leq (1 /$ 
       $(1 - y^2) / \text{of\_nat } (2 * n + 1))$ 
    by simp
    hence  $(\ln x - (\sum_{k < n}. 2 * y^{(2 * k + 1)} / \text{of\_nat } (2 * k + 1))) / c \leq (1 / (1 -$ 
       $y^2) / \text{of\_nat } (2 * n + 1))$ 
    by (auto simp add: field_split_simps)
    with c_pos have  $\ln x \leq c / (1 - y^2) / \text{of\_nat } (2 * n + 1) + \text{approx}$ 
    by (subst (asm) pos_divide_le_eq) (simp_all add: mult_ac approx_def)
    moreover {
      from  $A$  c_pos have  $0 \leq c * (\text{inverse } c * (\ln x - (\sum_{k < n}. 2 * y^{(2 * k + 1)} /$ 
       $\text{of\_nat } (2 * k + 1))))$ 
      by (intro mult_nonneg_nonneg[of c]) simp_all
      also have  $\dots = (c * \text{inverse } c) * (\ln x - (\sum_{k < n}. 2 * y^{(2 * k + 1)} / \text{of\_nat}$ 
       $(2 * k + 1)))$ 
      by (simp add: mult_ac)
      also from c_pos have  $c * \text{inverse } c = 1$  by simp
      finally have  $\ln x \geq \text{approx}$  by (simp add: approx_def)
    }
    ultimately show  $\ln x \in \{\text{approx}.. \text{approx} + 2 * d\}$  by (simp add: c_def d_def)
    thus  $\text{abs } (\ln x - (\text{approx} + d)) \leq d$  by auto
  qed

```

end

lemma euler_mascheroni_bounds:

```

    fixes  $n :: \text{nat}$  assumes  $n \geq 1$  defines  $t \equiv \text{harm } n - \ln (\text{of\_nat } (\text{Suc } n)) :: \text{real}$ 
    shows  $\text{euler\_mascheroni} \in \{t + \text{inverse } (\text{of\_nat } (2 * (n + 1))).. t + \text{inverse } (\text{of\_nat}$ 

```

```

(2*n))}
using assms euler_mascheroni_upper[of n-1] euler_mascheroni_lower[of n-1]
unfolding t_def by (cases n) (simp_all add: harm_Suc t_def inverse_eq_divide)

```

```

lemma euler_mascheroni_bounds':
  fixes n :: nat assumes n ≥ 1 ln (real_of_nat (Suc n)) ∈ {l<..u}
  shows euler_mascheroni ∈
    {harm n - u + inverse (of_nat (2*(n+1)))<..harm n - l + inverse
(of_nat (2*n))}
  using euler_mascheroni_bounds[OF assms(1)] assms(2) by auto

```

Approximation of $\ln(2::'a)$. The lower bound is accurate to about 0.03; the upper bound is accurate to about 0.0015.

```

lemma ln2_ge_two_thirds:  $2/3 \leq \ln(2::\text{real})$ 
  and ln2_le_25_over_36:  $\ln(2::\text{real}) \leq 25/36$ 
  using ln_approx_bounds[of 2 1, simplified, simplified eval_nat_numeral, simplified] by simp_all

```

Approximation of the Euler-Mascheroni constant. The lower bound is accurate to about 0.0015; the upper bound is accurate to about 0.015.

```

lemma euler_mascheroni_gt_19_over_33: (euler_mascheroni :: real) > 19/33
(is ?th1)
  and euler_mascheroni_less_13_over_22: (euler_mascheroni :: real) < 13/22
(is ?th2)
proof -
  have  $\ln(\text{real}(\text{Suc } 7)) = 3 * \ln 2$  by (simp add: ln_powr [symmetric])
  also from ln_approx_bounds[of 2 3] have ... ∈ {3*307/443<..3*4615/6658}
  by (simp add: eval_nat_numeral)
  finally have  $\ln(\text{real}(\text{Suc } 7)) \in \dots$ 
  from euler_mascheroni_bounds'[OF _ this] have ?th1 ∧ ?th2 by (simp_all add: harm_expand)
  thus ?th1 ?th2 by blast+
qed

```

end

6.23 The Gamma Function

```

theory Gamma_Function
imports
  Equivalence_Lebesgue_Henstock_Integration
  Summation_Tests
  Harmonic_Numbers
  HOL-Library.Nonpos_Ints
  HOL-Library.Periodic_Fun
begin

```

Several equivalent definitions of the Gamma function and its most important properties. Also contains the definition and some properties of the

log-Gamma function and the Digamma function and the other Polygamma functions.

Based on the Gamma function, we also prove the Weierstraß product form of the sin function and, based on this, the solution of the Basel problem (the sum over all $1 / \text{real } (n^2)$).

lemma *pochhammer_eq_0_imp_nonpos_Int*:
 $\text{pochhammer } (x::'a::\text{field\_char\_0}) \ n = 0 \implies x \in \mathbb{Z}_{\leq 0}$
by (auto simp: pochhammer_eq_0_iff)

lemma *closed_nonpos_Ints [simp]*: $\text{closed } (\mathbb{Z}_{\leq 0} :: 'a :: \text{real\_normed\_algebra\_1 set})$

proof –

have $\mathbb{Z}_{\leq 0} = (\text{of\_int } ' \{n. n \leq 0\} :: 'a \text{ set})$
by (auto elim!: nonpos_Ints_cases intro!: nonpos_Ints_of_int)
also have $\text{closed } \dots$ **by** (rule closed_of_int_image)
finally show ?thesis .

qed

lemma *plus_one_in_nonpos_Ints_imp*: $z + 1 \in \mathbb{Z}_{\leq 0} \implies z \in \mathbb{Z}_{\leq 0}$
using nonpos_Ints_diff_Nats[of z+1 1] **by** simp_all

lemma *of_int_in_nonpos_Ints_iff*:
 $(\text{of\_int } n :: 'a :: \text{ring\_char\_0}) \in \mathbb{Z}_{\leq 0} \longleftrightarrow n \leq 0$
by (auto simp: nonpos_Ints_def)

lemma *one_plus_of_int_in_nonpos_Ints_iff*:
 $(1 + \text{of\_int } n :: 'a :: \text{ring\_char\_0}) \in \mathbb{Z}_{\leq 0} \longleftrightarrow n \leq -1$

proof –

have $1 + \text{of\_int } n = (\text{of\_int } (n + 1) :: 'a)$ **by** simp
also have $\dots \in \mathbb{Z}_{\leq 0} \longleftrightarrow n + 1 \leq 0$ **by** (subst of_int_in_nonpos_Ints_iff) simp_all
also have $\dots \longleftrightarrow n \leq -1$ **by** presburger
finally show ?thesis .

qed

lemma *one_minus_of_nat_in_nonpos_Ints_iff*:
 $(1 - \text{of\_nat } n :: 'a :: \text{ring\_char\_0}) \in \mathbb{Z}_{\leq 0} \longleftrightarrow n > 0$

proof –

have $(1 - \text{of\_nat } n :: 'a) = \text{of\_int } (1 - \text{int } n)$ **by** simp
also have $\dots \in \mathbb{Z}_{\leq 0} \longleftrightarrow n > 0$ **by** (subst of_int_in_nonpos_Ints_iff) presburger
finally show ?thesis .

qed

lemma *fraction_not_in_ints*:
assumes $\neg(n \text{ dvd } m) \ n \neq 0$
shows $\text{of\_int } m / \text{of\_int } n \notin (\mathbb{Z} :: 'a :: \{\text{division\_ring}, \text{ring\_char\_0}\} \text{ set})$
proof

assume $of\_int\ m / (of\_int\ n :: 'a) \in \mathbb{Z}$
 then obtain k where $of\_int\ m / of\_int\ n = (of\_int\ k :: 'a)$ by (elim Ints_cases)
 with assms have $of\_int\ m = (of\_int\ (k * n) :: 'a)$ by (auto simp add: field_split_simps)
 hence $m = k * n$ by (subst (asm) of_int_eq_iff)
 hence $n \text{ dvd } m$ by simp
 with assms(1) show False by contradiction
 qed

lemma fraction_not_in_nats:
 assumes $\neg n \text{ dvd } m \ n \neq 0$
 shows $of\_int\ m / of\_int\ n \notin (\mathbb{N} :: 'a :: \{\text{division\_ring, ring\_char\_0}\} \text{ set})$
 proof
 assume $of\_int\ m / of\_int\ n \in (\mathbb{N} :: 'a \text{ set})$
 also note Nats_subset_Ints
 finally have $of\_int\ m / of\_int\ n \in (\mathbb{Z} :: 'a \text{ set})$.
 moreover have $of\_int\ m / of\_int\ n \notin (\mathbb{Z} :: 'a \text{ set})$
 using assms by (intro fraction_not_in_ints)
 ultimately show False by contradiction
 qed

lemma not_in_Ints_imp_not_in_nonpos_Ints: $z \notin \mathbb{Z} \implies z \notin \mathbb{Z}_{\leq 0}$
 by (auto simp: Ints_def nonpos_Ints_def)

lemma double_in_nonpos_Ints_imp:
 assumes $2 * (z :: 'a :: \text{field\_char\_0}) \in \mathbb{Z}_{\leq 0}$
 shows $z \in \mathbb{Z}_{\leq 0} \vee z + 1/2 \in \mathbb{Z}_{\leq 0}$
 proof -
 from assms obtain k where $k: 2 * z = - of\_nat\ k$ by (elim nonpos_Ints_cases')
 thus ?thesis by (cases even k) (auto elim!: evenE oddE simp: field_simps)
 qed

lemma sin_series: $(\lambda n. ((-1)^n / \text{fact } (2*n+1)) * _R\ z^{(2*n+1)}) \text{ sums } \sin z$
 proof -
 from sin_converges[of z] have $(\lambda n. \sin\_coeff\ n * _R\ z^n) \text{ sums } \sin z$.
 also have $(\lambda n. \sin\_coeff\ n * _R\ z^n) \text{ sums } \sin z \longleftrightarrow$
 $(\lambda n. ((-1)^n / \text{fact } (2*n+1)) * _R\ z^{(2*n+1)}) \text{ sums } \sin z$
 by (subst sums_mono_reindex[of $\lambda n. 2*n+1$, symmetric])
 (auto simp: sin_coeff_def strict_mono_def ac_simps elim!: oddE)
 finally show ?thesis.
 qed

lemma cos_series: $(\lambda n. ((-1)^n / \text{fact } (2*n)) * _R\ z^{(2*n)}) \text{ sums } \cos z$
 proof -
 from cos_converges[of z] have $(\lambda n. \cos\_coeff\ n * _R\ z^n) \text{ sums } \cos z$.
 also have $(\lambda n. \cos\_coeff\ n * _R\ z^n) \text{ sums } \cos z \longleftrightarrow$
 $(\lambda n. ((-1)^n / \text{fact } (2*n)) * _R\ z^{(2*n)}) \text{ sums } \cos z$
 by (subst sums_mono_reindex[of $\lambda n. 2*n$, symmetric])
 (auto simp: cos_coeff_def strict_mono_def ac_simps elim!: evenE)

finally show ?thesis .
qed

lemma sin_z_over_z_series:
fixes z :: 'a :: {real_normed_field,banach}
assumes z $\neq$ 0
shows $(\lambda n. (-1)^n / \text{fact } (2*n+1) * z^{(2*n)}) \text{ sums } (\sin z / z)$
proof -
from sin_series[of z] have $(\lambda n. z * ((-1)^n / \text{fact } (2*n+1)) * z^{(2*n)}) \text{ sums } \sin z$
by (simp add: field_simps scaleR_conv_of_real)
from sums_mult[OF this, of inverse z] and assms show ?thesis
by (simp add: field_simps)
qed

lemma sin_z_over_z_series':
fixes z :: 'a :: {real_normed_field,banach}
assumes z $\neq$ 0
shows $(\lambda n. \sin\_coeff (n+1) * z^n) \text{ sums } (\sin z / z)$
proof -
from sums_split_initial_segment[OF sin_converges[of z], of 1]
have $(\lambda n. z * (\sin\_coeff (n+1) * z^n)) \text{ sums } \sin z$ by simp
from sums_mult[OF this, of inverse z] assms show ?thesis by (simp add: field_simps)
qed

lemma has_field_derivative_sin_z_over_z:
fixes A :: 'a :: {real_normed_field,banach} set
shows $((\lambda z. \text{if } z = 0 \text{ then } 1 \text{ else } \sin z / z) \text{ has\_field\_derivative } 0) \text{ (at } 0 \text{ within } A)$
 $(\text{is } (?f \text{ has\_field\_derivative } ?f') \_)$
proof (rule has_field_derivative_at_within)
have $((\lambda z::'a. \sum n. \text{of\_real } (\sin\_coeff (n+1)) * z^n) \text{ has\_field\_derivative } (\sum n. \text{diffs } (\lambda n. \text{of\_real } (\sin\_coeff (n+1))) n * 0^n)) \text{ (at } 0)$
proof (rule termdiffs_strong)
from summable_ignore_initial_segment[OF sums_summable[OF sin_converges[of 1::'a]], of 1]
show summable $(\lambda n. \text{of\_real } (\sin\_coeff (n+1)) * (1::'a)^n)$ by (simp add: of_real_def)
qed simp
also have $(\lambda z::'a. \sum n. \text{of\_real } (\sin\_coeff (n+1)) * z^n) = ?f$
proof
fix z
show $(\sum n. \text{of\_real } (\sin\_coeff (n+1)) * z^n) = ?f z$
by (cases z = 0) (insert sin_z_over_z_series'[of z],
simp_all add: scaleR_conv_of_real sums_iff sin_coeff_def)
qed
also have $(\sum n. \text{diffs } (\lambda n. \text{of\_real } (\sin\_coeff (n+1))) n * (0::'a)^n) =$

$\text{diffs } (\lambda n. \text{ of\_real } (\text{sin\_coeff } (\text{Suc } n))) \ 0 \text{ by simp}$
also have $\dots = 0$ **by** $(\text{simp add: sin\_coeff\_def diffs\_def})$
finally show $((\lambda z::'a. \text{ if } z = 0 \text{ then } 1 \text{ else } \text{sin } z / z) \text{ has\_field\_derivative } 0)$ $(\text{at } 0)$.
qed

lemma *round_Re_minimises_norm*:
 $\text{norm } ((z::\text{complex}) - \text{of\_int } m) \geq \text{norm } (z - \text{of\_int } (\text{round } (\text{Re } z)))$
proof –
let $?n = \text{round } (\text{Re } z)$
have $\text{norm } (z - \text{of\_int } ?n) = \text{sqrt } ((\text{Re } z - \text{of\_int } ?n)^2 + (\text{Im } z)^2)$
by $(\text{simp add: cmod\_def})$
also have $|\text{Re } z - \text{of\_int } ?n| \leq |\text{Re } z - \text{of\_int } m|$ **by** $(\text{rule round\_diff\_minimal})$
hence $\text{sqrt } ((\text{Re } z - \text{of\_int } ?n)^2 + (\text{Im } z)^2) \leq \text{sqrt } ((\text{Re } z - \text{of\_int } m)^2 + (\text{Im } z)^2)$
by $(\text{intro real\_sqrt\_le\_mono add\_mono})$ $(\text{simp\_all add: abs\_le\_square\_iff})$
also have $\dots = \text{norm } (z - \text{of\_int } m)$ **by** $(\text{simp add: cmod\_def})$
finally show $?thesis$.
qed

lemma *Re_pos_in_ball*:
assumes $\text{Re } z > 0 \ t \in \text{ball } z \ (\text{Re } z / 2)$
shows $\text{Re } t > 0$
proof –
have $\text{Re } (z - t) \leq \text{norm } (z - t)$ **by** $(\text{rule complex\_Re\_le\_cmod})$
also from *assms* **have** $\dots < \text{Re } z / 2$ **by** $(\text{simp add: dist\_complex\_def})$
finally show $\text{Re } t > 0$ **using** *assms* **by** *simp*
qed

lemma *no_nonpos_Int_in_ball_complex*:
assumes $\text{Re } z > 0 \ t \in \text{ball } z \ (\text{Re } z / 2)$
shows $t \notin \mathbb{Z}_{\leq 0}$
using *Re_pos_in_ball*[*OF* *assms*] **by** $(\text{force elim!: nonpos\_Ints\_cases})$

lemma *no_nonpos_Int_in_ball*:
assumes $t \in \text{ball } z \ (\text{dist } z \ (\text{round } (\text{Re } z)))$
shows $t \notin \mathbb{Z}_{\leq 0}$
proof
assume $t \in \mathbb{Z}_{\leq 0}$
then obtain n **where** $t = \text{of\_int } n$ **by** $(\text{auto elim!: nonpos\_Ints\_cases})$
have $\text{dist } z \ (\text{of\_int } n) \leq \text{dist } z \ t + \text{dist } t \ (\text{of\_int } n)$ **by** $(\text{rule dist\_triangle})$
also from *assms* **have** $\text{dist } z \ t < \text{dist } z \ (\text{round } (\text{Re } z))$ **by** *simp*
also have $\dots \leq \text{dist } z \ (\text{of\_int } n)$
using *round_Re_minimises_norm*[*of* z] **by** $(\text{simp add: dist\_complex\_def})$
finally have $\text{dist } t \ (\text{of\_int } n) > 0$ **by** *simp*
with $\langle t = \text{of\_int } n \rangle$ **show** *False* **by** *simp*
qed

lemma *no_nonpos_Int_in_ball'*:

```

    assumes (z :: 'a :: {euclidean_space, real_normed_algebra_1})  $\notin \mathbb{Z}_{\leq 0}$ 
    obtains d where  $d > 0 \wedge t. t \in \text{ball } z \ d \implies t \notin \mathbb{Z}_{\leq 0}$ 
  proof (rule that)
    from assms show  $\text{setdist } \{z\} \ \mathbb{Z}_{\leq 0} > 0$  by (subst setdist_gt_0_compact_closed)
  auto
  next
    fix t assume  $t \in \text{ball } z \ (\text{setdist } \{z\} \ \mathbb{Z}_{\leq 0})$ 
    thus  $t \notin \mathbb{Z}_{\leq 0}$  using setdist_le_dist[of z {z} t  $\mathbb{Z}_{\leq 0}$ ] by force
  qed

  lemma no_nonpos_Real_in_ball:
    assumes z:  $z \notin \mathbb{R}_{\leq 0}$  and t:  $t \in \text{ball } z \ (\text{if } \text{Im } z = 0 \text{ then } \text{Re } z / 2 \text{ else } \text{abs } (\text{Im } z) / 2)$ 
    shows  $t \notin \mathbb{R}_{\leq 0}$ 
  using z
  proof (cases  $\text{Im } z = 0$ )
    assume A:  $\text{Im } z = 0$ 
    with z have  $\text{Re } z > 0$  by (force simp add: complex_nonpos_Reals_iff)
    with t A Re_pos_in_ball[of z t] show ?thesis by (force simp add: complex_nonpos_Reals_iff)
  next
    assume A:  $\text{Im } z \neq 0$ 
    have  $\text{abs } (\text{Im } z) - \text{abs } (\text{Im } t) \leq \text{abs } (\text{Im } z - \text{Im } t)$  by linarith
    also have  $\dots = \text{abs } (\text{Im } (z - t))$  by simp
    also have  $\dots \leq \text{norm } (z - t)$  by (rule abs_Im_le_cmod)
    also from A t have  $\dots \leq \text{abs } (\text{Im } z) / 2$  by (simp add: dist_complex_def)
    finally have  $\text{abs } (\text{Im } t) > 0$  using A by simp
    thus ?thesis by (force simp add: complex_nonpos_Reals_iff)
  qed

```

6.23.1 The Euler form and the logarithmic Gamma function

We define the Gamma function by first defining its multiplicative inverse $rGamma$. This is more convenient because $rGamma$ is entire, which makes proofs of its properties more convenient because one does not have to watch out for discontinuities. (e.g. $rGamma$ fulfils $rGamma \ z = z * rGamma \ (z + 1)$ everywhere, whereas the Γ function does not fulfil the analogous equation on the non-positive integers)

We define the Γ function (resp. its reciprocal) in the Euler form. This form has the advantage that it is a relatively simple limit that converges everywhere. The limit at the poles is 0 (due to division by 0). The functional equation $\Gamma \ (z + 1) = z * \Gamma \ z$ follows immediately from the definition.

definition $\Gamma\_series :: ('a :: \{banach, real\_normed\_field\}) \Rightarrow nat \Rightarrow 'a$ where
 $\Gamma\_series \ z \ n = \text{fact } n * \exp \ (z * \text{of\_real } (\ln \ (\text{of\_nat } n))) / \text{pochhammer } z \ (n+1)$

definition $\Gamma\_series' :: ('a :: \{banach, real\_normed\_field\}) \Rightarrow nat \Rightarrow 'a$

where

$\text{Gamma\_series}' z n = \text{fact } (n - 1) * \exp (z * \text{of\_real } (\ln (\text{of\_nat } n))) / \text{pochhammer } z n$

definition $\text{rGamma\_series} :: ('a :: \{\text{banach}, \text{real\_normed\_field}\}) \Rightarrow \text{nat} \Rightarrow 'a$
 where

$\text{rGamma\_series } z n = \text{pochhammer } z (n+1) / (\text{fact } n * \exp (z * \text{of\_real } (\ln (\text{of\_nat } n))))$

lemma $\text{Gamma\_series\_altdef}$: $\text{Gamma\_series } z n = \text{inverse } (\text{rGamma\_series } z n)$

and $\text{rGamma\_series\_altdef}$: $\text{rGamma\_series } z n = \text{inverse } (\text{Gamma\_series } z n)$

unfolding Gamma_series_def rGamma_series_def **by** simp_all

lemma $\text{rGamma\_series\_minus\_of\_nat}$:

$\text{eventually } (\lambda n. \text{rGamma\_series } (- \text{of\_nat } k) n = 0) \text{ sequentially}$

using $\text{eventually\_ge\_at\_top}[of k]$

by $\text{eventually\_elim } (\text{auto simp: rGamma\_series\_def pochhammer\_of\_nat\_eq\_0\_iff})$

lemma $\text{Gamma\_series\_minus\_of\_nat}$:

$\text{eventually } (\lambda n. \text{Gamma\_series } (- \text{of\_nat } k) n = 0) \text{ sequentially}$

using $\text{eventually\_ge\_at\_top}[of k]$

by $\text{eventually\_elim } (\text{auto simp: Gamma\_series\_def pochhammer\_of\_nat\_eq\_0\_iff})$

lemma $\text{Gamma\_series}'\_minus\_of\_nat$:

$\text{eventually } (\lambda n. \text{Gamma\_series}' (- \text{of\_nat } k) n = 0) \text{ sequentially}$

using $\text{eventually\_gt\_at\_top}[of k]$

by $\text{eventually\_elim } (\text{auto simp: Gamma\_series'\_def pochhammer\_of\_nat\_eq\_0\_iff})$

lemma $\text{rGamma\_series\_nonpos\_Ints\_LIMSEQ}$: $z \in \mathbb{Z}_{\leq 0} \implies \text{rGamma\_series } z \longrightarrow 0$

by $(\text{elim nonpos\_Ints\_cases}', \text{hypsubst}, \text{subst tendsto\_cong}, \text{rule rGamma\_series\_minus\_of\_nat}, \text{simp})$

lemma $\text{Gamma\_series\_nonpos\_Ints\_LIMSEQ}$: $z \in \mathbb{Z}_{\leq 0} \implies \text{Gamma\_series } z \longrightarrow 0$

by $(\text{elim nonpos\_Ints\_cases}', \text{hypsubst}, \text{subst tendsto\_cong}, \text{rule Gamma\_series\_minus\_of\_nat}, \text{simp})$

lemma $\text{Gamma\_series}'\_nonpos\_Ints\_LIMSEQ$: $z \in \mathbb{Z}_{\leq 0} \implies \text{Gamma\_series}' z \longrightarrow 0$

by $(\text{elim nonpos\_Ints\_cases}', \text{hypsubst}, \text{subst tendsto\_cong}, \text{rule Gamma\_series}'\_minus\_of\_nat}, \text{simp})$

lemma $\text{Gamma\_series\_Gamma\_series}'$:

assumes $z: z \notin \mathbb{Z}_{\leq 0}$

shows $(\lambda n. \text{Gamma\_series}' z n / \text{Gamma\_series } z n) \longrightarrow 1$

proof $(\text{rule Lim\_transform\_eventually})$


```

from eventually_gt_at_top[of 0::nat]
show eventually ( $\lambda n. z / \text{of\_nat } n + 1 = \text{Gamma\_series}' z n / \text{Gamma\_series}$ 
 $z n$ ) sequentially
proof eventually_elim
  fix  $n :: \text{nat}$  assume  $n > 0$ 
  from  $n z$  have  $\text{Gamma\_series}' z n / \text{Gamma\_series } z n = (z + \text{of\_nat } n) /$ 
 $\text{of\_nat } n$ 
  by (cases  $n$ , simp)
  (auto simp add: Gamma_series_def Gamma_series'_def pochhammer_rec'
    dest: pochhammer_eq_0_imp_nonpos_Int_plus_of_nat_eq_0_imp)
  also from  $n$  have  $\dots = z / \text{of\_nat } n + 1$  by (simp add: field_split_simps)
  finally show  $z / \text{of\_nat } n + 1 = \text{Gamma\_series}' z n / \text{Gamma\_series } z n$  ..
qed
have ( $\lambda x. z / \text{of\_nat } x$ )  $\longrightarrow 0$ 
by (rule tendsto_norm_zero_cancel)
  (insert tendsto_mult[OF tendsto_const[of norm  $z$ ] lim_inverse_n],
    simp add: norm_divide inverse_eq_divide)
from tendsto_add[OF this tendsto_const[of 1]] show ( $\lambda n. z / \text{of\_nat } n + 1$ )
 $\longrightarrow 1$  by simp
qed

```

We now show that the series that defines the Γ function in the Euler form converges and that the function defined by it is continuous on the complex halfspace with positive real part.

We do this by showing that the logarithm of the Euler series is continuous and converges locally uniformly, which means that the log-Gamma function defined by its limit is also continuous.

This will later allow us to lift holomorphicity and continuity from the log-Gamma function to the inverse of the Gamma function, and from that to the Gamma function itself.

definition $\text{ln\_Gamma\_series} :: ('a :: \{\text{banach, real\_normed\_field}, \text{ln}\}) \Rightarrow \text{nat} \Rightarrow 'a$ **where**
 $\text{ln\_Gamma\_series } z n = z * \text{ln } (\text{of\_nat } n) - \text{ln } z - (\sum k=1..n. \text{ln } (z / \text{of\_nat } k + 1))$

definition $\text{ln\_Gamma\_series}' :: ('a :: \{\text{banach, real\_normed\_field}, \text{ln}\}) \Rightarrow \text{nat} \Rightarrow 'a$ **where**
 $\text{ln\_Gamma\_series}' z n =$
 $- \text{euler\_mascheroni} * z - \text{ln } z + (\sum k=1..n. z / \text{of\_nat } n - \text{ln } (z / \text{of\_nat } k + 1))$

definition $\text{ln\_Gamma} :: ('a :: \{\text{banach, real\_normed\_field}, \text{ln}\}) \Rightarrow 'a$ **where**
 $\text{ln\_Gamma } z = \text{lim } (\text{ln\_Gamma\_series } z)$

We now show that the log-Gamma series converges locally uniformly for all complex numbers except the non-positive integers. We do this by proving that the series is locally Cauchy.

context

begin

```

private lemma ln_Gamma_series_complex_converges_aux:
  fixes z :: complex and k :: nat
  assumes z: z ≠ 0 and k: of_nat k ≥ 2 * norm z k ≥ 2
  shows norm (z * ln (1 - 1 / of_nat k) + ln (z / of_nat k + 1)) ≤ 2 * (norm z +
norm z ^ 2) / of_nat k ^ 2
proof -
  let ?k = of_nat k :: complex and ?z = norm z
  have z * ln (1 - 1 / ?k) + ln (z / ?k + 1) = z * (ln (1 - 1 / ?k :: complex) + 1 / ?k)
+ (ln (1 + z / ?k) - z / ?k)
  by (simp add: algebra_simps)
  also have norm ... ≤ ?z * norm (ln (1 - 1 / ?k) + 1 / ?k :: complex) + norm (ln
(1 + z / ?k) - z / ?k)
  by (subst norm_mult [symmetric], rule norm_triangle_ineq)
  also have norm (Ln (1 + -1 / ?k) - (-1 / ?k)) ≤ (norm (-1 / ?k)) ^ 2 / (1 -
norm (-1 / ?k))
  using k by (intro Ln_approx_linear) (simp add: norm_divide)
  hence ?z * norm (ln (1 - 1 / ?k) + 1 / ?k) ≤ ?z * ((norm (1 / ?k)) ^ 2 / (1 - norm
(1 / ?k)))
  by (intro mult_left_mono) simp_all
  also have ... ≤ (?z * (of_nat k / (of_nat k - 1))) / of_nat k ^ 2 using k
  by (simp add: field_simps power2_eq_square norm_divide)
  also have ... ≤ (?z * 2) / of_nat k ^ 2 using k
  by (intro divide_right_mono mult_left_mono) (simp_all add: field_simps)
  also have norm (ln (1 + z / ?k) - z / ?k) ≤ norm (z / ?k) ^ 2 / (1 - norm (z / ?k))
using k
  by (intro Ln_approx_linear) (simp add: norm_divide)
  hence norm (ln (1 + z / ?k) - z / ?k) ≤ ?z ^ 2 / of_nat k ^ 2 / (1 - ?z / of_nat k)
  by (simp add: field_simps norm_divide)
  also have ... ≤ (?z ^ 2 * (of_nat k / (of_nat k - ?z))) / of_nat k ^ 2 using k
  by (simp add: field_simps power2_eq_square)
  also have ... ≤ (?z ^ 2 * 2) / of_nat k ^ 2 using k
  by (intro divide_right_mono mult_left_mono) (simp_all add: field_simps)
  also note add_divide_distrib [symmetric]
  finally show ?thesis by (simp only: distrib_left mult.commute)
qed

```

```

lemma ln_Gamma_series_complex_converges:
  assumes z: z ∉ ℤ≤0
  assumes d: d > 0 ∧ n. n ∈ ℤ≤0 ⇒ norm (z - of_int n) > d
  shows uniformly_convergent_on (ball z d) (λn z. ln_Gamma_series z n :: com-
plex)
proof (intro Cauchy_uniformly_convergent uniformly_Cauchy_onI')
  fix e :: real assume e: e > 0
  define e'' where e'' = (SUP t ∈ ball z d. norm t + norm t ^ 2)
  define e' where e' = e / (2 * e'')
  have bounded ((λt. norm t + norm t ^ 2) ` cball z d)
  by (intro compact_imp_bounded compact_continuous_image) (auto intro!: con-

```

```

tinuous intros)
  hence bounded (( $\lambda t. \text{norm } t + \text{norm } t^2$ ) ' ball z d) by (rule bounded_subset)
auto
  hence bdd: bdd_above (( $\lambda t. \text{norm } t + \text{norm } t^2$ ) ' ball z d) by (rule bounded_imp_bdd_above)

  with z d(1) d(2)[of -1] have e''_pos:  $e'' > 0$  unfolding e''_def
  by (subst less_cSUP_iff) (auto intro!: add_pos_nonneg bexI[of _ z])
  have e'':  $\text{norm } t + \text{norm } t^2 \leq e''$  if  $t \in \text{ball } z d$  for t unfolding e''_def using
that
  by (rule cSUP_upper[OF _ bdd])
  from e z e''_pos have e':  $e' > 0$  unfolding e'_def
  by (intro divide_pos_pos mult_pos_pos add_pos_pos) (simp_all add: field_simps)

  have summable ( $\lambda k. \text{inverse } ((\text{real\_of\_nat } k)^2)$ )
  by (rule inverse_power_summable) simp
  from summable_partial_sum_bound[OF this e']
  obtain M where M:  $\bigwedge m n. M \leq m \implies \text{norm } (\sum k = m..n. \text{inverse } ((\text{real } k)^2))$ 
<  $e'$ 
  by auto

define N where  $N = \max 2 (\max (\text{nat } \lceil 2 * (\text{norm } z + d) \rceil) M)$ 
{
  from d have  $\lceil 2 * (\text{cmod } z + d) \rceil \geq \lceil 0::\text{real} \rceil$ 
  by (intro ceiling_mono mult_nonneg_nonneg add_nonneg_nonneg) simp_all
  hence  $2 * (\text{norm } z + d) \leq \text{of\_nat } (\text{nat } \lceil 2 * (\text{norm } z + d) \rceil)$  unfolding N_def
  by (simp_all)
  also have  $\dots \leq \text{of\_nat } N$  unfolding N_def
  by (subst of_nat_le_iff) (rule max.coboundedI2, rule max.cobounded1)
  finally have  $\text{of\_nat } N \geq 2 * (\text{norm } z + d)$  .
  moreover have  $N \geq 2$   $N \geq M$  unfolding N_def by simp_all
  moreover have  $(\sum k=m..n. 1/(\text{of\_nat } k)^2) < e'$  if  $m \geq N$  for m n
  using M[OF order.trans[OF  $\langle N \geq m \rangle$  that]] unfolding real_norm_def
  by (subst (asm) abs_of_nonneg) (auto intro: sum_nonneg simp: field_split_simps)
  moreover note calculation
} note N = this

show  $\exists M. \forall t \in \text{ball } z d. \forall m \geq M. \forall n > m. \text{dist } (\text{ln\_Gamma\_series } t m) (\text{ln\_Gamma\_series } t n) < e$ 
  unfolding dist_complex_def
  proof (intro exI[of _ N] ballI allI impI)
    fix t m n assume t:  $t \in \text{ball } z d$  and mn:  $m \geq N$   $n > m$ 
    from d(2)[of 0] t have  $0 < \text{dist } z 0 - \text{dist } z t$  by (simp add: field_simps
dist_complex_def)
    also have  $\text{dist } z 0 - \text{dist } z t \leq \text{dist } 0 t$  using dist_triangle[of 0 z t]
    by (simp add: dist_commute)
    finally have t_nz:  $t \neq 0$  by auto

    have  $\text{norm } t \leq \text{norm } z + \text{norm } (t - z)$  by (rule norm_triangle_sub)
    also from t have  $\text{norm } (t - z) < d$  by (simp add: dist_complex_def norm_minus_commute)

```

```

also have  $2 * (\text{norm } z + d) \leq \text{of\_nat } N$  by (rule  $N$ )
also have  $N \leq m$  by (rule  $mn$ )
finally have  $\text{norm\_}t: 2 * \text{norm } t < \text{of\_nat } m$  by simp

have  $\text{ln\_Gamma\_series } t \ m - \text{ln\_Gamma\_series } t \ n =$ 
   $(-(t * \text{Ln } (\text{of\_nat } n)) - (-(t * \text{Ln } (\text{of\_nat } m)))) +$ 
   $((\sum k=1..n. \text{Ln } (t / \text{of\_nat } k + 1)) - (\sum k=1..m. \text{Ln } (t / \text{of\_nat } k$ 
   $+ 1)))$ 
  by (simp add:  $\text{ln\_Gamma\_series\_def algebra\_simps}$ )
also have  $(\sum k=1..n. \text{Ln } (t / \text{of\_nat } k + 1)) - (\sum k=1..m. \text{Ln } (t / \text{of\_nat } k$ 
   $+ 1)) =$ 
   $(\sum k \in \{1..n\} - \{1..m\}. \text{Ln } (t / \text{of\_nat } k + 1))$  using  $mn$ 
  by (simp add:  $\text{sum\_diff}$ )
also from  $mn$  have  $\{1..n\} - \{1..m\} = \{\text{Suc } m..n\}$  by fastforce
also have  $-(t * \text{Ln } (\text{of\_nat } n)) - (-(t * \text{Ln } (\text{of\_nat } m))) =$ 
   $(\sum k = \text{Suc } m..n. t * \text{Ln } (\text{of\_nat } (k - 1)) - t * \text{Ln } (\text{of\_nat } k))$ 
using  $mn$ 
  by (subst  $\text{sum\_telescope'' [symmetric]}$ ) simp_all
also have  $\dots = (\sum k = \text{Suc } m..n. t * \text{Ln } (\text{of\_nat } (k - 1) / \text{of\_nat } k))$  using
 $mn \ N$ 
  by (intro  $\text{sum\_cong\_Suc}$ )
  (simp_all del:  $\text{of\_nat\_Suc}$  add:  $\text{field\_simps Ln\_of\_nat Ln\_of\_nat\_over\_of\_nat}$ )
also have  $\text{of\_nat } (k - 1) / \text{of\_nat } k = 1 - 1 / (\text{of\_nat } k :: \text{complex})$  if  $k \in$ 
 $\{\text{Suc } m..n\}$  for  $k$ 
  using  $\text{that of\_nat\_eq\_0\_iff}[\text{of } \text{Suc } i \text{ for } i]$  by (cases  $k$ ) (simp_all add:
 $\text{field\_split\_simps}$ )
hence  $(\sum k = \text{Suc } m..n. t * \text{Ln } (\text{of\_nat } (k - 1) / \text{of\_nat } k)) =$ 
   $(\sum k = \text{Suc } m..n. t * \text{Ln } (1 - 1 / \text{of\_nat } k))$  using  $mn \ N$ 
  by (intro  $\text{sum.cong}$ ) simp_all
also note  $\text{sum.distrib [symmetric]}$ 
also have  $\text{norm } (\sum k=\text{Suc } m..n. t * \text{Ln } (1 - 1/\text{of\_nat } k) + \text{Ln } (t/\text{of\_nat } k$ 
   $+ 1)) \leq$ 
   $(\sum k=\text{Suc } m..n. 2 * (\text{norm } t + (\text{norm } t)^2) / (\text{real\_of\_nat } k)^2)$  using  $t\_nz$ 
 $N(2) \ mn \ \text{norm\_}t$ 
  by (intro  $\text{order.trans}[\text{OF norm\_sum sum\_mono}[\text{OF ln\_Gamma\_series\_complex\_converges\_aux}]]$ )
  simp_all
also have  $\dots \leq 2 * (\text{norm } t + \text{norm } t^2) * (\sum k=\text{Suc } m..n. 1 / (\text{of\_nat } k)^2)$ 
  by (simp add:  $\text{sum\_distrib\_left}$ )
also have  $\dots < 2 * (\text{norm } t + \text{norm } t^2) * e'$  using  $mn \ z \ t\_nz$ 
  by (intro  $\text{mult\_strict\_left\_mono } N \ \text{mult\_pos\_pos add\_pos\_pos}$ ) simp_all
also from  $e''\_pos$  have  $\dots = e * ((\text{cmod } t + (\text{cmod } t)^2) / e'')$ 
  by (simp add:  $e'\_def \text{field\_simps power2\_eq\_square}$ )
also from  $e''[\text{OF } t] \ e''\_pos \ e$ 
  have  $\dots \leq e * 1$  by (intro  $\text{mult\_left\_mono}$ ) (simp_all add:  $\text{field\_simps}$ )
finally show  $\text{norm } (\text{ln\_Gamma\_series } t \ m - \text{ln\_Gamma\_series } t \ n) < e$  by
  simp
  qed
  qed

```

end

lemma *ln_Gamma_series_complex_converges'*:

assumes $z: (z :: \text{complex}) \notin \mathbb{Z}_{\leq 0}$

shows $\exists d > 0. \text{uniformly\_convergent\_on } (\text{ball } z \ d) \ (\lambda n \ z. \text{ln\_Gamma\_series } z \ n)$

proof –

define d' where $d' = \text{Re } z$

define d where $d = (\text{if } d' > 0 \text{ then } d' / 2 \text{ else } \text{norm } (z - \text{of\_int } (\text{round } d')) / 2)$

have $\text{of\_int } (\text{round } d') \in \mathbb{Z}_{\leq 0}$ if $d' \leq 0$ using that

by (intro nonpos_Ints_of_int) (simp_all add: round_def)

with assms have $d\_pos: d > 0$ unfolding d_def by (force simp: not_less)

have $d < \text{cmod } (z - \text{of\_int } n)$ if $n \in \mathbb{Z}_{\leq 0}$ for n

proof (cases $\text{Re } z > 0$)

case True

from nonpos_Ints_nonpos[OF that] have $n: n \leq 0$ by simp

from True have $d = \text{Re } z / 2$ by (simp add: $d\_def \ d'_def$)

also from $n \ \text{True}$ have $\dots < \text{Re } (z - \text{of\_int } n)$ by simp

also have $\dots \leq \text{norm } (z - \text{of\_int } n)$ by (rule complex_Re_le_cmod)

finally show ?thesis .

next

case False

with assms nonpos_Ints_of_int[of round (Re z)]

have $z \neq \text{of\_int } (\text{round } d')$ by (auto simp: not_less)

with False have $d < \text{norm } (z - \text{of\_int } (\text{round } d'))$ by (simp add: $d\_def \ d'_def$)

also have $\dots \leq \text{norm } (z - \text{of\_int } n)$ unfolding d'_def by (rule round_Re_minimises_norm)

finally show ?thesis .

qed

hence $\text{conv}: \text{uniformly\_convergent\_on } (\text{ball } z \ d) \ (\lambda n \ z. \text{ln\_Gamma\_series } z \ n)$

by (intro *ln_Gamma_series_complex_converges* $d\_pos \ z$) simp_all

from $d\_pos \ \text{conv}$ show ?thesis by blast

qed

lemma *ln_Gamma_series_complex_converges''*: $(z :: \text{complex}) \notin \mathbb{Z}_{\leq 0} \implies \text{convergent } (\text{ln\_Gamma\_series } z)$

by (drule *ln_Gamma_series_complex_converges'*) (auto intro: uniformly_convergent_imp_convergent)

theorem *ln_Gamma_complex_LIMSEQ*: $(z :: \text{complex}) \notin \mathbb{Z}_{\leq 0} \implies \text{ln\_Gamma\_series } z \longrightarrow \text{ln\_Gamma } z$

using *ln_Gamma_series_complex_converges''* by (simp add: convergent_LIMSEQ_iff ln_Gamma_def)

lemma *exp_ln_Gamma_series_complex*:

assumes $n > 0 \ z \notin \mathbb{Z}_{\leq 0}$

shows $\text{exp } (\text{ln\_Gamma\_series } z \ n :: \text{complex}) = \text{Gamma\_series } z \ n$

proof –

```

from assms obtain m where m:  $n = \text{Suc } m$  by (cases n) blast
from assms have  $z \neq 0$  by (intro notI) auto
with assms have  $\exp (\ln\_Gamma\_series\ z\ n) =$ 
  ( $\text{of\_nat } n$ )  $\text{powr } z / (z * (\prod_{k=1..n} \exp (\ln (z / \text{of\_nat } k + 1))))$ 
  unfolding ln_Gamma_series_def powr_def by (simp add: exp_diff exp_sum)
also from assms have  $(\prod_{k=1..n} \exp (\ln (z / \text{of\_nat } k + 1))) = (\prod_{k=1..n}.$ 
 $z / \text{of\_nat } k + 1)$ 
  by (intro prod.cong[OF refl], subst exp_Ln) (auto simp: field_simps plus_of_nat_eq_0_imp)
also have  $\dots = (\prod_{k=1..n} z + k) / \text{fact } n$ 
  by (simp add: fact_prod)
  (subst prod_dividef [symmetric], simp_all add: field_simps)
also from m have  $z * \dots = (\prod_{k=0..n} z + k) / \text{fact } n$ 
  by (simp add: prod.atLeast0_atMost_Suc_shift prod.atLeast_Suc_atMost_Suc_shift
del: prod.cl_ivl_Suc)
also have  $(\prod_{k=0..n} z + k) = \text{pochhammer } z\ (\text{Suc } n)$ 
  unfolding pochhammer_prod
  by (simp add: prod.atLeast0_atMost_Suc atLeastLessThanSuc_atLeastAtMost)
also have  $\text{of\_nat } n \text{ powr } z / (\text{pochhammer } z\ (\text{Suc } n) / \text{fact } n) = \text{Gamma\_series}$ 
 $z\ n$ 
  unfolding Gamma_series_def using assms by (simp add: field_split_simps
powr_def)
  finally show ?thesis .
qed

```

```

lemma ln_Gamma_series'_aux:
  assumes  $(z::\text{complex}) \notin \mathbb{Z}_{\leq 0}$ 
  shows  $(\lambda k. z / \text{of\_nat } (\text{Suc } k) - \ln (1 + z / \text{of\_nat } (\text{Suc } k))) \text{ sums}$ 
 $(\ln\_Gamma\ z + \text{euler\_mascheroni} * z + \ln z)$  (is ?f sums ?s)
unfolding sums_def
proof (rule Lim_transform)
  show  $(\lambda n. \ln\_Gamma\_series\ z\ n + \text{of\_real } (\text{harm } n - \ln (\text{of\_nat } n)) * z + \ln$ 
 $z) \longrightarrow ?s$ 
  (is ?g  $\longrightarrow$  ?)
  by (intro tendsto_intros ln_Gamma_complex LIMSEQ euler_mascheroni LIMSEQ_of_real
assms)

```

```

have A: eventually  $(\lambda n. (\sum_{k < n} ?f\ k) - ?g\ n = 0)$  sequentially
  using eventually_gt_at_top[of 0::nat]
proof eventually_elim
  fix n :: nat assume n:  $n > 0$ 
  have  $(\sum_{k < n} ?f\ k) = (\sum_{k=1..n} z / \text{of\_nat } k - \ln (1 + z / \text{of\_nat } k))$ 
    by (subst atLeast0LessThan [symmetric], subst sum.shift_bounds_Suc_ivl
[symmetric],
      subst atLeastLessThanSuc_atLeastAtMost simp_all)
  also have  $\dots = z * \text{of\_real } (\text{harm } n) - (\sum_{k=1..n} \ln (1 + z / \text{of\_nat } k))$ 
    by (simp add: harm_def sum_subtractf sum_distrib_left divide_inverse)
  also from n have  $\dots - ?g\ n = 0$ 
    by (simp add: ln_Gamma_series_def sum_subtractf algebra_simps)

```

```

    finally show  $(\sum_{k < n} ?f\ k) - ?g\ n = 0$  .
  qed
  show  $(\lambda n. (\sum_{k < n} ?f\ k) - ?g\ n) \longrightarrow 0$  by (subst tendsto_cong[OF A])
simp_all
qed

```

lemma *uniformly_summable_deriv_ln_Gamma*:

assumes $z: (z :: 'a :: \{\text{real\_normed\_field}, \text{banach}\}) \neq 0$ **and** $d: d > 0 \wedge d \leq \text{norm } z / 2$

shows *uniformly_convergent_on* (ball $z\ d$)
 $(\lambda k\ z. \sum_{i < k} \text{inverse}(\text{of\_nat}(\text{Suc } i)) - \text{inverse}(z + \text{of\_nat}(\text{Suc } i)))$
(is uniformly_convergent_on _ $(\lambda k\ z. \sum_{i < k} ?f\ i\ z))$

proof (rule *Weierstrass_m_test'_ev*)

```

{
  fix t assume t:  $t \in \text{ball } z\ d$ 
  have  $\text{norm } z = \text{norm } (t + (z - t))$  by simp
  have  $\text{norm } (t + (z - t)) \leq \text{norm } t + \text{norm } (z - t)$  by (rule norm_triangle_ineq)
  also from t d have  $\text{norm } (z - t) < \text{norm } z / 2$  by (simp add: dist_norm)
  finally have A:  $\text{norm } t > \text{norm } z / 2$  using z by (simp add: field_simps)

  have  $\text{norm } t = \text{norm } (z + (t - z))$  by simp
  also have  $\dots \leq \text{norm } z + \text{norm } (t - z)$  by (rule norm_triangle_ineq)
  also from t d have  $\text{norm } (t - z) \leq \text{norm } z / 2$  by (simp add: dist_norm
norm_minus_commute)
  also from z have  $\dots < \text{norm } z$  by simp
  finally have B:  $\text{norm } t < 2 * \text{norm } z$  by simp
  note A B
} note ball = this

```

show *eventually* $(\lambda n. \forall t \in \text{ball } z\ d. \text{norm } (?f\ n\ t) \leq 4 * \text{norm } z * \text{inverse}(\text{of\_nat}(\text{Suc } n)^2))$ *sequentially*

using *eventually_gt_at_top* **apply** *eventually_elim*

proof *safe*

```

  fix t :: 'a assume t:  $t \in \text{ball } z\ d$ 
  from z ball[OF t] have t_nz:  $t \neq 0$  by auto
  fix n :: nat assume n:  $n > \text{nat } \lceil 4 * \text{norm } z \rceil$ 
  from ball[OF t] t_nz have  $4 * \text{norm } z > 2 * \text{norm } t$  by simp
  also from n have  $\dots < \text{of\_nat } n$  by linarith
  finally have n:  $\text{of\_nat } n > 2 * \text{norm } t$  .
  hence  $\text{of\_nat } n > \text{norm } t$  by simp
  hence t':  $t \neq -\text{of\_nat}(\text{Suc } n)$  by (intro notI) (simp del: of_nat_Suc)

```

```

  with t_nz have  $?f\ n\ t = 1 / (\text{of\_nat}(\text{Suc } n) * (1 + \text{of\_nat}(\text{Suc } n)/t))$ 
  by (simp add: field_split_simps eq_neg_iff_add_eq_0 del: of_nat_Suc)
  also have  $\text{norm } \dots = \text{inverse}(\text{of\_nat}(\text{Suc } n)) * \text{inverse}(\text{norm } (\text{of\_nat}(\text{Suc } n)/t + 1))$ 
  by (simp add: norm_divide norm_mult field_split_simps del: of_nat_Suc)
  also {

```

```

    from z t_nz ball[OF t] have of_nat (Suc n) / (4 * norm z) ≤ of_nat (Suc
n) / (2 * norm t)
    by (intro divide_left_mono mult_pos_pos) simp_all
    also have ... < norm (of_nat (Suc n) / t) - norm (1 :: 'a)
    using t_nz n by (simp add: field_simps norm_divide del: of_nat_Suc)
    also have ... ≤ norm (of_nat (Suc n)/t + 1) by (rule norm_diff_ineq)
    finally have inverse (norm (of_nat (Suc n)/t + 1)) ≤ 4 * norm z / of_nat
(Suc n)
    using z by (simp add: field_split_simps norm_divide mult_ac del: of_nat_Suc)
  }
  also have inverse (real_of_nat (Suc n)) * (4 * norm z / real_of_nat (Suc
n)) =
    4 * norm z * inverse (of_nat (Suc n)^2)
    by (simp add: field_split_simps power2_eq_square del: of_nat_Suc)
  finally show norm (?f n t) ≤ 4 * norm z * inverse (of_nat (Suc n)^2)
    by (simp del: of_nat_Suc)
qed
next
show summable (λn. 4 * norm z * inverse ((of_nat (Suc n))^2))
  by (subst summable_Suc_iff) (simp add: summable_mult inverse_power_summable)
qed

```

6.23.2 The Polygamma functions

lemma *summable_deriv_ln_Gamma*:

```

z ≠ 0 :: 'a :: {real_normed_field,banach} ⇒
  summable (λn. inverse (of_nat (Suc n)) - inverse (z + of_nat (Suc n)))
unfolding summable_iff_convergent
by (rule uniformly_convergent_imp_convergent,
    rule uniformly_summable_deriv_ln_Gamma[of z norm z/2]) simp_all

```

definition *Polygamma* :: nat ⇒ ('a :: {real_normed_field,banach}) ⇒ 'a **where**
Polygamma n z = (if n = 0 then
 $(\sum k. \text{inverse} (\text{of\_nat} (\text{Suc } k)) - \text{inverse} (z + \text{of\_nat } k)) - \text{euler\_mascheroni}$
else
 $(-1)^{\text{Suc } n} * \text{fact } n * (\sum k. \text{inverse} ((z + \text{of\_nat } k)^{\text{Suc } n}))$)

abbreviation *Digamma* :: ('a :: {real_normed_field,banach}) ⇒ 'a **where**
Digamma ≡ *Polygamma* 0

lemma *Digamma_def*:

```

Digamma z = (∑ k. inverse (of_nat (Suc k)) - inverse (z + of_nat k)) -
euler_mascheroni
by (simp add: Polygamma_def)

```

lemma *summable_Digamma*:

```

assumes (z :: 'a :: {real_normed_field,banach}) ≠ 0
shows summable (λn. inverse (of_nat (Suc n)) - inverse (z + of_nat n))

```


proof –

have $\text{sums} : (\lambda n. \text{inverse } (z + \text{of\_nat } (\text{Suc } n)) - \text{inverse } (z + \text{of\_nat } n)) \text{ sums}$
 $(0 - \text{inverse } (z + \text{of\_nat } 0))$
by ($\text{intro telescope\_sums filterlim\_compose} [OF \text{ tendsto\_inverse\_0}]$
 $\text{tendsto\_add\_filterlim\_at\_infinity} [OF \text{ tendsto\_const}] \text{ tendsto\_of\_nat}$)
from $\text{summable\_add} [OF \text{ summable\_deriv\_ln\_Gamma} [OF \text{ assms}] \text{ sums\_summable} [OF$
 $\text{sums}]$
show $\text{summable } (\lambda n. \text{inverse } (\text{of\_nat } (\text{Suc } n)) - \text{inverse } (z + \text{of\_nat } n))$ **by**
 simp
qed

lemma summable_offset :

assumes $\text{summable } (\lambda n. f (n + k)) :: 'a :: \text{real\_normed\_vector}$
shows $\text{summable } f$

proof –

from assms **have** $\text{convergent } (\lambda m. \sum_{n < m}. f (n + k))$
using $\text{summable\_iff\_convergent}$ **by** blast
hence $\text{convergent } (\lambda m. (\sum_{n < k}. f n) + (\sum_{n < m}. f (n + k)))$
by ($\text{intro convergent\_add convergent\_const}$)
also **have** $(\lambda m. (\sum_{n < k}. f n) + (\sum_{n < m}. f (n + k))) = (\lambda m. \sum_{n < m+k}. f n)$
proof
fix $m :: \text{nat}$
have $\{.. < m+k\} = \{.. < k\} \cup \{k.. < m+k\}$ **by** auto
also **have** $(\sum_{n \in \dots}. f n) = (\sum_{n < k}. f n) + (\sum_{n = k.. < m+k}. f n)$
by ($\text{rule sum.union\_disjoint}$) auto
also **have** $(\sum_{n = k.. < m+k}. f n) = (\sum_{n = 0.. < m+k-k}. f (n + k))$
using $\text{sum.shift\_bounds\_nat\_ivl} [of f 0 k m]$ **by** simp
finally **show** $(\sum_{n < k}. f n) + (\sum_{n < m}. f (n + k)) = (\sum_{n < m+k}. f n)$ **by**
 $(\text{simp add: atLeast0LessThan})$
qed
finally **have** $(\lambda a. \text{sum } f \{.. < a\}) \longrightarrow \text{lim } (\lambda m. \text{sum } f \{.. < m + k\})$
by ($\text{auto simp: convergent\_LIMSEQ\_iff dest: LIMSEQ\_offset}$)
thus $?thesis$ **by** ($\text{auto simp: summable\_iff\_convergent convergent\_def}$)
qed

lemma $\text{Polygamma\_converges}$:

fixes $z :: 'a :: \{\text{real\_normed\_field}, \text{banach}\}$
assumes $z : z \neq 0$ **and** $n : n \geq 2$
shows $\text{uniformly\_convergent\_on } (\text{ball } z d) (\lambda k z. \sum_{i < k}. \text{inverse } ((z + \text{of\_nat } i)^n))$

proof ($\text{rule Weierstrass\_m\_test' ev}$)

define e **where** $e = (1 + d / \text{norm } z)$

define m **where** $m = \text{nat } \lceil \text{norm } z * e \rceil$

{

fix t **assume** $t : t \in \text{ball } z d$

have $\text{norm } t = \text{norm } (z + (t - z))$ **by** simp

also **have** $\dots \leq \text{norm } z + \text{norm } (t - z)$ **by** ($\text{rule norm\_triangle\_ineq}$)

also **from** t **have** $\text{norm } (t - z) < d$ **by** ($\text{simp add: dist\_norm norm\_minus\_commute}$)

finally **have** $\text{norm } t < \text{norm } z * e$ **using** z **by** ($\text{simp add: divide\_simps e\_def}$)

```

} note ball = this

show eventually ( $\lambda k. \forall t \in \text{ball } z \, d. \text{norm } (\text{inverse } ((t + \text{of\_nat } k)^\wedge n)) \leq$ 
   $\text{inverse } (\text{of\_nat } (k - m)^\wedge n)$ ) sequentially
  using eventually_gt_at_top[of m] apply eventually_elim
proof (intro ballI)
  fix k :: nat and t :: 'a assume k:  $k > m$  and t:  $t \in \text{ball } z \, d$ 
  from k have  $\text{real\_of\_nat } (k - m) = \text{of\_nat } k - \text{of\_nat } m$  by (simp add:
of_nat_diff)
  also have  $\dots \leq \text{norm } (\text{of\_nat } k :: 'a) - \text{norm } z * e$ 
  unfolding m_def by (subst norm_of_nat) linarith
  also from ball[OF t] have  $\dots \leq \text{norm } (\text{of\_nat } k :: 'a) - \text{norm } t$  by simp
  also have  $\dots \leq \text{norm } (\text{of\_nat } k + t)$  by (rule norm_diff_ineq)
  finally have  $\text{inverse } ((\text{norm } (t + \text{of\_nat } k))^\wedge n) \leq \text{inverse } (\text{real\_of\_nat } (k -$ 
 $m)^\wedge n)$  using k n
  by (intro le_imp_inverse_le power_mono) (simp_all add: add_ac del:
of_nat_Suc)
  thus  $\text{norm } (\text{inverse } ((t + \text{of\_nat } k)^\wedge n)) \leq \text{inverse } (\text{of\_nat } (k - m)^\wedge n)$ 
  by (simp add: norm_inverse norm_power power_inverse)
qed

have summable ( $\lambda k. \text{inverse } ((\text{real\_of\_nat } k)^\wedge n)$ )
  using inverse_power_summable[of n] n by simp
hence summable ( $\lambda k. \text{inverse } ((\text{real\_of\_nat } (k + m - m))^\wedge n)$ ) by simp
thus summable ( $\lambda k. \text{inverse } ((\text{real\_of\_nat } (k - m))^\wedge n)$ ) by (rule summable_offset)
qed

lemma Polygamma_converges':
  fixes z :: 'a :: {real_normed_field,banach}
  assumes z:  $z \neq 0$  and n:  $n \geq 2$ 
  shows summable ( $\lambda k. \text{inverse } ((z + \text{of\_nat } k)^\wedge n)$ )
  using uniformly_convergent_imp_convergent[OF Polygamma_converges[OF assms,
of 1], of z]
  by (simp add: summable_iff_convergent)

theorem Digamma_LIMSEQ:
  fixes z :: 'a :: {banach,real_normed_field}
  assumes z:  $z \neq 0$ 
  shows  $(\lambda m. \text{of\_real } (\ln (\text{real } m)) - (\sum n < m. \text{inverse } (z + \text{of\_nat } n))) \longrightarrow$ 
  Digamma z
proof -
  have  $(\lambda n. \text{of\_real } (\ln (\text{real } n / (\text{real } (\text{Suc } n)))) \longrightarrow (\text{of\_real } (\ln 1) :: 'a)$ 
  by (intro tendsto_intros LIMSEQ_n_over_Suc_n) simp_all
  hence  $(\lambda n. \text{of\_real } (\ln (\text{real } n / (\text{real } n + 1)))) \longrightarrow (0 :: 'a)$  by (simp add:
add_ac)
  hence  $\lim: (\lambda n. \text{of\_real } (\ln (\text{real } n)) - \text{of\_real } (\ln (\text{real } n + 1))) \longrightarrow (0 :: 'a)$ 
  proof (rule Lim_transform_eventually)
    show eventually ( $\lambda n. \text{of\_real } (\ln (\text{real } n / (\text{real } n + 1))) =$ 
 $\text{of\_real } (\ln (\text{real } n)) - (\text{of\_real } (\ln (\text{real } n + 1)) :: 'a)$ ) at_top

```

using *eventually_gt_at_top*[*of 0::nat*] **by** *eventually_elim* (*simp add: ln_div*)
qed

from *summable_Digamma*[*OF z*]
have ($\lambda n. \text{inverse} (\text{of\_nat } (n+1)) - \text{inverse} (z + \text{of\_nat } n)$)
 $\text{sums } (\text{Digamma } z + \text{euler\_mascheroni})$
by (*simp add: Digamma_def summable_sums*)
from *sums_diff*[*OF this euler_mascheroni_sum*]
have ($\lambda n. \text{of\_real } (\ln (\text{real } (\text{Suc } n) + 1)) - \text{of\_real } (\ln (\text{real } n + 1)) - \text{inverse}$
 $(z + \text{of\_nat } n)$)
 $\text{sums } \text{Digamma } z$ **by** (*simp add: add_ac*)
hence ($\lambda m. (\sum n < m. \text{of\_real } (\ln (\text{real } (\text{Suc } n) + 1)) - \text{of\_real } (\ln (\text{real } n +$
 $1))) -$
 $(\sum n < m. \text{inverse } (z + \text{of\_nat } n))) \longrightarrow \text{Digamma } z$
by (*simp add: sums_def sum_subtractf*)
also have ($\lambda m. (\sum n < m. \text{of\_real } (\ln (\text{real } (\text{Suc } n) + 1)) - \text{of\_real } (\ln (\text{real } n$
 $+ 1))) =$
 $(\lambda m. \text{of\_real } (\ln (m + 1))) :: 'a$)
by (*subst sum_lessThan_telescope simp_all*)
finally show *?thesis* **by** (*rule Lim_transform*) (*insert lim, simp*)
qed

theorem *Polygamma_LIMSEQ*:

fixes $z :: 'a :: \{\text{banach, real\_normed\_field}\}$
assumes $z \neq 0$ **and** $n > 0$
shows $(\lambda k. \text{inverse } ((z + \text{of\_nat } k)^{\text{Suc } n})) \text{sums } ((-1)^{\text{Suc } n} * \text{Polygamma}$
 $n \text{ } z / \text{fact } n)$
using *Polygamma_converges*'[*OF assms(1), of Suc n*] *assms(2)*
by (*simp add: sums_iff Polygamma_def*)

theorem *has_field_derivative_ln_Gamma_complex* [*derivative_intros*]:

fixes $z :: \text{complex}$
assumes $z: z \notin \mathbb{R}_{\leq 0}$
shows $(\ln\_Gamma \text{ has\_field\_derivative } \text{Digamma } z) (\text{at } z)$
proof –
have *not_nonpos_Int* [*simp*]: $t \notin \mathbb{Z}_{\leq 0}$ **if** $\text{Re } t > 0$ **for** t
using *that* **by** (*auto elim!: nonpos_Ints_cases*)
from z **have** $z': z \notin \mathbb{Z}_{\leq 0}$ **and** $z'': z \neq 0$ **using** *nonpos_Ints_subset_nonpos_Reals*
 $\text{nonpos_Reals\_zero}_I$
by *blast+*
let $?f' = \lambda z \ k. \text{inverse} (\text{of\_nat } (\text{Suc } k)) - \text{inverse} (z + \text{of\_nat } (\text{Suc } k))$
let $?f = \lambda z \ k. z / \text{of\_nat } (\text{Suc } k) - \ln (1 + z / \text{of\_nat } (\text{Suc } k))$ **and** $?F' = \lambda z.$
 $\sum n. ?f' z \ n$
define d **where** $d = \min (\text{norm } z / 2)$ (*if* $\text{Im } z = 0$ *then* $\text{Re } z / 2$ *else* $\text{abs } (\text{Im } z)$
 $/ 2$)
from z **have** $d: d > 0$ $\text{norm } z / 2 \geq d$ **by** (*auto simp add: complex_nonpos_Reals_iff*
 d_def)
have *ball*: $\text{Im } t = 0 \longrightarrow \text{Re } t > 0$ **if** $\text{dist } z \ t < d$ **for** t
using *no_nonpos_Real_in_ball*[*OF z, of t*] **that** *unfolding* d_def **by** (*force*)

```

simp add: complex_nonpos_Reals_iff)
  have sums: (λn. inverse (z + of_nat (Suc n)) - inverse (z + of_nat n)) sums
    (0 - inverse (z + of_nat 0))
  by (intro telescope_sums filterlim_compose[OF tendsto_inverse_0]
      tendsto_add_filterlim_at_infinity[OF tendsto_const] tendsto_of_nat)

  have ((λz. ∑ n. ?f z n) has_field_derivative ?F' z) (at z)
  using d z ln_Gamma_series'_aux[OF z']
  apply (intro has_field_derivative_series'(2)[of ball z d _ z] uniformly_summable_deriv_ln_Gamma)
  apply (auto intro!: derivative_eq_intros add_pos_pos mult_pos_pos dest!: ball
      simp: field_simps sums_iff nonpos_Reals_divide_of_nat_iff
      simp del: of_nat_Suc)
  apply (auto simp add: complex_nonpos_Reals_iff)
  done
with z have ((λz. (∑ k. ?f z k) - euler_mascheroni * z - Ln z) has_field_derivative
  ?F' z - euler_mascheroni - inverse z) (at z)
  by (force intro!: derivative_eq_intros simp: Digamma_def)
  also have ?F' z - euler_mascheroni - inverse z = (?F' z + -inverse z) -
euler_mascheroni by simp
  also from sums have -inverse z = (∑ n. inverse (z + of_nat (Suc n)) -
inverse (z + of_nat n))
  by (simp add: sums_iff)
  also from sums summable_deriv_ln_Gamma[OF z']
  have ?F' z + ... = (∑ n. inverse (of_nat (Suc n)) - inverse (z + of_nat
n))
  by (subst suminf_add) (simp_all add: add_ac sums_iff)
  also have ... - euler_mascheroni = Digamma z by (simp add: Digamma_def)
  finally have ((λz. (∑ k. ?f z k) - euler_mascheroni * z - Ln z)
    has_field_derivative Digamma z) (at z) .
  moreover from eventually_nhds_ball[OF d(1), of z]
  have eventually (λz. ln_Gamma z = (∑ k. ?f z k) - euler_mascheroni * z -
Ln z) (nhds z)
  proof eventually_elim
    fix t assume t ∈ ball z d
    hence t ∉ ℤ≤0 by (auto dest!: ball_elim!: nonpos_Ints_cases)
    from ln_Gamma_series'_aux[OF this]
    show ln_Gamma t = (∑ k. ?f t k) - euler_mascheroni * t - Ln t by (simp
add: sums_iff)
  qed
  ultimately show ?thesis by (subst DERIV_cong_ev[OF refl _ refl])
qed

```

```

declare has_field_derivative_ln_Gamma_complex[THEN DERIV_chain2, deriva-
tive_intros]

```

```

lemma Digamma_1 [simp]: Digamma (1 :: 'a :: {real_normed_field,banach}) =
- euler_mascheroni
  by (simp add: Digamma_def)

```

lemma *Digamma_plus1*:

assumes $z \neq 0$

shows $\text{Digamma } (z+1) = \text{Digamma } z + 1/z$

proof –

have *sums*: $(\lambda k. \text{inverse } (z + \text{of\_nat } k) - \text{inverse } (z + \text{of\_nat } (\text{Suc } k)))$
 $\text{sums } (\text{inverse } (z + \text{of\_nat } 0) - 0)$

by (intro *telescope_sums'*[*OF filterlim_compose*[*OF tendsto_inverse_0*]]
tendsto_add_filterlim_at_infinity[*OF tendsto_const*] *tendsto_of_nat*)

have $\text{Digamma } (z+1) = (\sum k. \text{inverse } (\text{of\_nat } (\text{Suc } k)) - \text{inverse } (z + \text{of\_nat } (\text{Suc } k))) -$

euler_mascheroni (**is** $\_ = \text{suminf } ?f - \_$) **by** (*simp add: Digamma_def add_ac*)

also have $\text{suminf } ?f = (\sum k. \text{inverse } (\text{of\_nat } (\text{Suc } k)) - \text{inverse } (z + \text{of\_nat } k)) +$

$(\sum k. \text{inverse } (z + \text{of\_nat } k) - \text{inverse } (z + \text{of\_nat } (\text{Suc } k)))$

using *summable_Digamma*[*OF assms*] *sums* **by** (*subst suminf_add*) (*simp_all add: add_ac sums_iff*)

also have $(\sum k. \text{inverse } (z + \text{of\_nat } k) - \text{inverse } (z + \text{of\_nat } (\text{Suc } k))) = 1/z$

using *sums* **by** (*simp add: sums_iff inverse_eq_divide*)

finally show *?thesis* **by** (*simp add: Digamma_def*[*of z*])

qed

theorem *Polygamma_plus1*:

assumes $z \neq 0$

shows $\text{Polygamma } n \ (z + 1) = \text{Polygamma } n \ z + (-1)^n * \text{fact } n / (z \wedge \text{Suc } n)$

proof (*cases n = 0*)

assume $n: n \neq 0$

let $?f = \lambda k. \text{inverse } ((z + \text{of\_nat } k) \wedge \text{Suc } n)$

have $\text{Polygamma } n \ (z + 1) = (-1)^{\text{Suc } n} * \text{fact } n * (\sum k. ?f \ (k+1))$

using *n* **by** (*simp add: Polygamma_def add_ac*)

also have $(\sum k. ?f \ (k+1)) + (\sum k < 1. ?f \ k) = (\sum k. ?f \ k)$

using *Polygamma_converges'*[*OF assms, of Suc n*] *n*

by (*subst suminf_split_initial_segment* [*symmetric*]) *simp_all*

hence $(\sum k. ?f \ (k+1)) = (\sum k. ?f \ k) - \text{inverse } (z \wedge \text{Suc } n)$ **by** (*simp add: algebra_simps*)

also have $(-1)^{\text{Suc } n} * \text{fact } n * ((\sum k. ?f \ k) - \text{inverse } (z \wedge \text{Suc } n)) =$
 $\text{Polygamma } n \ z + (-1)^n * \text{fact } n / (z \wedge \text{Suc } n)$ **using** *n*

by (*simp add: inverse_eq_divide algebra_simps Polygamma_def*)

finally show *?thesis* .

qed (*insert assms, simp add: Digamma_plus1 inverse_eq_divide*)

theorem *Digamma_of_nat*:

$\text{Digamma } (\text{of\_nat } (\text{Suc } n)) :: 'a :: \{\text{real\_normed\_field}, \text{banach}\} = \text{harm } n -$
euler_mascheroni

proof (*induction n*)

case (*Suc n*)

have $\text{Digamma } (\text{of\_nat } (\text{Suc } (\text{Suc } n))) :: 'a = \text{Digamma } (\text{of\_nat } (\text{Suc } n) + 1)$

by *simp*
also have $\dots = \text{Digamma } (\text{of\_nat } (\text{Suc } n)) + \text{inverse } (\text{of\_nat } (\text{Suc } n))$
by (*subst Digamma_plus1*) (*simp_all add: inverse_eq_divide del: of_nat_Suc*)
also have $\text{Digamma } (\text{of\_nat } (\text{Suc } n) :: 'a) = \text{harm } n - \text{euler\_mascheroni}$ **by**
(*rule Suc*)
also have $\dots + \text{inverse } (\text{of\_nat } (\text{Suc } n)) = \text{harm } (\text{Suc } n) - \text{euler\_mascheroni}$
by (*simp add: harm_Suc*)
finally show ?case .
qed (*simp add: harm_def*)

lemma *Digamma_numeral*: $\text{Digamma } (\text{numeral } n) = \text{harm } (\text{pred\_numeral } n) - \text{euler\_mascheroni}$
by (*subst of_nat_numeral[symmetric], subst numeral_eq_Suc, subst Digamma_of_nat*)
(*rule refl*)

lemma *Polygamma_of_real*: $x \neq 0 \implies \text{Polygamma } n \text{ (of\_real } x) = \text{of\_real } (\text{Polygamma } n x)$
(*Polygamma n x*)
unfolding *Polygamma_def* **using** *summable_Digamma[of x] Polygamma_converges'[of x Suc n]*
by (*simp_all add: suminf_of_real*)

lemma *Polygamma_Real*: $z \in \mathbb{R} \implies z \neq 0 \implies \text{Polygamma } n z \in \mathbb{R}$
by (*elim Reals_cases, hypsubst, subst Polygamma_of_real*) *simp_all*

lemma *Digamma_half_integer*:
 $\text{Digamma } (\text{of\_nat } n + 1/2 :: 'a :: \{\text{real\_normed\_field}, \text{banach}\}) =$
 $(\sum_{k < n. 2 / (\text{of\_nat } (2*k+1))) - \text{euler\_mascheroni} - \text{of\_real } (2 * \ln 2)$
proof (*induction n*)
case 0
have $\text{Digamma } (1/2 :: 'a) = \text{of\_real } (\text{Digamma } (1/2))$ **by** (*simp add: Polygamma_of_real [symmetric]*)
also have $\text{Digamma } (1/2 :: \text{real}) =$
 $(\sum k. \text{inverse } (\text{of\_nat } (\text{Suc } k)) - \text{inverse } (\text{of\_nat } k + 1/2)) - \text{euler\_mascheroni}$
by (*simp add: Digamma_def add_ac*)
also have $(\sum k. \text{inverse } (\text{of\_nat } (\text{Suc } k) :: \text{real}) - \text{inverse } (\text{of\_nat } k + 1/2)) =$
 $(\sum k. \text{inverse } (1/2) * (\text{inverse } (2 * \text{of\_nat } (\text{Suc } k)) - \text{inverse } (2 * \text{of\_nat } k + 1)))$
by (*simp_all add: add_ac inverse_mult_distrib[symmetric] ring_distrib del: inverse_divide*)
also have $\dots = -2 * \ln 2$ **using** *sums_minus[OF alternating_harmonic_series_sums]*
by (*subst suminf_mult*) (*simp_all add: algebra_simps sums_iff*)
finally show ?case **by** *simp*

next
case (*Suc n*)
have $nz: 2 * \text{of\_nat } n + (1 :: 'a) \neq 0$
using *of_nat_neq_0[of 2*n]* **by** (*simp only: of_nat_Suc*) (*simp add: add_ac*)
hence $nz': \text{of\_nat } n + (1/2 :: 'a) \neq 0$ **by** (*simp add: field_simps*)
have $\text{Digamma } (\text{of\_nat } (\text{Suc } n) + 1/2 :: 'a) = \text{Digamma } (\text{of\_nat } n + 1/2 +$

```

1) by simp
also from nz' have ... = Digamma (of_nat n + 1/2) + 1 / (of_nat n + 1/2)
  by (rule Digamma_plus1)
also from nz nz' have 1 / (of_nat n + 1/2 :: 'a) = 2 / (2 * of_nat n + 1)
  by (subst divide_eq_eq) simp_all
also note Suc
finally show ?case by (simp add: add_ac)
qed

```

```

lemma Digamma_one_half: Digamma (1/2) = - euler_mascheroni - of_real
(2 * ln 2)
  using Digamma_half_integer[of 0] by simp

```

```

lemma Digamma_real_three_halves_pos: Digamma (3/2 :: real) > 0
proof -
  have -Digamma (3/2 :: real) = -Digamma (of_nat 1 + 1/2) by simp
  also have ... = 2 * ln 2 + euler_mascheroni - 2 by (subst Digamma_half_integer)
  simp
  also note euler_mascheroni_less_13_over_22
  also note ln2_le_25_over_36
  finally show ?thesis by simp
qed

```

```

theorem has_field_derivative_Polygamma [derivative_intros]:
  fixes z :: 'a :: {real_normed_field, euclidean_space}
  assumes z: z ∉ ℤ≤0
  shows (Polygamma n has_field_derivative Polygamma (Suc n) z) (at z within
A)
proof (rule has_field_derivative_at_within, cases n = 0)
  assume n: n = 0
  let ?f = λk z. inverse (of_nat (Suc k)) - inverse (z + of_nat k)
  let ?F = λz. ∑ k. ?f k z and ?f' = λk z. inverse ((z + of_nat k)2)
  from no_nonpos_Int_in_ball'[OF z] obtain d where d: 0 < d ∧ t. t ∈ ball z
d ⇒ t ∉ ℤ≤0
  by auto
  from z have summable: summable (λk. inverse (of_nat (Suc k)) - inverse (z +
of_nat k))
  by (intro summable_Digamma) force
  from z have conv: uniformly_convergent_on (ball z d) (λk z. ∑ i<k. inverse
((z + of_nat i)2))
  by (intro Polygamma_converges) auto
  with d have summable (λk. inverse ((z + of_nat k)2)) unfolding summable_iff_convergent
  by (auto dest!: uniformly_convergent_imp_convergent simp: summable_iff_convergent)
)

have (?F has_field_derivative (∑ k. ?f' k z)) (at z)
proof (rule has_field_derivative_series'[of ball z d _ z])
  fix k :: nat and t :: 'a assume t: t ∈ ball z d

```

```

from  $t$   $d(2)[\text{of } t]$  show  $((\lambda z. ?f\ k\ z)\ \text{has\_field\_derivative}\ ?f'\ k\ t)$   $(\text{at } t\ \text{within ball } z\ d)$ 
by  $(\text{auto intro!}:\ \text{derivative\_eq\_intros simp: power2\_eq\_square simp del: of\_nat\_Suc}$ 
 $\text{dest!}:\ \text{plus\_of\_nat\_eq\_0\_imp elim!}:\ \text{nonpos\_Ints\_cases})$ 
qed  $(\text{insert } d(1)\ \text{summable conv, (assumption|simp)})$ 
with  $z$  show  $(\text{Polygamma } n\ \text{has\_field\_derivative Polygamma (Suc } n)\ z)$   $(\text{at } z)$ 
unfolding  $\text{Digamma\_def [abs\_def] Polygamma\_def [abs\_def]}$  using  $n$ 
by  $(\text{force simp: power2\_eq\_square intro!}:\ \text{derivative\_eq\_intros})$ 
next
assume  $n: n \neq 0$ 
from  $z$  have  $z': z \neq 0$  by  $\text{auto}$ 
from  $\text{no\_nonpos\_Int\_in\_ball}[OF\ z]$  obtain  $d$  where  $d: 0 < d \wedge t. t \in \text{ball } z\ d \implies t \notin \mathbb{Z}_{\leq 0}$ 
by  $\text{auto}$ 
define  $n'$  where  $n' = \text{Suc } n$ 
from  $n$  have  $n': n' \geq 2$  by  $(\text{simp add: } n'\text{\_def})$ 
have  $((\lambda z. \sum k. \text{inverse } ((z + \text{of\_nat } k) ^{n'}))\ \text{has\_field\_derivative}$ 
 $(\sum k. -\ \text{of\_nat } n' * \text{inverse } ((z + \text{of\_nat } k) ^{(n'+1)})))$   $(\text{at } z)$ 
proof  $(\text{rule has\_field\_derivative\_series'}[\text{of ball } z\ d \_ \_ z])$ 
fix  $k :: \text{nat}$  and  $t :: 'a$  assume  $t: t \in \text{ball } z\ d$ 
with  $d$  have  $t': t \notin \mathbb{Z}_{\leq 0}\ t \neq 0$  by  $\text{auto}$ 
show  $((\lambda a. \text{inverse } ((a + \text{of\_nat } k) ^{n'}))\ \text{has\_field\_derivative}$ 
 $-\ \text{of\_nat } n' * \text{inverse } ((t + \text{of\_nat } k) ^{(n'+1)}))$   $(\text{at } t\ \text{within ball } z\ d)$  using  $t'$ 
by  $(\text{fastforce intro!}:\ \text{derivative\_eq\_intros simp: divide\_simps power\_diff dest: plus\_of\_nat\_eq\_0\_imp})$ 
next
have  $\text{uniformly\_convergent\_on } (\text{ball } z\ d)$ 
 $(\lambda k\ z. (-\ \text{of\_nat } n' :: 'a) * (\sum i < k. \text{inverse } ((z + \text{of\_nat } i) ^{(n'+1)})))$ 
using  $z'\ n$  by  $(\text{intro uniformly\_convergent\_mult Polygamma\_converges})$ 
 $(\text{simp\_all add: } n'\text{\_def})$ 
thus  $\text{uniformly\_convergent\_on } (\text{ball } z\ d)$ 
 $(\lambda k\ z. \sum i < k. -\ \text{of\_nat } n' * \text{inverse } ((z + \text{of\_nat } i :: 'a) ^{(n'+1)}))$ 
by  $(\text{subst (asm) sum\_distrib\_left})\ \text{simp}$ 
qed  $(\text{insert Polygamma\_converges'}[OF\ z'\ n]\ d, \text{simp\_all})$ 
also have  $(\sum k. -\ \text{of\_nat } n' * \text{inverse } ((z + \text{of\_nat } k) ^{(n' + 1)})) =$ 
 $(-\ \text{of\_nat } n') * (\sum k. \text{inverse } ((z + \text{of\_nat } k) ^{(n' + 1)}))$ 
using  $\text{Polygamma\_converges'}[OF\ z', \text{of } n'+1]\ n'$  by  $(\text{subst suminf\_mult})$ 
 $\text{simp\_all}$ 
finally have  $((\lambda z. \sum k. \text{inverse } ((z + \text{of\_nat } k) ^{n'}))\ \text{has\_field\_derivative}$ 
 $-\ \text{of\_nat } n' * (\sum k. \text{inverse } ((z + \text{of\_nat } k) ^{(n' + 1)})))$   $(\text{at } z)$  .
from  $\text{DERIV\_cmult}[OF\ \text{this, of } (-1)^{\text{Suc } n} * \text{fact } n :: 'a]$ 
show  $(\text{Polygamma } n\ \text{has\_field\_derivative Polygamma (Suc } n)\ z)$   $(\text{at } z)$ 
unfolding  $n'\text{\_def Polygamma\_def[abs\_def]}$  using  $n$  by  $(\text{simp add: algebra\_simps})$ 
qed

```

declare $\text{has\_field\_derivative\_Polygamma}[\text{THEN DERIV\_chain2, derivative\_intros}]$

lemma *isCont_Polygamma* [continuous_intros]:
fixes $f :: \_ \Rightarrow 'a :: \{\text{real\_normed\_field}, \text{euclidean\_space}\}$
shows $\text{isCont } f \ z \Longrightarrow f \ z \notin \mathbb{Z}_{\leq 0} \Longrightarrow \text{isCont } (\lambda x. \text{Polygamma } n \ (f \ x)) \ z$
by (rule *isCont_o2*[OF *DERIV_isCont*[OF *has_field_derivative_Polygamma*]])

lemma *continuous_on_Polygamma*:
 $A \cap \mathbb{Z}_{\leq 0} = \{\} \Longrightarrow \text{continuous\_on } A \ (\text{Polygamma } n :: \_ \Rightarrow 'a :: \{\text{real\_normed\_field}, \text{euclidean\_space}\})$
by (intro *continuous_at_imp_continuous_on* *isCont_Polygamma*[OF *continuous_ident*] *ballI*) *blast*

lemma *isCont_ln_Gamma_complex* [continuous_intros]:
fixes $f :: 'a :: t2\_space \Rightarrow \text{complex}$
shows $\text{isCont } f \ z \Longrightarrow f \ z \notin \mathbb{R}_{\leq 0} \Longrightarrow \text{isCont } (\lambda z. \text{ln\_Gamma } (f \ z)) \ z$
by (rule *isCont_o2*[OF *DERIV_isCont*[OF *has_field_derivative_ln_Gamma_complex*]])

lemma *continuous_on_ln_Gamma_complex* [continuous_intros]:
fixes $A :: \text{complex set}$
shows $A \cap \mathbb{R}_{\leq 0} = \{\} \Longrightarrow \text{continuous\_on } A \ \text{ln\_Gamma}$
by (intro *continuous_at_imp_continuous_on* *ballI* *isCont_ln_Gamma_complex*[OF *continuous_ident*])
fastforce

lemma *deriv_Polygamma*:
assumes $z \notin \mathbb{Z}_{\leq 0}$
shows $\text{deriv } (\text{Polygamma } m) \ z = \text{Polygamma } (\text{Suc } m) \ (z :: 'a :: \{\text{real\_normed\_field}, \text{euclidean\_space}\})$
by (intro *DERIV_imp_deriv* *has_field_derivative_Polygamma* *assms*)
thm *has_field_derivative_Polygamma*

lemma *higher_deriv_Polygamma*:
assumes $z \notin \mathbb{Z}_{\leq 0}$
shows $(\text{deriv } \widetilde{\sim}^n) \ (\text{Polygamma } m) \ z = \text{Polygamma } (m + n) \ (z :: 'a :: \{\text{real\_normed\_field}, \text{euclidean\_space}\})$

proof –
have *eventually* $(\lambda u. (\text{deriv } \widetilde{\sim}^n) \ (\text{Polygamma } m) \ u = \text{Polygamma } (m + n) \ u)$
(nhds z)
proof (*induction n*)
case (*Suc n*)
from *Suc.IH* **have** *eventually* $(\lambda z. \text{eventually } (\lambda u. (\text{deriv } \widetilde{\sim}^n) \ (\text{Polygamma } m) \ u = \text{Polygamma } (m + n) \ u) \ (\text{nhds } z)) \ (\text{nhds } z)$
by (*simp add: eventually_eventually*)
hence *eventually* $(\lambda z. \text{deriv } ((\text{deriv } \widetilde{\sim}^n) \ (\text{Polygamma } m)) \ z = \text{deriv } (\text{Polygamma } (m + n)) \ z) \ (\text{nhds } z)$
by *eventually_elim* (*intro deriv_cong_ev refl*)
moreover **have** *eventually* $(\lambda z. z \in \text{UNIV} - \mathbb{Z}_{\leq 0}) \ (\text{nhds } z)$ **using** *assms*
by (*intro eventually_nhds_in_open open_Diff open_UNIV*) *auto*
ultimately **show** ?*case* **by** *eventually_elim* (*simp_all add: deriv_Polygamma*)
qed *simp_all*

thus *?thesis* **by** (rule eventually_nhds_x_imp_x)
qed

lemma *deriv_ln_Gamma_complex*:
assumes $z \notin \mathbb{R}_{\leq 0}$
shows $\text{deriv\_ln\_Gamma } z = \text{Digamma } (z :: \text{complex})$
by (intro DERIV_imp_deriv has_field_derivative_ln_Gamma_complex assms)

lemma *higher_deriv_ln_Gamma_complex*:
assumes $(x :: \text{complex}) \notin \mathbb{R}_{\leq 0}$
shows $(\text{deriv } \sim j) \text{ ln\_Gamma } x = (\text{if } j = 0 \text{ then } \text{ln\_Gamma } x \text{ else } \text{Polygamma } (j - 1) x)$
proof (cases j)
case (Suc j')
have $(\text{deriv } \sim j') (\text{deriv\_ln\_Gamma } x) = (\text{deriv } \sim j') \text{ Digamma } x$
using eventually_nhds_in_open[of UNIV - $\mathbb{R}_{\leq 0}$] assms
by (intro higher_deriv_cong_ev refl)
(auto elim!: eventually_mono simp: open_Diff deriv_ln_Gamma_complex)
also have $\dots = \text{Polygamma } j' x$ **using** assms
by (subst higher_deriv_Polygamma)
(auto elim!: nonpos_Ints_cases simp: complex_nonpos_Reals_iff)
finally show *?thesis* **using** Suc **by** (simp del: funpow.simps add: funpow_Suc_right)
qed simp_all

We define a type class that captures all the fundamental properties of the inverse of the Gamma function and defines the Gamma function upon that. This allows us to instantiate the type class both for the reals and for the complex numbers with a minimal amount of proof duplication.

class *Gamma* = *real_normed_field* + *complete_space* +
fixes $r\text{Gamma} :: 'a \Rightarrow 'a$
assumes $r\text{Gamma\_eq\_zero\_iff\_aux}: r\text{Gamma } z = 0 \longleftrightarrow (\exists n. z = - \text{of\_nat } n)$
assumes *differentiable_rGamma_aux1*:
 $(\bigwedge n. z \neq - \text{of\_nat } n) \implies$
 $\text{let } d = (\text{THE } d. (\lambda n. \sum k < n. \text{inverse } (\text{of\_nat } (\text{Suc } k)) - \text{inverse } (z + \text{of\_nat } k)))$
 $\longrightarrow d) - \text{scaleR } \text{euler\_mascheroni } 1$
 $\text{in } \text{filterlim } (\lambda y. (r\text{Gamma } y - r\text{Gamma } z + r\text{Gamma } z * d * (y - z)) /_R$
 $\text{norm } (y - z)) (\text{nhds } 0) (\text{at } z)$
assumes *differentiable_rGamma_aux2*:
 $\text{let } z = - \text{of\_nat } n$
 $\text{in } \text{filterlim } (\lambda y. (r\text{Gamma } y - r\text{Gamma } z - (-1)^n * (\text{prod of\_nat } \{1..n\})$
 $* (y - z)) /_R$
 $\text{norm } (y - z)) (\text{nhds } 0) (\text{at } z)$
assumes $r\text{Gamma\_series\_aux}: (\bigwedge n. z \neq - \text{of\_nat } n) \implies$
 $\text{let fact}' = (\lambda n. \text{prod of\_nat } \{1..n\});$
 $\text{exp} = (\lambda x. \text{THE } e. (\lambda n. \sum k < n. x^k /_R \text{fact } k) \longrightarrow e);$
 $\text{pochhammer}' = (\lambda a n. (\prod n = 0..n. a + \text{of\_nat } n))$

```

      in filterlim ( $\lambda n. \text{ pochhammer}' z n / (\text{fact}' n * \exp (z * (\ln (\text{of\_nat } n) *_{\mathbb{R}} 1))))$ 

```

```

      (nhds (rGamma z)) sequentially

```

```

begin

```

```

subclass banach ..

```

```

end

```

```

definition Gamma z = inverse (rGamma z)

```

6.23.3 Basic properties

```

lemma Gamma_nonpos_Int:  $z \in \mathbb{Z}_{\leq 0} \implies \text{Gamma } z = 0$ 
  and rGamma_nonpos_Int:  $z \in \mathbb{Z}_{\leq 0} \implies \text{rGamma } z = 0$ 
  using rGamma_eq_zero_iff_aux[of z] unfolding Gamma_def by (auto elim!:
nonpos_Ints_cases')

```

```

lemma Gamma_nonzero:  $z \notin \mathbb{Z}_{\leq 0} \implies \text{Gamma } z \neq 0$ 
  and rGamma_nonzero:  $z \notin \mathbb{Z}_{\leq 0} \implies \text{rGamma } z \neq 0$ 
  using rGamma_eq_zero_iff_aux[of z] unfolding Gamma_def by (auto elim!:
nonpos_Ints_cases')

```

```

lemma Gamma_eq_zero_iff:  $\text{Gamma } z = 0 \longleftrightarrow z \in \mathbb{Z}_{\leq 0}$ 
  and rGamma_eq_zero_iff:  $\text{rGamma } z = 0 \longleftrightarrow z \in \mathbb{Z}_{\leq 0}$ 
  using rGamma_eq_zero_iff_aux[of z] unfolding Gamma_def by (auto elim!:
nonpos_Ints_cases')

```

```

lemma rGamma_inverse_Gamma:  $\text{rGamma } z = \text{inverse } (\text{Gamma } z)$ 
  unfolding Gamma_def by simp

```

```

lemma rGamma_series_LIMSEQ [tendsto_intros]:
  rGamma_series z  $\longrightarrow$  rGamma z
proof (cases  $z \in \mathbb{Z}_{\leq 0}$ )
  case False
  hence  $z \neq -\text{of\_nat } n$  for  $n$  by auto
  from rGamma_series_aux[OF this] show ?thesis
    by (simp add: rGamma_series_def[abs_def] fact_prod pochhammer_Suc_prod
      exp_def of_real_def[symmetric] suminf_def sums_def[abs_def]
atLeast0AtMost)
qed (insert rGamma_eq_zero_iff[of z], simp_all add: rGamma_series_nonpos_Ints_LIMSEQ)

```

```

theorem Gamma_series_LIMSEQ [tendsto_intros]:
  Gamma_series z  $\longrightarrow$  Gamma z
proof (cases  $z \in \mathbb{Z}_{\leq 0}$ )
  case False
  hence  $(\lambda n. \text{inverse } (\text{rGamma\_series } z n)) \longrightarrow \text{inverse } (\text{rGamma } z)$ 
    by (intro tendsto_intros) (simp_all add: rGamma_eq_zero_iff)
  also have  $(\lambda n. \text{inverse } (\text{rGamma\_series } z n)) = \text{Gamma\_series } z$ 
    by (simp add: rGamma_series_def Gamma_series_def[abs_def])
  finally show ?thesis by (simp add: Gamma_def)

```

qed (*insert Gamma_eq_zero_iff*[*of z*], *simp_all add: Gamma_series_nonpos_Ints_LIMSEQ*)

lemma *Gamma_altdef*: $\text{Gamma } z = \lim (\text{Gamma\_series } z)$
using *Gamma_series_LIMSEQ*[*of z*] **by** (*simp add: limI*)

lemma *rGamma_1* [*simp*]: $r\text{Gamma } 1 = 1$

proof –

have *A*: *eventually* ($\lambda n. r\text{Gamma\_series } 1 \ n = \text{of\_nat } (\text{Suc } n) / \text{of\_nat } n$)
sequentially

using *eventually_gt_at_top*[*of 0::nat*]

by (*force elim!*: *eventually_mono simp: rGamma_series_def exp_of_real pochhammer_fact*

field_split_simps pochhammer_rec' dest!: pochhammer_eq_0_imp_nonpos_Int)

have $r\text{Gamma\_series } 1 \longrightarrow 1$ **by** (*subst tendsto_cong[OF A]*) (*rule LIMSEQ_Suc_n_over_n*)

moreover have $r\text{Gamma\_series } 1 \longrightarrow r\text{Gamma } 1$ **by** (*rule tendsto_intros*)

ultimately show *?thesis* **by** (*intro LIMSEQ_unique*)

qed

lemma *rGamma_plus1*: $z * r\text{Gamma } (z + 1) = r\text{Gamma } z$

proof –

let *?f* = $\lambda n. (z + 1) * \text{inverse } (\text{of\_nat } n) + 1$

have *eventually* ($\lambda n. ?f \ n * r\text{Gamma\_series } z \ n = z * r\text{Gamma\_series } (z + 1) \ n$)
sequentially

using *eventually_gt_at_top*[*of 0::nat*]

proof *eventually_elim*

fix *n* :: *nat* **assume** *n* > 0

hence $z * r\text{Gamma\_series } (z + 1) \ n = \text{inverse } (\text{of\_nat } n) *$

$\text{pochhammer } z \ (\text{Suc } (\text{Suc } n)) / (\text{fact } n * \text{exp } (z * \text{of\_real } (\ln (\text{of\_nat } n))))$

by (*subst pochhammer_rec*) (*simp add: rGamma_series_def field_simps exp_add exp_of_real*)

also from *n* **have** ... = $?f \ n * r\text{Gamma\_series } z \ n$

by (*subst pochhammer_rec'*) (*simp_all add: field_split_simps rGamma_series_def*)

finally show $?f \ n * r\text{Gamma\_series } z \ n = z * r\text{Gamma\_series } (z + 1) \ n$..

qed

moreover have ($\lambda n. ?f \ n * r\text{Gamma\_series } z \ n$) $\longrightarrow ((z+1) * 0 + 1) * r\text{Gamma } z$

by (*intro tendsto_intros lim_inverse_n*)

hence ($\lambda n. ?f \ n * r\text{Gamma\_series } z \ n$) $\longrightarrow r\text{Gamma } z$ **by** *simp*

ultimately have ($\lambda n. z * r\text{Gamma\_series } (z + 1) \ n$) $\longrightarrow r\text{Gamma } z$

by (*blast intro: Lim_transform_eventually*)

moreover have ($\lambda n. z * r\text{Gamma\_series } (z + 1) \ n$) $\longrightarrow z * r\text{Gamma } (z + 1)$

by (*intro tendsto_intros*)

ultimately show $z * r\text{Gamma } (z + 1) = r\text{Gamma } z$ **using** *LIMSEQ_unique*

by *blast*

qed

lemma pochhammer_rGamma: $rGamma\ z = pochhammer\ z\ n * rGamma\ (z + of\_nat\ n)$
proof (induction n arbitrary: z)
 case (Suc n z)
 have $rGamma\ z = pochhammer\ z\ n * rGamma\ (z + of\_nat\ n)$ **by** (rule Suc.IH)
 also **note** rGamma_plus1 [symmetric]
 finally **show** ?case **by** (simp add: add_ac pochhammer_rec')
qed simp_all

theorem Gamma_plus1: $z \notin \mathbb{Z}_{\leq 0} \implies Gamma\ (z + 1) = z * Gamma\ z$
using rGamma_plus1 [of z] **by** (simp add: rGamma_inverse_Gamma_field_simps Gamma_eq_zero_iff)

theorem pochhammer_Gamma: $z \notin \mathbb{Z}_{\leq 0} \implies pochhammer\ z\ n = Gamma\ (z + of\_nat\ n) / Gamma\ z$
using pochhammer_rGamma [of z]
by (simp add: rGamma_inverse_Gamma Gamma_eq_zero_iff field_simps)

lemma Gamma_0 [simp]: $Gamma\ 0 = 0$
and rGamma_0 [simp]: $rGamma\ 0 = 0$
and Gamma_neg_1 [simp]: $Gamma\ (-1) = 0$
and rGamma_neg_1 [simp]: $rGamma\ (-1) = 0$
and Gamma_neg_numeral [simp]: $Gamma\ (-numeral\ n) = 0$
and rGamma_neg_numeral [simp]: $rGamma\ (-numeral\ n) = 0$
and Gamma_neg_of_nat [simp]: $Gamma\ (-of\_nat\ m) = 0$
and rGamma_neg_of_nat [simp]: $rGamma\ (-of\_nat\ m) = 0$
by (simp_all add: rGamma_eq_zero_iff Gamma_eq_zero_iff)

lemma Gamma_1 [simp]: $Gamma\ 1 = 1$ **unfolding** Gamma_def **by** simp

theorem Gamma_fact: $Gamma\ (1 + of\_nat\ n) = fact\ n$
by (simp add: pochhammer_fact pochhammer_Gamma of_nat_in_nonpos_Ints_iff flip: of_nat_Suc)

lemma Gamma_numeral: $Gamma\ (numeral\ n) = fact\ (pred\_numeral\ n)$
by (subst of_nat_numeral[symmetric], subst numeral_eq_Suc,
 subst of_nat_Suc, subst Gamma_fact) (rule refl)

lemma Gamma_of_int: $Gamma\ (of\_int\ n) = (if\ n > 0\ then\ fact\ (nat\ (n - 1))\ else\ 0)$

proof (cases n > 0)
 case True
 hence $Gamma\ (of\_int\ n) = Gamma\ (of\_nat\ (Suc\ (nat\ (n - 1))))$ **by** (subst of_nat_Suc) simp_all
 with True **show** ?thesis **by** (subst (asm) of_nat_Suc, subst (asm) Gamma_fact) simp
qed (simp_all add: Gamma_eq_zero_iff nonpos_Ints_of_int)

lemma *rGamma_of_int*: $r\text{Gamma}(\text{of\_int } n) = (\text{if } n > 0 \text{ then inverse (fact (nat (n - 1))) else } 0)$

by (*simp add: Gamma_of_int rGamma_inverse_Gamma*)

lemma *Gamma_seriesI*:

assumes $(\lambda n. g \ n / \text{Gamma\_series } z \ n) \longrightarrow 1$

shows $g \longrightarrow \text{Gamma } z$

proof (*rule Lim_transform_eventually*)

have $1/2 > (0::\text{real})$ **by** *simp*

from *tendstoD[OF assms, OF this]*

show *eventually* $(\lambda n. g \ n / \text{Gamma\_series } z \ n * \text{Gamma\_series } z \ n = g \ n)$ *sequentially*

by (*force elim!: eventually_mono simp: dist_real_def*)

from *assms* **have** $(\lambda n. g \ n / \text{Gamma\_series } z \ n * \text{Gamma\_series } z \ n) \longrightarrow 1 * \text{Gamma } z$

by (*intro tendsto_intros*)

thus $(\lambda n. g \ n / \text{Gamma\_series } z \ n * \text{Gamma\_series } z \ n) \longrightarrow \text{Gamma } z$ **by** *simp*

qed

lemma *Gamma_seriesI'*:

assumes $f \longrightarrow r\text{Gamma } z$

assumes $(\lambda n. g \ n * f \ n) \longrightarrow 1$

assumes $z \notin \mathbb{Z}_{\leq 0}$

shows $g \longrightarrow \text{Gamma } z$

proof (*rule Lim_transform_eventually*)

have $1/2 > (0::\text{real})$ **by** *simp*

from *tendstoD[OF assms(2), OF this]* **show** *eventually* $(\lambda n. g \ n * f \ n / f \ n = g \ n)$ *sequentially*

by (*force elim!: eventually_mono simp: dist_real_def*)

from *assms* **have** $(\lambda n. g \ n * f \ n / f \ n) \longrightarrow 1 / r\text{Gamma } z$

by (*intro tendsto_divide assms*) (*simp_all add: rGamma_eq_zero_iff*)

thus $(\lambda n. g \ n * f \ n / f \ n) \longrightarrow \text{Gamma } z$ **by** (*simp add: Gamma_def divide_inverse*)

qed

lemma *Gamma_series'_LIMSEQ*: $\text{Gamma\_series}' \ z \longrightarrow \text{Gamma } z$

by (*cases* $z \in \mathbb{Z}_{\leq 0}$) (*simp_all add: Gamma_nonpos_Int Gamma_seriesI[OF Gamma_series_Gamma_series']*)

Gamma_series'_nonpos_Ints_LIMSEQ[of z])

6.23.4 Differentiability

lemma *has_field_derivative_rGamma_no_nonpos_int*:

assumes $z \notin \mathbb{Z}_{\leq 0}$

shows $(r\text{Gamma } \text{has\_field\_derivative } -r\text{Gamma } z * \text{Digamma } z)$ (at z within A)

proof (*rule has_field_derivative_at_within*)

from *assms* **have** $z \neq - \text{of\_nat } n$ **for** n **by** *auto*

```

from differentiable_rGamma_aux1 [OF this]
show (rGamma has_field_derivative -rGamma z * Digamma z) (at z)
unfolding Digamma_def suminf_def sums_def [abs_def]
has_field_derivative_def has_derivative_def netlimit_at
by (simp add: Let_def bounded_linear_mult_right mult_ac of_real_def [symmetric])
qed

```

```

lemma has_field_derivative_rGamma_nonpos_int:
  (rGamma has_field_derivative  $(-1)^n * \text{fact } n$ ) (at  $(- \text{ of\_nat } n)$  within A)
apply (rule has_field_derivative_at_within)
using differentiable_rGamma_aux2 [of n]
unfolding Let_def has_field_derivative_def has_derivative_def netlimit_at
by (simp only: bounded_linear_mult_right mult_ac of_real_def [symmetric]
  fact_prod) simp

```

```

lemma has_field_derivative_rGamma [derivative_intros]:
  (rGamma has_field_derivative (if  $z \in \mathbb{Z}_{\leq 0}$  then  $(-1)^{\text{nat } \lfloor \text{norm } z \rfloor} * \text{fact } (\text{nat } \lfloor \text{norm } z \rfloor)$ 
  else  $-rGamma z * Digamma z$ ) (at  $z$  within A)
using has_field_derivative_rGamma_no_nonpos_int [of z A]
  has_field_derivative_rGamma_nonpos_int [of nat  $\lfloor \text{norm } z \rfloor$  A]
by (auto elim!: nonpos_Ints_cases')

```

```

declare has_field_derivative_rGamma_no_nonpos_int [THEN DERIV_chain2,
  derivative_intros]
declare has_field_derivative_rGamma [THEN DERIV_chain2, derivative_intros]
declare has_field_derivative_rGamma_nonpos_int [derivative_intros]
declare has_field_derivative_rGamma_no_nonpos_int [derivative_intros]
declare has_field_derivative_rGamma [derivative_intros]

```

```

theorem has_field_derivative_Gamma [derivative_intros]:
   $z \notin \mathbb{Z}_{\leq 0} \implies (\text{Gamma has\_field\_derivative } \text{Gamma } z * \text{Digamma } z)$  (at  $z$  within A)
unfolding Gamma_def [abs_def]
by (fastforce intro!: derivative_eq_intros simp: rGamma_eq_zero_iff)

```

```

declare has_field_derivative_Gamma [THEN DERIV_chain2, derivative_intros]

```

```

hide_fact rGamma_eq_zero_iff_aux differentiable_rGamma_aux1 differentiable_rGamma_aux2
  differentiable_rGamma_aux2 rGamma_series_aux Gamma_class.rGamma_eq_zero_iff_aux

```

```

lemma continuous_on_rGamma [continuous_intros]: continuous_on A rGamma
by (rule DERIV_continuous_on has_field_derivative_rGamma)+

```

```

lemma continuous_on_Gamma [continuous_intros]:  $A \cap \mathbb{Z}_{\leq 0} = \{\}$   $\implies$  continuous_on A Gamma
by (rule DERIV_continuous_on has_field_derivative_Gamma)+ blast

```

lemma *isCont_rGamma* [*continuous_intros*]:
 $isCont\ f\ z \implies isCont\ (\lambda x. rGamma\ (f\ x))\ z$
by (rule *isCont_o2*[*OF* $\_DERIV\_isCont$ [*OF has_field_derivative_rGamma*]])

lemma *isCont_Gamma* [*continuous_intros*]:
 $isCont\ f\ z \implies f\ z \notin \mathbb{Z}_{\leq 0} \implies isCont\ (\lambda x. Gamma\ (f\ x))\ z$
by (rule *isCont_o2*[*OF* $\_DERIV\_isCont$ [*OF has_field_derivative_Gamma*]])

6.23.5 The complex Gamma function

instantiation *complex* :: *Gamma*
begin

definition *rGamma_complex* :: *complex* $\Rightarrow$ *complex* **where**
 $rGamma\_complex\ z = \lim\ (rGamma\_series\ z)$

lemma *rGamma_series_complex_converges*:
 $convergent\ (rGamma\_series\ (z :: complex))\ (\text{is } ?thesis1)$
and *rGamma_complex_altdef*:
 $rGamma\ z = (\text{if } z \in \mathbb{Z}_{\leq 0} \text{ then } 0 \text{ else } exp\ (-ln\_Gamma\ z))\ (\text{is } ?thesis2)$
proof –
have *?thesis1* $\wedge$ *?thesis2*
proof (cases $z \in \mathbb{Z}_{\leq 0}$)
case *False*
have $rGamma\_series\ z \longrightarrow exp\ (-ln\_Gamma\ z)$
proof (rule *Lim_transform_eventually*)
from $ln\_Gamma\_series\_complex\_converges'$ [*OF False*]
obtain *d* **where** $0 < d$ *uniformly_convergent_on* (*ball* $z\ d$) ($\lambda n\ z. ln\_Gamma\_series\ z\ n$)
by *auto*
from *this*(1) *uniformly_convergent_imp_convergent*[*OF this*(2), *of* z]
have $ln\_Gamma\_series\ z \longrightarrow \lim\ (ln\_Gamma\_series\ z)$ **by** (*simp add*:
convergent_LIMSEQ_iff)
thus ($\lambda n. exp\ (-ln\_Gamma\_series\ z\ n)$) $\longrightarrow exp\ (-ln\_Gamma\ z)$
unfolding *convergent_def* ln_Gamma_def **by** (*intro tendsto_exp tendsto_minus*)
from *eventually_gt_at_top*[*of* $0 :: nat$] $exp\_ln\_Gamma\_series\_complex\ False$
show *eventually* ($\lambda n. exp\ (-ln\_Gamma\_series\ z\ n) = rGamma\_series\ z\ n$) *sequentially*
by (*force elim*!: *eventually_mono simp: exp_minus Gamma_series_def rGamma_series_def*)
qed
with *False* **show** *?thesis*
by (*auto simp: convergent_def rGamma_complex_def intro*!: *limI*)
next
case *True*
then obtain *k* **where** $z = -\ of\_nat\ k$ **by** (*erule nonpos_Ints_cases'*)
also have $rGamma\_series\ \dots \longrightarrow 0$
by (*subst tendsto_cong*[*OF rGamma_series_minus_of_nat*]) (*simp_all add*:


```

convergent_const)
  finally show ?thesis using True
    by (auto simp: rGamma_complex_def convergent_def intro!: limI)
qed
thus ?thesis1 ?thesis2 by blast+
qed

context
begin

private lemma rGamma_complex_plus1:  $z * rGamma (z + 1) = rGamma (z :: complex)$ 
proof -
  let ?f =  $\lambda n. (z + 1) * inverse (of\_nat n) + 1$ 
  have eventually ( $\lambda n. ?f n * rGamma\_series z n = z * rGamma\_series (z + 1) n$ ) sequentially
    using eventually_gt_at_top[of 0::nat]
  proof eventually_elim
    fix n :: nat assume n:  $n > 0$ 
    hence  $z * rGamma\_series (z + 1) n = inverse (of\_nat n) * pochhammer z (Suc (Suc n)) / (fact n * exp (z * of\_real (ln (of\_nat n))))$ 
    by (subst pochhammer_rec) (simp add: rGamma_series_def field_simps exp_add exp_of_real)
    also from n have  $\dots = ?f n * rGamma\_series z n$ 
    by (subst pochhammer_rec') (simp_all add: field_split_simps rGamma_series_def add_ac)
    finally show ?f n * rGamma_series z n =  $z * rGamma\_series (z + 1) n$  ..
  qed
  moreover have ( $\lambda n. ?f n * rGamma\_series z n$ )  $\longrightarrow ((z+1) * 0 + 1) * rGamma z$ 
    using rGamma_series_complex_converges
    by (intro tendsto_intros lim_inverse_n)
    (simp_all add: convergent_LIMSEQ_iff rGamma_complex_def)
  hence ( $\lambda n. ?f n * rGamma\_series z n$ )  $\longrightarrow rGamma z$  by simp
  ultimately have ( $\lambda n. z * rGamma\_series (z + 1) n$ )  $\longrightarrow rGamma z$ 
    by (blast intro: Lim_transform_eventually)
  moreover have ( $\lambda n. z * rGamma\_series (z + 1) n$ )  $\longrightarrow z * rGamma (z + 1)$ 
    using rGamma_series_complex_converges
    by (auto intro!: tendsto_mult simp: rGamma_complex_def convergent_LIMSEQ_iff)
  ultimately show  $z * rGamma (z + 1) = rGamma z$  using LIMSEQ_unique
by blast
qed

private lemma has_field_derivative_rGamma_complex_no_nonpos_Int:
  assumes  $(z :: complex) \notin \mathbb{Z}_{\leq 0}$ 
  shows  $(rGamma has\_field\_derivative - rGamma z * Digamma z) (at z)$ 

```

```

proof –
  have diff: (rGamma has_field_derivative – rGamma z * Digamma z) (at z) if
    Re z > 0 for z
  proof (subst DERIV_cong_ev[OF refl _ refl])
    from that have eventually (λt. t ∈ ball z (Re z/2)) (nhds z)
    by (intro eventually_nhds_in_nhd) simp_all
    thus eventually (λt. rGamma t = exp (– ln_Gamma t)) (nhds z)
    using no_nonpos_Int_in_ball_complex[OF that]
    by (auto elim!: eventually_mono simp: rGamma_complex_altdef)
  next
    have z ∉ ℝ≤0 using that by (simp add: complex_nonpos_Reals_iff)
    with that show ((λt. exp (– ln_Gamma t)) has_field_derivative (–rGamma
    z * Digamma z)) (at z)
    by (force elim!: nonpos_Ints_cases intro!: derivative_eq_intros simp: rGamma_complex_altdef)
  qed

from assms show (rGamma has_field_derivative – rGamma z * Digamma z)
(at z)
proof (induction nat [1 – Re z] arbitrary: z)
  case (Suc n z)
    from Suc.prem1 have z: z ≠ 0 by auto
    from Suc.hyps have n = nat [– Re z] by linarith
    hence A: n = nat [1 – Re (z + 1)] by simp
    from Suc.prem2 have B: z + 1 ∉ ℤ≤0 by (force dest: plus_one_in_nonpos_Ints_imp)

    have ((λz. z * (rGamma ∘ (λz. z + 1)) z) has_field_derivative
    –rGamma (z + 1) * (Digamma (z + 1) * z – 1)) (at z)
    by (rule derivative_eq_intros DERIV_chain Suc refl A B)+ (simp add:
    algebra_simps)
    also have (λz. z * (rGamma ∘ (λz. z + 1 :: complex)) z) = rGamma
    by (simp add: rGamma_complex_plus1)
    also from z have Digamma (z + 1) * z – 1 = z * Digamma z
    by (subst Digamma_plus1) (simp_all add: field_simps)
    also have –rGamma (z + 1) * (z * Digamma z) = –rGamma z * Digamma z
    by (simp add: rGamma_complex_plus1[of z, symmetric])
    finally show ?case .
  qed (intro diff, simp)
qed

private lemma rGamma_complex_1: rGamma (1 :: complex) = 1
proof –
  have A: eventually (λn. rGamma_series 1 n = of_nat (Suc n) / of_nat n)
    sequentially
    using eventually_gt_at_top[of 0::nat]
    by (force elim!: eventually_mono simp: rGamma_series_def exp_of_real pochhammer_fact
    field_split_simps pochhammer_rec' dest!: pochhammer_eq_0_imp_nonpos_Int)
  have rGamma_series 1 → 1 by (subst tendsto_cong[OF A]) (rule LIM-
    SEQ_Suc_n_over_n)

```

thus $rGamma\ 1 = (1 :: complex)$ **unfolding** $rGamma\_complex\_def$ **by** $(rule\ limI)$
qed

private lemma $has\_field\_derivative\_rGamma\_complex\_nonpos\_Int$:
 $(rGamma\ has\_field\_derivative\ (-1)^n * fact\ n)\ (at\ (-\ of\_nat\ n :: complex))$
proof $(induction\ n)$
case 0
have $A: (0 :: complex) + 1 \notin \mathbb{Z}_{\leq 0}$ **by** $simp$
have $((\lambda z. z * (rGamma \circ (\lambda z. z + 1 :: complex)))\ z)\ has\_field\_derivative\ 1)$
 $(at\ 0)$
by $(rule\ derivative\_eq\_intros\ DERIV\_chain\ refl$
 $has\_field\_derivative\_rGamma\_complex\_no\_nonpos\_Int\ A) + (simp$
 $add: rGamma\_complex\_1)$
thus $?case$ **by** $(simp\ add: rGamma\_complex\_plus1)$
next
case $(Suc\ n)$
hence $A: (rGamma\ has\_field\_derivative\ (-1)^n * fact\ n)$
 $(at\ (-\ of\_nat\ (Suc\ n) + 1 :: complex))$ **by** $simp$
have $((\lambda z. z * (rGamma \circ (\lambda z. z + 1 :: complex)))\ z)\ has\_field\_derivative$
 $(-1)^{Suc\ n} * fact\ (Suc\ n)\ (at\ (-\ of\_nat\ (Suc\ n)))$
by $(rule\ derivative\_eq\_intros\ refl\ A\ DERIV\_chain) +$
 $(simp\ add: algebra\_simps\ rGamma\_complex\_altdef)$
thus $?case$ **by** $(simp\ add: rGamma\_complex\_plus1)$
qed

instance proof

fix $z :: complex$ **show** $(rGamma\ z = 0) \longleftrightarrow (\exists n. z = -\ of\_nat\ n)$
by $(auto\ simp: rGamma\_complex\_altdef\ elim!: nonpos\_Ints\_cases')$
next
fix $z :: complex$ **assume** $\bigwedge n. z \neq -\ of\_nat\ n$
hence $z \notin \mathbb{Z}_{\leq 0}$ **by** $(auto\ elim!: nonpos\_Ints\_cases')$
from $has\_field\_derivative\_rGamma\_complex\_no\_nonpos\_Int$ $[OF\ this]$
show $let\ d = (THE\ d. (\lambda n. \sum k < n. inverse\ (of\_nat\ (Suc\ k)) - inverse\ (z +$
 $of\_nat\ k))$
 $\longrightarrow d) - euler\_mascheroni *_{\mathbb{R}} 1\ in\ (\lambda y. (rGamma\ y -$
 $rGamma\ z +$
 $rGamma\ z * d * (y - z)) /_{\mathbb{R}}\ cmod\ (y - z)) - z \rightarrow 0$
by $(simp\ add: has\_field\_derivative\_def\ has\_derivative\_def\ Digamma\_def$
 $sums\_def\ [abs\_def]$
 $of\_real\_def[symmetric]\ suminf\_def)$
next
fix $n :: nat$
from $has\_field\_derivative\_rGamma\_complex\_nonpos\_Int$ $[of\ n]$
show $let\ z = -\ of\_nat\ n\ in\ (\lambda y. (rGamma\ y - rGamma\ z - (-1)^n * prod$
 $of\_nat\ \{1..n\} *$
 $(y - z)) /_{\mathbb{R}}\ cmod\ (y - z)) - z \rightarrow 0$
by $(simp\ add: has\_field\_derivative\_def\ has\_derivative\_def\ fact\_prod\ Let\_def)$
next

```

fix z :: complex
from rGamma_series_complex_converges[of z] have rGamma_series z  $\longrightarrow$ 
rGamma z
  by (simp add: convergent_LIMSEQ_iff rGamma_complex_def)
  thus let fact' =  $\lambda n.$  prod of_nat {1..n};
      exp =  $\lambda x.$  THE e. ( $\lambda n.$   $\sum k < n. x^k /_R \text{fact } k$ )  $\longrightarrow$  e;
      pochhammer' =  $\lambda a n.$   $\prod n = 0..n. a + \text{of\_nat } n$ 
      in ( $\lambda n.$  pochhammer' z n / (fact' n * exp (z * ln (real_of_nat n) *_R 1)))
 $\longrightarrow$  rGamma z
  by (simp add: fact_prod pochhammer_Suc_prod rGamma_series_def [abs_def]
exp_def
      of_real_def [symmetric] suminf_def sums_def [abs_def] atLeast0At-
Most)
qed

end
end

```

```

lemma Gamma_complex_altdef:
  Gamma z = (if z  $\in \mathbb{Z}_{\leq 0}$  then 0 else exp (ln_Gamma (z :: complex)))
  unfolding Gamma_def rGamma_complex_altdef by (simp add: exp_minus)

```

```

lemma cnj_rGamma: cnj (rGamma z) = rGamma (cnj z)
proof -
  have rGamma_series (cnj z) = ( $\lambda n.$  cnj (rGamma_series z n))
    by (intro ext) (simp_all add: rGamma_series_def exp_cnj)
  also have ...  $\longrightarrow$  cnj (rGamma z) by (intro tendsto_cnj tendsto_intros)
  finally show ?thesis unfolding rGamma_complex_def by (intro sym[OF limI])
qed

```

```

lemma cnj_Gamma: cnj (Gamma z) = Gamma (cnj z)
  unfolding Gamma_def by (simp add: cnj_rGamma)

```

```

lemma Gamma_complex_real:
  z  $\in \mathbb{R} \implies$  Gamma z  $\in (\mathbb{R} :: \text{complex set})$  and rGamma_complex_real: z  $\in \mathbb{R} \implies$ 
rGamma z  $\in \mathbb{R}$ 
  by (simp_all add: Reals_cnj_iff cnj_Gamma cnj_rGamma)

```

```

lemma field_differentiable_rGamma: rGamma field_differentiable (at z within A)
  using has_field_derivative_rGamma[of z] unfolding field_differentiable_def
by blast

```

```

lemma holomorphic_rGamma [holomorphic_intros]: rGamma holomorphic_on A
  unfolding holomorphic_on_def by (auto intro!: field_differentiable_rGamma)

```

```

lemma holomorphic_rGamma' [holomorphic_intros]:
  assumes f holomorphic_on A
  shows ( $\lambda x.$  rGamma (f x)) holomorphic_on A

```

proof –

have $r\text{Gamma} \circ f$ *holomorphic_on* A **using** *assms*
 by (*intro holomorphic_on_compose assms holomorphic_rGamma*)
 thus ?thesis **by** (*simp only: o_def*)

qed

lemma *analytic_rGamma*: $r\text{Gamma}$ *analytic_on* A
unfolding *analytic_on_def* **by** (*auto intro!: exI[of _ 1] holomorphic_rGamma*)

lemma *field_differentiable_Gamma*: $z \notin \mathbb{Z}_{\leq 0} \implies \text{Gamma}$ *field_differentiable* (at z within A)
using *has_field_derivative_Gamma[of z]* **unfolding** *field_differentiable_def* **by** *auto*

lemma *holomorphic_Gamma* [*holomorphic_intros*]: $A \cap \mathbb{Z}_{\leq 0} = \{\}$ $\implies \text{Gamma}$ *holomorphic_on* A
unfolding *holomorphic_on_def* **by** (*auto intro!: field_differentiable_Gamma*)

lemma *holomorphic_Gamma'* [*holomorphic_intros*]:
assumes f *holomorphic_on* A **and** $\bigwedge x. x \in A \implies f\ x \notin \mathbb{Z}_{\leq 0}$
shows $(\lambda x. \text{Gamma}\ (f\ x))$ *holomorphic_on* A

proof –

have $\text{Gamma} \circ f$ *holomorphic_on* A **using** *assms*
 by (*intro holomorphic_on_compose assms holomorphic_Gamma*) *auto*
 thus ?thesis **by** (*simp only: o_def*)

qed

lemma *analytic_Gamma*: $A \cap \mathbb{Z}_{\leq 0} = \{\}$ $\implies \text{Gamma}$ *analytic_on* A
by (*rule analytic_on_subset[of _ UNIV - $\mathbb{Z}_{\leq 0}$], subst analytic_on_open*)
 (*auto intro!: holomorphic_Gamma*)

lemma *field_differentiable_ln_Gamma_complex*:
 $z \notin \mathbb{R}_{\leq 0} \implies \ln\_Gamma$ *field_differentiable* (at $(z::\text{complex})$ within A)
by (*rule field_differentiable_within_subset[of _ UNIV]*)
 (*force simp: field_differentiable_def intro!: derivative_intros*)**+**

lemma *holomorphic_ln_Gamma* [*holomorphic_intros*]: $A \cap \mathbb{R}_{\leq 0} = \{\}$ $\implies \ln\_Gamma$ *holomorphic_on* A
unfolding *holomorphic_on_def* **by** (*auto intro!: field_differentiable_ln_Gamma_complex*)

lemma *analytic_ln_Gamma*: $A \cap \mathbb{R}_{\leq 0} = \{\}$ $\implies \ln\_Gamma$ *analytic_on* A
by (*rule analytic_on_subset[of _ UNIV - $\mathbb{R}_{\leq 0}$], subst analytic_on_open*)
 (*auto intro!: holomorphic_ln_Gamma*)

lemma *has_field_derivative_rGamma_complex'* [*derivative_intros*]:
 $(r\text{Gamma}\ \text{has\_field\_derivative}\ (\text{if } z \in \mathbb{Z}_{\leq 0} \text{ then } (-1)^{\wedge(\text{nat } \lfloor -\text{Re } z \rfloor)} * \text{fact } (\text{nat } \lfloor -\text{Re } z \rfloor))$

```

[ $-Re\ z$ ]) else
   $-rGamma\ z * Digamma\ z$ ) (at  $z$  within  $A$ )
  using has_field_derivative_rGamma[of  $z$ ] by (auto elim!: nonpos_Ints_cases')

```

```

declare has_field_derivative_rGamma_complex'[THEN DERIV_chain2, derivative_intros]

```

```

lemma field_differentiable_Polygamma:
  fixes  $z :: complex$ 
  shows
     $z \notin \mathbb{Z}_{\leq 0} \implies Polygamma\ n\ field\_differentiable$  (at  $z$  within  $A$ )
  using has_field_derivative_Polygamma[of  $z\ n$ ] unfolding field_differentiable_def
  by auto

```

```

lemma holomorphic_on_Polygamma [holomorphic_intros]:  $A \cap \mathbb{Z}_{\leq 0} = \{\}$   $\implies$ 
   $Polygamma\ n\ holomorphic\_on\ A$ 
  unfolding holomorphic_on_def by (auto intro!: field_differentiable_Polygamma)

```

```

lemma analytic_on_Polygamma:  $A \cap \mathbb{Z}_{\leq 0} = \{\}$   $\implies Polygamma\ n\ analytic\_on\ A$ 
  by (rule analytic_on_subset[of UNIV  $-\mathbb{Z}_{\leq 0}$ ], subst analytic_on_open)
  (auto intro!: holomorphic_on_Polygamma)

```

6.23.6 The real Gamma function

```

lemma rGamma_series_real:
  eventually ( $\lambda n. rGamma\_series\ x\ n = Re\ (rGamma\_series\ (of\_real\ x)\ n)$ ) sequentially
  using eventually_gt_at_top[of 0 :: nat]
proof eventually_elim
  fix  $n :: nat$  assume  $n > 0$ 
  have  $Re\ (rGamma\_series\ (of\_real\ x)\ n) =$ 
     $Re\ (of\_real\ (pochhammer\ x\ (Suc\ n)) / (fact\ n * exp\ (of\_real\ (x * ln\ (real\_of\_nat\ n))))))$ 
  using  $n$  by (simp add: rGamma_series_def powr_def pochhammer_of_real)
  also from  $n$  have  $\dots = Re\ (of\_real\ ((pochhammer\ x\ (Suc\ n)) /$ 
     $(fact\ n * (exp\ (x * ln\ (real\_of\_nat\ n))))))$ 
  by (subst exp_of_real) simp
  also from  $n$  have  $\dots = rGamma\_series\ x\ n$ 
  by (subst Re_complex_of_real) (simp add: rGamma_series_def powr_def)
  finally show  $rGamma\_series\ x\ n = Re\ (rGamma\_series\ (of\_real\ x)\ n) ..$ 
qed

```

```

instantiation real :: Gamma
begin

```

```

definition rGamma_real  $x = Re\ (rGamma\ (of\_real\ x :: complex))$ 

```

instance proof

```

  fix x :: real
  have rGamma x = Re (rGamma (of_real x)) by (simp add: rGamma_real_def)
  also have of_real ... = rGamma (of_real x :: complex)
    by (intro of_real_Re rGamma_complex_real) simp_all
  also have ... = 0  $\longleftrightarrow$   $x \in \mathbb{Z}_{\leq 0}$  by (simp add: rGamma_eq_zero_iff_of_real_in_nonpos_Ints_iff)
  also have ...  $\longleftrightarrow$   $(\exists n. x = - \text{of\_nat } n)$  by (auto elim!: nonpos_Ints_cases')
  finally show (rGamma x) = 0  $\longleftrightarrow$   $(\exists n. x = - \text{real\_of\_nat } n)$  by simp
next
  fix x :: real assume  $\bigwedge n. x \neq - \text{of\_nat } n$ 
  hence x: complex_of_real x  $\notin \mathbb{Z}_{\leq 0}$ 
    by (subst of_real_in_nonpos_Ints_iff) (auto elim!: nonpos_Ints_cases')
  then have x  $\neq 0$  by auto
  with x have (rGamma has_field_derivative - rGamma x * Digamma x) (at x)
    by (fastforce intro!: derivative_eq_intros has_vector_derivative_real_field
      simp: Polygamma_of_real rGamma_real_def [abs_def])
  thus let d = (THE d.  $(\lambda n. \sum k < n. \text{inverse} (\text{of\_nat } (\text{Suc } k)) - \text{inverse} (x + \text{of\_nat } k))$ 
     $\longrightarrow d$ ) - euler_mascheroni *R 1 in  $(\lambda y. (rGamma y - rGamma x +$ 
     $rGamma x * d * (y - x)) /_R \text{norm } (y - x)) - x \rightarrow 0$ 
    by (simp add: has_field_derivative_def has_derivative_def Digamma_def
      sums_def [abs_def]
      of_real_def[symmetric] suminf_def)
next
  fix n :: nat
  have (rGamma has_field_derivative  $(-1)^n * \text{fact } n$ ) (at  $(- \text{of\_nat } n :: \text{real})$ )
    by (fastforce intro!: derivative_eq_intros has_vector_derivative_real_field
      simp: Polygamma_of_real rGamma_real_def [abs_def])
  thus let x =  $- \text{of\_nat } n$  in  $(\lambda y. (rGamma y - rGamma x - (-1)^n * \text{prod of\_nat } \{1..n\} *$ 
     $(y - x)) /_R \text{norm } (y - x)) - x :: \text{real} \rightarrow 0$ 
    by (simp add: has_field_derivative_def has_derivative_def fact_prod Let_def)
next
  fix x :: real
  have rGamma_series x  $\longrightarrow$  rGamma x
  proof (rule Lim_transform_eventually)
    show  $(\lambda n. \text{Re } (rGamma\_series (\text{of\_real } x) n)) \longrightarrow rGamma x$  unfolding
      rGamma_real_def
    by (intro tendsto_intros)
  qed (insert rGamma_series_real, simp add: eq_commute)
  thus let fact' =  $\lambda n. \text{prod of\_nat } \{1..n\}$ ;
    exp =  $\lambda x. \text{THE } e. (\lambda n. \sum k < n. x^k /_R \text{fact } k) \longrightarrow e$ ;
    pochhammer' =  $\lambda a n. \prod n = 0..n. a + \text{of\_nat } n$ 
    in  $(\lambda n. \text{pochhammer}' x n / (\text{fact}' n * \text{exp } (x * \ln (\text{real\_of\_nat } n) *_{\text{R}} 1)))$ 
 $\longrightarrow$  rGamma x
  by (simp add: fact_prod pochhammer_Suc_prod rGamma_series_def [abs_def]
    exp_def
    of_real_def [symmetric] suminf_def sums_def [abs_def] atLeast0At-
```

2444

Most)

qed

end

lemma *rGamma_complex_of_real*: $rGamma\ (complex\_of\_real\ x) = complex\_of\_real\ (rGamma\ x)$

unfolding *rGamma_real_def* **using** *rGamma_complex_real* **by** *simp*

lemma *Gamma_complex_of_real*: $Gamma\ (complex\_of\_real\ x) = complex\_of\_real\ (Gamma\ x)$

unfolding *Gamma_def* **by** (*simp add: rGamma_complex_of_real*)

lemma *rGamma_real_altdef*: $rGamma\ x = \lim\ (rGamma\_series\ (x :: real))$

by (*rule sym, rule limI, rule tendsto_intros*)

lemma *Gamma_real_altdef1*: $Gamma\ x = \lim\ (Gamma\_series\ (x :: real))$

by (*rule sym, rule limI, rule tendsto_intros*)

lemma *Gamma_real_altdef2*: $Gamma\ x = Re\ (Gamma\ (of\_real\ x))$

using *rGamma_complex_real*[*OF Reals_of_real*[*of x*]]

by (*elim Reals_cases*)

(*simp only: Gamma_def rGamma_real_def of_real_inverse[symmetric] Re_complex_of_real*)

lemma *ln_Gamma_series_complex_of_real*:

$x > 0 \implies n > 0 \implies ln\_Gamma\_series\ (complex\_of\_real\ x)\ n = of\_real\ (ln\_Gamma\_series\ x\ n)$

proof –

assume *xn*: $x > 0\ n > 0$

have $Ln\ (complex\_of\_real\ x / of\_nat\ k + 1) = of\_real\ (ln\ (x / of\_nat\ k + 1))$

if $k \geq 1$ **for** *k*

using *that xn* **by** (*subst Ln_of_real [symmetric]*) (*auto intro!: add_nonneg_pos simp: field_simps*)

with *xn* **show** *?thesis* **by** (*simp add: ln_Gamma_series_def Ln_of_real*)

qed

lemma *ln_Gamma_real_converges*:

assumes $(x :: real) > 0$

shows *convergent* ($ln\_Gamma\_series\ x$)

proof –

have $(\lambda n. ln\_Gamma\_series\ (complex\_of\_real\ x)\ n) \longrightarrow ln\_Gamma\ (of\_real\ x)$ **using** *assms*

by (*intro ln_Gamma_complex_LIMSEQ*) (*auto simp: of_real_in_nonpos_Ints_iff*)

moreover from *eventually_gt_at_top*[*of 0::nat*]

have *eventually* $(\lambda n. complex\_of\_real\ (ln\_Gamma\_series\ x\ n) =$

$ln\_Gamma\_series\ (complex\_of\_real\ x)\ n)$ *sequentially*

by *eventually_elim* (*simp add: ln_Gamma_series_complex_of_real assms*)

ultimately have $(\lambda n. complex\_of\_real\ (ln\_Gamma\_series\ x\ n)) \longrightarrow ln\_Gamma$


```

(of_real x)
  by (subst tendsto_cong) assumption+
  from tendsto_Re[OF this] show ?thesis by (auto simp: convergent_def)
qed

```

```

lemma ln_Gamma_real_LIMSEQ: (x::real) > 0  $\implies$  ln_Gamma_series x  $\longrightarrow$ 
ln_Gamma x
  using ln_Gamma_real_converges[of x] unfolding ln_Gamma_def by (simp
add: convergent_LIMSEQ_iff)

```

```

lemma ln_Gamma_complex_of_real: x > 0  $\implies$  ln_Gamma (complex_of_real
x) = of_real (ln_Gamma x)
proof (unfold ln_Gamma_def, rule limI, rule Lim_transform_eventually)
  assume x: x > 0
  show eventually ( $\lambda n$ . of_real (ln_Gamma_series x n) =
    ln_Gamma_series (complex_of_real x) n) sequentially
    using eventually_gt_at_top[of 0::nat]
    by eventually_elim (simp add: ln_Gamma_series_complex_of_real x)
qed (intro tendsto_of_real, insert ln_Gamma_real_LIMSEQ[of x], simp add:
ln_Gamma_def)

```

```

lemma Gamma_real_pos_exp: x > (0 :: real)  $\implies$  Gamma x = exp (ln_Gamma
x)
  by (auto simp: Gamma_real_altdef2 Gamma_complex_altdef of_real_in_nonpos_Ints_iff
    ln_Gamma_complex_of_real exp_of_real)

```

```

lemma ln_Gamma_real_pos: x > 0  $\implies$  ln_Gamma x = ln (Gamma x :: real)
  unfolding Gamma_real_pos_exp by simp

```

```

lemma ln_Gamma_complex_conv_fact: n > 0  $\implies$  ln_Gamma (of_nat n :: com-
plex) = ln (fact (n - 1))
  using ln_Gamma_complex_of_real[of real n] Gamma_fact[of n - 1, where 'a
= real]
  by (simp add: ln_Gamma_real_pos of_nat_diff Ln_of_real [symmetric])

```

```

lemma ln_Gamma_real_conv_fact: n > 0  $\implies$  ln_Gamma (real n) = ln (fact (n
- 1))
  using Gamma_fact[of n - 1, where 'a = real]
  by (simp add: ln_Gamma_real_pos of_nat_diff Ln_of_real [symmetric])

```

```

lemma Gamma_real_pos [simp, intro]: x > (0::real)  $\implies$  Gamma x > 0
  by (simp add: Gamma_real_pos_exp)

```

```

lemma Gamma_real_nonneg [simp, intro]: x > (0::real)  $\implies$  Gamma x  $\geq$  0
  by (simp add: Gamma_real_pos_exp)

```

```

lemma has_field_derivative_ln_Gamma_real [derivative_intros]:
  assumes x: x > (0::real)
  shows (ln_Gamma has_field_derivative Digamma x) (at x)

```

```

proof (subst DERIV_cong_ev[OF refl _ refl])
  from assms show ((Re ∘ ln_Gamma ∘ complex_of_real) has_field_derivative
    Digamma x) (at x)
  by (auto intro!: derivative_eq_intros has_vector_derivative_real_field
    simp: Polygamma_of_real o_def)
  from eventually_nhds_in_nhd[of x {0<..}] assms
  show eventually (λy. ln_Gamma y = (Re ∘ ln_Gamma ∘ of_real) y) (nhds x)
  by (auto elim!: eventually_mono simp: ln_Gamma_complex_of_real inte-
    rior_open)
qed

```

```

lemma field_differentiable_ln_Gamma_real:
  x > 0 ⟹ ln_Gamma field_differentiable (at (x::real) within A)
  by (rule field_differentiable_within_subset[of _ UNIV])
  (auto simp: field_differentiable_def intro!: derivative_intros)+

```

```

declare has_field_derivative_ln_Gamma_real[THEN DERIV_chain2, derivative_intros]

```

```

lemma deriv_ln_Gamma_real:
  assumes z > 0
  shows deriv ln_Gamma z = Digamma (z :: real)
  by (intro DERIV_imp_deriv has_field_derivative_ln_Gamma_real assms)

```

```

lemma higher_deriv_ln_Gamma_real:
  assumes (x::real) > 0
  shows (deriv  $\sim$  j) ln_Gamma x = (if j = 0 then ln_Gamma x else Polygamma
    (j - 1) x)
proof (cases j)
  case (Suc j')
  have (deriv  $\sim$  j') (deriv ln_Gamma) x = (deriv  $\sim$  j') Digamma x
  using eventually_nhds_in_open[of {0<..} x] assms
  by (intro higher_deriv_cong_ev refl)
  (auto elim!: eventually_mono simp: open_Diff deriv_ln_Gamma_real)
  also have ... = Polygamma j' x using assms
  by (subst higher_deriv_Polygamma)
  (auto elim!: nonpos_Ints_cases simp: complex_nonpos_Reals_iff)
  finally show ?thesis using Suc by (simp del: funpow.simps add: funpow_Suc_right)
qed simp_all

```

```

lemma higher_deriv_ln_Gamma_complex_of_real:
  assumes (x :: real) > 0
  shows (deriv  $\sim$  j) ln_Gamma (complex_of_real x) = of_real ((deriv  $\sim$  j)
    ln_Gamma x)
  using assms
  by (auto simp: higher_deriv_ln_Gamma_real higher_deriv_ln_Gamma_complex
    ln_Gamma_complex_of_real Polygamma_of_real)

```

```

lemma has_field_derivative_rGamma_real' [derivative_intros]:
  (rGamma has_field_derivative (if x ∈ ℤ≤₀ then (-1)^(nat ⌊-x⌋) * fact (nat

```

```

| $-x$ ]) else
   $-rGamma\ x * Digamma\ x$ ) (at  $x$  within  $A$ )
  using has_field_derivative_rGamma[of  $x$ ] by (force elim!: nonpos_Ints_cases')

declare has_field_derivative_rGamma_real'[THEN DERIV_chain2, derivative_intros]

lemma Polygamma_real_odd_pos:
  assumes  $(x::real) \notin \mathbb{Z}_{\leq 0}$  odd  $n$ 
  shows  $Polygamma\ n\ x > 0$ 
proof -
  from assms have  $x \neq 0$  by auto
  with assms show ?thesis
    unfolding Polygamma_def using Polygamma_converges'[of  $x\ Suc\ n$ ]
    by (auto simp: zero_less_power_eq simp del: power_Suc
      dest: plus_of_nat_eq_0_imp intro!: mult_pos_pos suminf_pos)
qed

lemma Polygamma_real_even_neg:
  assumes  $(x::real) > 0$   $n > 0$  even  $n$ 
  shows  $Polygamma\ n\ x < 0$ 
  using assms unfolding Polygamma_def using Polygamma_converges'[of  $x\ Suc\ n$ ]
  by (auto intro!: mult_pos_pos suminf_pos)

lemma Polygamma_real_strict_mono:
  assumes  $x > 0$   $x < (y::real)$  even  $n$ 
  shows  $Polygamma\ n\ x < Polygamma\ n\ y$ 
proof -
  have  $\exists \xi. x < \xi \wedge \xi < y \wedge Polygamma\ n\ y - Polygamma\ n\ x = (y - x) * Polygamma\ (Suc\ n)\ \xi$ 
  using assms by (intro MVT2 derivative_intros impI allI) (auto elim!: nonpos_Ints_cases)
  then obtain  $\xi$ 
  where  $\xi: x < \xi \wedge \xi < y$ 
  and Polygamma:  $Polygamma\ n\ y - Polygamma\ n\ x = (y - x) * Polygamma\ (Suc\ n)\ \xi$ 
  by auto
  note Polygamma
  also from  $\xi$  assms have  $(y - x) * Polygamma\ (Suc\ n)\ \xi > 0$ 
  by (intro mult_pos_pos Polygamma_real_odd_pos) (auto elim!: nonpos_Ints_cases)
  finally show ?thesis by simp
qed

lemma Polygamma_real_strict_antimono:
  assumes  $x > 0$   $x < (y::real)$  odd  $n$ 
  shows  $Polygamma\ n\ x > Polygamma\ n\ y$ 
proof -
  have  $\exists \xi. x < \xi \wedge \xi < y \wedge Polygamma\ n\ y - Polygamma\ n\ x = (y - x) * Polygamma\ (Suc\ n)\ \xi$ 

```

```

    using assms by (intro MVT2 derivative_intros impI allI) (auto elim!: non-
pos_Ints_cases)
  then obtain  $\xi$ 
    where  $\xi: x < \xi < y$ 
    and Polygamma:  $\text{Polygamma } n \ y - \text{Polygamma } n \ x = (y - x) * \text{Polygamma}$ 
    (Suc n)  $\xi$ 
    by auto
  note Polygamma
  also from  $\xi$  assms have  $(y - x) * \text{Polygamma} (\text{Suc } n) \ \xi < 0$ 
    by (intro mult_pos_neg Polygamma_real_even_neg) simp_all
  finally show ?thesis by simp
qed

```

```

lemma Polygamma_real_mono:
  assumes  $x > 0 \ x \leq (y::\text{real})$  even n
  shows  $\text{Polygamma } n \ x \leq \text{Polygamma } n \ y$ 
  using Polygamma_real_strict_mono[OF assms(1) _ assms(3), of y] assms(2)
  by (cases  $x = y$ ) simp_all

```

```

lemma Digamma_real_strict_mono:  $(0::\text{real}) < x \implies x < y \implies \text{Digamma } x <$ 
 $\text{Digamma } y$ 
  by (rule Polygamma_real_strict_mono) simp_all

```

```

lemma Digamma_real_mono:  $(0::\text{real}) < x \implies x \leq y \implies \text{Digamma } x \leq \text{Digamma}$ 
 $y$ 
  by (rule Polygamma_real_mono) simp_all

```

```

lemma Digamma_real_ge_three_halves_pos:
  assumes  $x \geq 3/2$ 
  shows  $\text{Digamma } (x :: \text{real}) > 0$ 
proof -
  have  $0 < \text{Digamma } (3/2 :: \text{real})$  by (fact Digamma_real_three_halves_pos)
  also from assms have  $\dots \leq \text{Digamma } x$  by (intro Polygamma_real_mono)
  simp_all
  finally show ?thesis .
qed

```

```

lemma ln_Gamma_real_strict_mono:
  assumes  $x \geq 3/2 \ x < y$ 
  shows  $\ln\_Gamma \ (x :: \text{real}) < \ln\_Gamma \ y$ 
proof -
  have  $\exists \xi. x < \xi \wedge \xi < y \wedge \ln\_Gamma \ y - \ln\_Gamma \ x = (y - x) * \text{Digamma}$ 
 $\xi$ 
    using assms by (intro MVT2 derivative_intros impI allI) (auto elim!: non-
pos_Ints_cases)
  then obtain  $\xi$  where  $\xi: x < \xi < y$ 
    and ln_Gamma:  $\ln\_Gamma \ y - \ln\_Gamma \ x = (y - x) * \text{Digamma } \xi$ 
    by auto
  note ln_Gamma

```

```

also from  $\xi$  assms have  $(y - x) * \text{Digamma } \xi > 0$ 
by (intro mult_pos_pos Digamma_real_ge_three_halves_pos) simp_all
finally show ?thesis by simp
qed

lemma Gamma_real_strict_mono:
  assumes  $x \geq 3/2$   $x < y$ 
  shows  $\text{Gamma } (x :: \text{real}) < \text{Gamma } y$ 
proof -
  from Gamma_real_pos_exp[of  $x$ ] assms have  $\text{Gamma } x = \exp (\ln \text{Gamma } x)$ 
by simp
  also have  $\dots < \exp (\ln \text{Gamma } y)$  by (intro exp_less_mono ln_Gamma_real_strict_mono
assms)
  also from Gamma_real_pos_exp[of  $y$ ] assms have  $\dots = \text{Gamma } y$  by simp
  finally show ?thesis .
qed

theorem log_convex_Gamma_real: convex_on  $\{0 < ..\}$   $(\ln \circ \text{Gamma} :: \text{real} \Rightarrow \text{real})$ 
by (rule convex_on_realI[of _ _ Digamma])
  (auto intro!: derivative_eq_intros Polygamma_real_mono Gamma_real_pos
    simp: o_def Gamma_eq_zero_iff elim!: nonpos_Ints_cases')

```

6.23.7 The uniqueness of the real Gamma function

The following is a proof of the Bohr–Mollerup theorem, which states that any log-convex function G on the positive reals that fulfils $G(1) = 1$ and satisfies the functional equation $G(x + 1) = x G(x)$ must be equal to the Gamma function. In principle, if G is a holomorphic complex function, one could then extend this from the positive reals to the entire complex plane (minus the non-positive integers, where the Gamma function is not defined).

```

context
  fixes  $G :: \text{real} \Rightarrow \text{real}$ 
  assumes  $G\_1$ :  $G\ 1 = 1$ 
  assumes  $G\_plus1$ :  $x > 0 \implies G\ (x + 1) = x * G\ x$ 
  assumes  $G\_pos$ :  $x > 0 \implies G\ x > 0$ 
  assumes  $\text{log\_convex\_}G$ : convex_on  $\{0 < ..\}$   $(\ln \circ G)$ 
begin

```

```

private lemma  $G\_fact$ :  $G\ (\text{of\_nat } n + 1) = \text{fact } n$ 
  using  $G\_plus1$  [of  $\text{real } n + 1$  for  $n$ ]
  by (induction  $n$ ) (simp_all add:  $G\_1$   $G\_plus1$ )

```

```

private definition  $S :: \text{real} \Rightarrow \text{real} \Rightarrow \text{real}$  where
   $S\ x\ y = (\ln (G\ y) - \ln (G\ x)) / (y - x)$ 

```

```

private lemma  $S\_eq$ :
   $n \geq 2 \implies S\ (\text{of\_nat } n)\ (\text{of\_nat } n + x) = (\ln (G\ (\text{real } n + x)) - \ln (\text{fact } (n -$ 

```

$1))) / x$
by (subst G_fact [symmetric]) (simp add: S_def add_ac of_nat_diff)

private lemma G_lower :
assumes $x: x > 0$ **and** $n: n \geq 1$
shows $\Gamma\_series\ x\ n \leq G\ x$
proof –
have $(\ln \circ G)\ (\text{real}\ (Suc\ n)) \leq ((\ln \circ G)\ (\text{real}\ (Suc\ n) + x) -$
 $(\ln \circ G)\ (\text{real}\ (Suc\ n) - 1)) / (\text{real}\ (Suc\ n) + x - (\text{real}\ (Suc\ n) - 1)) *$
 $(\text{real}\ (Suc\ n) - (\text{real}\ (Suc\ n) - 1)) + (\ln \circ G)\ (\text{real}\ (Suc\ n) - 1)$
using $x\ n$ **by** (intro convex_onD_Icc' convex_on_subset[OF log_convex_G])
auto
hence $S\ (\text{of\_nat}\ n)\ (\text{of\_nat}\ (Suc\ n)) \leq S\ (\text{of\_nat}\ (Suc\ n))\ (\text{of\_nat}\ (Suc\ n) +$
 $x)$
unfolding S_def **using** x **by** (simp add: field_simps)
also have $S\ (\text{of\_nat}\ n)\ (\text{of\_nat}\ (Suc\ n)) = \ln\ (\text{fact}\ n) - \ln\ (\text{fact}\ (n-1))$
unfolding S_def **using** n
by (subst (1 2) G_fact [symmetric]) (simp_all add: add_ac of_nat_diff)
also have $\dots = \ln\ (\text{fact}\ n / \text{fact}\ (n-1))$ **by** (subst ln_div) simp_all
also from n **have** $\text{fact}\ n / \text{fact}\ (n-1) = n$ **by** (cases n) simp_all
finally have $x * \ln\ (\text{real}\ n) + \ln\ (\text{fact}\ n) \leq \ln\ (G\ (\text{real}\ (Suc\ n) + x))$
using $x\ n$ **by** (subst (asm) S_eq) (simp_all add: field_simps)
also have $x * \ln\ (\text{real}\ n) + \ln\ (\text{fact}\ n) = \ln\ (\exp\ (x * \ln\ (\text{real}\ n)) * \text{fact}\ n)$
using x **by** (simp add: ln_mult)
finally have $\exp\ (x * \ln\ (\text{real}\ n)) * \text{fact}\ n \leq G\ (\text{real}\ (Suc\ n) + x)$ **using** x
by (subst (asm) ln_le_cancel_iff) (simp_all add: G_pos)
also have $G\ (\text{real}\ (Suc\ n) + x) = \text{pochhammer}\ x\ (Suc\ n) * G\ x$
using G_plus1 [of $\text{real}\ (Suc\ n) + x$ for n] G_plus1 [of x] x
by (induction n) (simp_all add: pochhammer_Suc add_ac)
finally show $\Gamma\_series\ x\ n \leq G\ x$
using x **by** (simp add: field_simps pochhammer_pos Γ_series_def)
qed

private lemma G_upper :
assumes $x: x > 0\ x \leq 1$ **and** $n: n \geq 2$
shows $G\ x \leq \Gamma\_series\ x\ n * (1 + x / \text{real}\ n)$
proof –
have $(\ln \circ G)\ (\text{real}\ n + x) \leq ((\ln \circ G)\ (\text{real}\ n + 1) -$
 $(\ln \circ G)\ (\text{real}\ n)) / (\text{real}\ n + 1 - (\text{real}\ n)) *$
 $((\text{real}\ n + x) - \text{real}\ n) + (\ln \circ G)\ (\text{real}\ n)$
using $x\ n$ **by** (intro convex_onD_Icc' convex_on_subset[OF log_convex_G])
auto
hence $S\ (\text{of\_nat}\ n)\ (\text{of\_nat}\ n + x) \leq S\ (\text{of\_nat}\ n)\ (\text{of\_nat}\ n + 1)$
unfolding S_def **using** x **by** (simp add: field_simps)
also from n **have** $S\ (\text{of\_nat}\ n)\ (\text{of\_nat}\ n + 1) = \ln\ (\text{fact}\ n) - \ln\ (\text{fact}\ (n-1))$
by (subst (1 2) G_fact [symmetric]) (simp add: S_def add_ac of_nat_diff)
also have $\dots = \ln\ (\text{fact}\ n / \text{fact}\ (n-1))$ **using** n **by** (subst ln_div) simp_all
also from n **have** $\text{fact}\ n / \text{fact}\ (n-1) = n$ **by** (cases n) simp_all
finally have $\ln\ (G\ (\text{real}\ n + x)) \leq x * \ln\ (\text{real}\ n) + \ln\ (\text{fact}\ (n-1))$

```

    using x n by (subst (asm) S_eq) (simp_all add: field_simps)
  also have ... = ln (exp (x * ln (real n)) * fact (n - 1)) using x
    by (simp add: ln_mult)
  finally have G (real n + x) ≤ exp (x * ln (real n)) * fact (n - 1) using x
    by (subst (asm) ln_le_cancel_iff) (simp_all add: G_pos)
  also have G (real n + x) = pochhammer x n * G x
    using G_plus1[of real n + x for n] x
    by (induction n) (simp_all add: pochhammer_Suc add_ac)
  finally have G x ≤ exp (x * ln (real n)) * fact (n - 1) / pochhammer x n
    using x by (simp add: field_simps pochhammer_pos)
  also from n have fact (n - 1) = fact n / n by (cases n) simp_all
  also have exp (x * ln (real n)) * ... / pochhammer x n =
    Gamma_series x n * (1 + x / real n) using n x
    by (simp add: Gamma_series_def divide_simps pochhammer_Suc)
  finally show ?thesis .
qed

private lemma G_eq_Gamma_aux:
  assumes x: x > 0 x ≤ 1
  shows G x = Gamma x
proof (rule antisym)
  show G x ≥ Gamma x
  proof (rule tendsto_upperbound)
    from G_lower[of x] show eventually (λn. Gamma_series x n ≤ G x) sequentially
    using x by (auto intro: eventually_mono[OF eventually_ge_at_top[of 1::nat]])
  qed (simp_all add: Gamma_series_LIMSEQ)
next
  show G x ≤ Gamma x
  proof (rule tendsto_lowerbound)
    have (λn. Gamma_series x n * (1 + x / real n)) ⟶ Gamma x * (1 + 0)
    by (rule tendsto_intros real_tendsto_divide_at_top
      Gamma_series_LIMSEQ filterlim_real_sequentially)+
    thus (λn. Gamma_series x n * (1 + x / real n)) ⟶ Gamma x by simp
  next
    from G_upper[of x] show eventually (λn. Gamma_series x n * (1 + x / real
n) ≥ G x) sequentially
    using x by (auto intro: eventually_mono[OF eventually_ge_at_top[of 2::nat]])
  qed simp_all
qed

theorem Gamma_pos_real_unique:
  assumes x: x > 0
  shows G x = Gamma x
proof -
  have G_eq: G (real n + x) = Gamma (real n + x) if x ∈ {0<..1} for n x using
that
  proof (induction n)

```

```

    case (Suc n)
    from Suc have  $x + \text{real } n > 0$  by simp
    hence  $x + \text{real } n \notin \mathbb{Z}_{\leq 0}$  by auto
    with Suc show ?case using  $G\_plus1[\text{of real } n + x]$   $\Gamma\_plus1[\text{of real } n + x]$ 
    by (auto simp: add_ac)
  qed (simp_all add:  $G\_eq\_Gamma\_aux$ )

  show ?thesis
  proof (cases  $\text{frac } x = 0$ )
    case True
    hence  $x = \text{of\_int } (\text{floor } x)$  by (simp add:  $\text{frac\_def}$ )
    with x have  $x\_eq: x = \text{of\_nat } (\text{nat } (\text{floor } x) - 1) + 1$  by simp
    show ?thesis by (subst (1 2)  $x\_eq$ , rule  $G\_eq$ ) simp_all
  next
    case False
    from assms have  $x\_eq: x = \text{of\_nat } (\text{nat } (\text{floor } x)) + \text{frac } x$ 
    by (simp add:  $\text{frac\_def}$ )
    have  $\text{frac\_le\_1}: \text{frac } x \leq 1$  unfolding  $\text{frac\_def}$  by linarith
    show ?thesis
    by (subst (1 2)  $x\_eq$ , rule  $G\_eq$ , insert False  $\text{frac\_le\_1}$ ) simp_all
  qed
qed
end

```

6.23.8 The Beta function

definition *Beta* where $\text{Beta } a \ b = \Gamma a * \Gamma b / \Gamma (a + b)$

lemma *Beta_altdef*: $\text{Beta } a \ b = \Gamma a * \Gamma b * r\Gamma (a + b)$
 by (simp add: inverse_eq_divide *Beta_def* Γ_def)

lemma *Beta_commute*: $\text{Beta } a \ b = \text{Beta } b \ a$
 unfolding *Beta_def* by (simp add: ac_simps)

lemma *has_field_derivative_Beta1* [*derivative_intros*]:
 assumes $x \notin \mathbb{Z}_{\leq 0}$ $x + y \notin \mathbb{Z}_{\leq 0}$
 shows $((\lambda x. \text{Beta } x \ y) \text{ has\_field\_derivative } (\text{Beta } x \ y * (\text{Digamma } x - \text{Digamma } (x + y))))$
 (at x within A) unfolding *Beta_altdef*
 by (rule *DERIV_cong*, (rule *derivative_intros* assms)+) (simp add: algebra_simps)

lemma *Beta_pole1*: $x \in \mathbb{Z}_{\leq 0} \implies \text{Beta } x \ y = 0$
 by (auto simp add: *Beta_def* elim! : $\text{nonpos\_Ints\_cases'}$)

lemma *Beta_pole2*: $y \in \mathbb{Z}_{\leq 0} \implies \text{Beta } x \ y = 0$
 by (auto simp add: *Beta_def* elim! : $\text{nonpos\_Ints\_cases'}$)

lemma *Beta_zero*: $x + y \in \mathbb{Z}_{\leq 0} \implies \text{Beta } x \ y = 0$
by (*auto simp add: Beta_def elim!: nonpos_Ints_cases'*)

lemma *has_field_derivative_Beta2* [*derivative_intros*]:
assumes $y \notin \mathbb{Z}_{\leq 0} \ x + y \notin \mathbb{Z}_{\leq 0}$
shows $((\lambda y. \text{Beta } x \ y) \text{ has\_field\_derivative } (\text{Beta } x \ y * (\text{Digamma } y - \text{Digamma } (x + y))))$
(at y within A)
using *has_field_derivative_Beta1* [*of y x A*] **assms** **by** (*simp add: Beta_commute add_ac*)

theorem *Beta_plus1_plus1*:
assumes $x \notin \mathbb{Z}_{\leq 0} \ y \notin \mathbb{Z}_{\leq 0}$
shows $\text{Beta } (x + 1) \ y + \text{Beta } x \ (y + 1) = \text{Beta } x \ y$
proof –
have $\text{Beta } (x + 1) \ y + \text{Beta } x \ (y + 1) =$
 $(\text{Gamma } (x + 1) * \text{Gamma } y + \text{Gamma } x * \text{Gamma } (y + 1)) * r\text{Gamma}$
 $((x + y) + 1)$
by (*simp add: Beta_altdef add_divide_distrib algebra_simps*)
also have $\dots = (\text{Gamma } x * \text{Gamma } y) * ((x + y) * r\text{Gamma } ((x + y) + 1))$
by (*subst assms[THEN Gamma_plus1] + (simp add: algebra_simps)*)
also from assms have $\dots = \text{Beta } x \ y$ **unfolding** *Beta_altdef* **by** (*subst rGamma_plus1*)
simp
finally show ?thesis .
qed

theorem *Beta_plus1_left*:
assumes $x \notin \mathbb{Z}_{\leq 0}$
shows $(x + y) * \text{Beta } (x + 1) \ y = x * \text{Beta } x \ y$
proof –
have $(x + y) * \text{Beta } (x + 1) \ y = \text{Gamma } (x + 1) * \text{Gamma } y * ((x + y) * r\text{Gamma } ((x + y) + 1))$
unfolding *Beta_altdef* **by** (*simp only: ac_simps*)
also have $\dots = x * \text{Beta } x \ y$ **unfolding** *Beta_altdef*
by (*subst assms[THEN Gamma_plus1] rGamma_plus1 + (simp only: ac_simps)*)
finally show ?thesis .
qed

theorem *Beta_plus1_right*:
assumes $y \notin \mathbb{Z}_{\leq 0}$
shows $(x + y) * \text{Beta } x \ (y + 1) = y * \text{Beta } x \ y$
using *Beta_plus1_left* [*of y x*] **assms** **by** (*simp_all add: Beta_commute add.commute*)

lemma *Gamma_Gamma_Beta*:
assumes $x + y \notin \mathbb{Z}_{\leq 0}$
shows $\text{Gamma } x * \text{Gamma } y = \text{Beta } x \ y * \text{Gamma } (x + y)$
unfolding *Beta_altdef* **using** *assms Gamma_eq_zero_iff* [*of x+y*]
by (*simp add: rGamma_inverse_Gamma*)

6.23.9 Legendre duplication theorem

context

begin

private lemma *Gamma_legendre_duplication_aux*:

fixes $z :: 'a :: \text{Gamma}$

assumes $z \notin \mathbb{Z}_{\leq 0} \quad z + 1/2 \notin \mathbb{Z}_{\leq 0}$

shows $\text{Gamma } z * \text{Gamma } (z + 1/2) = \exp ((1 - 2*z) * \text{of\_real } (\ln 2)) * \text{Gamma } (1/2) * \text{Gamma } (2*z)$

proof –

let $?powr = \lambda b \ a. \exp (a * \text{of\_real } (\ln (\text{of\_nat } b)))$

let $?h = \lambda n. (\text{fact } (n-1))^2 / \text{fact } (2*n-1) * \text{of\_nat } (2^{(2*n)}) * \exp (1/2 * \text{of\_real } (\ln (\text{real\_of\_nat } n)))$

{

fix $z :: 'a$ **assume** $z: z \notin \mathbb{Z}_{\leq 0} \quad z + 1/2 \notin \mathbb{Z}_{\leq 0}$

let $?g = \lambda n. ?powr 2 (2*z) * \text{Gamma\_series}' z n * \text{Gamma\_series}' (z + 1/2)$

$n /$

$\text{Gamma\_series}' (2*z) (2*n)$

have *eventually* $(\lambda n. ?g n = ?h n)$ **sequentially using** *eventually_gt_at_top*

proof *eventually_elim*

fix $n :: \text{nat}$ **assume** $n: n > 0$

let $?f = \text{fact } (n - 1) :: 'a$ **and** $?f' = \text{fact } (2*n - 1) :: 'a$

have $A: \exp t * \exp t = \exp (2*t :: 'a)$ **for** t **by** $(\text{subst } \exp\_add [\text{symmetric}])$

simp

have $A: \text{Gamma\_series}' z n * \text{Gamma\_series}' (z + 1/2) n = ?f^2 * ?powr n (2*z + 1/2) /$

$(\text{pochhammer } z n * \text{pochhammer } (z + 1/2) n)$

by $(\text{simp add: Gamma\_series'\_def } \exp\_add \text{ ring\_distribs power2\_eq\_square } A \text{ mult\_ac})$

have $B: \text{Gamma\_series}' (2*z) (2*n) =$

$?f' * ?powr 2 (2*z) * ?powr n (2*z) /$

$(\text{of\_nat } (2^{(2*n)})) * \text{pochhammer } z n * \text{pochhammer } (z+1/2)$

$n)$ **using** n

by $(\text{simp add: Gamma\_series'\_def } \ln\_mult \exp\_add \text{ ring\_distribs pochhammer\_double})$

from z **have** $\text{pochhammer } z n \neq 0$ **by** $(\text{auto dest: pochhammer\_eq\_0\_imp\_nonpos\_Int})$

moreover from z **have** $\text{pochhammer } (z + 1/2) n \neq 0$ **by** $(\text{auto dest: pochhammer\_eq\_0\_imp\_nonpos\_Int})$

ultimately have $?powr 2 (2*z) * (\text{Gamma\_series}' z n * \text{Gamma\_series}' (z + 1/2) n) / \text{Gamma\_series}' (2*z) (2*n) =$

$?f^2 / ?f' * \text{of\_nat } (2^{(2*n)}) * (?powr n ((4*z + 1)/2) * ?powr n (-2*z))$

using n **unfolding** $A \ B$ **by** $(\text{simp add: field\_split\_simps } \exp\_minus)$

also have $?powr n ((4*z + 1)/2) * ?powr n (-2*z) = ?powr n (1/2)$

by $(\text{simp add: algebra\_sims } \exp\_add [\text{symmetric}] \text{ add\_divide\_distrib})$

finally show $?g n = ?h n$ **by** $(\text{simp only: mult\_ac})$

qed

moreover from z *double_in_nonpos_Ints_imp*[*of* z] **have** $2 * z \notin \mathbb{Z}_{\leq 0}$ **by** *auto*

```

  hence ?g ⟶ ?powr 2 (2*z) * Gamma z * Gamma (z+1/2) / Gamma
    (2*z)
  using LIMSEQ_subseq_LIMSEQ[OF Gamma_series'_LIMSEQ, of (*) 2 2*z]
  by (intro tendsto_intros Gamma_series'_LIMSEQ)
    (simp_all add: o_def strict_mono_def Gamma_eq_zero_iff)
  ultimately have ?h ⟶ ?powr 2 (2*z) * Gamma z * Gamma (z+1/2) /
    Gamma (2*z)
  by (blast intro: Lim_transform_eventually)
} note lim = this

from assms double_in_nonpos_Ints_imp[of z] have z': 2 * z ∉ ℤ≤₀ by auto
from fraction_not_in_ints[of 2 1] have (1/2 :: 'a) ∉ ℤ≤₀
  by (intro not_in_Ints_imp_not_in_nonpos_Ints) simp_all
with lim[of 1/2 :: 'a] have ?h ⟶ 2 * Gamma (1/2 :: 'a) by (simp add:
exp_of_real)
from LIMSEQ_unique[OF this lim[OF assms]] z' show ?thesis
  by (simp add: field_split_simps Gamma_eq_zero_iff ring_distrib exp_diff
exp_of_real)
qed

```

The following lemma is somewhat annoying. With a little bit of complex analysis (Cauchy's integral theorem, to be exact), this would be completely trivial. However, we want to avoid depending on the complex analysis session at this point, so we prove it the hard way.

```

private lemma Gamma_reflection_aux:
  defines h ≡ λz::complex. if z ∈ ℤ then 0 else
    (of_real pi * cot (of_real pi*z) + Digamma z - Digamma (1 - z))
  defines a ≡ complex_of_real pi
  obtains h' where continuous_on UNIV h' ∧ z. (h has_field_derivative (h' z))
    (at z)
proof -
  define f where f n = a * of_real (cos_coeff (n+1) - sin_coeff (n+2)) for n
  define F where F z = (if z = 0 then 0 else (cos (a*z) - sin (a*z)/(a*z)) / z)
  for z
  define g where g n = complex_of_real (sin_coeff (n+1)) for n
  define G where G z = (if z = 0 then 1 else sin (a*z)/(a*z)) for z
  have a_nz: a ≠ 0 unfolding a_def by simp

  have (λn. f n * (a*z)ⁿ) sums (F z) ∧ (λn. g n * (a*z)ⁿ) sums (G z)
    if abs (Re z) < 1 for z
  proof (cases z = 0; rule conjI)
    assume z ≠ 0
    note z = this that

    from z have sin_nz: sin (a*z) ≠ 0 unfolding a_def by (auto simp: sin_eq_0)
    have (λn. of_real (sin_coeff n) * (a*z)ⁿ) sums (sin (a*z)) using sin_converges[of
a*z]
      by (simp add: scaleR_conv_of_real)
    from sums_split_initial_segment[OF this, of 1]

```

```

    have (λn. (a*z) * of_real (sin_coeff (n+1)) * (a*z)^n) sums (sin (a*z)) by
(simp add: mult_ac)
    from sums_mult[OF this, of inverse (a*z)] z a_nz
    have A: (λn. g n * (a*z)^n) sums (sin (a*z)/(a*z))
    by (simp add: field_simps g_def)
    with z show (λn. g n * (a*z)^n) sums (G z) by (simp add: G_def)
    from A z a_nz sin_nz have g_nz: (∑ n. g n * (a*z)^n) ≠ 0 by (simp add:
sums_iff g_def)

    have [simp]: sin_coeff (Suc 0) = 1 by (simp add: sin_coeff_def)
    from sums_split_initial_segment[OF sums_diff[OF cos_converges[of a*z] A],
of 1]
    have (λn. z * f n * (a*z)^n) sums (cos (a*z) - sin (a*z) / (a*z))
    by (simp add: mult_ac scaleR_conv_of_real ring_distrib f_def g_def)
    from sums_mult[OF this, of inverse z] z assms
    show (λn. f n * (a*z)^n) sums (F z) by (simp add: divide_simps mult_ac
f_def F_def)
  next
    assume z: z = 0
    have (λn. f n * (a * z) ^ n) sums f 0 using powser_sums_zero[of f] z by simp
    with z show (λn. f n * (a * z) ^ n) sums (F z)
    by (simp add: f_def F_def sin_coeff_def cos_coeff_def)
    have (λn. g n * (a * z) ^ n) sums g 0 using powser_sums_zero[of g] z by
simp
    with z show (λn. g n * (a * z) ^ n) sums (G z)
    by (simp add: g_def G_def sin_coeff_def cos_coeff_def)
  qed
  note sums = conjunct1[OF this] conjunct2[OF this]

  define h2 where [abs_def]:
    h2 z = (∑ n. f n * (a*z)^n) / (∑ n. g n * (a*z)^n) + Digamma (1 + z) -
Digamma (1 - z) for z
  define POWSER where [abs_def]: POWSER f z = (∑ n. f n * (z^n :: complex))
for f z
  define POWSER' where [abs_def]: POWSER' f z = (∑ n. diffs f n * (z^n))
for f and z :: complex
  define h2' where [abs_def]:
    h2' z = a * (POWSER g (a*z) * POWSER' f (a*z) - POWSER f (a*z) *
POWSER' g (a*z)) /
    (POWSER g (a*z))^2 + Polygamma 1 (1 + z) + Polygamma 1 (1 - z) for
z

  have h_eq: h t = h2 t if abs (Re t) < 1 for t
  proof -
    from that have t: t ∈ ℤ ⟷ t = 0 by (auto elim!: Ints_cases)
    hence h t = a*cot (a*t) - 1/t + Digamma (1 + t) - Digamma (1 - t)
    unfolding h_def using Digamma_plus1[of t] by (force simp: field_simps
a_def)
    also have a*cot (a*t) - 1/t = (F t) / (G t)

```

```

    using t by (auto simp add: divide_simps sin_eq_0 cot_def a_def F_def
G_def)
    also have ... = ( $\sum n. f\ n * (a*t)^n$ ) / ( $\sum n. g\ n * (a*t)^n$ )
    using sums[of t] that by (simp add: sums_iff)
    finally show h t = h2 t by (simp only: h2_def)
qed

let ?A = {z. abs (Re z) < 1}
have open ({z. Re z < 1}  $\cap$  {z. Re z > -1})
  using open_halfspace_Re_gt open_halfspace_Re_lt by auto
also have ({z. Re z < 1}  $\cap$  {z. Re z > -1}) = {z. abs (Re z) < 1} by auto
finally have open_A: open ?A .
hence [simp]: interior ?A = ?A by (simp add: interior_open)

have summable_f: summable ( $\lambda n. f\ n * z^n$ ) for z
  by (rule powser_inside, rule sums_summable, rule sums[of i * of_real (norm
z + 1) / a])
  (simp_all add: norm_mult a_def del: of_real_add)
have summable_g: summable ( $\lambda n. g\ n * z^n$ ) for z
  by (rule powser_inside, rule sums_summable, rule sums[of i * of_real (norm
z + 1) / a])
  (simp_all add: norm_mult a_def del: of_real_add)
have summable_fg': summable ( $\lambda n. \text{diffs } f\ n * z^n$ ) summable ( $\lambda n. \text{diffs } g\ n *
z^n$ ) for z
  by (intro termdiff_converges_all summable_f summable_g)+
have (POWSER f has_field_derivative (POWSER' f z)) (at z)
  (POWSER g has_field_derivative (POWSER' g z)) (at z) for z
  unfolding POWSER_def POWSER'_def
  by (intro termdiffs_strong_converges_everywhere summable_f summable_g)+
note derivs = this[THEN DERIV_chain2[OF DERIV_cmult[OF DERIV_ident]],
unfolded POWSER_def]
have isCont (POWSER f) z isCont (POWSER g) z isCont (POWSER' f) z
isCont (POWSER' g) z
  for z unfolding POWSER_def POWSER'_def
  by (intro isCont_powser_converges_everywhere summable_f summable_g summable_fg')+
note cont = this[THEN isCont_o2[rotated], unfolded POWSER_def POWSER'_def]

{
  fix z :: complex assume z: abs (Re z) < 1
  define d where d = i * of_real (norm z + 1)
  have d: abs (Re d) < 1 norm z < norm d by (simp_all add: d_def norm_mult
del: of_real_add)
  have eventually ( $\lambda z. h\ z = h2\ z$ ) (nhds z)
    using eventually_nhds_in_nhd[of z ?A] using h_eq z
    by (auto elim!: eventually_mono)

  moreover from sums(2)[OF z] z have nz: ( $\sum n. g\ n * (a * z)^n$ )  $\neq 0$ 
    unfolding G_def by (auto simp: sums_iff sin_eq_0 a_def)
  have A:  $z \in \mathbb{Z} \iff z = 0$  using z by (auto elim!: Ints_cases)

```

```

have no_int:  $1 + z \in \mathbb{Z} \longleftrightarrow z = 0$  using z Ints_diff[of 1+z 1] A
  by (auto elim!: nonpos_Ints_cases)
have no_int':  $1 - z \in \mathbb{Z} \longleftrightarrow z = 0$  using z Ints_diff[of 1 1-z] A
  by (auto elim!: nonpos_Ints_cases)
from no_int no_int' have no_int:  $1 - z \notin \mathbb{Z}_{\leq 0} \wedge 1 + z \notin \mathbb{Z}_{\leq 0}$  by auto
have (h2 has_field_derivative h2' z) (at z) unfolding h2_def
  by (rule DERIV_cong, (rule derivative_intros refl derivs[unfolded POWSER_def]
nz no_int)+)
  (auto simp: h2'_def POWSER_def field_simps power2_eq_square)
ultimately have deriv: (h has_field_derivative h2' z) (at z)
  by (subst DERIV_cong_ev[OF refl _ refl])

from sums(2)[OF z] z have  $(\sum n. g\ n * (a * z) ^ n) \neq 0$ 
  unfolding G_def by (auto simp: sums_iff a_def sin_eq_0)
hence isCont h2' z using no_int unfolding h2'_def[abs_def] POWSER_def
POWSER'_def
  by (intro continuous_intros cont
continuous_on_compose2[OF _ continuous_on_Polygamma[of {z. Re
z > 0}]] auto
note deriv and this
} note A = this

interpret h: periodic_fun_simple' h
proof
fix z :: complex
show h (z + 1) = h z
proof (cases z ∈ ℤ)
assume z: z ∉ ℤ
hence A:  $z + 1 \notin \mathbb{Z} \wedge z \neq 0$  using Ints_diff[of z+1 1] by auto
hence Digamma (z + 1) - Digamma (-z) = Digamma z - Digamma (-z
+ 1)
  by (subst (1 2) Digamma_plus1) simp_all
with A z show h (z + 1) = h z
  by (simp add: h_def sin_plus_pi cos_plus_pi ring_distrib cot_def)
qed (simp add: h_def)
qed

have h2'_eq:  $h2'(z - 1) = h2' z$  if z:  $\text{Re } z > 0 \wedge \text{Re } z < 1$  for z
proof -
have ((λz. h (z - 1)) has_field_derivative h2' (z - 1)) (at z)
  by (rule DERIV_cong, rule DERIV_chain'[OF _ A(1)])
  (insert z, auto intro!: derivative_eq_intros)
hence (h has_field_derivative h2' (z - 1)) (at z) by (subst (asm) h.minus_1)
moreover from z have (h has_field_derivative h2' z) (at z) by (intro A)
simp_all
ultimately show  $h2'(z - 1) = h2' z$  by (rule DERIV_unique)
qed

define h2'' where  $h2''\ z = h2'(z - \text{of\_int } \lfloor \text{Re } z \rfloor)$  for z

```

```

have deriv: (h has_field_derivative h2'' z) (at z) for z
proof -
  fix z :: complex
  have B: |Re z - real_of_int ⌊Re z⌋| < 1 by linarith
  have ((λt. h (t - of_int ⌊Re z⌋)) has_field_derivative h2'' z) (at z)
    unfolding h2''_def by (rule DERIV_cong, rule DERIV_chain'[OF _ A(1)])
    (insert B, auto intro!: derivative_intros)
  thus (h has_field_derivative h2'' z) (at z) by (simp add: h.minus_of_int)
qed

have cont: continuous_on UNIV h2''
proof (intro continuous_at_imp_continuous_on ballI)
  fix z :: complex
  define r where r = ⌊Re z⌋
  define A where A = {t. of_int r - 1 < Re t ∧ Re t < of_int r + 1}
  have continuous_on A (λt. h2' (t - of_int r)) unfolding A_def
    by (intro continuous_at_imp_continuous_on isCont_o2[OF _ A(2)] ballI
    continuous_intros)
    (simp_all add: abs_real_def)
  moreover have h2'' t = h2' (t - of_int r) if t: t ∈ A for t
  proof (cases Re t ≥ of_int r)
    case True
    from t have of_int r - 1 < Re t Re t < of_int r + 1 by (simp_all add:
    A_def)
    with True have ⌊Re t⌋ = ⌊Re z⌋ unfolding r_def by linarith
    thus ?thesis by (auto simp: r_def h2''_def)
  next
    case False
    from t have t: of_int r - 1 < Re t Re t < of_int r + 1 by (simp_all add:
    A_def)
    with False have t': ⌊Re t⌋ = ⌊Re z⌋ - 1 unfolding r_def by linarith
    moreover from t False have h2' (t - of_int r + 1 - 1) = h2' (t - of_int
    r + 1)
    by (intro h2'_eq) simp_all
    ultimately show ?thesis by (auto simp: r_def h2''_def algebra_simps t')
  qed
  ultimately have continuous_on A h2'' by (subst continuous_on_cong[OF
  refl])
  moreover {
    have open ({t. of_int r - 1 < Re t} ∩ {t. of_int r + 1 > Re t})
    by (intro open_Int open_halfspace_Re_gt open_halfspace_Re_lt)
    also have {t. of_int r - 1 < Re t} ∩ {t. of_int r + 1 > Re t} = A
    unfolding A_def by blast
    finally have open A .
  }
  ultimately have C: isCont h2'' t if t ∈ A for t using that
    by (subst (asm) continuous_on_eq_continuous_at) auto
  have of_int r - 1 < Re z Re z < of_int r + 1 unfolding r_def by linarith+
  thus isCont h2'' z by (intro C) (simp_all add: A_def)

```

qed

from that[*OF cont deriv*] show ?thesis .
qed

lemma *Gamma_reflection_complex*:

fixes $z :: \text{complex}$

shows $\text{Gamma } z * \text{Gamma } (1 - z) = \text{of\_real } \pi / \sin (\text{of\_real } \pi * z)$

proof -

let $?g = \lambda z :: \text{complex}. \text{Gamma } z * \text{Gamma } (1 - z) * \sin (\text{of\_real } \pi * z)$

define g where [abs_def]: $g \ z = (\text{if } z \in \mathbb{Z} \text{ then } \text{of\_real } \pi \text{ else } ?g \ z)$ for $z :: \text{complex}$

let $?h = \lambda z :: \text{complex}. (\text{of\_real } \pi * \cot (\text{of\_real } \pi * z) + \text{Digamma } z - \text{Digamma } (1 - z))$

define h where [abs_def]: $h \ z = (\text{if } z \in \mathbb{Z} \text{ then } 0 \text{ else } ?h \ z)$ for $z :: \text{complex}$

— g is periodic with period 1.

interpret g : *periodic_fun_simple'* g

proof

fix $z :: \text{complex}$

show $g \ (z + 1) = g \ z$

proof (cases $z \in \mathbb{Z}$)

case False

hence $z * g \ z = z * \text{Beta } z \ (-z + 1) * \sin (\text{of\_real } \pi * z)$ by (simp add: $g\_def \ \text{Beta\_def}$)

also have $z * \text{Beta } z \ (-z + 1) = (z + 1 + -z) * \text{Beta } (z + 1) \ (-z + 1)$

using False *Ints_diff*[of 1 1 - z] *nonpos_Ints_subset_Ints*

by (subst *Beta_plus1_left* [symmetric]) auto

also have $\dots * \sin (\text{of\_real } \pi * z) = z * (\text{Beta } (z + 1) \ (-z) * \sin (\text{of\_real } \pi * (z + 1)))$

using False *Ints_diff*[of $z+1$ 1] *Ints_minus*[of -z] *nonpos_Ints_subset_Ints*

by (subst *Beta_plus1_right*) (auto simp: *ring_distrib sin_plus_pi*)

also from False have $\text{Beta } (z + 1) \ (-z) * \sin (\text{of\_real } \pi * (z + 1)) = g \ (z + 1)$

using *Ints_diff*[of $z+1$ 1] by (auto simp: $g\_def \ \text{Beta\_def}$)

finally show $g \ (z + 1) = g \ z$ using False by (subst (asm) *mult_left_cancel*) auto

qed (simp add: g_def)

qed

— g is entire.

have g_g' : $(g \text{ has\_field\_derivative } (h \ z * g \ z)) \ (at \ z)$ for $z :: \text{complex}$

proof (cases $z \in \mathbb{Z}$)

let $?h' = \lambda z. \text{Beta } z \ (1 - z) * ((\text{Digamma } z - \text{Digamma } (1 - z)) * \sin (z * \text{of\_real } \pi) +$

$\text{of\_real } \pi * \cos (z * \text{of\_real } \pi))$

case False

from False have eventually $(\lambda t. t \in \text{UNIV} - \mathbb{Z}) \ (nhds \ z)$

by (intro *eventually_nhds_in_open*) (auto simp: *open_Diff*)


```

    hence eventually ( $\lambda t. g\ t = ?g\ t$ ) (nhds  $z$ ) by eventually_elim (simp add:
g_def)
  moreover {
    from False Ints_diff[of 1 1-z] have  $1 - z \notin \mathbb{Z}$  by auto
    hence ( $?g$  has_field_derivative  $?h'\ z$ ) (at  $z$ ) using nonpos_Ints_subset_Ints
      by (auto intro!: derivative_eq_intros simp: algebra_simps Beta_def)
    also from False have  $\sin(\text{of\_real } \pi * z) \neq 0$  by (subst sin_eq_0) auto
    hence  $?h'\ z = h\ z * g\ z$ 
      using False unfolding g_def h_def cot_def by (simp add: field_simps
Beta_def)
    finally have ( $?g$  has_field_derivative ( $h\ z * g\ z$ )) (at  $z$ ) .
  }
  ultimately show ?thesis by (subst DERIV_cong_ev[OF refl _ refl])
next
case True
then obtain  $n$  where  $z: z = \text{of\_int } n$  by (auto elim!: Ints_cases)
let  $?t = (\lambda z::\text{complex}. \text{if } z = 0 \text{ then } 1 \text{ else } \sin z / z) \circ (\lambda z. \text{of\_real } \pi * z)$ 
have deriv_0: ( $g$  has_field_derivative 0) (at 0)
proof (subst DERIV_cong_ev[OF refl _ refl])
  show eventually ( $\lambda z. g\ z = \text{of\_real } \pi * \Gamma(1 + z) * \Gamma(1 - z)$ 
* ?t  $z$ ) (nhds 0)
    using eventually_nhds_ball[OF zero_less_one, of 0::complex]
  proof eventually_elim
    fix  $z::\text{complex}$  assume  $z: z \in \text{ball } 0\ 1$ 
    show  $g\ z = \text{of\_real } \pi * \Gamma(1 + z) * \Gamma(1 - z) * ?t\ z$ 
    proof (cases  $z = 0$ )
      assume  $z': z \neq 0$ 
      with  $z$  have  $z'': z \notin \mathbb{Z}_{\leq 0} \wedge z \notin \mathbb{Z}$  by (auto elim!: Ints_cases)
      from Gamma_plus1[OF this(1)] have  $\Gamma\ z = \Gamma(z + 1) / z$ 
    by simp
      with  $z''\ z'$  show ?thesis by (simp add: g_def ac_simps)
    qed (simp add: g_def)
  qed
  have ( $?t$  has_field_derivative ( $0 * \text{of\_real } \pi$ )) (at 0)
    using has_field_derivative_sin_z_over_z[of UNIV :: complex set]
    by (intro DERIV_chain) simp_all
  thus (( $\lambda z. \text{of\_real } \pi * \Gamma(1 + z) * \Gamma(1 - z) * ?t\ z$ ) has_field_derivative
0) (at 0)
    by (auto intro!: derivative_eq_intros simp: o_def)
  qed

  have (( $g \circ (\lambda x. x - \text{of\_int } n)$ ) has_field_derivative  $0 * 1$ ) (at ( $\text{of\_int } n$ ))
    using deriv_0 by (intro DERIV_chain) (auto intro!: derivative_eq_intros)
  also have  $g \circ (\lambda x. x - \text{of\_int } n) = g$  by (intro ext) (simp add: g.minus_of_int)
  finally show ( $g$  has_field_derivative ( $h\ z * g\ z$ )) (at  $z$ ) by (simp add: z h_def)
qed

  have g_eq:  $g(z/2) * g((z+1)/2) = \Gamma(1/2)^2 * g\ z$  if  $\text{Re } z > -1$   $\text{Re } z < 2$  for  $z$ 

```

```

proof (cases z ∈ ℤ)
  case True
    with that have z = 0 ∨ z = 1 by (force elim!: Ints_cases)
    moreover have g 0 * g (1/2) = Gamma (1/2)^2 * g 0
      using fraction_not_in_ints[where 'a = complex, of 2 1] by (simp add: g_def
power2_eq_square)
    moreover have g (1/2) * g 1 = Gamma (1/2)^2 * g 1
      using fraction_not_in_ints[where 'a = complex, of 2 1]
      by (simp add: g_def power2_eq_square Beta_def algebra_simps)
    ultimately show ?thesis by force
  next
    case False
      hence z: z/2 ∉ ℤ (z+1)/2 ∉ ℤ using Ints_diff[of z+1 1] by (auto elim!:
Ints_cases)
      hence z': z/2 ∉ ℤ≤0 (z+1)/2 ∉ ℤ≤0 by (auto elim!: nonpos_Ints_cases)
      from z have 1-z/2 ∉ ℤ 1-((z+1)/2) ∉ ℤ
        using Ints_diff[of 1 1-z/2] Ints_diff[of 1 1-((z+1)/2)] by auto
      hence z'': 1-z/2 ∉ ℤ≤0 1-((z+1)/2) ∉ ℤ≤0 by (auto elim!: nonpos_Ints_cases)
      from z have g (z/2) * g ((z+1)/2) =
        (Gamma (z/2) * Gamma ((z+1)/2)) * (Gamma (1-z/2) * Gamma (1-((z+1)/2)))
      *
        (sin (of_real pi * z/2) * sin (of_real pi * (z+1)/2))
      by (simp add: g_def)
      also from z' Gamma_legendre_duplication_aux[of z/2]
      have Gamma (z/2) * Gamma ((z+1)/2) = exp ((1-z) * of_real (ln 2)) *
Gamma (1/2) * Gamma z
      by (simp add: add_divide_distrib)
      also from z'' Gamma_legendre_duplication_aux[of 1-(z+1)/2]
      have Gamma (1-z/2) * Gamma (1-(z+1)/2) =
        Gamma (1-z) * Gamma (1/2) * exp (z * of_real (ln 2))
      by (simp add: add_divide_distrib ac_simps)
      finally have g (z/2) * g ((z+1)/2) = Gamma (1/2)^2 * (Gamma z * Gamma
(1-z) *
        (2 * (sin (of_real pi*z/2) * sin (of_real pi*(z+1)/2))))
      by (simp add: add_ac power2_eq_square exp_add ring_distrib exp_diff
exp_of_real)
      also have sin (of_real pi*(z+1)/2) = cos (of_real pi*z/2)
      using cos_sin_eq[of - of_real pi * z/2, symmetric]
      by (simp add: ring_distrib add_divide_distrib ac_simps)
      also have 2 * (sin (of_real pi*z/2) * cos (of_real pi*z/2)) = sin (of_real pi
* z)
      by (subst sin_times_cos) (simp add: field_simps)
      also have Gamma z * Gamma (1 - z) * sin (complex_of_real pi * z) = g z
      using ⟨z ∉ ℤ⟩ by (simp add: g_def)
      finally show ?thesis .
qed
have g_eq: g (z/2) * g ((z+1)/2) = Gamma (1/2)^2 * g z for z
proof -
  define r where r = ⌊Re z / 2⌋

```

have $\Gamma(1/2)^2 * g\ z = \Gamma(1/2)^2 * g\ (z - \text{of\_int}\ (2*r))$ by
 (simp only: $g.\text{minus\_of\_int}$)
 also have $\text{of\_int}\ (2*r) = 2 * \text{of\_int}\ r$ by simp
 also have $\text{Re}\ z - 2 * \text{of\_int}\ r > -1$ $\text{Re}\ z - 2 * \text{of\_int}\ r < 2$ unfolding
 r_def by linarith+
 hence $\Gamma(1/2)^2 * g\ (z - 2 * \text{of\_int}\ r) =$
 $g\ ((z - 2 * \text{of\_int}\ r)/2) * g\ ((z - 2 * \text{of\_int}\ r + 1)/2)$
 unfolding r_def by (intro $g\_eq[\text{symmetric}]$) simp_all
 also have $(z - 2 * \text{of\_int}\ r) / 2 = z/2 - \text{of\_int}\ r$ by simp
 also have $g\ \dots = g\ (z/2)$ by (rule $g.\text{minus\_of\_int}$)
 also have $(z - 2 * \text{of\_int}\ r + 1) / 2 = (z + 1)/2 - \text{of\_int}\ r$ by simp
 also have $g\ \dots = g\ ((z+1)/2)$ by (rule $g.\text{minus\_of\_int}$)
 finally show ?thesis ..
 qed

have g_nz [simp]: $g\ z \neq 0$ for $z :: \text{complex}$
 unfolding g_def using $\text{Ints\_diff}[\text{of}\ 1\ 1 - z]$
 by (auto simp: $\Gamma\_eq\_zero\_iff\ \sin\_eq\_0$ dest!: nonpos_Ints_Int)

 have h_eq : $h\ z = (h\ (z/2) + h\ ((z+1)/2)) / 2$ for z
 proof -
 have $((\lambda t. g\ (t/2) * g\ ((t+1)/2)) \text{ has\_field\_derivative}$
 $(g\ (z/2) * g\ ((z+1)/2)) * ((h\ (z/2) + h\ ((z+1)/2)) / 2))$ (at z)
 by (auto intro!: $\text{derivative\_eq\_intros}\ g\_g'[\text{THEN}\ \text{DERIV\_chain2}]$ simp:
 field_simps)
 hence $((\lambda t. \Gamma(1/2)^2 * g\ t) \text{ has\_field\_derivative}$
 $\Gamma(1/2)^2 * g\ z * ((h\ (z/2) + h\ ((z+1)/2)) / 2))$ (at z)
 by (subst (1 2) $g\_eq[\text{symmetric}]$) simp
 from $\text{DERIV\_cmult}[\text{OF}\ \text{this}, \text{of}\ \text{inverse}\ ((\Gamma(1/2))^2)]$
 have $(g \text{ has\_field\_derivative}\ (g\ z * ((h\ (z/2) + h\ ((z+1)/2))/2)))$ (at z)
 using $\text{fraction\_not\_in\_ints}[\text{where}\ 'a = \text{complex}, \text{of}\ 2\ 1]$
 by (simp add: $\text{divide\_simps}\ \Gamma\_eq\_zero\_iff\ \text{not\_in}\ \text{Ints\_imp\_not\_in}\ \text{nonpos\_Ints}$)
 moreover have $(g \text{ has\_field\_derivative}\ (g\ z * h\ z))$ (at z)
 using $g\_g'[\text{of}\ z]$ by (simp add: ac_simps)
 ultimately have $g\ z * h\ z = g\ z * ((h\ (z/2) + h\ ((z+1)/2))/2)$
 by (intro DERIV_unique)
 thus $h\ z = (h\ (z/2) + h\ ((z+1)/2)) / 2$ by simp
 qed

obtain h' where h'_cont : $\text{continuous\_on}\ \text{UNIV}\ h'$ and
 h_h' : $\bigwedge z. (h \text{ has\_field\_derivative}\ h'\ z)$ (at z)
 unfolding h_def by (erule $\Gamma\_reflection\_aux$)

have h'_eq : $h'\ z = (h'\ (z/2) + h'\ ((z+1)/2)) / 4$ for z
 proof -
 have $((\lambda t. (h\ (t/2) + h\ ((t+1)/2)) / 2) \text{ has\_field\_derivative}$
 $((h'\ (z/2) + h'\ ((z+1)/2)) / 4))$ (at z)
 by (fastforce intro!: $\text{derivative\_eq\_intros}\ h\_h'[\text{THEN}\ \text{DERIV\_chain2}]$)
 hence $(h \text{ has\_field\_derivative}\ ((h'\ (z/2) + h'\ ((z+1)/2))/4))$ (at z)

```

    by (subst (asm) h_eq[symmetric])
    from h_h' and this show h' z = (h' (z/2) + h' ((z+1)/2)) / 4 by (rule
DERIV_unique)
  qed

  have h'_zero: h' z = 0 for z
  proof -
    define m where m = max 1 |Re z|
    define B where B = {t. abs (Re t) ≤ m ∧ abs (Im t) ≤ abs (Im z)}
    have closed ({t. Re t ≥ -m} ∩ {t. Re t ≤ m} ∩
      {t. Im t ≥ -|Im z|} ∩ {t. Im t ≤ |Im z|})
    (is closed ?B) by (intro closed_Int closed_halfspace_Re_ge closed_halfspace_Re_le
      closed_halfspace_Im_ge closed_halfspace_Im_le)
    also have ?B = B unfolding B_def by fastforce
    finally have closed B .
    moreover have bounded B unfolding bounded_iff
    proof (intro ballI exI)
      fix t assume t: t ∈ B
      have norm t ≤ |Re t| + |Im t| by (rule cmod_le)
      also from t have |Re t| ≤ m unfolding B_def by blast
      also from t have |Im t| ≤ |Im z| unfolding B_def by blast
      finally show norm t ≤ m + |Im z| by - simp
    qed
    ultimately have compact: compact B by (subst compact_eq_bounded_closed)
  blast

```

```

  define M where M = (SUP z∈B. norm (h' z))
  have compact (h' ` B)
    by (intro compact_continuous_image continuous_on_subset[OF h'_cont]
compact) blast+
  hence bdd: bdd_above ((λz. norm (h' z)) ` B)
    using bdd_above_norm[of h' ` B] by (simp add: image_comp o_def com-
pact_imp_bounded)
  have norm (h' z) ≤ M unfolding M_def by (intro cSUP_upper bdd) (simp_all
add: B_def m_def)
  also have M ≤ M/2
  proof (subst M_def, subst cSUP_le_iff)
    have z ∈ B unfolding B_def m_def by simp
    thus B ≠ {} by auto
  next
    show ∀ z∈B. norm (h' z) ≤ M/2
    proof
      fix t :: complex assume t: t ∈ B
      from h'_eq[of t] t have h' t = (h' (t/2) + h' ((t+1)/2)) / 4 by (simp)
      also have norm ... = norm (h' (t/2) + h' ((t+1)/2)) / 4 by simp
      also have norm (h' (t/2) + h' ((t+1)/2)) ≤ norm (h' (t/2)) + norm (h'
((t+1)/2))
      by (rule norm_triangle_ineq)
      also from t have abs (Re ((t + 1)/2)) ≤ m unfolding m_def B_def by

```

auto

with t have $t/2 \in B$ $(t+1)/2 \in B$ unfolding B_def by auto
 hence $\text{norm } (h' (t/2)) + \text{norm } (h' ((t+1)/2)) \leq M + M$ unfolding M_def
 by (intro add_mono cSUP_upper bdd) (auto simp: B_def)
 also have $(M + M) / 4 = M / 2$ by simp
 finally show $\text{norm } (h' t) \leq M/2$ by - simp_all
 qed
 qed (insert bdd, auto)
 hence $M \leq 0$ by simp
 finally show $h' z = 0$ by simp

qed

have $h\_h'\_2$: $(h \text{ has\_field\_derivative } 0) \text{ (at } z) \text{ for } z$
 using $h\_h'[of\ z]$ $h'\_zero[of\ z]$ by simp

have g_real : $g\ z \in \mathbb{R}$ if $z \in \mathbb{R}$ for z
 unfolding g_def using that by (auto intro!: $Reals\_mult$ $Gamma\_complex\_real$)
 have h_real : $h\ z \in \mathbb{R}$ if $z \in \mathbb{R}$ for z
 unfolding h_def using that by (auto intro!: $Reals\_mult$ $Reals\_add$ $Reals\_diff$ $Polygamma\_Real$)
 have g_nz : $g\ z \neq 0$ for z unfolding g_def using $Ints\_diff[of\ 1\ 1-z]$
 by (auto simp: $Gamma\_eq\_zero\_iff\ sin\_eq\_0$)

from h'_zero $h\_h'\_2$ have $\exists c. \forall z \in UNIV. h\ z = c$
 by (intro has_field_derivative_zero_constant) (simp_all add: dist_0_norm)
 then obtain c where $c: \bigwedge z. h\ z = c$ by auto
 have $\exists u. u \in \text{closed\_segment } 0\ 1 \wedge \text{Re } (g\ 1) - \text{Re } (g\ 0) = \text{Re } (h\ u * g\ u * (1 - 0))$
 by (intro complex_mvt_line g_g')
 then obtain u where $u: u \in \text{closed\_segment } 0\ 1 \wedge \text{Re } (g\ 1) - \text{Re } (g\ 0) = \text{Re } (h\ u * g\ u)$
 by auto
 from $u(1)$ have u' : $u \in \mathbb{R}$ unfolding $\text{closed\_segment\_def}$
 by (auto simp: $\text{scaleR\_conv\_of\_real}$)
 from u' $g\_real[of\ u]$ $g\_nz[of\ u]$ have $\text{Re } (g\ u) \neq 0$ by (auto elim!: $Reals\_cases$)
 with $u(2)$ $c[of\ u]$ $g\_real[of\ u]$ $g\_nz[of\ u]$ u'
 have $\text{Re } c = 0$ by (simp add: $\text{complex\_is\_Real\_iff } g.of\_1$)
 with $h\_real[of\ 0]$ $c[of\ 0]$ have $c = 0$ by (auto elim!: $Reals\_cases$)
 with c have $A: h\ z * g\ z = 0$ for z by simp
 hence $(g \text{ has\_field\_derivative } 0) \text{ (at } z) \text{ for } z$ using $g\_g'[of\ z]$ by simp
 hence $\exists c'. \forall z \in UNIV. g\ z = c'$ by (intro has_field_derivative_zero_constant) simp_all
 then obtain c' where $c: \bigwedge z. g\ z = c'$ by (force)
 from $this[of\ 0]$ have $c' = \pi$ unfolding g_def by simp
 with c have $g\ z = \pi$ by simp

show ?thesis

proof (cases $z \in \mathbb{Z}$)

case False

with $\langle g\ z = \pi \rangle$ show ?thesis by (auto simp: g_def divide_simps)

```

next
  case True
  then obtain n where n: z = of_int n by (elim Ints_cases)
  with sin_eq_0[of of_real pi * z] have sin (of_real pi * z) = 0 by force
  moreover have of_int (1 - n) ∈ ℤ≤0 if n > 0 using that by (intro non-
pos_Ints_of_int) simp
  ultimately show ?thesis using n
  by (cases n ≤ 0) (auto simp: Gamma_eq_zero_iff nonpos_Ints_of_int)
qed
qed

```

```

lemma rGamma_reflection_complex:
  rGamma z * rGamma (1 - z :: complex) = sin (of_real pi * z) / of_real pi
  using Gamma_reflection_complex[of z]
  by (simp add: Gamma_def field_split_simps split: if_split_asm)

```

```

lemma rGamma_reflection_complex':
  rGamma z * rGamma (- z :: complex) = -z * sin (of_real pi * z) / of_real pi
proof -
  have rGamma z * rGamma (-z) = -z * (rGamma z * rGamma (1 - z))
  using rGamma_plus1[of -z, symmetric] by simp
  also have rGamma z * rGamma (1 - z) = sin (of_real pi * z) / of_real pi
  by (rule rGamma_reflection_complex)
  finally show ?thesis by simp
qed

```

```

lemma Gamma_reflection_complex':
  Gamma z * Gamma (- z :: complex) = - of_real pi / (z * sin (of_real pi * z))
  using rGamma_reflection_complex'[of z] by (force simp add: Gamma_def field_split_simps)

```

```

lemma Gamma_one_half_real: Gamma (1/2 :: real) = sqrt pi
proof -
  from Gamma_reflection_complex[of 1/2] fraction_not_in_ints[where 'a =
complex, of 2 1]
  have Gamma (1/2 :: complex)2 = of_real pi by (simp add: power2_eq_square)
  hence of_real pi = Gamma (complex_of_real (1/2))2 by simp
  also have ... = of_real ((Gamma (1/2))2) by (subst Gamma_complex_of_real)
  simp_all
  finally have Gamma (1/2)2 = pi by (subst (asm) of_real_eq_iff) simp_all
  moreover have Gamma (1/2 :: real) ≥ 0 using Gamma_real_pos[of 1/2] by
  simp
  ultimately show ?thesis by (rule real_sqrt_unique [symmetric])
qed

```

```

lemma Gamma_one_half_complex: Gamma (1/2 :: complex) = of_real (sqrt pi)
proof -
  have Gamma (1/2 :: complex) = Gamma (of_real (1/2)) by simp

```

also have $\dots = \text{of\_real } (\text{sqrt } \pi)$ **by** (*simp only: Gamma_complex_of_real Gamma_one_half_real*)

finally show ?thesis .

qed

theorem Gamma_legendre_duplication:

fixes $z :: \text{complex}$

assumes $z \notin \mathbb{Z}_{\leq 0}$ $z + 1/2 \notin \mathbb{Z}_{\leq 0}$

shows $\text{Gamma } z * \text{Gamma } (z + 1/2) =$

$\exp ((1 - 2z) * \text{of\_real } (\ln 2)) * \text{of\_real } (\text{sqrt } \pi) * \text{Gamma } (2z)$

using Gamma_legendre_duplication_aux[OF assms] **by** (*simp add: Gamma_one_half_complex*)

end

6.23.10 Limits and residues

The inverse of the Gamma function has simple zeros:

lemma rGamma_zeros:

$(\lambda z. \text{rGamma } z / (z + \text{of\_nat } n)) - (- \text{of\_nat } n) \rightarrow ((-1)^n * \text{fact } n :: 'a :: \text{Gamma})$

proof (*subst tendsto_cong*)

let ?f = $\lambda z. \text{pochhammer } z \ n * \text{rGamma } (z + \text{of\_nat } (\text{Suc } n)) :: 'a$

from eventually_at_ball'[OF zero_less_one, of - of_nat n :: 'a UNIV]

show eventually $(\lambda z. \text{rGamma } z / (z + \text{of\_nat } n) = ?f \ z)$ (at (- of_nat n))

by (*subst pochhammer_rGamma[of _ Suc n]*)

(auto elim!: eventually_mono simp: field_split_simps pochhammer_rec' eq_neg_iff_add_eq_0)

have isCont ?f (- of_nat n) **by** (*intro continuous_intros*)

thus ?f - (- of_nat n) $\rightarrow (-1)^n * \text{fact } n$ **unfolding** isCont_def

by (*simp add: pochhammer_same*)

qed

The simple zeros of the inverse of the Gamma function correspond to simple poles of the Gamma function, and their residues can easily be computed from the limit we have just proven:

lemma Gamma_poles: *filterlim* Gamma at_infinity (at (- of_nat n :: 'a :: Gamma))

proof -

from eventually_at_ball'[OF zero_less_one, of - of_nat n :: 'a UNIV]

have eventually $(\lambda z. \text{rGamma } z \neq (0 :: 'a))$ (at (- of_nat n))

by (*auto elim!: eventually_mono nonpos_Ints_cases'*)

simp: rGamma_eq_zero_iff dist_of_nat dist_minus

with isCont_rGamma[of - of_nat n :: 'a, OF continuous_ident]

have *filterlim* $(\lambda z. \text{inverse } (\text{rGamma } z) :: 'a)$ at_infinity (at (- of_nat n))

unfolding isCont_def **by** (*intro filterlim_compose[OF filterlim_inverse_at_infinity]*)

(simp_all add: filterlim_at)

moreover have $(\lambda z. \text{inverse } (\text{rGamma } z) :: 'a) = \text{Gamma}$

by (*intro ext*) (*simp add: rGamma_inverse_Gamma*)

ultimately show ?thesis **by** (*simp only:*)

qed

lemma *Gamma_residues*:

$(\lambda z. \text{Gamma } z * (z + \text{of\_nat } n)) - (- \text{of\_nat } n) \rightarrow ((-1)^n / \text{fact } n :: 'a :: \text{Gamma})$
proof (*subst tendsto_cong*)
 let ?c = $(-1)^n / \text{fact } n :: 'a$
 from *eventually_at_ball'[OF zero_less_one, of - of_nat n :: 'a UNIV]*
 show *eventually* $(\lambda z. \text{Gamma } z * (z + \text{of\_nat } n) = \text{inverse } (r\text{Gamma } z / (z + \text{of\_nat } n)))$
 (at (- of_nat n))
 by (*auto elim!: eventually_mono simp: field_split_simps rGamma_inverse_Gamma*)
 have $(\lambda z. \text{inverse } (r\text{Gamma } z / (z + \text{of\_nat } n))) - (- \text{of\_nat } n) \rightarrow$
 $\text{inverse } ((-1)^n * \text{fact } n :: 'a)$
 by (*intro tendsto_intros rGamma_zeros simp_all*)
 also have $\text{inverse } ((-1)^n * \text{fact } n) = ?c$
 by (*simp_all add: field_simps flip: power_mult_distrib*)
 finally show $(\lambda z. \text{inverse } (r\text{Gamma } z / (z + \text{of\_nat } n))) - (- \text{of\_nat } n) \rightarrow ?c$
 .
 qed

6.23.11 Alternative definitions

Variant of the Euler form

definition *Gamma_series_euler'* where

Gamma_series_euler' z n =
 $\text{inverse } z * (\prod_{k=1..n. \exp (z * \text{of\_real } (\ln (1 + \text{inverse } (\text{of\_nat } k)))) / (1 + z / \text{of\_nat } k))$

context

begin

private lemma *Gamma_euler'_aux1*:

fixes $z :: 'a :: \{\text{real\_normed\_field}, \text{banach}\}$
assumes $n: n > 0$
shows $\exp (z * \text{of\_real } (\ln (\text{of\_nat } n + 1))) = (\prod_{k=1..n. \exp (z * \text{of\_real } (\ln (1 + 1 / \text{of\_nat } k))))$
proof -
 have $(\prod_{k=1..n. \exp (z * \text{of\_real } (\ln (1 + 1 / \text{of\_nat } k)))) =$
 $\exp (z * \text{of\_real } (\sum_{k=1..n. \ln (1 + 1 / \text{real\_of\_nat } k)))$
 by (*subst exp_sum [symmetric] (simp_all add: sum_distrib_left)*)
 also have $(\sum_{k=1..n. \ln (1 + 1 / \text{of\_nat } k) :: \text{real}) = \ln (\prod_{k=1..n. 1 + 1 / \text{real\_of\_nat } k})$
 by (*subst ln_prod [symmetric] (auto intro!: add_pos_nonneg)*)
 also have $(\prod_{k=1..n. 1 + 1 / \text{of\_nat } k :: \text{real}) = (\prod_{k=1..n. (\text{of\_nat } k + 1) / \text{of\_nat } k})$
 by (*intro prod.cong (simp_all add: field_split_simps)*)
 also have $(\prod_{k=1..n. (\text{of\_nat } k + 1) / \text{of\_nat } k :: \text{real}) = \text{of\_nat } n + 1$
 by (*induction n (simp_all add: prod_nat_ivl_Suc' field_split_simps)*)
 finally show ?thesis ..

qed

theorem *Gamma_series_euler'*:

assumes $z: (z :: 'a :: \text{Gamma}) \notin \mathbb{Z}_{\leq 0}$

shows $(\lambda n. \text{Gamma\_series\_euler}' z n) \longrightarrow \text{Gamma } z$

proof (rule *Gamma_seriesI*, rule *Lim_transform_eventually*)

let $?f = \lambda n. \text{fact } n * \exp (z * \text{of\_real } (\ln (\text{of\_nat } n + 1))) / \text{pochhammer } z (n + 1)$

let $?r = \lambda n. ?f n / \text{Gamma\_series } z n$

let $?r' = \lambda n. \exp (z * \text{of\_real } (\ln (\text{of\_nat } (\text{Suc } n) / \text{of\_nat } n)))$

from z **have** $z' : z \neq 0$ **by** *auto*

have *eventually* $(\lambda n. ?r' n = ?r n)$ *sequentially*

using z **by** (auto simp: *field_split_simps* *Gamma_series_def* *ring_distrib* *exp_diff* *ln_div*

intro: eventually_mono eventually_gt_at_top[of $0::\text{nat}$] *dest:* *pochhammer_eq_0_imp_nonpos_Int*)

moreover **have** $?r' \longrightarrow \exp (z * \text{of\_real } (\ln 1))$

by (*intro tendsto_intros LIMSEQ_Suc_n_over_n*) *simp_all*

ultimately show $?r \longrightarrow 1$ **by** (*force intro: Lim_transform_eventually*)

from *eventually_gt_at_top*[of $0::\text{nat}$]

show *eventually* $(\lambda n. ?r n = \text{Gamma\_series\_euler}' z n / \text{Gamma\_series } z n)$ *sequentially*

proof *eventually_elim*

fix $n :: \text{nat}$ **assume** $n : n > 0$

from n z' **have** $\text{Gamma\_series\_euler}' z n =$

$\exp (z * \text{of\_real } (\ln (\text{of\_nat } n + 1))) / (z * (\prod_{k=1..n}. (1 + z / \text{of\_nat } k)))$

by (*subst Gamma_euler'_aux1*)

(*simp_all add: Gamma_series_euler'_def prod.distrib*

prod_inversef[symmetric] divide_inverse)

also have $(\prod_{k=1..n}. (1 + z / \text{of\_nat } k)) = \text{pochhammer } (z + 1) n / \text{fact } n$

proof (*cases n*)

case (*Suc n'*)

then show *?thesis*

unfolding *pochhammer_prod fact_prod*

by (*simp add: atLeastLessThanSuc_atLeastAtMost field_simps prod_dividef*

prod.atLeast_Suc_atMost_Suc_shift del: prod.cl_ivl_Suc)

qed *auto*

also have $z * \dots = \text{pochhammer } z (\text{Suc } n) / \text{fact } n$ **by** (*simp add: pochhammer_rec*)

finally show $?r n = \text{Gamma\_series\_euler}' z n / \text{Gamma\_series } z n$ **by** *simp*

qed

qed

end

Weierstrass form

definition *Gamma_series_Weierstrass* :: 'a :: {banach,real_normed_field} $\Rightarrow$ nat $\Rightarrow$ 'a **where**

Gamma_series_Weierstrass z n =
 $\exp(-\text{euler\_mascheroni} * z) / z * (\prod_{k=1..n} \exp(z / \text{of\_nat } k) / (1 + z / \text{of\_nat } k))$

definition

rGamma_series_Weierstrass :: 'a :: {banach,real_normed_field} $\Rightarrow$ nat $\Rightarrow$ 'a **where**

rGamma_series_Weierstrass z n =
 $\exp(\text{euler\_mascheroni} * z) * z * (\prod_{k=1..n} (1 + z / \text{of\_nat } k) * \exp(-z / \text{of\_nat } k))$

lemma *Gamma_series_Weierstrass_nonpos_Ints*:

eventually ($\lambda k. \text{Gamma\_series\_Weierstrass } (- \text{of\_nat } n) k = 0$) *sequentially*

using *eventually_ge_at_top*[of n] **by** *eventually_elim* (auto simp: *Gamma_series_Weierstrass_def*)

lemma *rGamma_series_Weierstrass_nonpos_Ints*:

eventually ($\lambda k. \text{rGamma\_series\_Weierstrass } (- \text{of\_nat } n) k = 0$) *sequentially*

using *eventually_ge_at_top*[of n] **by** *eventually_elim* (auto simp: *rGamma_series_Weierstrass_def*)

theorem *Gamma_Weierstrass_complex*: *Gamma_series_Weierstrass* z $\longrightarrow$ *Gamma* (z :: complex)

proof (cases z $\in \mathbb{Z}_{\leq 0}$)

case *True*

then obtain n **where** z = - of_nat n **by** (*elim nonpos_Ints_cases'*)

also from *True* **have** *Gamma_series_Weierstrass* ... $\longrightarrow$ *Gamma* z

by (simp add: *tendsto_cong*[OF *Gamma_series_Weierstrass_nonpos_Ints*]
Gamma_nonpos_Int)

finally show ?thesis .

next

case *False*

hence z: z $\neq 0$ **by** auto

let ?f = ($\lambda x. \prod_{x = \text{Suc } 0..x} \exp(z / \text{of\_nat } x) / (1 + z / \text{of\_nat } x)$)

have A: $\exp(\ln(1 + z / \text{of\_nat } n)) = (1 + z / \text{of\_nat } n)$ **if** n ≥ 1 **for** n :: nat

using *False* **that** **by** (*subst exp_Ln*) (auto simp: *field_simps* *dest!*: *plus_of_nat_eq_0_imp*)

have ($\lambda n. \sum_{k=1..n} z / \text{of\_nat } k - \ln(1 + z / \text{of\_nat } k)$) $\longrightarrow$ $\ln\_Gamma$
z + *euler_mascheroni* * z + $\ln z$

using *ln_Gamma_series'_aux*[OF *False*]

by (simp only: *atLeastLessThanSuc_atLeastAtMost* [*symmetric*] *One_nat_def*
sum.shift_bounds_Suc_ivl *sums_def* *atLeast0LessThan*)

from *tendsto_exp*[OF *this*] *False* z **have** ?f $\longrightarrow$ z * $\exp(\text{euler\_mascheroni} * z)$ * *Gamma* z

by (simp add: *exp_add exp_sum exp_diff mult_ac* *Gamma_complex_altdef* A)

from *tendsto_mult*[OF *tendsto_const*[of $\exp(-\text{euler\_mascheroni} * z) / z$] *this*]

z

show *Gamma_series_Weierstrass* z $\longrightarrow$ *Gamma* z

by (simp add: *exp_minus field_split_simps* *Gamma_series_Weierstrass_def*)

[abs_def])
qed

lemma *tendsto_complex_of_real_iff*: $((\lambda x. \text{complex\_of\_real } (f x)) \longrightarrow \text{of\_real } c) \iff F = (f \longrightarrow c) F$
by (rule tendsto_of_real_iff)

lemma *Gamma_Weierstrass_real*: $\text{Gamma\_series\_Weierstrass } x \longrightarrow \text{Gamma}$
($x :: \text{real}$)
using *Gamma_Weierstrass_complex*[of *of_real* x] **unfolding** *Gamma_series_Weierstrass_def*[*abs_def*]
by (subst *tendsto_complex_of_real_iff* [symmetric])
(simp_all add: *exp_of_real*[symmetric] *Gamma_complex_of_real*)

lemma *rGamma_Weierstrass_complex*: $r\text{Gamma\_series\_Weierstrass } z \longrightarrow r\text{Gamma}$
($z :: \text{complex}$)
proof (cases $z \in \mathbb{Z}_{\leq 0}$)
case True
then obtain n where $z = - \text{of\_nat } n$ by (elim *nonpos_Ints_cases'*)
also from True have $r\text{Gamma\_series\_Weierstrass } \dots \longrightarrow r\text{Gamma } z$
by (simp add: *tendsto_cong*[OF *rGamma_series_Weierstrass_nonpos_Ints*]
rGamma_nonpos_Int)
finally show ?thesis .
next
case False
have $r\text{Gamma\_series\_Weierstrass } z = (\lambda n. \text{inverse } (\text{Gamma\_series\_Weierstrass } z n))$
by (simp add: *rGamma_series_Weierstrass_def*[*abs_def*] *Gamma_series_Weierstrass_def*
exp_minus *divide_inverse* *prod_inversef*[symmetric] *mult_ac*)
also from False have $\dots \longrightarrow \text{inverse } (\text{Gamma } z)$
by (intro *tendsto_intros* *Gamma_Weierstrass_complex*) (simp add: *Gamma_eq_zero_iff*)
finally show ?thesis by (simp add: *Gamma_def*)
qed

Binomial coefficient form

lemma *Gamma_gbinomial*:
 $(\lambda n. ((z + \text{of\_nat } n) \text{gchoose } n) * \text{exp } (-z * \text{of\_real } (\ln (\text{of\_nat } n)))) \longrightarrow r\text{Gamma } (z+1)$
proof (cases $z = 0$)
case False
show ?thesis
proof (rule *Lim_transform_eventually*)
let ?powr = $\lambda a b. \text{exp } (b * \text{of\_real } (\ln (\text{of\_nat } a)))$
show eventually $(\lambda n. r\text{Gamma\_series } z n / z = ((z + \text{of\_nat } n) \text{gchoose } n) * ?powr n (-z))$ sequentially
proof (intro *always_eventually_allI*)
fix $n :: \text{nat}$
from False have $((z + \text{of\_nat } n) \text{gchoose } n) = \text{pochhammer } z (\text{Suc } n) / z / \text{fact } n$

```

    by (simp add: gbinomial_pochhammer' pochhammer_rec)
    also have pochhammer z (Suc n) / z / fact n * ?powr n (-z) = rGamma_series
z n / z
    by (simp add: rGamma_series_def field_split_simps exp_minus)
    finally show rGamma_series z n / z = ((z + of_nat n) gchoose n) * ?powr
n (-z) ..
qed

from False have (λn. rGamma_series z n / z) ⟶ rGamma z / z by (intro
tendsto_intros)
also from False have rGamma z / z = rGamma (z + 1) using rGamma_plus1[of
z]
    by (simp add: field_simps)
    finally show (λn. rGamma_series z n / z) ⟶ rGamma (z+1) .
qed
qed (simp_all add: binomial_gbinomial [symmetric])

lemma gbinomial_minus': (a + of_nat b) gchoose b = (- 1) ^ b * (- (a + 1)
gchoose b)
    by (subst gbinomial_minus) (simp add: power_mult_distrib [symmetric])

lemma gbinomial_asymptotic:
    fixes z :: 'a :: Gamma
    shows (λn. (z gchoose n) / ((-1)^n / exp ((z+1) * of_real (ln (real n)))))
⟶
    inverse (Gamma (- z))
    unfolding rGamma_inverse_Gamma [symmetric] using Gamma_gbinomial[of
-z-1]
    by (subst (asm) gbinomial_minus')
    (simp add: add_ac mult_ac divide_inverse power_inverse [symmetric])

lemma fact_binomial_limit:
    (λn. of_nat ((k + n) choose n) / of_nat (n ^ k) :: 'a :: Gamma) ⟶ 1 /
fact k
proof (rule Lim_transform_eventually)
    have (λn. of_nat ((k + n) choose n) / of_real (exp (of_nat k * ln (real_of_nat
n)))))
    ⟶ 1 / Gamma (of_nat (Suc k) :: 'a) (is ?f ⟶ _)
    using Gamma_gbinomial[of of_nat k :: 'a]
    by (simp add: binomial_gbinomial Gamma_def field_split_simps exp_of_real
[symmetric] exp_minus)
    also have Gamma (of_nat (Suc k)) = fact k by (simp add: Gamma_fact)
    finally show ?f ⟶ 1 / fact k .

show eventually (λn. ?f n = of_nat ((k + n) choose n) / of_nat (n ^ k))
sequentially
    using eventually_gt_at_top[of 0::nat]
proof eventually_elim
    fix n :: nat assume n: n > 0

```

```

    from n have exp (real_of_nat k * ln (real_of_nat n)) = real_of_nat (n ^ k)
    by (simp add: exp_of_nat_mult)
    thus ?f n = of_nat ((k + n) choose n) / of_nat (n ^ k) by simp
qed
qed

lemma binomial_asymptotic':
  (λn. of_nat ((k + n) choose n) / (of_nat (n ^ k) / fact k) :: 'a :: Gamma)
  ⟶ 1
  using tendsto_mult[OF fact_binomial_limit[of k] tendsto_const[of fact k :: 'a]]
  by simp

lemma gbinomial_Beta:
  assumes z + 1 ∉ ℤ≤₀
  shows ((z::'a::Gamma) gchoose n) = inverse ((z + 1) * Beta (z - of_nat n
+ 1) (of_nat n + 1))
  using assms
  proof (induction n arbitrary: z)
    case 0
    hence z + 2 ∉ ℤ≤₀
    using plus_one_in_nonpos_Ints_imp[of z+1] by (auto simp: add.commute)
    with 0 show ?case
    by (auto simp: Beta_def Gamma_eq_zero_iff Gamma_plus1 [symmetric]
add.commute)
  next
    case (Suc n z)
    show ?case
    proof (cases z ∈ ℤ≤₀)
      case True
      with Suc.premis have z = 0
      by (auto elim!: nonpos_Ints_cases simp: algebra_simps one_plus_of_int_in_nonpos_Ints_iff)
      show ?thesis
      proof (cases n = 0)
        case True
        with ⟨z = 0⟩ show ?thesis
        by (simp add: Beta_def Gamma_eq_zero_iff Gamma_plus1 [symmetric])
      next
        case False
        with ⟨z = 0⟩ show ?thesis
        by (simp_all add: Beta_pole1 one_minus_of_nat_in_nonpos_Ints_iff)
      qed
    next
      case False
      have (z gchoose (Suc n)) = ((z - 1 + 1) gchoose (Suc n)) by simp
      also have ... = (z - 1 gchoose n) * ((z - 1) + 1) / of_nat (Suc n)
      by (subst gbinomial_factors) (simp add: field_simps)
      also from False have ... = inverse (of_nat (Suc n) * Beta (z - of_nat n)
(of_nat (Suc n)))
      (is _ = inverse ?x) by (subst Suc.IH) (simp_all add: field_simps Beta_pole1)
    qed
  qed

```

also have $\text{of\_nat } (\text{Suc } n) \notin (\mathbb{Z}_{\leq 0} :: 'a \text{ set})$ by (subst of_nat_in_nonpos_Ints_iff) simp_all
 hence $?x = (z + 1) * \text{Beta } (z - \text{of\_nat } (\text{Suc } n) + 1) (\text{of\_nat } (\text{Suc } n) + 1)$
 by (subst Beta_plus1_right [symmetric]) simp_all
 finally show ?thesis .
 qed
 qed

theorem gbinomial_Gamma:

assumes $z + 1 \notin \mathbb{Z}_{\leq 0}$
 shows $(z \text{ gchoose } n) = \text{Gamma } (z + 1) / (\text{fact } n * \text{Gamma } (z - \text{of\_nat } n + 1))$
 proof -
 have $(z \text{ gchoose } n) = \text{Gamma } (z + 2) / (z + 1) / (\text{fact } n * \text{Gamma } (z - \text{of\_nat } n + 1))$
 by (subst gbinomial_Beta[OF assms]) (simp_all add: Beta_def Gamma_fact [symmetric] add_ac)
 also from assms have $\text{Gamma } (z + 2) / (z + 1) = \text{Gamma } (z + 1)$
 using Gamma_plus1[of z+1] by (auto simp add: field_split_simps)
 finally show ?thesis .
 qed

Integral form

lemma integrable_on_pwr_from_0':

assumes $a > (-1::\text{real})$ and $c: c \geq 0$
 shows $(\lambda x. x \text{ pwr } a) \text{ integrable\_on } \{0 <..c\}$
 proof -
 from c have *: $\{0 <..c\} - \{0..c\} = \{\}$ $\{0..c\} - \{0 <..c\} = \{0\}$ by auto
 show ?thesis
 by (rule integrable_spike_set [OF integrable_on_pwr_from_0 [OF a c]]) (simp_all add: *)
 qed

lemma absolutely_integrable_Gamma_integral:

assumes $\text{Re } z > 0$ $a > 0$
 shows $(\lambda t. \text{complex\_of\_real } t \text{ pwr } (z - 1) / \text{of\_real } (\exp (a * t)))$
 $\text{absolutely\_integrable\_on } \{0 <..\}$ (is ?f absolutely_integrable_on _)
 proof -
 have $((\lambda x. (\text{Re } z - 1) * (\ln x / x)) \longrightarrow (\text{Re } z - 1) * 0) \text{ at\_top}$
 by (intro tendsto_intros ln_x_over_x_tendsto_0)
 hence $((\lambda x. ((\text{Re } z - 1) * \ln x) / x) \longrightarrow 0) \text{ at\_top}$ by simp
 from order_tendstoD(2)[OF this, of a/2] and $\langle a > 0 \rangle$
 have eventually $(\lambda x. (\text{Re } z - 1) * \ln x / x < a/2) \text{ at\_top}$ by simp
 from eventually_conj[OF this eventually_gt_at_top[of 0]]
 obtain $x0$ where $\forall x \geq x0. (\text{Re } z - 1) * \ln x / x < a/2 \wedge x > 0$
 by (auto simp: eventually_at_top_linorder)
 hence $x0 > 0$ by simp
 have $x \text{ pwr } (\text{Re } z - 1) / \exp (a * x) < \exp (-(a/2) * x)$ if $x \geq x0$ for x

```

proof -
  from that and  $\langle \forall x \geq x0. \_ \rangle$  have  $x: (Re\ z - 1) * \ln\ x / x < a / 2\ x > 0$  by
auto
  have  $x\ powr\ (Re\ z - 1) = exp\ ((Re\ z - 1) * \ln\ x)$ 
    using  $\langle x > 0 \rangle$  by (simp add: powr_def)
  also from  $x$  have  $(Re\ z - 1) * \ln\ x < (a * x) / 2$  by (simp add: field_simps)
  finally show ?thesis by (simp add: field_simps exp_add [symmetric])
qed
note  $x0 = \langle x0 > 0 \rangle$  this

have ?f absolutely_integrable_on  $(\{0 <..x0\} \cup \{x0..\})$ 
proof (rule set_integrable_Un)
  show ?f absolutely_integrable_on  $\{0 <..x0\}$ 
    unfolding set_integrable_def
  proof (rule Bochner_Integration.integrable_bound [OF _ _ AE_I2])
    show integrable lebesgue  $(\lambda x. \text{indicat\_real}\ \{0 <..x0\}\ x *_R x\ powr\ (Re\ z - 1))$ 

      using  $x0(1)$  assms
      by (intro nonnegative_absolutely_integrable_1 [unfolded set_integrable_def]
integrable_on_powr_from_0') auto
      show  $(\lambda x. \text{indicat\_real}\ \{0 <..x0\}\ x *_R (x\ powr\ (z - 1) / exp\ (a * x))) \in$ 
borel_measurable lebesgue
        by (intro measurable_completion)
        (auto intro!: borel_measurable_continuous_on_indicator continuous_intros)
      fix  $x :: real$ 
      have  $x\ powr\ (Re\ z - 1) / exp\ (a * x) \leq x\ powr\ (Re\ z - 1) / 1$  if  $x \geq 0$ 
        using that assms by (intro divide_left_mono) auto
      thus norm  $(\text{indicator}\ \{0 <..x0\}\ x *_R\ ?f\ x) \leq$ 
        norm  $(\text{indicator}\ \{0 <..x0\}\ x *_R\ x\ powr\ (Re\ z - 1))$ 
        by (simp_all add: norm_divide norm_powr_real_powr indicator_def)
    qed
  next
    show ?f absolutely_integrable_on  $\{x0..\}$ 
      unfolding set_integrable_def
    proof (rule Bochner_Integration.integrable_bound [OF _ _ AE_I2])
      show integrable lebesgue  $(\lambda x. \text{indicat\_real}\ \{x0..\}\ x *_R exp\ (-(a / 2) * x))$ 
using assms
      by (intro nonnegative_absolutely_integrable_1 [unfolded set_integrable_def]
integrable_on_exp_minus_to_infinity) auto
      show  $(\lambda x. \text{indicat\_real}\ \{x0..\}\ x *_R (x\ powr\ (z - 1) / exp\ (a * x))) \in$ 
borel_measurable lebesgue using  $x0(1)$ 
        by (intro measurable_completion)
        (auto intro!: borel_measurable_continuous_on_indicator continuous_intros)
      fix  $x :: real$ 
      show norm  $(\text{indicator}\ \{x0..\}\ x *_R\ ?f\ x) \leq$ 
        norm  $(\text{indicator}\ \{x0..\}\ x *_R exp\ (-(a/2) * x))$  using  $x0$ 
        by (auto simp: norm_divide norm_powr_real_powr indicator_def less_imp_le)
    qed
qed auto

```

also have $\{0 <..x0\} \cup \{x0..\} = \{0 <..\}$ using $x0(1)$ by *auto*
 finally show *?thesis* .
 qed

lemma *integrable_Gamma_integral_bound*:
 fixes $a\ c :: \text{real}$
 assumes $a: a > -1$ and $c: c \geq 0$
 defines $f \equiv \lambda x. \text{if } x \in \{0..c\} \text{ then } x \text{ powr } a \text{ else } \exp(-x/2)$
 shows $f \text{ integrable\_on } \{0..\}$
proof -
 have $f \text{ integrable\_on } \{0..c\}$
 by (rule *integrable_spike_finite*[of $\{ \}$, $OF\_ \_ \text{integrable\_on\_powr\_from\_0}$ [of
 $a\ c$]])
 (insert $a\ c$, *simp_all add: f_def*)
 moreover have $A: (\lambda x. \exp(-x/2)) \text{ integrable\_on } \{c..\}$
 using *integrable_on_exp_minus_to_infinity*[of $1/2$] by *simp*
 have $f \text{ integrable\_on } \{c..\}$
 by (rule *integrable_spike_finite*[of $\{c\}$, $OF\_ \_ A$]) (*simp_all add: f_def*)
 ultimately show $f \text{ integrable\_on } \{0..\}$
 by (rule *integrable_Un'*) (insert c , *auto simp: max_def*)
 qed

theorem *Gamma_integral_complex*:
 assumes $z: \text{Re } z > 0$
 shows $((\lambda t. \text{of\_real } t \text{ powr } (z - 1) / \text{of\_real } (\exp t)) \text{ has\_integral } \Gamma z)$
 $\{0..\}$
proof -
 have $A: ((\lambda t. (\text{of\_real } t) \text{ powr } (z - 1) * \text{of\_real } ((1 - t) ^ n))$
 $\text{has\_integral } (\text{fact } n / \text{pochhammer } z (n+1))) \{0..1\}$
 if $\text{Re } z > 0$ for $n\ z$ using *that*
proof (*induction n arbitrary: z*)
 case 0
 have $((\lambda t. \text{complex\_of\_real } t \text{ powr } (z - 1)) \text{ has\_integral }$
 $(\text{of\_real } 1 \text{ powr } z / z - \text{of\_real } 0 \text{ powr } z / z)) \{0..1\}$ using 0
 by (*intro fundamental_theorem_of_calculus_interior*)
 (*auto intro!: continuous_intros derivative_eq_intros has_vector_derivative_real_field*)
 thus *?case* by *simp*
 next
 case (*Suc n*)
 let $?f = \lambda t. \text{complex\_of\_real } t \text{ powr } z / z$
 let $?f' = \lambda t. \text{complex\_of\_real } t \text{ powr } (z - 1)$
 let $?g = \lambda t. (1 - \text{complex\_of\_real } t) ^ \text{Suc } n$
 let $?g' = \lambda t. - ((1 - \text{complex\_of\_real } t) ^ n) * \text{of\_nat } (\text{Suc } n)$
 have $((\lambda t. ?f' t * ?g t) \text{ has\_integral }$
 $(\text{of\_nat } (\text{Suc } n) * \text{fact } n / \text{pochhammer } z (n+2))) \{0..1\}$
 (*is* $\_ \text{has\_integral } ?I$) $\_$
proof (*rule integration_by_parts_interior*[**where** $f' = ?f'$ and $g = ?g$])
from *Suc.prem*s **show** *continuous_on* $\{0..1\}$ $?f \text{ continuous\_on } \{0..1\}$ $?g$
 by (*auto intro!: continuous_intros*)


```

next
  fix t :: real assume t: t ∈ {0<.. $1$ }
  show (?f has_vector_derivative ?f' t) (at t) using t Suc.prem
    by (auto intro!: derivative_eq_intros has_vector_derivative_real_field)
  show (?g has_vector_derivative ?g' t) (at t)
    by (rule has_vector_derivative_real_field derivative_eq_intros refl)+
simp_all
next
  from Suc.prem have [simp]: z ≠ 0 by auto
  from Suc.prem have A: Re (z + of_nat n) > 0 for n by simp
  have [simp]: z + of_nat n ≠ 0 z + 1 + of_nat n ≠ 0 for n
  using A[of n] A[of Suc n] by (auto simp add: add.assoc simp del: plus_complex.sel)
  have ((λx. of_real x powr z * of_real ((1 - x) ^ n) * (- of_nat (Suc n) /
z)) has_integral
    fact n / pochhammer (z+1) (n+1) * (- of_nat (Suc n) / z)) {0.. $1$ }
    (is (?A has_integral ?B) _)
  using Suc.IH[of z+1] Suc.prem by (intro has_integral_mult_left) (simp_all
add: add_ac pochhammer_rec)
  also have ?A = (λt. ?f t * ?g' t) by (intro ext) (simp_all add: field_simps)
  also have ?B = - (of_nat (Suc n) * fact n / pochhammer z (n+2))
    by (simp add: field_split_simps pochhammer_rec
prod.shift_bounds_cl_Suc_ivl del: of_nat_Suc)
  finally show ((λt. ?f t * ?g' t) has_integral (?f 1 * ?g 1 - ?f 0 * ?g 0 -
?I)) {0.. $1$ }
    by simp
qed (simp_all add: bounded_bilinear_mult)
thus ?case by simp
qed

have B: ((λt. if t ∈ {0.. $of\_nat\ n$ } then
  of_real t powr (z - 1) * (1 - of_real t / of_nat n) ^ n else 0)
  has_integral (of_nat n powr z * fact n / pochhammer z (n+1))) {0.. $1$ }
for n
proof (cases n > 0)
case [simp]: True
hence [simp]: n ≠ 0 by auto
with has_integral_affinity01[OF A[OF z, of n], of inverse (of_nat n) 0]
have ((λx. (of_nat n - of_real x) ^ n * (of_real x / of_nat n) powr (z -
1) / of_nat n ^ n)
  has_integral fact n * of_nat n / pochhammer z (n+1)) ((λx. real n *
x) {0.. $1$ })
  (is (?f has_integral ?I) ?ivl) by (simp add: field_simps scaleR_conv_of_real)
also from True have ((λx. real n*x) {0.. $1$ }) = {0.. $real\ n$ }
  by (subst image_mult_atLeastAtMost) simp_all
also have ?f = (λx. (of_real x / of_nat n) powr (z - 1) * (1 - of_real x /
of_nat n) ^ n)
  using True by (intro ext) (simp add: field_simps)
finally have ((λx. (of_real x / of_nat n) powr (z - 1) * (1 - of_real x /
of_nat n) ^ n)

```

```

      has_integral ?I) {0..real n} (is ?P) .
    also have ?P  $\longleftrightarrow$  (( $\lambda x.$  exp ((z - 1) * of_real (ln (x / of_nat n))) * (1 -
of_real x / of_nat n) ^ n)
      has_integral ?I) {0..real n}
    by (intro has_integral_spike_finite_eq[of {0}]) (auto simp: powr_def Ln_of_real
[symmetric])
    also have ...  $\longleftrightarrow$  (( $\lambda x.$  exp ((z - 1) * of_real (ln x - ln (of_nat n))) * (1
- of_real x / of_nat n) ^ n)
      has_integral ?I) {0..real n}
    by (intro has_integral_spike_finite_eq[of {0}]) (simp_all add: ln_div)
  finally have ... .
  note B = has_integral_mult_right[OF this, of exp ((z - 1) * ln (of_nat n))]
  have (( $\lambda x.$  exp ((z - 1) * of_real (ln x)) * (1 - of_real x / of_nat n) ^ n)
    has_integral (?I * exp ((z - 1) * ln (of_nat n)))) {0..real n} (is ?P)
    by (insert B, subst (asm) mult.assoc [symmetric], subst (asm) exp_add
[symmetric])
      (simp add: algebra_simps)
    also have ?P  $\longleftrightarrow$  (( $\lambda x.$  of_real x powr (z - 1) * (1 - of_real x / of_nat n)
^ n)
      has_integral (?I * exp ((z - 1) * ln (of_nat n)))) {0..real n}
    by (intro has_integral_spike_finite_eq[of {0}]) (simp_all add: powr_def
Ln_of_real)
    also have fact n * of_nat n / pochhammer z (n+1) * exp ((z - 1) * Ln
(of_nat n)) =
      (of_nat n powr z * fact n / pochhammer z (n+1))
    by (auto simp add: powr_def algebra_simps exp_diff exp_of_real)
  finally show ?thesis by (subst has_integral_restrict) simp_all
next
case False
thus ?thesis by (subst has_integral_restrict) (simp_all add: has_integral_refl)
qed

have eventually ( $\lambda n.$  Gamma_series z n =
  of_nat n powr z * fact n / pochhammer z (n+1)) sequentially
  using eventually_gt_at_top[of 0::nat]
  by eventually_elim (simp add: powr_def algebra_simps Gamma_series_def)
from this and Gamma_series_LIMSEQ[of z]
have C: ( $\lambda k.$  of_nat k powr z * fact k / pochhammer z (k+1))  $\longrightarrow$  Gamma
z
  by (blast intro: Lim_transform_eventually)
{
  fix x :: real assume x: x  $\geq$  0
  have lim_exp: ( $\lambda k.$  (1 - x / real k) ^ k)  $\longrightarrow$  exp (-x)
    using tendsto_exp_limit_sequentially[of -x] by simp
  have ( $\lambda k.$  of_real x powr (z - 1) * of_real ((1 - x / of_nat k) ^ k))
     $\longrightarrow$  of_real x powr (z - 1) * of_real (exp (-x)) (is ?P)
    by (intro tendsto_intros lim_exp)
  also from eventually_gt_at_top[of nat [x]]
  have eventually ( $\lambda k.$  of_nat k > x) sequentially by eventually_elim linarith

```

```

hence ?P  $\longleftrightarrow$  ( $\lambda k$ . if  $x \leq \text{of\_nat } k$  then
  of_real  $x \text{ powr } (z - 1) * \text{of\_real } ((1 - x / \text{of\_nat } k) ^ k)$  else 0)
   $\longrightarrow$  of_real  $x \text{ powr } (z - 1) * \text{of\_real } (\exp (-x))$ 
by (intro tendsto_cong) (auto elim!: eventually_mono)
finally have ... .
}
hence D:  $\forall x \in \{0.. \}$ . ( $\lambda k$ . if  $x \in \{0.. \text{real } k\}$  then
  of_real  $x \text{ powr } (z - 1) * (1 - \text{of\_real } x / \text{of\_nat } k) ^ k$  else 0)
   $\longrightarrow$  of_real  $x \text{ powr } (z - 1) / \text{of\_real } (\exp x)$ 
by (simp add: exp_minus field_simps cong: if_cong)

have (( $\lambda x$ .  $(\text{Re } z - 1) * (\ln x / x)$ )  $\longrightarrow$   $(\text{Re } z - 1) * 0$ ) at_top
by (intro tendsto_intros ln_x_over_x_tendsto_0)
hence (( $\lambda x$ .  $((\text{Re } z - 1) * \ln x) / x$ )  $\longrightarrow$  0) at_top by simp
from order_tendstoD(2)[OF this, of 1/2]
  have eventually ( $\lambda x$ .  $(\text{Re } z - 1) * \ln x / x < 1/2$ ) at_top by simp
from eventually_conj[OF this eventually_gt_at_top[of 0]]
  obtain  $x0$  where  $\forall x \geq x0$ .  $(\text{Re } z - 1) * \ln x / x < 1/2 \wedge x > 0$ 
  by (auto simp: eventually_at_top_linorder)
hence  $x0$ :  $x0 > 0 \wedge x. x \geq x0 \implies (\text{Re } z - 1) * \ln x < x / 2$  by auto

define h where  $h = (\lambda x$ . if  $x \in \{0..x0\}$  then  $x \text{ powr } (\text{Re } z - 1)$  else  $\exp (-x/2)$ )
have le_h:  $x \text{ powr } (\text{Re } z - 1) * \exp (-x) \leq h \ x$  if  $x$ :  $x \geq 0$  for  $x$ 
proof (cases  $x > x0$ )
  case True
    from True  $x0(1)$  have  $x \text{ powr } (\text{Re } z - 1) * \exp (-x) = \exp ((\text{Re } z - 1) * \ln$ 
 $x - x)$ 
    by (simp add: powr_def exp_diff exp_minus field_simps exp_add)
    also from  $x0(2)$ [of  $x$ ] True have  $\dots < \exp (-x/2)$ 
    by (simp add: field_simps)
    finally show ?thesis using True by (auto simp add: h_def)
  next
  case False
    from  $x$  have  $x \text{ powr } (\text{Re } z - 1) * \exp (-x) \leq x \text{ powr } (\text{Re } z - 1) * 1$ 
    by (intro mult_left_mono) simp_all
    with False show ?thesis by (auto simp add: h_def)
qed

have E:  $\forall x \in \{0.. \}$ .  $\text{cmod } (\text{if } x \in \{0.. \text{real } k\} \text{ then } \text{of\_real } x \text{ powr } (z - 1) * (1 - \text{complex\_of\_real } x / \text{of\_nat } k) ^ k \text{ else } 0) \leq h \ x$ 
(is  $\forall x \in \_.$   $?f \ x \leq \_$ ) for  $k$ 
proof safe
  fix  $x :: \text{real}$  assume  $x: x \geq 0$ 
  {
    fix  $x :: \text{real}$  and  $n :: \text{nat}$  assume  $x: x \leq \text{of\_nat } n$ 
    have  $(1 - \text{complex\_of\_real } x / \text{of\_nat } n) = \text{complex\_of\_real } ((1 - x / \text{of\_nat } n))$  by simp
    also have  $\text{norm } \dots = |(1 - x / \text{real } n)|$  by (subst norm_of_real) (rule refl)
    also from  $x$  have  $\dots = (1 - x / \text{real } n)$  by (intro abs_of_nonneg) (simp_all

```

```

add: field_split_simps)
  finally have cmod (1 - complex_of_real x / of_nat n) = 1 - x / real n .
} note D = this
from D[of x k] x
  have ?f x ≤ (if of_nat k ≥ x ∧ k > 0 then x powr (Re z - 1) * (1 - x /
real k) ^ k else 0)
  by (auto simp: norm_mult norm_powr_real_powr norm_power intro!: mult_nonneg_nonneg)
  also have ... ≤ x powr (Re z - 1) * exp (-x)
  by (auto intro!: mult_left_mono exp_ge_one_minus_x_over_n_power_n)
  also from x have ... ≤ h x by (rule le_h)
  finally show ?f x ≤ h x .
qed

```

```

have F: h integrable_on {0..} unfolding h_def
  by (rule integrable_Gamma_integral_bound) (insert assms x0(1), simp_all)
show ?thesis
  by (rule has_integral_dominated_convergence[OF B F E D C])
qed

```

```

lemma Gamma_integral_real:
  assumes x: x > (0 :: real)
  shows ((λt. t powr (x - 1) / exp t) has_integral Gamma x) {0..}
proof -
  have A: ((λt. complex_of_real t powr (complex_of_real x - 1) /
    complex_of_real (exp t)) has_integral complex_of_real (Gamma x)) {0..}
  using Gamma_integral_complex[of x] assms by (simp_all add: Gamma_complex_of_real
    powr_of_real)
  have ((λt. complex_of_real (t powr (x - 1) / exp t)) has_integral of_real
    (Gamma x)) {0..}
  by (rule has_integral_eq[OF _ A]) (simp_all add: powr_of_real [symmetric])
  from has_integral_linear[OF this bounded_linear_Re] show ?thesis by (simp
    add: o_def)
qed

```

```

lemma absolutely_integrable_Gamma_integral':
  assumes Re z > 0
  shows (λt. complex_of_real t powr (z - 1) / of_real (exp t)) absolutely_integrable_on
    {0<..}
  using absolutely_integrable_Gamma_integral [OF assms zero_less_one] by simp

```

```

lemma Gamma_integral_complex':
  assumes z: Re z > 0
  shows ((λt. of_real t powr (z - 1) / of_real (exp t)) has_integral Gamma z)
    {0<..}
proof -
  have ((λt. of_real t powr (z - 1) / of_real (exp t)) has_integral Gamma z)
    {0..}
  by (rule Gamma_integral_complex) fact+
  hence ((λt. if t ∈ {0<..} then of_real t powr (z - 1) / of_real (exp t) else 0)

```

```

      has_integral Gamma z) {0..}
    by (rule has_integral_spike [of {0}, rotated 2]) auto
  also have ?this = ?thesis
    by (subst has_integral_restrict) auto
  finally show ?thesis .
qed

```

```

lemma Gamma_conv_nn_integral_real:
  assumes  $s > (0::real)$ 
  shows  $\text{Gamma } s = \text{nn\_integral lborel } (\lambda t. \text{ennreal } (\text{indicator } \{0..\} t * t \text{ powr } (s - 1) / \exp t))$ 
  using nn_integral_has_integral_lebesgue[OF Gamma_integral_real[OF assms]]
  by simp

```

```

lemma integrable_Beta:
  assumes  $a > 0$   $b > (0::real)$ 
  shows  $\text{set\_integrable lborel } \{0..1\} (\lambda t. t \text{ powr } (a - 1) * (1 - t) \text{ powr } (b - 1))$ 
proof -
  define C where  $C = \max 1 ((1/2) \text{ powr } (b - 1))$ 
  define D where  $D = \max 1 ((1/2) \text{ powr } (a - 1))$ 
  have  $C: (1 - x) \text{ powr } (b - 1) \leq C$  if  $x \in \{0..1/2\}$  for  $x$ 
  proof (cases  $b < 1$ )
    case False
      with that have  $(1 - x) \text{ powr } (b - 1) \leq (1 \text{ powr } (b - 1))$  by (intro powr_mono2) auto
      thus ?thesis by (auto simp: C_def)
    qed (insert that, auto simp: max.coboundedI1 max.coboundedI2 powr_mono2' powr_mono2 C_def)
  have  $D: x \text{ powr } (a - 1) \leq D$  if  $x \in \{1/2..1\}$  for  $x$ 
  proof (cases  $a < 1$ )
    case False
      with that have  $x \text{ powr } (a - 1) \leq (1 \text{ powr } (a - 1))$  by (intro powr_mono2) auto
      thus ?thesis by (auto simp: D_def)
    next
    case True
      qed (insert that, auto simp: max.coboundedI1 max.coboundedI2 powr_mono2' powr_mono2 D_def)
  have [simp]:  $C \geq 0$   $D \geq 0$  by (simp_all add: C_def D_def)

  have I1:  $\text{set\_integrable lborel } \{0..1/2\} (\lambda t. t \text{ powr } (a - 1) * (1 - t) \text{ powr } (b - 1))$ 
  unfolding set_integrable_def
  proof (rule Bochner_Integration.integrable_bound[OF _ AE_I2])
    have  $(\lambda t. t \text{ powr } (a - 1)) \text{ integrable\_on } \{0..1/2\}$ 
      by (rule integrable_on_powr_from_0) (use assms in auto)
    hence  $(\lambda t. t \text{ powr } (a - 1)) \text{ absolutely\_integrable\_on } \{0..1/2\}$ 
      by (subst absolutely_integrable_on_iff_nonneg) auto

```

```

from integrable_mult_right[OF this [unfolded set_integrable_def], of C]
show integrable lborel ( $\lambda x. \text{indicat\_real } \{0..1/2\} x *_R (C * x \text{ powr } (a - 1))$ )
  by (subst (asm) integrable_completion) (auto simp: mult_ac)
next
  fix x :: real
  have  $x \text{ powr } (a - 1) * (1 - x) \text{ powr } (b - 1) \leq x \text{ powr } (a - 1) * C$  if  $x \in \{0..1/2\}$ 
    using that by (intro mult_left_mono powr_mono2 C) auto
  thus norm (indicator  $\{0..1/2\} x *_R (x \text{ powr } (a - 1) * (1 - x) \text{ powr } (b - 1))$ )  $\leq$ 
    norm (indicator  $\{0..1/2\} x *_R (C * x \text{ powr } (a - 1))$ )
    by (auto simp: indicator_def abs_mult mult_ac)
qed (auto intro!: AE_I2 simp: indicator_def)

  have I2: set_integrable lborel  $\{1/2..1\} (\lambda t. t \text{ powr } (a - 1) * (1 - t) \text{ powr } (b - 1))$ 
    unfolding set_integrable_def
  proof (rule Bochner_Integration.integrable_bound[OF _ AE_I2])
    have  $(\lambda t. t \text{ powr } (b - 1)) \text{ integrable\_on } \{0..1/2\}$ 
      by (rule integrable_on_powr_from_0) (use assms in auto)
    hence  $(\lambda t. t \text{ powr } (b - 1)) \text{ integrable\_on } (\text{cbox } 0 (1/2))$  by simp
    from integrable_affinity[OF this, of -1 1]
    have  $(\lambda t. (1 - t) \text{ powr } (b - 1)) \text{ integrable\_on } \{1/2..1\}$  by simp
    hence  $(\lambda t. (1 - t) \text{ powr } (b - 1)) \text{ absolutely\_integrable\_on } \{1/2..1\}$ 
      by (subst absolutely_integrable_on_iff_nonneg) auto
    from integrable_mult_right[OF this [unfolded set_integrable_def], of D]
    show integrable lborel ( $\lambda x. \text{indicat\_real } \{1/2..1\} x *_R (D * (1 - x) \text{ powr } (b - 1))$ )
      by (subst (asm) integrable_completion) (auto simp: mult_ac)
  next
    fix x :: real
    have  $x \text{ powr } (a - 1) * (1 - x) \text{ powr } (b - 1) \leq D * (1 - x) \text{ powr } (b - 1)$  if  $x \in \{1/2..1\}$ 
      using that by (intro mult_right_mono powr_mono2 D) auto
    thus norm (indicator  $\{1/2..1\} x *_R (x \text{ powr } (a - 1) * (1 - x) \text{ powr } (b - 1))$ )  $\leq$ 
      norm (indicator  $\{1/2..1\} x *_R (D * (1 - x) \text{ powr } (b - 1))$ )
      by (auto simp: indicator_def abs_mult mult_ac)
    qed (auto intro!: AE_I2 simp: indicator_def)

    have set_integrable lborel  $(\{0..1/2\} \cup \{1/2..1\}) (\lambda t. t \text{ powr } (a - 1) * (1 - t) \text{ powr } (b - 1))$ 
      by (intro set_integrable_Un I1 I2) auto
    also have  $\{0..1/2\} \cup \{1/2..1\} = \{0..(1::\text{real})\}$  by auto
    finally show ?thesis .
  qed

lemma integrable_Beta':
  assumes  $a > 0$   $b > (0::\text{real})$ 

```

```

shows (λt. t powr (a - 1) * (1 - t) powr (b - 1)) integrable_on {0..1}
using integrable_Beta[OF assms] by (rule set_borel_integral_eq_integral)

theorem has_integral_Beta_real:
  assumes a: a > 0 and b: b > (0 :: real)
  shows ((λt. t powr (a - 1) * (1 - t) powr (b - 1)) has_integral Beta a b)
{0..1}
proof -
  define B where B = integral {0..1} (λx. x powr (a - 1) * (1 - x) powr (b - 1))
  have [simp]: B ≥ 0 unfolding B_def using a b
  by (intro integral_nonneg integrable_Beta') auto
  from a b have ennreal (Gamma a * Gamma b) =
    (∫+ t. ennreal (indicator {0..} t * t powr (a - 1) / exp t) ∂lborel) *
    (∫+ t. ennreal (indicator {0..} t * t powr (b - 1) / exp t) ∂lborel)
  by (subst ennreal_mult') (simp_all add: Gamma_conv_nn_integral_real)
  also have ... = (∫+ t. ∫+ u. ennreal (indicator {0..} t * t powr (a - 1) / exp
t) *
ennreal (indicator {0..} u * u powr (b - 1) / exp u) ∂lborel
∂lborel)
  by (simp add: nn_integral_cmult nn_integral_multc)
  also have ... = (∫+ t. ∫+ u. ennreal (indicator ({0..} × {0..}) (t,u) * t powr (a
- 1) * u powr (b - 1)
/ exp (t + u)) ∂lborel ∂lborel)
  by (intro nn_integral_cong)
  (auto simp: indicator_def divide_ennreal ennreal_mult' [symmetric] exp_add)
  also have ... = (∫+ t. ∫+ u. ennreal (indicator ({0..} × {t..}) (t,u) * t powr (a
- 1) *
(u - t) powr (b - 1) / exp u) ∂lborel ∂lborel)
  proof (rule nn_integral_cong, goal_cases)
    case (1 t)
    have (∫+ u. ennreal (indicator ({0..} × {0..}) (t,u) * t powr (a - 1) *
u powr (b - 1) / exp (t + u)) ∂distr lborel borel ((+)
(-t))) =
      (∫+ u. ennreal (indicator ({0..} × {t..}) (t,u) * t powr (a - 1) *
(u - t) powr (b - 1) / exp u) ∂lborel)
    by (subst nn_integral_distr) (auto intro!: nn_integral_cong simp: indica
tor_def)
    thus ?case by (subst (asm) lborel_distr_plus)
  qed
  also have ... = (∫+ u. ∫+ t. ennreal (indicator ({0..} × {t..}) (t,u) * t powr (a
- 1) *
(u - t) powr (b - 1) / exp u) ∂lborel ∂lborel)
  by (subst lborel_pair.Fubini')
  (auto simp: case_prod_unfold indicator_def cong: measurable_cong_sets)
  also have ... = (∫+ u. ∫+ t. ennreal (indicator {0..u} t * t powr (a - 1) * (u
- t) powr (b - 1)) *
ennreal (indicator {0..} u / exp u) ∂lborel ∂lborel)
  by (intro nn_integral_cong) (auto simp: indicator_def ennreal_mult' [symmetric])

```

```

also have ... = (∫+u. (∫+t. ennreal (indicator {0..u} t * t powr (a - 1) * (u
- t) powr (b - 1)))
    ∂lborel) * ennreal (indicator {0..} u / exp u) ∂lborel)
by (subst nn_integral_multc [symmetric]) auto
also have ... = (∫+u. (∫+t. ennreal (indicator {0..u} t * t powr (a - 1) * (u
- t) powr (b - 1)))
    ∂lborel) * ennreal (indicator {0<..} u / exp u) ∂lborel)
by (intro nn_integral_cong_AE eventually_mono[OF AE_lborel_singleton[of
0]])
    (auto simp: indicator_def)
also have ... = (∫+u. ennreal B * ennreal (indicator {0..} u / exp u * u powr
(a + b - 1)) ∂lborel)
proof (intro nn_integral_cong, goal_cases)
  case (1 u)
  show ?case
  proof (cases u > 0)
    case True
    have (∫+t. ennreal (indicator {0..u} t * t powr (a - 1) * (u - t) powr (b
- 1)) ∂lborel) =
      (∫+t. ennreal (indicator {0..1} t * (u * t) powr (a - 1) * (u - u *
t) powr (b - 1)))
      ∂distr lborel borel ((* (1 / u))) (is _ = nn_integral _ ?f)
    using True
    by (subst nn_integral_distr) (auto simp: indicator_def field_simps intro!:
nn_integral_cong)
    also have distr lborel borel ((* (1 / u)) = density lborel (λ_. u)
    using ⟨u > 0⟩ by (subst lborel_distr_mult) auto
    also have nn_integral ... ?f = (∫+x. ennreal (indicator {0..1} x * (u * (u
* x) powr (a - 1) *
      (u * (1 - x)) powr (b - 1))) ∂lborel) using
    ⟨u > 0⟩
    by (subst nn_integral_density) (auto simp: ennreal_mult' [symmetric]
algebra_simps)
    also have ... = (∫+x. ennreal (u powr (a + b - 1)) *
      ennreal (indicator {0..1} x * x powr (a - 1) *
      (1 - x) powr (b - 1)) ∂lborel) using ⟨u > 0⟩ a b
    by (intro nn_integral_cong)
      (auto simp: indicator_def powr_mult powr_add powr_diff mult_ac
ennreal_mult' [symmetric])
    also have ... = ennreal (u powr (a + b - 1)) *
      (∫+x. ennreal (indicator {0..1} x * x powr (a - 1) *
      (1 - x) powr (b - 1)) ∂lborel)
    by (subst nn_integral_cmult) auto
    also have ((λx. x powr (a - 1) * (1 - x) powr (b - 1)) has_integral
      integral {0..1} (λx. x powr (a - 1) * (1 - x) powr (b - 1)))
    {0..1}
    using a b by (intro integrable_integral integrable_Beta')
    from nn_integral_has_integral_lebesgue[OF _ this] a b
    have (∫+x. ennreal (indicator {0..1} x * x powr (a - 1) *

```



```

      (1 - x) powr (b - 1)) ∂lborel) = B by (simp add: mult_ac
B_def)
  finally show ?thesis using ‹u > 0› by (simp add: ennreal_mult' [symmetric]
mult_ac)
  qed auto
  qed
  also have ... = ennreal B * ennreal (Gamma (a + b))
  using a b by (subst nn_integral_cmult) (auto simp: Gamma_conv_nn_integral_real)
  also have ... = ennreal (B * Gamma (a + b))
  by (subst (1 2) mult.commute, intro ennreal_mult' [symmetric]) (use a b in
auto)
  finally have B = Beta a b using a b Gamma_real_pos[of a + b]
  by (subst (asm) ennreal_inj) (auto simp: field_simps Beta_def Gamma_eq_zero_iff)
  moreover have (λt. t powr (a - 1) * (1 - t) powr (b - 1)) integrable_on
{0..1}
  by (intro integrable_Beta' a b)
  ultimately show ?thesis by (simp add: has_integral_iff B_def)
  qed

```

6.23.12 The Weierstraß product formula for the sine

theorem *sin_product_formula_complex*:

```

  fixes z :: complex
  shows (λn. (− of_real pi * inverse z) * (∏ k=1..n. 1 - z^2 / of_nat k^2)) ⟶ sin
(of_real pi * z)
  proof −
    let ?f = rGamma_series_Weierstrass
    have (λn. (− of_real pi * inverse z) * (?f z n * ?f (− z) n))
      ⟶ (− of_real pi * inverse z) * (rGamma z * rGamma (− z))
    by (intro tendsto_intros rGamma_Weierstrass_complex)
    also have (λn. (− of_real pi * inverse z) * (?f z n * ?f (− z) n)) =
      (λn. of_real pi * z * (∏ k=1..n. 1 - z^2 / of_nat k^2))
    proof
      fix n :: nat
      have (− of_real pi * inverse z) * (?f z n * ?f (− z) n) =
        of_real pi * z * (∏ k=1..n. (of_nat k - z) * (of_nat k + z) / of_nat
k^2)
      by (simp add: rGamma_series_Weierstrass_def mult_ac exp_minus
        divide_simps prod.distrib[symmetric] power2_eq_square)
      also have (∏ k=1..n. (of_nat k - z) * (of_nat k + z) / of_nat k^2) =
        (∏ k=1..n. 1 - z^2 / of_nat k^2)
      by (intro prod.cong) (simp_all add: power2_eq_square field_simps)
      finally show (− of_real pi * inverse z) * (?f z n * ?f (− z) n) = of_real pi *
z * ...
      by (simp add: field_split_simps)
    qed
    also have (− of_real pi * inverse z) * (rGamma z * rGamma (− z)) = sin
(of_real pi * z)
    by (subst rGamma_reflection_complex') (simp add: field_split_simps)
  end

```

finally show *?thesis* .
qed

lemma *sin_product_formula_real*:
 $(\lambda n. \pi * (x :: \text{real}) * (\prod_{k=1..n}. 1 - x^2 / \text{of\_nat } k^2)) \longrightarrow \sin (\pi * x)$
proof –
from *sin_product_formula_complex* [of *of_real x*]
have $(\lambda n. \text{of\_real } \pi * \text{of\_real } x * (\prod_{k=1..n}. 1 - (\text{of\_real } x)^2 / (\text{of\_nat } k)^2))$
 $\longrightarrow \sin (\text{of\_real } \pi * \text{of\_real } x :: \text{complex})$ (**is** *?f* $\longrightarrow$ *?y*) .
also have *?f* = $(\lambda n. \text{of\_real } (\pi * x * (\prod_{k=1..n}. 1 - x^2 / (\text{of\_nat } k^2))))$
by *simp*
also have *?y* = *of_real (sin (pi * x))* **by** (*simp only: sin_of_real [symmetric]*
of_real_mult)
finally show *?thesis* **by** (*subst (asm) tendsto_of_real_iff*)
qed

lemma *sin_product_formula_real'*:
assumes $x \neq (0 :: \text{real})$
shows $(\lambda n. (\prod_{k=1..n}. 1 - x^2 / \text{of\_nat } k^2)) \longrightarrow \sin (\pi * x) / (\pi * x)$
using *tendsto_divide* [OF *sin_product_formula_real* [of *x*] *tendsto_const* [of $\pi * x$]] *assms*
by *simp*

theorem *wallis*: $(\lambda n. \prod_{k=1..n}. (4 * \text{real } k^2) / (4 * \text{real } k^2 - 1)) \longrightarrow \pi / 2$
proof –
from *tendsto_inverse* [OF *tendsto_mult* [OF
sin_product_formula_real [of $1/2$] *tendsto_const* [of $2/\pi$]]]
have $(\lambda n. (\prod_{k=1..n}. \text{inverse } (1 - (1/2)^2 / (\text{real } k)^2))) \longrightarrow \pi/2$
by (*simp add: prod_inversef [symmetric]*)
also have $(\lambda n. (\prod_{k=1..n}. \text{inverse } (1 - (1/2)^2 / (\text{real } k)^2))) =$
 $(\lambda n. (\prod_{k=1..n}. (4 * \text{real } k^2) / (4 * \text{real } k^2 - 1)))$
by (*intro ext prod.cong refl*) (*simp add: field_split_simps*)
finally show *?thesis* .
qed

6.23.13 The Solution to the Basel problem

theorem *inverse_squares_sums*: $(\lambda n. 1 / (n + 1)^2) \text{ sums } (\pi^2 / 6)$
proof –
define *P* **where** $P \ x \ n = (\prod_{k=1..n}. 1 - x^2 / \text{of\_nat } k^2)$ **for** $x :: \text{real}$ **and** n
define *K* **where** $K = (\sum n. \text{inverse } (\text{real\_of\_nat } (\text{Suc } n))^2)$
define *f* **where** [*abs_def*]: $f \ x = (\sum n. P \ x \ n / \text{of\_nat } (\text{Suc } n)^2)$ **for** x
define *g* **where** [*abs_def*]: $g \ x = (1 - \sin (\pi * x) / (\pi * x))$ **for** x
have *sums*: $(\lambda n. P \ x \ n / \text{of\_nat } (\text{Suc } n)^2) \text{ sums } (\text{if } x = 0 \text{ then } K \text{ else } g \ x / x^2)$ **for** x
proof (*cases x = 0*)

```

  assume x: x = 0
  have summable (λn. inverse ((real_of_nat (Suc n))2))
    using inverse_power_summable[of 2] by (subst summable_Suc_iff) simp
  thus ?thesis by (simp add: x_g_def P_def K_def inverse_eq_divide power_divide
    summable_sums)
  next
    assume x: x ≠ 0
    have (λn. P x n - P x (Suc n)) sums (P x 0 - sin (pi * x) / (pi * x))
      unfolding P_def using x by (intro telescope_sums' sin_product_formula_real')
    also have (λn. P x n - P x (Suc n)) = (λn. (x2 / of_nat (Suc n)2) * P x
      n)
      unfolding P_def by (simp add: prod_nat_ivl_Suc' algebra_simps)
    also have P x 0 = 1 by (simp add: P_def)
    finally have (λn. x2 / (of_nat (Suc n))2 * P x n) sums (1 - sin (pi * x) /
      (pi * x)) .
    from sums_divide[OF this, of x2] x show ?thesis unfolding g_def by simp
  qed

  have continuous_on (ball 0 1) f
  proof (rule uniform_limit_theorem; (intro always_eventually_allI)?)
    show uniform_limit (ball 0 1) (λn x. ∑ k<n. P x k / of_nat (Suc k)2) f
      sequentially
    proof (unfold f_def, rule Weierstrass_m_test)
      fix n :: nat and x :: real assume x: x ∈ ball 0 1
      {
        fix k :: nat assume k: k ≥ 1
        from x have x2 < 1 by (auto simp: abs_square_less_1)
        also from k have ... ≤ of_nat k2 by simp
        finally have (1 - x2 / of_nat k2) ∈ {0..1} using k
          by (simp_all add: field_simps del: of_nat_Suc)
      }
      hence (∏ k=1..n. abs (1 - x2 / of_nat k2)) ≤ (∏ k=1..n. 1) by (intro
        prod_mono) simp
      thus norm (P x n / (of_nat (Suc n)2)) ≤ 1 / of_nat (Suc n)2
        unfolding P_def by (simp add: field_simps abs_prod del: of_nat_Suc)
    qed (subst summable_Suc_iff, insert inverse_power_summable[of 2], simp
      add: inverse_eq_divide)
    qed (auto simp: P_def intro!: continuous_intros)
    hence isCont f 0 by (subst (asm) continuous_on_eq_continuous_at) simp_all
    hence (f - 0 → f 0) by (simp add: isCont_def)
    also have f 0 = K unfolding f_def P_def K_def by (simp add: inverse_eq_divide
      power_divide)
    finally have f - 0 → K .

  moreover have f - 0 → pi2 / 6
  proof (rule Lim_transform_eventually)
    define f' where [abs_def]: f' x = (∑ n. - sin_coeff (n+3) * pi ^ (n+2) *
      xn) for x
    have eventually (λx. x ≠ (0::real)) (at 0)

```

```

    by (auto simp add: eventually_at intro!: exI[of _ 1])
  thus eventually ( $\lambda x. f' x = f x$ ) (at 0)
proof eventually_elim
  fix x :: real assume x:  $x \neq 0$ 
  have sin_coeff 1 = (1 :: real) sin_coeff 2 = (0 :: real) by (simp_all add:
sin_coeff_def)
  with sums_split_initial_segment[OF sums_minus[OF sin_converges], of 3
pi*x]
  have ( $\lambda n. - (sin\_coeff (n+3) * (pi*x)^{(n+3)})$ ) sums ( $pi * x - sin (pi*x)$ )
  by (simp add: eval_nat_numeral)
  from sums_divide[OF this, of  $x^3 * pi$ ] x
  have ( $\lambda n. - (sin\_coeff (n+3) * pi^{(n+2)} * x^n)$ ) sums ((1 - sin (pi*x)
/ (pi*x)) /  $x^2$ )
  by (simp add: field_split_simps eval_nat_numeral)
  with x have ( $\lambda n. - (sin\_coeff (n+3) * pi^{(n+2)} * x^n)$ ) sums ( $g x / x^2$ )
  by (simp add: g_def)
  hence  $f' x = g x / x^2$  by (simp add: sums_iff f'_def)
  also have ... =  $f x$  using sums[of x] x by (simp add: sums_iff g_def f_def)
  finally show  $f' x = f x$  .
qed

have isCont f' 0 unfolding f'_def
proof (intro isCont_powser_converges_everywhere)
  fix x :: real show summable ( $\lambda n. -sin\_coeff (n+3) * pi^{(n+2)} * x^n$ )
  proof (cases x = 0)
    assume x:  $x \neq 0$ 
    from summable_divide[OF sums_summable[OF sums_split_initial_segment[OF
sin_converges[of pi*x]], of 3], of  $-pi*x^3$ ] x
    show ?thesis by (simp add: field_split_simps eval_nat_numeral)
  qed (simp only: summable_0_powser)
qed
hence  $f' - 0 \rightarrow f' 0$  by (simp add: isCont_def)
also have  $f' 0 = pi * pi / fact 3$  unfolding f'_def
  by (subst powser_zero) (simp add: sin_coeff_def)
  finally show  $f' - 0 \rightarrow pi^2 / 6$  by (simp add: eval_nat_numeral)
qed

ultimately have  $K = pi^2 / 6$  by (rule LIM_unique)
moreover from inverse_power_summable[of 2]
  have summable ( $\lambda n. (inverse (real_of_nat (Suc n)))^2$ )
  by (subst summable_Suc_iff) (simp add: power_inverse)
ultimately show ?thesis unfolding K_def
  by (auto simp add: sums_iff power_divide inverse_eq_divide)
qed

end

```

theory Interval_Integral

imports *Equivalence_Lebesgue_Henstock_Integration*
begin

definition *einterval* $a\ b = \{x. a < \text{ereal } x \wedge \text{ereal } x < b\}$

lemma *einterval_eq[simp]*:

shows *einterval_eq_Icc*: $\text{einterval } (\text{ereal } a) (\text{ereal } b) = \{a <..< b\}$
and *einterval_eq_Ici*: $\text{einterval } (\text{ereal } a) \infty = \{a <..\}$
and *einterval_eq_Iic*: $\text{einterval } (-\infty) (\text{ereal } b) = \{..< b\}$
and *einterval_eq_UNIV*: $\text{einterval } (-\infty) \infty = \text{UNIV}$
by (*auto simp: einterval_def*)

lemma *einterval_same*: $\text{einterval } a\ a = \{\}$

by (*auto simp: einterval_def*)

lemma *einterval_iff*: $x \in \text{einterval } a\ b \longleftrightarrow a < \text{ereal } x \wedge \text{ereal } x < b$

by (*simp add: einterval_def*)

lemma *einterval_nonempty*: $a < b \implies \exists c. c \in \text{einterval } a\ b$

by (*cases a b rule: ereal2_cases, auto simp: einterval_def intro!: dense_gt_ex lt_ex*)

lemma *open_einterval[simp]*: $\text{open } (\text{einterval } a\ b)$

by (*cases a b rule: ereal2_cases*)

(*auto simp: einterval_def intro!: open_Collect_conj open_Collect_less continuous_intros*)

lemma *borel_einterval[measurable]*: $\text{einterval } a\ b \in \text{sets borel}$

unfolding *einterval_def* **by** *measurable*

6.23.14 Approximating a (possibly infinite) interval

lemma *filterlim_sup1*: $(\text{LIM } x\ F. f\ x :> G1) \implies (\text{LIM } x\ F. f\ x :> (\sup G1\ G2))$

unfolding *filterlim_def* **by** (*auto intro: le_supI1*)

lemma *ereal_incseq_approx*:

fixes $a\ b :: \text{ereal}$

assumes $a < b$

obtains $X :: \text{nat} \Rightarrow \text{real}$ **where** $\text{incseq } X \wedge i. a < X\ i \wedge i. X\ i < b \implies X \longrightarrow b$

proof (*cases b*)

case *PInf*

with $\langle a < b \rangle$ **have** $a = -\infty \vee (\exists r. a = \text{ereal } r)$

by (*cases a*) *auto*

moreover **have** $(\lambda x. \text{ereal } (\text{real } (\text{Suc } x))) \longrightarrow \infty$

by (*simp add: Lim_PInfthy filterlim_sequentially_Suc*) (*metis le_SucI of_nat_Suc of_nat_mono order_trans real_arch_simple*)

moreover **have** $\bigwedge r. (\lambda x. \text{ereal } (r + \text{real } (\text{Suc } x))) \longrightarrow \infty$

by (*simp add: filterlim_sequentially_Suc Lim_PInfthy*) (*metis add commute diff_le_eq nat_ceiling_le_eq*)

```

ultimately show thesis
  by (intro that[of  $\lambda i. \text{real\_of\_ereal } a + \text{Suc } i$ ])
    (auto simp: incseq_def PInf)
next
case (real b')
define d where  $d = b' - (\text{if } a = -\infty \text{ then } b' - 1 \text{ else } \text{real\_of\_ereal } a)$ 
with  $\langle a < b \rangle$  have  $a' : 0 < d$ 
  by (cases a) (auto simp: real)
moreover
have  $\bigwedge i r. r < b' \implies (b' - r) * 1 < (b' - r) * \text{real } (\text{Suc } (\text{Suc } i))$ 
  by (intro mult_strict_left_mono) auto
with  $\langle a < b \rangle$  a' have  $\bigwedge i. a < \text{ereal } (b' - d / \text{real } (\text{Suc } (\text{Suc } i)))$ 
  by (cases a) (auto simp: real d_def field_simps)
moreover
have  $(\lambda i. b' - d / \text{real } i) \longrightarrow b'$ 
  by (force intro: tendsto_eq_intros tendsto_divide_0[OF tendsto_const] filter-
lim_sup1
      simp: at_infinity_eq_at_top_bot filterlim_real_sequentially)
then have  $(\lambda i. b' - d / \text{Suc } (\text{Suc } i)) \longrightarrow b'$ 
  by (blast intro: dest: filterlim_sequentially_Suc [THEN iffD2])
ultimately show thesis
  by (intro that[of  $\lambda i. b' - d / \text{Suc } (\text{Suc } i)$ ])
    (auto simp: real incseq_def intro!: divide_left_mono)
qed (use  $\langle a < b \rangle$  in auto)

lemma ereal_decseq_approx:
  fixes a b :: ereal
  assumes a < b
  obtains X :: nat  $\Rightarrow$  real where
    decseq X  $\bigwedge i. a < X i \bigwedge i. X i < b$   $X \longrightarrow a$ 
proof -
  have  $-b < -a$  using  $\langle a < b \rangle$  by simp
  from ereal_incseq_approx[OF this] obtain X where
    incseq X
     $\bigwedge i. -b < \text{ereal } (X i)$ 
     $\bigwedge i. \text{ereal } (X i) < -a$ 
     $(\lambda x. \text{ereal } (X x)) \longrightarrow -a$ 
  by auto
  then show thesis
    apply (intro that[of  $\lambda i. -X i$ ])
    apply (auto simp: decseq_def incseq_def simp flip: uminus_ereal_simps)
    apply (metis ereal_minus_less_minus ereal_uminus_uminus ereal_Lim_uminus)+
  done
qed

proposition einterval_Icc_approximation:
  fixes a b :: ereal
  assumes a < b
  obtains u l :: nat  $\Rightarrow$  real where

```

```

    einterval a b = ( $\bigcup i. \{l\ i \ ..\ u\ i\}$ )
    incseq u decseq l  $\bigwedge i. l\ i < u\ i \bigwedge i. a < l\ i \bigwedge i. u\ i < b$ 
    l  $\longrightarrow$  a u  $\longrightarrow$  b
  proof -
    from dense[OF  $\langle a < b \rangle$ ] obtain c where a < c c < b by safe
    from ereal_incseq_approx[OF  $\langle c < b \rangle$ ] obtain u where u:
      incseq u
       $\bigwedge i. c < ereal\ (u\ i)$ 
       $\bigwedge i. ereal\ (u\ i) < b$ 
       $(\lambda x. ereal\ (u\ x)) \longrightarrow b$ 
    by auto
    from ereal_decseq_approx[OF  $\langle a < c \rangle$ ] obtain l where l:
      decseq l
       $\bigwedge i. a < ereal\ (l\ i)$ 
       $\bigwedge i. ereal\ (l\ i) < c$ 
       $(\lambda x. ereal\ (l\ x)) \longrightarrow a$ 
    by auto
    { fix i from less_trans[OF  $\langle l\ i < c \rangle \langle c < u\ i \rangle$ ] have l i < u i by simp }
    have einterval a b = ( $\bigcup i. \{l\ i \ ..\ u\ i\}$ )
  proof (auto simp: einterval_iff)
    fix x assume a < ereal x ereal x < b
    have eventually  $(\lambda i. ereal\ (l\ i) < ereal\ x)$  sequentially
      using l(4)  $\langle a < ereal\ x \rangle$  by (rule order_tendstoD)
    moreover
    have eventually  $(\lambda i. ereal\ x < ereal\ (u\ i))$  sequentially
      using u(4)  $\langle ereal\ x < b \rangle$  by (rule order_tendstoD)
    ultimately have eventually  $(\lambda i. l\ i < x \wedge x < u\ i)$  sequentially
      by eventually_elim auto
    then show  $\exists i. l\ i \leq x \wedge x \leq u\ i$ 
      by (auto intro: less_imp_le simp: eventually_sequentially)
  next
    fix x i assume l i  $\leq$  x x  $\leq$  u i
    with  $\langle a < ereal\ (l\ i) \rangle \langle ereal\ (u\ i) < b \rangle$ 
    show a < ereal x ereal x < b
      by (auto simp flip: ereal_less_eq(3))
  qed
  show thesis
    by (intro that) fact+
qed

```

definition interval_lebesgue_integral :: real measure $\Rightarrow$ ereal $\Rightarrow$ ereal $\Rightarrow$ (real $\Rightarrow$ 'a) $\Rightarrow$ 'a::{banach, second_countable_topology} **where**

interval_lebesgue_integral M a b f =

(if a $\leq$ b then (LINT x:einterval a b |M. f x) else - (LINT x:einterval b a |M. f x))

syntax

_ascii_interval_lebesgue_integral :: ptrn $\Rightarrow$ real $\Rightarrow$ real $\Rightarrow$ real measure $\Rightarrow$ real

$\Rightarrow \text{real}$
 $((5LINT \_ = \_ .. \_ | \_ . \_) [0,60,60,61,100] 60)$

translations

$LINT \ x = a..b | M. f == CONST \ interval\_lebesgue\_integral \ M \ a \ b \ (\lambda x. f)$

definition $interval\_lebesgue\_integrable :: real \ measure \Rightarrow ereal \Rightarrow ereal \Rightarrow (real \Rightarrow 'a :: \{banach, second\_countable\_topology\}) \Rightarrow bool$ **where**

$interval\_lebesgue\_integrable \ M \ a \ b \ f =$
 $(if \ a \leq b \ then \ set\_integrable \ M \ (einterval \ a \ b) \ f \ else \ set\_integrable \ M \ (einterval \ b \ a) \ f)$

syntax

$\_ascii\_interval\_lebesgue\_borel\_integral :: ptrn \Rightarrow real \Rightarrow real \Rightarrow real \Rightarrow real$
 $((4LBINT \_ = \_ .. \_ . \_) [0,60,60,61] 60)$

translations

$LBINT \ x = a..b. f == CONST \ interval\_lebesgue\_integral \ CONST \ lborel \ a \ b \ (\lambda x. f)$

6.23.15 Basic properties of integration over an interval

lemma $interval\_lebesgue\_integral\_cong$:

$a \leq b \Longrightarrow (\bigwedge x. x \in einterval \ a \ b \Longrightarrow f \ x = g \ x) \Longrightarrow einterval \ a \ b \in sets \ M \Longrightarrow$
 $interval\_lebesgue\_integral \ M \ a \ b \ f = interval\_lebesgue\_integral \ M \ a \ b \ g$
by (auto intro: set_lebesgue_integral_cong simp: interval_lebesgue_integral_def)

lemma $interval\_lebesgue\_integral\_cong\_AE$:

$f \in borel\_measurable \ M \Longrightarrow g \in borel\_measurable \ M \Longrightarrow$
 $a \leq b \Longrightarrow AE \ x \in einterval \ a \ b \ in \ M. f \ x = g \ x \Longrightarrow einterval \ a \ b \in sets \ M \Longrightarrow$
 $interval\_lebesgue\_integral \ M \ a \ b \ f = interval\_lebesgue\_integral \ M \ a \ b \ g$
by (auto intro: set_lebesgue_integral_cong_AE simp: interval_lebesgue_integral_def)

lemma $interval\_integrable\_mirror$:

shows $interval\_lebesgue\_integrable \ lborel \ a \ b \ (\lambda x. f \ (-x)) \longleftrightarrow$
 $interval\_lebesgue\_integrable \ lborel \ (-b) \ (-a) \ f$

proof –

have *: $indicator \ (einterval \ a \ b) \ (-x) = (indicator \ (einterval \ (-b) \ (-a)) \ x :: real)$

for $a \ b :: ereal$ **and** $x :: real$

by (cases $a \ b$ rule: ereal2_cases) (auto simp: einterval_def split: split_indicator)

show ?thesis

unfolding $interval\_lebesgue\_integrable\_def$

using $lborel\_integrable\_real\_affine\_iff[symmetric, of \ -1 \ \lambda x. indicator \ (einterval \ \_ \_) \ x \ *_R \ f \ x \ 0]$

by (simp add: * set_integrable_def)

qed

lemma $interval\_lebesgue\_integral\_add$ [intro, simp]:


```

fixes  $M$   $a$   $b$   $f$ 
assumes  $\text{interval\_lebesgue\_integrable } M$   $a$   $b$   $f$   $\text{interval\_lebesgue\_integrable } M$   $a$ 
 $b$   $g$ 
shows  $\text{interval\_lebesgue\_integrable } M$   $a$   $b$   $(\lambda x. f\ x + g\ x)$  and
 $\text{interval\_lebesgue\_integral } M$   $a$   $b$   $(\lambda x. f\ x + g\ x) =$ 
 $\text{interval\_lebesgue\_integral } M$   $a$   $b$   $f + \text{interval\_lebesgue\_integral } M$   $a$   $b$   $g$ 
using  $\text{assms}$  by ( $\text{auto simp: interval\_lebesgue\_integral\_def interval\_lebesgue\_integrable\_def}$ 
 $\text{field\_simps}$ )

```

```

lemma  $\text{interval\_lebesgue\_integral\_diff}$  [ $\text{intro}$ ,  $\text{simp}$ ]:
fixes  $M$   $a$   $b$   $f$ 
assumes  $\text{interval\_lebesgue\_integrable } M$   $a$   $b$   $f$ 
 $\text{interval\_lebesgue\_integrable } M$   $a$   $b$   $g$ 
shows  $\text{interval\_lebesgue\_integrable } M$   $a$   $b$   $(\lambda x. f\ x - g\ x)$  and
 $\text{interval\_lebesgue\_integral } M$   $a$   $b$   $(\lambda x. f\ x - g\ x) =$ 
 $\text{interval\_lebesgue\_integral } M$   $a$   $b$   $f - \text{interval\_lebesgue\_integral } M$   $a$   $b$   $g$ 
using  $\text{assms}$  by ( $\text{auto simp: interval\_lebesgue\_integral\_def interval\_lebesgue\_integrable\_def}$ 
 $\text{field\_simps}$ )

```

```

lemma  $\text{interval\_lebesgue\_integrable\_mult\_right}$  [ $\text{intro}$ ,  $\text{simp}$ ]:
fixes  $M$   $a$   $b$   $c$  and  $f :: \text{real} \Rightarrow 'a :: \{\text{banach, real\_normed\_field, second\_countable\_topology}\}$ 
shows  $(c \neq 0 \Rightarrow \text{interval\_lebesgue\_integrable } M$   $a$   $b$   $f) \Rightarrow$ 
 $\text{interval\_lebesgue\_integrable } M$   $a$   $b$   $(\lambda x. c * f\ x)$ 
by ( $\text{simp add: interval\_lebesgue\_integrable\_def}$ )

```

```

lemma  $\text{interval\_lebesgue\_integrable\_mult\_left}$  [ $\text{intro}$ ,  $\text{simp}$ ]:
fixes  $M$   $a$   $b$   $c$  and  $f :: \text{real} \Rightarrow 'a :: \{\text{banach, real\_normed\_field, second\_countable\_topology}\}$ 
shows  $(c \neq 0 \Rightarrow \text{interval\_lebesgue\_integrable } M$   $a$   $b$   $f) \Rightarrow$ 
 $\text{interval\_lebesgue\_integrable } M$   $a$   $b$   $(\lambda x. f\ x * c)$ 
by ( $\text{simp add: interval\_lebesgue\_integrable\_def}$ )

```

```

lemma  $\text{interval\_lebesgue\_integrable\_divide}$  [ $\text{intro}$ ,  $\text{simp}$ ]:
fixes  $M$   $a$   $b$   $c$  and  $f :: \text{real} \Rightarrow 'a :: \{\text{banach, real\_normed\_field, field, second\_countable\_topology}\}$ 
shows  $(c \neq 0 \Rightarrow \text{interval\_lebesgue\_integrable } M$   $a$   $b$   $f) \Rightarrow$ 
 $\text{interval\_lebesgue\_integrable } M$   $a$   $b$   $(\lambda x. f\ x / c)$ 
by ( $\text{simp add: interval\_lebesgue\_integrable\_def}$ )

```

```

lemma  $\text{interval\_lebesgue\_integral\_mult\_right}$  [ $\text{simp}$ ]:
fixes  $M$   $a$   $b$   $c$  and  $f :: \text{real} \Rightarrow 'a :: \{\text{banach, real\_normed\_field, second\_countable\_topology}\}$ 
shows  $\text{interval\_lebesgue\_integral } M$   $a$   $b$   $(\lambda x. c * f\ x) =$ 
 $c * \text{interval\_lebesgue\_integral } M$   $a$   $b$   $f$ 
by ( $\text{simp add: interval\_lebesgue\_integral\_def}$ )

```

```

lemma  $\text{interval\_lebesgue\_integral\_mult\_left}$  [ $\text{simp}$ ]:
fixes  $M$   $a$   $b$   $c$  and  $f :: \text{real} \Rightarrow 'a :: \{\text{banach, real\_normed\_field, second\_countable\_topology}\}$ 
shows  $\text{interval\_lebesgue\_integral } M$   $a$   $b$   $(\lambda x. f\ x * c) =$ 
 $\text{interval\_lebesgue\_integral } M$   $a$   $b$   $f * c$ 
by ( $\text{simp add: interval\_lebesgue\_integral\_def}$ )

```

lemma *interval_lebesgue_integral_divide* [simp]:
fixes M a b c **and** $f :: \text{real} \Rightarrow 'a :: \{\text{banach}, \text{real\_normed\_field}, \text{field}, \text{second\_countable\_topology}\}$
shows $\text{interval\_lebesgue\_integral } M \ a \ b \ (\lambda x. f \ x / c) =$
 $\text{interval\_lebesgue\_integral } M \ a \ b \ f / c$
by (simp add: interval_lebesgue_integral_def)

lemma *interval_lebesgue_integral_uminus*:
 $\text{interval\_lebesgue\_integral } M \ a \ b \ (\lambda x. - f \ x) = - \text{interval\_lebesgue\_integral } M$
 $a \ b \ f$
by (auto simp: interval_lebesgue_integral_def interval_lebesgue_integrable_def
set_lebesgue_integral_def)

lemma *interval_lebesgue_integral_of_real*:
 $\text{interval\_lebesgue\_integral } M \ a \ b \ (\lambda x. \text{complex\_of\_real } (f \ x)) =$
 $\text{of\_real } (\text{interval\_lebesgue\_integral } M \ a \ b \ f)$
unfolding interval_lebesgue_integral_def
by (auto simp: interval_lebesgue_integral_def set_integral_complex_of_real)

lemma *interval_lebesgue_integral_le_eq*:
fixes a b f
assumes $a \leq b$
shows $\text{interval\_lebesgue\_integral } M \ a \ b \ f = (\text{LINT } x : \text{einterval } a \ b \mid M. f \ x)$
using assms **by** (auto simp: interval_lebesgue_integral_def)

lemma *interval_lebesgue_integral_gt_eq*:
fixes a b f
assumes $a > b$
shows $\text{interval\_lebesgue\_integral } M \ a \ b \ f = -(\text{LINT } x : \text{einterval } b \ a \mid M. f \ x)$
using assms **by** (auto simp: interval_lebesgue_integral_def less_imp_le einterval_def)

lemma *interval_lebesgue_integral_gt_eq'*:
fixes a b f
assumes $a > b$
shows $\text{interval\_lebesgue\_integral } M \ a \ b \ f = - \text{interval\_lebesgue\_integral } M \ b \ a \ f$
using assms **by** (auto simp: interval_lebesgue_integral_def less_imp_le einterval_def)

lemma *interval_integral_endpoints_same* [simp]: $(\text{LBINT } x=a..a. f \ x) = 0$
by (simp add: interval_lebesgue_integral_def set_lebesgue_integral_def einterval_same)

lemma *interval_integral_endpoints_reverse*: $(\text{LBINT } x=a..b. f \ x) = -(\text{LBINT } x=b..a. f \ x)$
by (cases $a \ b$ rule: linorder_cases) (auto simp: interval_lebesgue_integral_def
set_lebesgue_integral_def einterval_same)

lemma *interval_integrable_endpoints_reverse*:

```

interval_lebesgue_integrable lborel a b f  $\longleftrightarrow$ 
  interval_lebesgue_integrable lborel b a f
by (cases a b rule: linorder_cases) (auto simp: interval_lebesgue_integrable_def
einterval_same)

lemma interval_integral_reflect:
  (LBINT x=a..b. f x) = (LBINT x=-b..-a. f (-x))
proof (induct a b rule: linorder_wlog)
  case (sym a b) then show ?case
    by (auto simp: interval_lebesgue_integral_def interval_integrable_endpoints_reverse
      split: if_split_asm)
next
  case (le a b)
    have LBINT x:{x. - x  $\in$  einterval a b}. f (- x) = LBINT x:einterval (- b) (-
a). f (- x)
    unfolding interval_lebesgue_integrable_def set_lebesgue_integral_def
    apply (rule Bochner_Integration.integral_cong [OF refl])
    by (auto simp: einterval_iff ereal_uminus_le_reorder ereal_uminus_less_reorder
not_less
      simp flip: uminus_ereal.simps
      split: split_indicator)
    then show ?case
      unfolding interval_lebesgue_integral_def
      by (subst set_integral_reflect) (simp add: le)
qed

```

```

lemma interval_lebesgue_integral_0_infty:
  interval_lebesgue_integrable M 0  $\infty$  f  $\longleftrightarrow$  set_integrable M {0<..} f
  interval_lebesgue_integral M 0  $\infty$  f = (LINT x:{0<..}|M. f x)
  unfolding zero_ereal_def
  by (auto simp: interval_lebesgue_integral_le_eq interval_lebesgue_integrable_def)

```

```

lemma interval_integral_to_infinity_eq: (LINT x=ereal a.. $\infty$  | M. f x) = (LINT
x : {a<..} | M. f x)
  unfolding interval_lebesgue_integral_def by auto

```

```

proposition interval_integrable_to_infinity_eq: (interval_lebesgue_integrable M
a  $\infty$  f) =
  (set_integrable M {a<..} f)
  unfolding interval_lebesgue_integrable_def by auto

```

6.23.16 Basic properties of integration over an interval wrt lebesgue measure

```

lemma interval_integral_zero [simp]:
  fixes a b :: ereal
  shows LBINT x=a..b. 0 = 0
  unfolding interval_lebesgue_integral_def set_lebesgue_integral_def einterval_eq
by simp

```

```

lemma interval_integral_const [intro, simp]:
  fixes a b c :: real
  shows interval_lebesgue_integrable lborel a b ( $\lambda x. c$ ) and LBINT x=a..b. c = c
  * (b - a)
  unfolding interval_lebesgue_integral_def interval_lebesgue_integrable_def interval_eq
  by (auto simp: less_imp_le field_simps measure_def set_integrable_def set_lebesgue_integral_def)

```

```

lemma interval_integral_cong_AE:
  assumes [measurable]: f ∈ borel_measurable lborel g ∈ borel_measurable lborel
  assumes AE x ∈ einterval (min a b) (max a b) in lborel. f x = g x
  shows interval_lebesgue_integral lborel a b f = interval_lebesgue_integral lborel
  a b g
  using assms
proof (induct a b rule: linorder_wlog)
  case (sym a b) then show ?case
    by (simp add: min.commute max.commute interval_integral_endpoints_reverse[of
  a b])
  next
  case (le a b) then show ?case
    by (auto simp: interval_lebesgue_integral_def max_def min_def
      intro!: set_lebesgue_integral_cong_AE)
qed

```

```

lemma interval_integral_cong:
  assumes  $\bigwedge x. x \in \text{einterval } (\text{min } a \ b) \ (\text{max } a \ b) \implies f \ x = g \ x$ 
  shows interval_lebesgue_integral lborel a b f = interval_lebesgue_integral lborel
  a b g
  using assms
proof (induct a b rule: linorder_wlog)
  case (sym a b) then show ?case
    by (simp add: min.commute max.commute interval_integral_endpoints_reverse[of
  a b])
  next
  case (le a b) then show ?case
    by (auto simp: interval_lebesgue_integral_def max_def min_def
      intro!: set_lebesgue_integral_cong)
qed

```

```

lemma interval_lebesgue_integrable_cong_AE:
  f ∈ borel_measurable lborel  $\implies$  g ∈ borel_measurable lborel  $\implies$ 
  AE x ∈ einterval (min a b) (max a b) in lborel. f x = g x  $\implies$ 
  interval_lebesgue_integrable lborel a b f = interval_lebesgue_integrable lborel a
  b g
  apply (simp add: interval_lebesgue_integrable_def)
  apply (intro conjI impI set_integrable_cong_AE)
  apply (auto simp: min_def max_def)
  done

```

lemma *interval_integrable_abs_iff*:
fixes $f :: \text{real} \Rightarrow \text{real}$
shows $f \in \text{borel\_measurable lborel} \implies$
 $\text{interval\_lebesgue\_integrable lborel } a \ b \ (\lambda x. |f x|) = \text{interval\_lebesgue\_integrable}$
 $\text{lborel } a \ b \ f$
unfolding *interval_lebesgue_integrable_def*
by (*subst (1 2) set_integrable_abs_iff'*) *simp_all*

lemma *interval_integral_Icc*:
fixes $a \ b :: \text{real}$
shows $a \leq b \implies (\text{LBINT } x=a..b. f x) = (\text{LBINT } x : \{a..b\}. f x)$
by (*auto intro!: set_integral_discrete_difference[where X={a, b}]*)
simp add: interval_lebesgue_integral_def)

lemma *interval_integral_Icc'*:
 $a \leq b \implies (\text{LBINT } x=a..b. f x) = (\text{LBINT } x : \{x. a \leq \text{ereal } x \wedge \text{ereal } x \leq b\}. f$
 $x)$
by (*auto intro!: set_integral_discrete_difference[where X={real_of_ereal a,*
 $\text{real_of_ereal } b\}])
simp add: interval_lebesgue_integral_def einterval_iff)$

lemma *interval_integral_Ioc*:
 $a \leq b \implies (\text{LBINT } x=a..b. f x) = (\text{LBINT } x : \{a<..b\}. f x)$
by (*auto intro!: set_integral_discrete_difference[where X={a, b}]*)
simp add: interval_lebesgue_integral_def einterval_iff)

lemma *interval_integral_Ioc'*:
 $a \leq b \implies (\text{LBINT } x=a..b. f x) = (\text{LBINT } x : \{x. a < \text{ereal } x \wedge \text{ereal } x \leq b\}. f$
 $x)$
by (*auto intro!: set_integral_discrete_difference[where X={real_of_ereal a,*
 $\text{real_of_ereal } b\}])
simp add: interval_lebesgue_integral_def einterval_iff)$

lemma *interval_integral_Ico*:
 $a \leq b \implies (\text{LBINT } x=a..b. f x) = (\text{LBINT } x : \{a..<b\}. f x)$
by (*auto intro!: set_integral_discrete_difference[where X={a, b}]*)
simp add: interval_lebesgue_integral_def einterval_iff)

lemma *interval_integral_Ioi*:
 $|a| < \infty \implies (\text{LBINT } x=a..\infty. f x) = (\text{LBINT } x : \{\text{real\_of\_ereal } a <..\}. f x)$
by (*auto simp: interval_lebesgue_integral_def einterval_iff*)

lemma *interval_integral_Ioo*:
 $a \leq b \implies |a| < \infty \implies |b| < \infty \implies (\text{LBINT } x=a..b. f x) = (\text{LBINT } x :$
 $\{\text{real_of_ereal } a <.. $\text{real\_of\_ereal } b\}. f x)$
by (*auto simp: interval_lebesgue_integral_def einterval_iff*)$

```

lemma interval_integral_discrete_difference:
  fixes f :: real  $\Rightarrow$  'b::{banach, second_countable_topology} and a b :: ereal
  assumes countable X
  and eq:  $\bigwedge x. a \leq b \implies a < x \implies x < b \implies x \notin X \implies f x = g x$ 
  and anti_eq:  $\bigwedge x. b \leq a \implies b < x \implies x < a \implies x \notin X \implies f x = g x$ 
  assumes  $\bigwedge x. x \in X \implies \text{emeasure } M \{x\} = 0$   $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$ 
  shows interval_lebesgue_integral M a b f = interval_lebesgue_integral M a b g
  unfolding interval_lebesgue_integral_def set_lebesgue_integral_def
  apply (intro if_cong refl arg_cong[where f= $\lambda x. - x$ ] integral_discrete_difference[of X] assms)
  apply (auto simp: eq anti_eq einterval_iff split: split_indicator)
  done

```

```

lemma interval_integral_sum:
  fixes a b c :: ereal
  assumes integrable: interval_lebesgue_integrable lborel (min a (min b c)) (max
a (max b c)) f
  shows (LBINT x=a..b. f x) + (LBINT x=b..c. f x) = (LBINT x=a..c. f x)
proof -
  let ?I =  $\lambda a b. \text{LBINT } x=a..b. f x$ 
  { fix a b c :: ereal assume interval_lebesgue_integrable lborel a c f a  $\leq$  b b  $\leq$  c
    then have ord: a  $\leq$  b b  $\leq$  c a  $\leq$  c and f': set_integrable lborel (einterval a c)
    f
    by (auto simp: interval_lebesgue_integrable_def)
    then have f: set_borel_measurable borel (einterval a c) f
    unfolding set_integrable_def set_borel_measurable_def
    by (drule_tac borel_measurable_integrable) simp
    have (LBINT x:einterval a c. f x) = (LBINT x:einterval a b  $\cup$  einterval b c. f
x)
    proof (rule set_integral_cong_set)
      show AE x in lborel. (x  $\in$  einterval a b  $\cup$  einterval b c) = (x  $\in$  einterval a c)
      using AE_lborel_singleton[of real_of_ereal b] ord
      by (cases a b c rule: ereal3_cases) (auto simp: einterval_iff)
      show set_borel_measurable lborel (einterval a c) f set_borel_measurable lborel
(einterval a b  $\cup$  einterval b c) f
      unfolding set_borel_measurable_def
      using ord by (auto simp: einterval_iff intro!: set_borel_measurable_subset[OF
f, unfolded set_borel_measurable_def])
    qed
    also have ... = (LBINT x:einterval a b. f x) + (LBINT x:einterval b c. f x)
    using ord
    by (intro set_integral_Un_AE) (auto intro!: set_integrable_subset[OF f]
simp: einterval_iff not_less)
    finally have ?I a b + ?I b c = ?I a c
    using ord by (simp add: interval_lebesgue_integral_def)
  } note 1 = this
  { fix a b c :: ereal assume interval_lebesgue_integrable lborel a c f a  $\leq$  b b  $\leq$  c
    from 1[OF this] have ?I b c + ?I a b = ?I a c
    by (metis add.commute)
  }

```

```

} note 2 = this
have 3:  $\bigwedge a b. b \leq a \implies (LBINT\ x=a..b. f\ x) = - (LBINT\ x=b..a. f\ x)$ 
  by (rule interval_integral_endpoints_reverse)
show ?thesis
  using integrable
  apply (cases a b b c a c rule: linorder_le_cases[case_product linorder_le_cases
linorder_cases])
  apply simp_all
  by (simp_all add: min_absorb1 min_absorb2 max_absorb1 max_absorb2 field_simps
1 2 3)
qed

lemma interval_integrable_isCont:
  fixes a b and f :: real  $\Rightarrow$  'a::{banach, second_countable_topology}
  shows ( $\bigwedge x. \min a b \leq x \implies x \leq \max a b \implies isCont\ f\ x$ )  $\implies$ 
    interval_lebesgue_integrable lborel a b f
proof (induct a b rule: linorder_wlog)
  case (le a b) then show ?case
    unfolding interval_lebesgue_integrable_def set_integrable_def
    by (auto simp: interval_lebesgue_integrable_def
      intro!: set_integrable_subset[unfolded set_integrable_def, OF borel_integrable_compact[of
{a .. b}]]
      continuous_at_imp_continuous_on)
qed (auto intro: interval_integrable_endpoints_reverse[THEN iffD1])

lemma interval_integrable_continuous_on:
  fixes a b :: real and f
  assumes  $a \leq b$  and continuous_on {a..b} f
  shows interval_lebesgue_integrable lborel a b f
using assms unfolding interval_lebesgue_integrable_def apply simp
  by (rule set_integrable_subset, rule borel_integrable_atLeastAtMost' [of a b],
auto)

lemma interval_integral_eq_integral:
  fixes f :: real  $\Rightarrow$  'a::euclidean_space
  shows  $a \leq b \implies set\_integrable\ lborel\ \{a..b\}\ f \implies LBINT\ x=a..b. f\ x = integral\ \{a..b\}\ f$ 
  by (subst interval_integral_Icc, simp) (rule set_borel_integral_eq_integral)

lemma interval_integral_eq_integral':
  fixes f :: real  $\Rightarrow$  'a::euclidean_space
  shows  $a \leq b \implies set\_integrable\ lborel\ (einterval\ a\ b)\ f \implies LBINT\ x=a..b. f\ x = integral\ (einterval\ a\ b)\ f$ 
  by (subst interval_lebesgue_integral_le_eq, simp) (rule set_borel_integral_eq_integral)

```

6.23.17 General limit approximation arguments

proposition *interval_integral_Icc_approx_nonneg*:
 fixes a b :: ereal

```

assumes  $a < b$ 
fixes  $u \, l :: \text{nat} \Rightarrow \text{real}$ 
assumes  $\text{approx}: \text{einterval } a \, b = (\bigcup i. \{l \, i .. u \, i\})$ 
 $\text{incseq } u \, \text{decseq } l \wedge i. l \, i < u \, i \wedge i. a < l \, i \wedge i. u \, i < b$ 
 $l \longrightarrow a \, u \longrightarrow b$ 
fixes  $f :: \text{real} \Rightarrow \text{real}$ 
assumes  $f\_integrable: \wedge i. \text{set\_integrable } \text{lborel } \{l \, i .. u \, i\} \, f$ 
assumes  $f\_nonneg: AE \, x \text{ in } \text{lborel}. a < \text{ereal } x \longrightarrow \text{ereal } x < b \longrightarrow 0 \leq f \, x$ 
assumes  $f\_measurable: \text{set\_borel\_measurable } \text{lborel } (\text{einterval } a \, b) \, f$ 
assumes  $\text{lbint\_lim}: (\lambda i. \text{LBINT } x=l \, i .. u \, i. f \, x) \longrightarrow C$ 
shows
 $\text{set\_integrable } \text{lborel } (\text{einterval } a \, b) \, f$ 
 $(\text{LBINT } x=a .. b. f \, x) = C$ 
proof -
  have 1  $[\text{unfolded } \text{set\_integrable\_def}]: \wedge i. \text{set\_integrable } \text{lborel } \{l \, i .. u \, i\} \, f$  by
   $(\text{rule } f\_integrable)$ 
  have 2:  $AE \, x \text{ in } \text{lborel}. \text{mono } (\lambda n. \text{indicator } \{l \, n .. u \, n\} \, x *_R f \, x)$ 
  proof -
    from  $f\_nonneg$  have  $AE \, x \text{ in } \text{lborel}. \forall i. l \, i \leq x \longrightarrow x \leq u \, i \longrightarrow 0 \leq f \, x$ 
    by  $\text{eventually\_elim}$ 
     $(\text{metis } \text{approx}(5) \, \text{approx}(6) \, \text{dual\_order.strict\_trans1 } \text{ereal\_less\_eq}(3) \, \text{le\_less\_trans})$ 
    then show  $?thesis$ 
    apply  $\text{eventually\_elim}$ 
    apply  $(\text{auto simp: mono\_def split: split\_indicator})$ 
    apply  $(\text{metis } \text{approx}(3) \, \text{decseqD } \text{order\_trans})$ 
    apply  $(\text{metis } \text{approx}(2) \, \text{incseqD } \text{order\_trans})$ 
    done
  qed
  have 3:  $AE \, x \text{ in } \text{lborel}. (\lambda i. \text{indicator } \{l \, i .. u \, i\} \, x *_R f \, x) \longrightarrow \text{indicator}$ 
   $(\text{einterval } a \, b) \, x *_R f \, x$ 
  proof -
    { fix  $x \, i$  assume  $l \, i \leq x \leq u \, i$ 
      then have  $\text{eventually } (\lambda i. l \, i \leq x \wedge x \leq u \, i) \text{ sequentially}$ 
      apply  $(\text{auto simp: eventually\_sequentially\_intro!: exI[of\_ ]})$ 
      apply  $(\text{metis } \text{approx}(3) \, \text{decseqD } \text{order\_trans})$ 
      apply  $(\text{metis } \text{approx}(2) \, \text{incseqD } \text{order\_trans})$ 
      done
      then have  $\text{eventually } (\lambda i. f \, x * \text{indicator } \{l \, i .. u \, i\} \, x = f \, x) \text{ sequentially}$ 
      by  $\text{eventually\_elim auto}$  }
    then show  $?thesis$ 
    unfolding  $\text{approx}(1)$  by  $(\text{auto intro!: AE\_I2 tendsto\_eventually split: split\_indicator})$ 
  qed
  have 4:  $(\lambda i. \int x. \text{indicator } \{l \, i .. u \, i\} \, x *_R f \, x \, \partial \text{lborel}) \longrightarrow C$ 
  using  $\text{lbint\_lim}$  by  $(\text{simp add: interval\_integral\_Icc [unfolded set\_lebesgue\_integral\_def] approx\_less\_imp\_le})$ 
  have 5:  $(\lambda x. \text{indicat\_real } (\text{einterval } a \, b) \, x *_R f \, x) \in \text{borel\_measurable } \text{lborel}$ 
  using  $f\_measurable \text{ set\_borel\_measurable\_def}$  by  $\text{blast}$ 
  have  $(\text{LBINT } x=a .. b. f \, x) = \text{lebesgue\_integral } \text{lborel } (\lambda x. \text{indicator } (\text{einterval } a$ 

```



```

b)  $x *_R f x$ )
  using assms by (simp add: interval_lebesgue_integral_def set_lebesgue_integral_def
less_imp_le)
  also have ... = C
    by (rule integral_monotone_convergence [OF 1 2 3 4 5])
  finally show (LBINT  $x=a..b$ .  $f x$ ) = C .
  show set_integrable lborel (einterval a b) f
    unfolding set_integrable_def
    by (rule integrable_monotone_convergence [OF 1 2 3 4 5])
qed

proposition interval_integral_Icc_approx_integrable:
  fixes  $u l :: \text{nat} \Rightarrow \text{real}$  and  $a b :: \text{ereal}$ 
  fixes  $f :: \text{real} \Rightarrow 'a :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
  assumes  $a < b$ 
  assumes approx:  $\text{einterval } a \ b = (\bigcup i. \{l \ i .. u \ i\})$ 
    incseq u decseq l  $\wedge i. l \ i < u \ i \wedge i. a < l \ i \wedge i. u \ i < b$ 
     $l \longrightarrow a \ u \longrightarrow b$ 
  assumes f_integrable: set_integrable lborel (einterval a b) f
  shows  $(\lambda i. \text{LBINT } x=l \ i .. u \ i. f x) \longrightarrow (\text{LBINT } x=a..b. f x)$ 
proof -
  have  $(\lambda i. \text{LBINT } x:\{l \ i .. u \ i\}. f x) \longrightarrow (\text{LBINT } x:\text{einterval } a \ b. f x)$ 
    unfolding set_lebesgue_integral_def
  proof (rule integral_dominated_convergence)
    show integrable lborel  $(\lambda x. \text{norm } (\text{indicator } (\text{einterval } a \ b) \ x *_R f x))$ 
      using f_integrable integrable_norm set_integrable_def by blast
    show  $(\lambda x. \text{indicat\_real } (\text{einterval } a \ b) \ x *_R f x) \in \text{borel\_measurable lborel}$ 
      using f_integrable by (simp add: set_integrable_def)
    then show  $\bigwedge i. (\lambda x. \text{indicat\_real } \{l \ i .. u \ i\} \ x *_R f x) \in \text{borel\_measurable lborel}$ 
      by (rule set_borel_measurable_subset [unfolded set_borel_measurable_def])
  (auto simp: approx)
    show  $\bigwedge i. \text{AE } x \text{ in lborel. } \text{norm } (\text{indicator } \{l \ i .. u \ i\} \ x *_R f x) \leq \text{norm } (\text{indicator } (\text{einterval } a \ b) \ x *_R f x)$ 
      by (intro AE_I2) (auto simp: approx_split: split_indicator)
    show  $\text{AE } x \text{ in lborel. } (\lambda i. \text{indicator } \{l \ i .. u \ i\} \ x *_R f x) \longrightarrow \text{indicator } (\text{einterval } a \ b) \ x *_R f x$ 
proof (intro AE_I2 tendsto_intros tendsto_eventually)
    fix x
    { fix i assume  $l \ i \leq x \leq u \ i$ 
      with  $\langle \text{incseq } u \rangle [\text{THEN } \text{incseqD}, \text{ of } i] \ \langle \text{decseq } l \rangle [\text{THEN } \text{decseqD}, \text{ of } i]$ 
      have eventually  $(\lambda i. l \ i \leq x \wedge x \leq u \ i)$  sequentially
        by (auto simp: eventually_sequentially decseq_def incseq_def intro: order_trans) }
    then show eventually  $(\lambda x a. \text{indicator } \{l \ x .. u \ x\} \ x = (\text{indicator } (\text{einterval } a \ b) \ x :: \text{real}))$  sequentially
      using approx_order_tendstoD(2) [OF  $\langle l \longrightarrow a \rangle$ , of x] order_tendstoD(1) [OF  $\langle u \longrightarrow b \rangle$ , of x]
      by (auto split: split_indicator)
  qed

```

```

qed
with  $\langle a < b \rangle \langle \bigwedge i. l\ i < u\ i \rangle$  show ?thesis
by (simp add: interval_lebesgue_integral_le_eq[symmetric] interval_integral_Icc
less_imp_le)
qed

```

6.23.18 A slightly stronger Fundamental Theorem of Calculus

Three versions: first, for finite intervals, and then two versions for arbitrary intervals.

```

lemma interval_integral_FTC_finite:
  fixes f F :: real  $\Rightarrow$  'a::euclidean_space and a b :: real
  assumes f: continuous_on {min a b..max a b} f
  assumes F:  $\bigwedge x. \min a\ b \leq x \Rightarrow x \leq \max a\ b \Rightarrow (F\ \text{has\_vector\_derivative}\ (f\ x))$ 
    (at x within {min a b..max a b})
  shows (LBINT x=a..b. f x) = F b - F a
proof (cases a  $\leq$  b)
case True
  have (LBINT x=a..b. f x) = (LBINT x. indicat_real {a..b} x *R f x)
  by (simp add: True interval_integral_Icc set_lebesgue_integral_def)
  also have ... = F b - F a
proof (rule interval_FTC_atLeastAtMost [OF True])
  show continuous_on {a..b} f
  using True f by linarith
  show  $\bigwedge x. \llbracket a \leq x; x \leq b \rrbracket \Rightarrow (F\ \text{has\_vector\_derivative}\ f\ x)$ 
    (at x within {a..b})
  by (metis F True max.commute max_absorb1 min_def)
qed
finally show ?thesis .
next
case False
  then have b  $\leq$  a
  by simp
  have - interval_lebesgue_integral lborel (ereal b) (ereal a) f = - (LBINT x.
    indicat_real {b..a} x *R f x)
  by (simp add:  $\langle b \leq a \rangle$  interval_integral_Icc set_lebesgue_integral_def)
  also have ... = F b - F a
proof (subst interval_FTC_atLeastAtMost [OF  $\langle b \leq a \rangle$ ])
  show continuous_on {b..a} f
  using False f by linarith
  show  $\bigwedge x. \llbracket b \leq x; x \leq a \rrbracket \Rightarrow (F\ \text{has\_vector\_derivative}\ f\ x)$ 
    (at x within {b..a})
  by (metis F False max_def min_def)
qed auto
finally show ?thesis
  by (metis interval_integral_endpoints_reverse)
qed

```

```

lemma interval_integral_FTC_nonneg:
  fixes  $f F :: \text{real} \Rightarrow \text{real}$  and  $a b :: \text{ereal}$ 
  assumes  $a < b$ 
  assumes  $F: \bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{DERIV } F \ x :> f \ x$ 
  assumes  $f: \bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{isCont } f \ x$ 
  assumes  $f\_nonneg: \text{AE } x \text{ in lborel. } a < \text{ereal } x \longrightarrow \text{ereal } x < b \longrightarrow 0 \leq f \ x$ 
  assumes  $A: ((F \circ \text{real\_of\_ereal}) \longrightarrow A) \text{ (at\_right } a)$ 
  assumes  $B: ((F \circ \text{real\_of\_ereal}) \longrightarrow B) \text{ (at\_left } b)$ 
  shows
    set_integrable lborel (einterval  $a$   $b$ )  $f$ 
    (LBINT  $x=a..b. f \ x$ ) =  $B - A$ 
proof -
  obtain  $u \ l$  where approx:
    einterval  $a \ b$  =  $(\bigcup i. \{l \ i .. u \ i\})$ 
    incseq  $u$  decseq  $l \ \bigwedge i. l \ i < u \ i \ \bigwedge i. a < l \ i \ \bigwedge i. u \ i < b$ 
     $l \longrightarrow a \ u \longrightarrow b$ 
  by (blast intro: einterval_Icc_approximation[OF  $\langle a < b \rangle$ ])
  have [simp]:  $\bigwedge x \ i. l \ i \leq x \implies a < \text{ereal } x$ 
  by (rule order_less_le_trans, rule approx, force)
  have [simp]:  $\bigwedge x \ i. x \leq u \ i \implies \text{ereal } x < b$ 
  by (rule order_le_less_trans, subst ereal_less_eq(3), assumption, rule approx)
  have FTCi:  $\bigwedge i. (\text{LBINT } x=l \ i..u \ i. f \ x) = F \ (u \ i) - F \ (l \ i)$ 
  using assms approx apply (intro interval_integral_FTC_finite)
  apply (auto simp: less_imp_le min_def max_def
    has_field_derivative_iff_has_vector_derivative[symmetric])
  apply (rule continuous_at_imp_continuous_on, auto intro!:  $f$ )
  by (rule DERIV_subset [OF  $F$ ], auto)
  have 1:  $\bigwedge i. \text{set\_integrable lborel } \{l \ i..u \ i\} \ f$ 
  proof -
    fix  $i$  show set_integrable lborel  $\{l \ i .. u \ i\} \ f$ 
    using  $\langle a < l \ i \rangle \langle u \ i < b \rangle$  unfolding set_integrable_def
    by (intro borel_integrable_compact  $f$  continuous_at_imp_continuous_on
      compact_Icc ballI)
    (auto simp flip: ereal_less_eq)
  qed
  have 2: set_borel_measurable lborel (einterval  $a \ b$ )  $f$ 
  unfolding set_borel_measurable_def
  by (auto simp del: real_scaleR_def intro!: borel_measurable_continuous_on_indicator
    simp: continuous_on_eq_continuous_at einterval_iff  $f$ )
  have 3:  $(\lambda i. \text{LBINT } x=l \ i..u \ i. f \ x) \longrightarrow B - A$ 
  apply (subst FTCi)
  apply (intro tendsto_intros)
  using  $B$  approx unfolding tendsto_at_iff_sequentially comp_def
  using tendsto_at_iff_sequentially[where 'a=real]
  apply (elim allE[of _  $\lambda i. \text{ereal } (u \ i)$ ], auto)
  using  $A$  approx unfolding tendsto_at_iff_sequentially comp_def
  by (elim allE[of _  $\lambda i. \text{ereal } (l \ i)$ ], auto)

```

```

show (LBINT x=a..b. f x) = B - A
  by (rule interval_integral_Icc_approx_nonneg [OF ‹a < b› approx 1 f_nonneg
2 3])
show set_integrable lborel (einterval a b) f
  by (rule interval_integral_Icc_approx_nonneg [OF ‹a < b› approx 1 f_nonneg
2 3])
qed

```

theorem *interval_integral_FTC_integrable*:

```

fixes f F :: real ⇒ 'a::euclidean_space and a b :: ereal
assumes a < b
assumes F:  $\bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies (F \text{ has\_vector\_derivative } f x)$ 
(at x)
assumes f:  $\bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{isCont } f x$ 
assumes f_integrable: set_integrable lborel (einterval a b) f
assumes A:  $((F \circ \text{real\_of\_ereal}) \longrightarrow A)$  (at_right a)
assumes B:  $((F \circ \text{real\_of\_ereal}) \longrightarrow B)$  (at_left b)
shows (LBINT x=a..b. f x) = B - A
proof -
  obtain u l where approx:
    einterval a b =  $(\bigcup i. \{l\ i \ ..\ u\ i\})$ 
    incseq u decseq l  $\bigwedge i. l\ i < u\ i \bigwedge i. a < l\ i \bigwedge i. u\ i < b$ 
    l  $\longrightarrow a$  u  $\longrightarrow b$ 
  by (blast intro: einterval_Icc_approximation[OF ‹a < b›])
have [simp]:  $\bigwedge x\ i. l\ i \leq x \implies a < \text{ereal } x$ 
  by (rule order_less_le_trans, rule approx, force)
have [simp]:  $\bigwedge x\ i. x \leq u\ i \implies \text{ereal } x < b$ 
  by (rule order_le_less_trans, subst ereal_less_eq(3), assumption, rule approx)
have FTCi:  $\bigwedge i. (\text{LBINT } x=l\ i..u\ i. f x) = F(u\ i) - F(l\ i)$ 
  using assms approx
  by (auto simp: less_imp_le min_def max_def
    intro!: f_continuous_at_imp_continuous_on interval_integral_FTC_finite
    intro: has_vector_derivative_at_within)
have  $(\lambda i. \text{LBINT } x=l\ i..u\ i. f x) \longrightarrow B - A$ 
  unfolding FTCi
proof (intro tendsto_intros)
  show  $(\lambda x. F(l\ x)) \longrightarrow A$ 
    using A approx unfolding tendsto_at_iff_sequentially_comp_def
    by (elim allE[of _  $\lambda i. \text{ereal } (l\ i)$ ], auto)
  show  $(\lambda x. F(u\ x)) \longrightarrow B$ 
    using B approx unfolding tendsto_at_iff_sequentially_comp_def
    by (elim allE[of _  $\lambda i. \text{ereal } (u\ i)$ ], auto)
qed
moreover have  $(\lambda i. \text{LBINT } x=l\ i..u\ i. f x) \longrightarrow (\text{LBINT } x=a..b. f x)$ 
  by (rule interval_integral_Icc_approx_integrable [OF ‹a < b› approx f_integrable])
ultimately show ?thesis
  by (elim LIMSEQ_unique)
qed

```

```

theorem interval_integral_FTC2:
  fixes  $a\ b\ c :: \text{real}$  and  $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
  assumes  $a \leq c \leq b$ 
  and  $\text{contf}: \text{continuous\_on } \{a..b\} f$ 
  fixes  $x :: \text{real}$ 
  assumes  $a \leq x$  and  $x \leq b$ 
  shows  $((\lambda u. \text{LBINT } y=c..u. f\ y) \text{ has\_vector\_derivative } (f\ x)) \text{ (at } x \text{ within } \{a..b\})$ 
proof -
  let  $?F = (\lambda u. \text{LBINT } y=a..u. f\ y)$ 
  have  $\text{intf}: \text{set\_integrable lborel } \{a..b\} f$ 
    by (rule borel_integrable_atLeastAtMost', rule contf)
  have  $((\lambda u. \text{integral } \{a..u\} f) \text{ has\_vector\_derivative } f\ x) \text{ (at } x \text{ within } \{a..b\})$ 
    using  $\langle a \leq x \rangle \langle x \leq b \rangle$ 
    by (auto intro: interval_has_vector_derivative continuous_on_subset [OF contf])
  then have  $((\lambda u. \text{integral } \{a..u\} f) \text{ has\_vector\_derivative } (f\ x)) \text{ (at } x \text{ within } \{a..b\})$ 
    by simp
  then have  $(?F \text{ has\_vector\_derivative } (f\ x)) \text{ (at } x \text{ within } \{a..b\})$ 
    by (rule has_vector_derivative_weaken)
    (auto intro!: assms interval_integral_eq_integral[symmetric] set_integrable_subset [OF intf])
  then have  $((\lambda x. (\text{LBINT } y=c..a. f\ y) + ?F\ x) \text{ has\_vector\_derivative } (f\ x)) \text{ (at } x \text{ within } \{a..b\})$ 
    by (auto intro!: derivative_eq_intros)
  then show ?thesis
    proof (rule has_vector_derivative_weaken)
    fix  $u$  assume  $u \in \{a..b\}$ 
    then show  $(\text{LBINT } y=c..a. f\ y) + (\text{LBINT } y=a..u. f\ y) = (\text{LBINT } y=c..u. f\ y)$ 
      using assms
      apply (intro interval_integral_sum)
      apply (auto simp: interval_lebesgue_integrable_def simp del: real_scaleR_def)
      by (rule set_integrable_subset [OF intf], auto simp: min_def max_def)
    qed (insert assms, auto)
qed

```

proposition einterval_antiderivative:

```

  fixes  $a\ b :: \text{ereal}$  and  $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
  assumes  $a < b$  and  $\text{contf}: \bigwedge x :: \text{real}. a < x \implies x < b \implies \text{isCont } f\ x$ 
  shows  $\exists F. \forall x :: \text{real}. a < x \longrightarrow x < b \longrightarrow (F \text{ has\_vector\_derivative } f\ x) \text{ (at } x)$ 
proof -
  from einterval_nonempty [OF  $\langle a < b \rangle$ ] obtain  $c :: \text{real}$  where [simp]:  $a < c < b$ 
    by (auto simp: einterval_def)
  let  $?F = (\lambda u. \text{LBINT } y=c..u. f\ y)$ 
  show ?thesis

```

```

proof (rule exI, clarsimp)
  fix x :: real
  assume [simp]: a < x < b
  have 1: a < min c x by simp
  from interval_nonempty [OF 1] obtain d :: real where [simp]: a < d < c
d < x
  by (auto simp: interval_def)
  have 2: max c x < b by simp
  from interval_nonempty [OF 2] obtain e :: real where [simp]: c < e < e
e < b
  by (auto simp: interval_def)
  have (?F has_vector_derivative f x) (at x within {d..e})
proof (rule has_vector_derivative_within_subset [of _ _ _ {d..e}])
  have continuous_on {d..e} f
  proof (intro continuous_at_imp_continuous_on ballI contf; clarsimp)
    show  $\bigwedge x. \llbracket d \leq x; x \leq e \rrbracket \implies a < \text{ereal } x$ 
      using <a < ereal d> ereal_less_ereal_Ex by auto
    show  $\bigwedge x. \llbracket d \leq x; x \leq e \rrbracket \implies \text{ereal } x < b$ 
      using <ereal e < b> ereal_less_eq(3) le_less_trans by blast
  qed
  then show (?F has_vector_derivative f x) (at x within {d..e})
    by (intro interval_integral_FTC2) (use <d < c> <c < e> <d < x> <x < e>
in <linarith>)
  qed auto
  then show (?F has_vector_derivative f x) (at x)
    by (force simp: has_vector_derivative_within_open [of _ {d..e}])
  qed
qed

```

6.23.19 The substitution theorem

Once again, three versions: first, for finite intervals, and then two versions for arbitrary intervals.

theorem interval_integral_substitution_finite:

```

  fixes a b :: real and f :: real  $\Rightarrow$  'a::euclidean_space
  assumes a  $\leq$  b
  and deriv:  $\bigwedge x. a \leq x \implies x \leq b \implies (g \text{ has\_real\_derivative } (g' x))$  (at x within {a..b})
  and contf : continuous_on (g ' {a..b}) f
  and contg': continuous_on {a..b} g'
  shows LBINT x=a..b. g' x *R f (g x) = LBINT y=g a..g b. f y
proof–
  have v_deriv:  $\bigwedge x. a \leq x \implies x \leq b \implies (g \text{ has\_vector\_derivative } (g' x))$  (at x within {a..b})
    using deriv unfolding has_field_derivative_iff_has_vector_derivative .
  then have contg [simp]: continuous_on {a..b} g
    by (rule continuous_on_vector_derivative) auto
  have 1:  $\exists x \in \{a..b\}. u = g x$  if min (g a) (g b)  $\leq u \leq$  max (g a) (g b) for u
    by (cases g a  $\leq$  g b) (use that assms IVT' [of g a u b] IVT2' [of g b u a] in

```

```

(auto simp: min_def max_def)
obtain c d where g_im:  $g \text{ ' } \{a..b\} = \{c..d\}$  and  $c \leq d$ 
  by (metis continuous_image_closed_interval contg  $\langle a \leq b \rangle$ )
obtain F where derivF:
   $\bigwedge x. \llbracket a \leq x; x \leq b \rrbracket \implies (F \text{ has\_vector\_derivative } (f \text{ (g x)})) \text{ (at (g x) within } (g \text{ ' } \{a..b\}))$ 
  using continuous_on_subset [OF contg] g_im
  by (metis antiderivative_continuous atLeastAtMost_iff image_subset_iff set_eq_subset)
have contfg: continuous_on  $\{a..b\}$  ( $\lambda x. f \text{ (g x)}$ )
  by (blast intro: continuous_on_compose2 contg contg)
have LBINT x. indicat_real  $\{a..b\} x *_R g' x *_R f \text{ (g x)} = F \text{ (g b)} - F \text{ (g a)}$ 
  apply (rule integral FTC_atLeastAtMost
    [OF  $\langle a \leq b \rangle$  vector_diff_chain_within [OF v_derivg derivF, unfolded comp_def]])
  apply (auto intro!: continuous_on_scaleR contg' contfg)
  done
then have LBINT  $x=a..b. g' x *_R f \text{ (g x)} = F \text{ (g b)} - F \text{ (g a)}$ 
  by (simp add: asms interval_integral_Icc set_lebesgue_integral_def)
moreover have LBINT  $y=(g a)..(g b). f y = F \text{ (g b)} - F \text{ (g a)}$ 
proof (rule interval_integral_FTC_finite)
  show continuous_on  $\{\min (g a) (g b).. \max (g a) (g b)\} f$ 
    by (rule continuous_on_subset [OF contg]) (auto simp: image_def 1)
  show  $(F \text{ has\_vector\_derivative } f y) \text{ (at y within } \{\min (g a) (g b).. \max (g a) (g b)\})$ 
    if  $y: \min (g a) (g b) \leq y \leq \max (g a) (g b)$  for y
  proof -
    obtain x where  $a \leq x \leq b$   $y = g x$ 
      using 1 y by force
    then show ?thesis
      by (auto simp: image_def intro!: 1 has_vector_derivative_within_subset [OF derivF])
  qed
qed
ultimately show ?thesis by simp
qed

```

theorem interval_integral_substitution_integrable:

```

fixes f :: real  $\Rightarrow$  'a::euclidean_space and a b u v :: ereal
assumes a < b
and deriv_g:  $\bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{DERIV } g x :> g' x$ 
and contf:  $\bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{isCont } f \text{ (g x)}$ 
and contg':  $\bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{isCont } g' x$ 
and g'_nonneg:  $\bigwedge x. a \leq \text{ereal } x \implies \text{ereal } x \leq b \implies 0 \leq g' x$ 
and A:  $((\text{ereal} \circ g \circ \text{real\_of\_ereal}) \longrightarrow A) \text{ (at\_right a)}$ 
and B:  $((\text{ereal} \circ g \circ \text{real\_of\_ereal}) \longrightarrow B) \text{ (at\_left b)}$ 
and integrable: set_integrable lborel (einterval a b) ( $\lambda x. g' x *_R f \text{ (g x)}$ )

```

```

and integrable2: set_integrable lborel (einterval A B) ( $\lambda x. f x$ )
shows (LBINT  $x=A..B. f x$ ) = (LBINT  $x=a..b. g' x *_R f (g x)$ )
proof –
  obtain u l where approx [simp]:
    einterval a b = ( $\bigcup i. \{l i .. u i\}$ )
    incseq u decseq l  $\wedge i. l i < u i \wedge i. a < l i \wedge i. u i < b$ 
     $l \longrightarrow a \quad u \longrightarrow b$ 
    by (blast intro: einterval_Icc_approximation[OF  $\langle a < b \rangle$ ])
  note less_imp_le [simp]
  have [simp]:  $\wedge x i. l i \leq x \implies a < ereal x$ 
    by (rule order_less_le_trans, rule approx, force)
  have [simp]:  $\wedge x i. x \leq u i \implies ereal x < b$ 
    by (rule order_le_less_trans, subst ereal_less_eq(3), assumption, rule approx)
  then have lessb[simp]:  $\wedge i. l i < b$ 
    using approx(4) less_eq_real_def by blast
  have [simp]:  $\wedge i. a < u i$ 
    by (rule order_less_trans, rule approx, auto, rule approx)
  have lle[simp]:  $\wedge i j. i \leq j \implies l j \leq l i$  by (rule decseqD, rule approx)
  have [simp]:  $\wedge i j. i \leq j \implies u i \leq u j$  by (rule incseqD, rule approx)
  have g_nondec [simp]:  $g x \leq g y$  if  $a < x \leq y < b$  for  $x y$ 
  proof (rule DERIV_nonneg_imp_nondecreasing [OF  $\langle x \leq y \rangle$ ], intro exI conjI)
    show  $\wedge u. x \leq u \implies u \leq y \implies (g \text{ has\_real\_derivative } g' u) \text{ (at } u)$ 
      by (meson deriv_g ereal_less_eq(3) le_less_trans less_le_trans that)
    show  $\wedge u. x \leq u \implies u \leq y \implies 0 \leq g' u$ 
      by (meson assms(5) dual_order.trans le_ereal_le less_imp_le order_refl
that)
  qed
  have  $A \leq B$  and un: einterval A B = ( $\bigcup i. \{g(l i) <..< g(u i)\}$ )
  proof –
    have A2: ( $\lambda i. g (l i)$ )  $\longrightarrow A$ 
      using A apply (auto simp: einterval_def tendsto_at_iff_sequentially_comp_def)
      by (drule_tac  $x = \lambda i. ereal (l i)$  in spec, auto)
    hence A3:  $\wedge i. g (l i) \geq A$ 
      by (intro decseq_ge, auto simp: decseq_def)
    have B2: ( $\lambda i. g (u i)$ )  $\longrightarrow B$ 
      using B apply (auto simp: einterval_def tendsto_at_iff_sequentially_comp_def)
      by (drule_tac  $x = \lambda i. ereal (u i)$  in spec, auto)
    hence B3:  $\wedge i. g (u i) \leq B$ 
      by (intro incseq_le, auto simp: incseq_def)
    have ereal (g (l 0))  $\leq$  ereal (g (u 0))
      by auto
    then show  $A \leq B$ 
      by (meson A3 B3 order.trans)
    { fix  $x :: real$ 
      assume  $A < x$  and  $x < B$ 
      then have eventually ( $\lambda i. ereal (g (l i)) < x \wedge x < ereal (g (u i))$ ) sequentially
        by (fast intro: eventually_conj order_tendstoD A2 B2)
      hence  $\exists i. g (l i) < x \wedge x < g (u i)$ 
        by (simp add: eventually_sequentially, auto)
    }
  
```



```

} note AB = this
show einterval A B = ( $\bigcup i. \{g(l\ i) <..< g(u\ i)\}$ )
proof
  show einterval A B  $\subseteq$  ( $\bigcup i. \{g(l\ i) <..< g(u\ i)\}$ )
    by (auto simp: einterval_def AB)
  show ( $\bigcup i. \{g(l\ i) <..< g(u\ i)\}$ )  $\subseteq$  einterval A B
  proof (clarsimp simp add: einterval_def, intro conjI)
    show  $\bigwedge x\ i. \llbracket g(l\ i) < x; x < g(u\ i) \rrbracket \implies A < \text{ereal } x$ 
      using A3 le_ereal_less by blast
    show  $\bigwedge x\ i. \llbracket g(l\ i) < x; x < g(u\ i) \rrbracket \implies \text{ereal } x < B$ 
      using B3 ereal_le_less by blast
  qed
qed
qed
qed

have eq1: (LBINT x=l i.. u i. g' x *R f (g x)) = (LBINT y=g (l i)..g (u i). f
y) for i
  apply (rule interval_integral_substitution_finite [OF _ DERIV_subset [OF
deriv_g]])
  unfolding has_field_derivative_iff_has_vector_derivative[symmetric]
  apply (auto intro!: continuous_at_imp_continuous_on contf contg')
  done
have ( $\lambda i. \text{LBINT } x=l\ i..u\ i. g' x *_{\text{R}} f (g x)$ )  $\longrightarrow$  ( $\text{LBINT } x=a..b. g' x *_{\text{R}} f$ 
( $g x$ ))
  apply (rule interval_integral_Icc_approx_integrable [OF  $\langle a < b \rangle$  approx])
  by (rule assms)
hence 2: ( $\lambda i. (\text{LBINT } y=g(l\ i)..g(u\ i). f\ y)$ )  $\longrightarrow$  ( $\text{LBINT } x=a..b. g' x *_{\text{R}} f$ 
( $g x$ ))
  by (simp add: eq1)
have incseq: incseq ( $\lambda i. \{g(l\ i) <..< g(u\ i)\}$ )
  apply (auto simp: incseq_def)
  using lessb lle approx(5) g_nondec le_less_trans apply blast
  by (force intro: less_le_trans)
have ( $\lambda i. \text{set\_lebesgue\_integral borel } \{g(l\ i) <..< g(u\ i)\} f$ )
 $\longrightarrow$  set_lebesgue_integral borel (einterval A B) f
  unfolding un by (rule set_integral_cont_up) (use incseq integrable2 un in
auto)
then have ( $\lambda i. (\text{LBINT } y=g(l\ i)..g(u\ i). f\ y)$ )  $\longrightarrow$  ( $\text{LBINT } x = A..B. f\ x$ )
  by (simp add: interval_lebesgue_integral_le_eq  $\langle A \leq B \rangle$ )
thus ?thesis by (intro LIMSEQ_unique [OF _ 2])
qed

```

theorem interval_integral_substitution_nonneg:

```

fixes f g g': real  $\Rightarrow$  real and a b u v :: ereal
assumes a < b
and deriv_g:  $\bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{DERIV } g\ x :> g' x$ 
and contf:  $\bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{isCont } f (g\ x)$ 

```

and contg' : $\bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{isCont } g' x$
and f_nonneg : $\bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies 0 \leq f (g x)$
and g'_nonneg : $\bigwedge x. a \leq \text{ereal } x \implies \text{ereal } x \leq b \implies 0 \leq g' x$
and A : $((\text{ereal} \circ g \circ \text{real\_of\_ereal}) \longrightarrow A) \text{ (at\_right } a)$
and B : $((\text{ereal} \circ g \circ \text{real\_of\_ereal}) \longrightarrow B) \text{ (at\_left } b)$
and integrable_fg : $\text{set\_integrable lborel (einterval } a \text{ } b) (\lambda x. f (g x) * g' x)$
shows

$\text{set\_integrable lborel (einterval } A \text{ } B) f$
 $(\text{LBINT } x=A..B. f x) = (\text{LBINT } x=a..b. (f (g x) * g' x))$

proof –

from $\text{einterval\_Icc\_approximation}[\text{OF } \langle a < b \rangle]$ **obtain** $u \text{ } l$ **where** approx [simp] :
 $\text{einterval } a \text{ } b = (\bigcup i. \{l \text{ } i..u \text{ } i\})$

$\text{incseq } u$

$\text{decseq } l$

$\bigwedge i. l \text{ } i < u \text{ } i$

$\bigwedge i. a < \text{ereal } (l \text{ } i)$

$\bigwedge i. \text{ereal } (u \text{ } i) < b$

$(\lambda x. \text{ereal } (l \text{ } x)) \longrightarrow a$

$(\lambda x. \text{ereal } (u \text{ } x)) \longrightarrow b$ **by** *this auto*

have aless[simp] : $\bigwedge x \text{ } i. l \text{ } i \leq x \implies a < \text{ereal } x$

by $(\text{rule order\_less\_le\_trans, rule approx, force})$

have lessb[simp] : $\bigwedge x \text{ } i. x \leq u \text{ } i \implies \text{ereal } x < b$

by $(\text{rule order\_le\_less\_trans, subst ereal\_less\_eq(3), assumption, rule approx})$

have llb[simp] : $\bigwedge i. l \text{ } i < b$

using $\text{lessb approx(4) less\_eq\_real\_def}$ **by** *blast*

have alu[simp] : $\bigwedge i. a < u \text{ } i$

by $(\text{rule order\_less\_trans, rule approx, auto, rule approx})$

have [simp] : $\bigwedge i \text{ } j. i \leq j \implies l \text{ } j \leq l \text{ } i$ **by** $(\text{rule decseqD, rule approx})$

have uleu[simp] : $\bigwedge i \text{ } j. i \leq j \implies u \text{ } i \leq u \text{ } j$ **by** $(\text{rule incseqD, rule approx})$

have $g\_nondec \text{ [simp]}$: $g x \leq g y$ **if** $a < x \leq y < b$ **for** $x \text{ } y$

proof $(\text{rule DERIV\_nonneg\_imp\_nondecreasing [OF } \langle x \leq y \rangle], \text{intro exI conjI})$

show $\bigwedge u. x \leq u \implies u \leq y \implies (g \text{ has\_real\_derivative } g' u) \text{ (at } u)$

by $(\text{meson deriv\_g ereal\_less\_eq(3) le\_less\_trans less\_le\_trans that})$

show $\bigwedge u. x \leq u \implies u \leq y \implies 0 \leq g' u$

by $(\text{meson } g'\_nonneg \text{ less\_ereal.simps(1) less\_trans not\_less that})$

qed

have $A \leq B$ **and** un : $\text{einterval } A \text{ } B = (\bigcup i. \{g(l \text{ } i) <..< g(u \text{ } i)\})$

proof –

have $A2$: $(\lambda i. g (l \text{ } i)) \longrightarrow A$

using A **apply** $(\text{auto simp: einterval\_def tendsto\_at\_iff\_sequentially comp\_def})$

by $(\text{drule\_tac } x = \lambda i. \text{ereal } (l \text{ } i) \text{ in spec, auto})$

hence $A3$: $\bigwedge i. g (l \text{ } i) \geq A$

by $(\text{intro decseq\_ge, auto simp: decseq\_def})$

have $B2$: $(\lambda i. g (u \text{ } i)) \longrightarrow B$

using B **apply** $(\text{auto simp: einterval\_def tendsto\_at\_iff\_sequentially comp\_def})$

by $(\text{drule\_tac } x = \lambda i. \text{ereal } (u \text{ } i) \text{ in spec, auto})$

hence $B3$: $\bigwedge i. g (u \text{ } i) \leq B$

by $(\text{intro incseq\_le, auto simp: incseq\_def})$

have $\text{ereal } (g (l \text{ } 0)) \leq \text{ereal } (g (u \text{ } 0))$

```

    by (auto simp: less_imp_le)
  then show  $A \leq B$ 
    by (meson A3 B3 order.trans)
  { fix  $x :: \text{real}$ 
    assume  $A < x$  and  $x < B$ 
    then have eventually  $(\lambda i. \text{ereal } (g \ (l \ i)) < x \wedge x < \text{ereal } (g \ (u \ i)))$  sequentially
      by (fast intro: eventually_conj order_tendstoD A2 B2)
    hence  $\exists i. g \ (l \ i) < x \wedge x < g \ (u \ i)$ 
      by (simp add: eventually_sequentially, auto)
  } note AB = this
  show  $\text{interval } A \ B = (\bigcup i. \{g(l \ i) <..< g(u \ i)\})$ 
  proof
    show  $\text{interval } A \ B \subseteq (\bigcup i. \{g \ (l \ i) <..< g \ (u \ i)\})$ 
      by (auto simp: interval_def AB)
    show  $(\bigcup i. \{g \ (l \ i) <..< g \ (u \ i)\}) \subseteq \text{interval } A \ B$ 
      apply (clarsimp simp: interval_def, intro conjI)
      using A3 le_ereal_less apply blast
      using B3 ereal_le_less by blast
  qed
qed

  have eq1:  $(\text{LBINT } x=l \ i.. u \ i. (f \ (g \ x) * g' \ x)) = (\text{LBINT } y=g \ (l \ i)..g \ (u \ i). f \ y)$ 
  for  $i$ 
  proof -
    have  $(\text{LBINT } x=l \ i.. u \ i. g' \ x *_R f \ (g \ x)) = (\text{LBINT } y=g \ (l \ i)..g \ (u \ i). f \ y)$ 
      apply (rule interval_integral_substitution_finite [OF _ DERIV_subset [OF deriv_g]])
      unfolding has_field_derivative_iff_has_vector_derivative[symmetric]
      apply (auto simp: less_imp_le intro!: continuous_at_imp_continuous_on
        contf contg')
      done
    then show ?thesis
      by (simp add: ac_simps)
  qed

  have incseq:  $\text{incseq } (\lambda i. \{g \ (l \ i) <..< g \ (u \ i)\})$ 
    apply (clarsimp simp add: incseq_def, intro conjI)
    apply (meson llb antimonono_def approx(3) approx(5) g_nondec le_less_trans)
    using alu uleu approx(6) g_nondec less_le_trans by blast
  have img:  $\exists c \geq l \ i. c \leq u \ i \wedge x = g \ c \text{ if } g \ (l \ i) \leq x \leq g \ (u \ i) \text{ for } x \ i$ 
  proof -
    have continuous_on  $\{l \ i..u \ i\} \ g$ 
      by (force intro!: DERIV_isCont deriv_g continuous_at_imp_continuous_on)
    with that show ?thesis
      using IVT' [of g] approx(4) dual_order.strict_implies_order by blast
  qed

  have continuous_on  $\{g \ (l \ i)..g \ (u \ i)\} \ f$  for  $i$ 
    apply (intro continuous_intros continuous_at_imp_continuous_on)
    using contf img by force
  then have int_f:  $\bigwedge i. \text{set\_integrable lborel } \{g \ (l \ i) <..< g \ (u \ i)\} \ f$ 

```

```

    by (rule set_integrable_subset [OF borel_integrable_atLeastAtMost']) (auto
intro: less_imp_le)
  have integrable: set_integrable lborel ( $\bigcup i. \{g(l\ i) <..< g(u\ i)\}$ ) f
  proof (intro pos_integrable_to_top incseq_int_f)
    let ?l = (LBINT x=a..b. f (g x) * g' x)
    have ( $\lambda i. LBINT\ x=l\ i..u\ i. f\ (g\ x) * g'\ x$ )  $\longrightarrow$  ?l
    by (intro assms interval_integral_Icc_approx_integrable [OF <a < b> approx])
    hence ( $\lambda i. (LBINT\ y=g\ (l\ i)..g\ (u\ i). f\ y)$ )  $\longrightarrow$  ?l
    by (simp add: eq1)
  then show ( $\lambda i. set\_lebesgue\_integral\ lborel\ \{g\ (l\ i) <..< g\ (u\ i)\}\ f$ )  $\longrightarrow$  ?l
    unfolding interval_lebesgue_integral_def by (auto simp: less_imp_le)
  have  $\bigwedge x\ i. g\ (l\ i) \leq x \implies x \leq g\ (u\ i) \implies 0 \leq f\ x$ 
    using aless_f_nonneg_img lessb by blast
  then show  $\bigwedge x\ i. x \in \{g\ (l\ i) <..< g\ (u\ i)\} \implies 0 \leq f\ x$ 
    using less_eq_real_def by auto
  qed (auto simp: greaterThanLessThan_borel)
  thus set_integrable lborel (einterval A B) f
    by (simp add: un)

```

```

  have (LBINT x=A..B. f x) = (LBINT x=a..b. g' x *R f (g x))
  proof (rule interval_integral_substitution_integrable)
    show set_integrable lborel (einterval a b) ( $\lambda x. g'\ x *_{\mathbb{R}} f\ (g\ x)$ )
      using integrable_fg by (simp add: ac_simps)
    qed fact+
  then show (LBINT x=A..B. f x) = (LBINT x=a..b. (f (g x) * g' x))
    by (simp add: ac_simps)
  qed

```

```

syntax _complex_lebesgue_borel_integral :: pttrn  $\Rightarrow$  real  $\Rightarrow$  complex
  ((2CLBINT _ . _) [0,60] 60)

```

```

translations CLBINT x. f == CONST complex_lebesgue_integral CONST lborel
  ( $\lambda x. f$ )

```

```

syntax _complex_set_lebesgue_borel_integral :: pttrn  $\Rightarrow$  real set  $\Rightarrow$  real  $\Rightarrow$  complex
  ((3CLBINT _ : _ . _) [0,60,61] 60)

```

```

translations
  CLBINT x:A. f == CONST complex_set_lebesgue_integral CONST lborel A
  ( $\lambda x. f$ )

```

```

abbreviation complex_interval_lebesgue_integral ::
  real measure  $\Rightarrow$  ereal  $\Rightarrow$  ereal  $\Rightarrow$  (real  $\Rightarrow$  complex)  $\Rightarrow$  complex where
  complex_interval_lebesgue_integral M a b f  $\equiv$  interval_lebesgue_integral M a b
  f

```

```

abbreviation complex_interval_lebesgue_integrable ::

```

real measure $\Rightarrow$ *ereal* $\Rightarrow$ *ereal* $\Rightarrow$ (*real* $\Rightarrow$ *complex*) $\Rightarrow$ *bool* **where**
complex_interval_lebesgue_integrable *M a b f* $\equiv$ *interval_lebesgue_integrable* *M*
a b f

syntax

_ascii_complex_interval_lebesgue_borel_integral :: *pttrn* $\Rightarrow$ *ereal* $\Rightarrow$ *ereal* $\Rightarrow$
real $\Rightarrow$ *complex*
 ((4*CLBINT* _=*...*. _) [0,60,60,61] 60)

translations

CLBINT *x=a..b. f* == *CONST complex_interval_lebesgue_integral* *CONST*
lborel a b ($\lambda x. f$)

proposition *interval_integral_norm*:

fixes *f* :: *real* $\Rightarrow$ '*a* :: {*banach*, *second_countable_topology*}

shows *interval_lebesgue_integrable lborel a b f* $\implies$ *a* $\leq$ *b* $\implies$
norm (*LBINT* *t=a..b. f t*) $\leq$ *LBINT* *t=a..b. norm* (*f t*)

using *integral_norm_bound*[of *lborel* $\lambda x. \text{indicator } (einterval\ a\ b)\ x *_R f\ x$]

by (*auto simp: interval_lebesgue_integral_def interval_lebesgue_integrable_def*
set_lebesgue_integral_def)

proposition *interval_integral_norm2*:

interval_lebesgue_integrable lborel a b f $\implies$
norm (*LBINT* *t=a..b. f t*) $\leq$ |*LBINT* *t=a..b. norm* (*f t*)|

proof (*induct a b rule: linorder_wlog*)

case (*sym a b*) **then show** ?*case*

by (*simp add: interval_integral_endpoints_reverse*[of *a b*] *interval_integrable_endpoints_reverse*[of
a b])

next

case (*le a b*)

then have |*LBINT* *t=a..b. norm* (*f t*)| = *LBINT* *t=a..b. norm* (*f t*)

using *integrable_norm*[of *lborel* $\lambda x. \text{indicator } (einterval\ a\ b)\ x *_R f\ x$]

by (*auto simp: interval_lebesgue_integral_def interval_lebesgue_integrable_def*
set_lebesgue_integral_def
intro!: integral_nonneg_AE abs_of_nonneg)

then show ?*case*

using *le* **by** (*simp add: interval_integral_norm*)

qed

lemma *integral_cos*: *t* $\neq 0 \implies$ *LBINT* *x=a..b. cos* (*t* * *x*) = *sin* (*t* * *b*) / *t* -
sin (*t* * *a*) / *t*

apply (*intro interval_integral_FTC_finite continuous_intros*)

by (*auto intro!: derivative_eq_intros simp: has_field_derivative_iff_has_vector_derivative[symmetric]*)

end

6.24 Integration by Substitution for the Lebesgue Integral

```
theory Lebesgue_Integral_Substitution
imports Interval_Integral
begin
```

```
lemma nn_integral_substitution_aux:
  fixes f :: real  $\Rightarrow$  ennreal
  assumes Mf: f  $\in$  borel_measurable borel
  assumes nonnegf:  $\bigwedge x. f\ x \geq 0$ 
  assumes derivg:  $\bigwedge x. x \in \{a..b\} \implies (g \text{ has\_real\_derivative } g'\ x) \text{ (at } x)$ 
  assumes contg': continuous_on {a..b} g'
  assumes derivg_nonneg:  $\bigwedge x. x \in \{a..b\} \implies g'\ x \geq 0$ 
  assumes a < b
  shows  $(\int^{+x}. f\ x * \text{indicator } \{g\ a..g\ b\}\ x\ \partial \text{lborel}) =$ 
     $(\int^{+x}. f\ (g\ x) * g'\ x * \text{indicator } \{a..b\}\ x\ \partial \text{lborel})$ 
proof -
  from  $\langle a < b \rangle$  have [simp]:  $a \leq b$  by simp
  from derivg have contg: continuous_on {a..b} g by (rule has_real_derivative_imp_continuous_on)
  from this and contg' have Mg: set_borel_measurable borel {a..b} g and
    Mg': set_borel_measurable borel {a..b} g'
    by (simp_all only: set_measurable_continuous_on_ivl)
  from derivg have derivg':  $\bigwedge x. x \in \{a..b\} \implies (g \text{ has\_vector\_derivative } g'\ x) \text{ (at } x)$ 
    by (simp only: has_field_derivative_iff_has_vector_derivative)

  have real_ind[simp]:  $\bigwedge A\ x. \text{enn2real } (\text{indicator } A\ x) = \text{indicator } A\ x$ 
    by (auto split: split_indicator)
  have ennreal_ind[simp]:  $\bigwedge A\ x. \text{ennreal } (\text{indicator } A\ x) = \text{indicator } A\ x$ 
    by (auto split: split_indicator)
  have [simp]:  $\bigwedge x\ A. \text{indicator } A\ (g\ x) = \text{indicator } (g\ -' A)\ x$ 
    by (auto split: split_indicator)

  from derivg derivg_nonneg have monog:  $\bigwedge x\ y. a \leq x \implies x \leq y \implies y \leq b \implies$ 
 $g\ x \leq g\ y$ 
    by (rule deriv_nonneg_imp_mono) simp_all
  with monog have [simp]:  $g\ a \leq g\ b$  by (auto intro: mono_onD)

  show ?thesis
  proof (induction rule: borel_measurable_induct[OF Mf, case_names cong set
    mult add sup])
    case (cong f1 f2)
    from cong.hyps(3) have f1 = f2 by auto
    with cong show ?case by simp
  next
    case (set A)
    from set.hyps show ?case
```

```

proof (induction rule: borel_set_induct)
  case empty
  thus ?case by simp
next
  case (interval c d)
  {
    fix u v :: real assume asm: {u..v}  $\subseteq$  {g a..g b} u  $\leq$  v

    obtain u' v' where u'v': {a..b}  $\cap$  g- '{u..v} = {u'..v'} u'  $\leq$  v' g u' = u g
    v' = v

    using asm by (rule_tac continuous_interval_vimage_Int[OF contg
monog, of u v]) simp_all
    hence {u'..v'}  $\subseteq$  {a..b} {u'..v'}  $\subseteq$  g - '{u..v} by blast+
    with u'v'(2) have u'  $\in$  g - '{u..v} v'  $\in$  g - '{u..v} by auto
    from u'v'(1) have [simp]: {a..b}  $\cap$  g - '{u..v}  $\in$  sets borel by simp

    have A: continuous_on {min u' v'..max u' v'} g'
      by (simp only: u'v' max_absorb2 min_absorb1)
      (intro continuous_on_subset[OF contg'], insert u'v', auto)
    have  $\bigwedge x. x \in \{u'..v'\} \implies (g \text{ has\_real\_derivative } g' x) \text{ (at } x \text{ within } \{u'..v'\})$ 
      using asm by (intro has_field_derivative_subset[OF derivg] subsetD[OF
{u'..v'}  $\subseteq$  {a..b}]) auto
    hence B:  $\bigwedge x. \min u' v' \leq x \implies x \leq \max u' v' \implies$ 
      (g has_vector_derivative g' x) (at x within {min u' v'..max u'
v'})
      by (simp only: u'v' max_absorb2 min_absorb1)
      (auto simp: has_field_derivative_iff_has_vector_derivative)
    have integrable lborel ( $\lambda x. \text{indicator } (\{a..b\} \cap g - '{u..v}) x *_R g' x$ )
      using set_integrable_subset borel_integrable_atLeastAtMost[OF contg']
    by (metis {u'..v'}  $\subseteq$  {a..b} eucl_ivals(5) set_integrable_def sets_lborel
u'v'(1))
    hence ( $\int^+ x. \text{ennreal } (g' x) * \text{indicator } (\{a..b\} \cap g - '{u..v}) x \partial \text{lborel}$ ) =
      LBINT x:{a..b}  $\cap$  g- '{u..v}. g' x
    unfolding set_lebesgue_integral_def
    by (subst nn_integral_eq_integral[symmetric])
      (auto intro!: derivg_nonneg nn_integral_cong split: split_indicator)
    also from interval_integral_FTC_finite[OF A B]
      have LBINT x:{a..b}  $\cap$  g- '{u..v}. g' x = v - u
      by (simp add: u'v' interval_integral_Icc {u  $\leq$  v})
    finally have ( $\int^+ x. \text{ennreal } (g' x) * \text{indicator } (\{a..b\} \cap g - '{u..v}) x$ 
 $\partial \text{lborel}$ ) =
      ennreal (v - u) .
  } note A = this

  have ( $\int^+ x. \text{indicator } \{c..d\} (g x) * \text{ennreal } (g' x) * \text{indicator } \{a..b\} x \partial \text{lborel}$ )
  =
    ( $\int^+ x. \text{ennreal } (g' x) * \text{indicator } (\{a..b\} \cap g - '{c..d}) x \partial \text{lborel}$ )
    by (intro nn_integral_cong) (simp split: split_indicator)
  also have {a..b}  $\cap$  g- '{c..d} = {a..b}  $\cap$  g- '{max (g a) c..min (g b) d}

```

```

    using ⟨a ≤ b⟩ ⟨c ≤ d⟩
    by (auto intro!: monog intro: order.trans)
  also have (∫+ x. ennreal (g' x) * indicator ... x ∂lborel) =
    (if max (g a) c ≤ min (g b) d then min (g b) d - max (g a) c else 0)
    using ⟨c ≤ d⟩ by (simp add: A)
  also have ... = (∫+ x. indicator ({g a..g b} ∩ {c..d}) x ∂lborel)
    by (subst nn_integral_indicator) (auto intro!: measurable_sets Mg simp:)
  also have ... = (∫+ x. indicator {c..d} x * indicator {g a..g b} x ∂lborel)
    by (intro nn_integral_cong) (auto split: split_indicator)
  finally show ?case ..

next

case (compl A)
note ⟨A ∈ sets borel⟩[measurable]
from emeasure_mono[of A ∩ {g a..g b} {g a..g b} lborel]
have [simp]: emeasure lborel (A ∩ {g a..g b}) ≠ top by (auto simp: top_unique)
have [simp]: g - 'A ∩ {a..b} ∈ sets borel
  by (rule set_borel_measurable_sets[OF Mg]) auto
have [simp]: g - '(-A) ∩ {a..b} ∈ sets borel
  by (rule set_borel_measurable_sets[OF Mg]) auto

have (∫+ x. indicator (-A) x * indicator {g a..g b} x ∂lborel) =
  (∫+ x. indicator (-A ∩ {g a..g b}) x ∂lborel)
  by (rule nn_integral_cong) (simp split: split_indicator)
  also from compl have ... = emeasure lborel ({g a..g b} - A) using de-
rivg_nonneg
  by (simp add: vimage_Compl diff_eq Int_commute[of -A])
  also have {g a..g b} - A = {g a..g b} - A ∩ {g a..g b} by blast
  also have emeasure lborel ... = g b - g a - emeasure lborel (A ∩ {g a..g b})
    using ⟨A ∈ sets borel⟩ by (subst emeasure_Diff) (auto simp: )
  also have emeasure lborel (A ∩ {g a..g b}) =
    ∫+ x. indicator A x * indicator {g a..g b} x ∂lborel
    using ⟨A ∈ sets borel⟩
    by (subst nn_integral_indicator[symmetric], simp, intro nn_integral_cong)
    (simp split: split_indicator)
  also have ... = ∫+ x. indicator (g - 'A ∩ {a..b}) x * ennreal (g' x * indicator
{a..b} x) ∂lborel (is _ = ?I)
    by (subst compl.IH, intro nn_integral_cong) (simp split: split_indicator)
  also have g b - g a = LBINT x:{a..b}. g' x using derivg'
    unfolding set_lebesgue_integral_def
    by (intro integral_FTC_atLeastAtMost[symmetric])
    (auto intro: continuous_on_subset[OF contg] has_field_derivative_subset[OF
derivg]
      has_vector_derivative_at_within)
  also have ennreal ... = ∫+ x. g' x * indicator {a..b} x ∂lborel
  using borel_integrable_atLeastAtMost'[OF contg] unfolding set_lebesgue_integral_def
    by (subst nn_integral_eq_integral)
    (simp_all add: mult.commute derivg_nonneg set_integrable_def split:

```



```

split_indicator)
  also have  $Mg''$ :  $(\lambda x. \text{indicator } (g - 'A \cap \{a..b\}) x * \text{ennreal } (g' x * \text{indicator } \{a..b\} x))$ 
     $\in \text{borel\_measurable borel using } Mg'$ 
  by (intro borel_measurable_times_ennreal borel_measurable_indicator)
    (simp_all add: mult.commute set_borel_measurable_def)
  have  $le$ :  $(\int^+ x. \text{indicator } (g - 'A \cap \{a..b\}) x * \text{ennreal } (g' x * \text{indicator } \{a..b\} x) \partial \text{lborel}) \leq$ 
     $(\int^+ x. \text{ennreal } (g' x) * \text{indicator } \{a..b\} x \partial \text{lborel})$ 
  by (intro nn_integral_mono) (simp split: split_indicator add: derivg_nonneg)
  note integrable = borel_integrable_atLeastAtMost[OF contg]
  with  $le$  have notinf:  $(\int^+ x. \text{indicator } (g - 'A \cap \{a..b\}) x * \text{ennreal } (g' x * \text{indicator } \{a..b\} x) \partial \text{lborel}) \neq \text{top}$ 
  by (auto simp: real_integrable_def nn_integral_set_ennreal mult.commute top_unique set_integrable_def)
  have  $(\int^+ x. g' x * \text{indicator } \{a..b\} x \partial \text{lborel}) - ?I =$ 
     $\int^+ x. \text{ennreal } (g' x * \text{indicator } \{a..b\} x) -$ 
     $\text{indicator } (g - 'A \cap \{a..b\}) x * \text{ennreal } (g' x * \text{indicator } \{a..b\} x) \partial \text{lborel}$ 
  apply (intro nn_integral_diff[symmetric])
  apply (insert  $Mg'$ , simp add: mult.commute set_borel_measurable_def) []
  apply (insert  $Mg''$ , simp) []
  apply (simp split: split_indicator add: derivg_nonneg)
  apply (rule notinf)
  apply (simp split: split_indicator add: derivg_nonneg)
  done
  also have  $\dots = \int^+ x. \text{indicator } (-A) (g x) * \text{ennreal } (g' x) * \text{indicator } \{a..b\} x \partial \text{lborel}$ 
  by (intro nn_integral_cong) (simp split: split_indicator)
  finally show ?case .

next
case (union f)
then have [simp]:  $\bigwedge i. \{a..b\} \cap g - 'f i \in \text{sets borel}$ 
  by (subst Int_commute, intro set_borel_measurable_sets[OF Mg]) auto
  have  $g - '(\bigcup i. f i) \cap \{a..b\} = (\bigcup i. \{a..b\} \cap g - 'f i)$  by auto
  hence  $g - '(\bigcup i. f i) \cap \{a..b\} \in \text{sets borel}$  by (auto simp del: UN_simps)

  have  $(\int^+ x. \text{indicator } (\bigcup i. f i) x * \text{indicator } \{g a..g b\} x \partial \text{lborel}) =$ 
     $\int^+ x. \text{indicator } (\bigcup i. \{g a..g b\} \cap f i) x \partial \text{lborel}$ 
  by (intro nn_integral_cong) (simp split: split_indicator)
  also from union have  $\dots = \text{emeasure lborel } (\bigcup i. \{g a..g b\} \cap f i)$  by simp
  also from union have  $\dots = (\sum i. \text{emeasure lborel } (\{g a..g b\} \cap f i))$ 
  by (intro suminf_emeasure[symmetric]) (auto simp: disjoint_family_on_def)
  also from union have  $\dots = (\sum i. \int^+ x. \text{indicator } (\{g a..g b\} \cap f i) x \partial \text{lborel})$ 
  by simp
  also have  $(\lambda i. \int^+ x. \text{indicator } (\{g a..g b\} \cap f i) x \partial \text{lborel}) =$ 
     $(\lambda i. \int^+ x. \text{indicator } (f i) x * \text{indicator } \{g a..g b\} x \partial \text{lborel})$ 
  by (intro ext nn_integral_cong) (simp split: split_indicator)

```

also from *union.IH* **have** $(\sum i. \int^+ x. \text{indicator } (f \ i) \ x * \text{indicator } \{g \ a..g \ b\} \ x \ \partial \text{lborel}) =$
 $(\sum i. \int^+ x. \text{indicator } (f \ i) \ (g \ x) * \text{ennreal } (g' \ x) * \text{indicator } \{a..b\} \ x \ \partial \text{lborel})$ **by** *simp*
also have $(\lambda i. \int^+ x. \text{indicator } (f \ i) \ (g \ x) * \text{ennreal } (g' \ x) * \text{indicator } \{a..b\} \ x \ \partial \text{lborel}) =$
 $(\lambda i. \int^+ x. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * \text{indicator } (\{a..b\} \cap g - 'f \ i) \ x \ \partial \text{lborel})$
by (*intro ext nn_integral_cong*) (*simp split: split_indicator*)
also have $(\sum i. \dots i) = \int^+ x. (\sum i. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * \text{indicator } (\{a..b\} \cap g - 'f \ i) \ x) \ \partial \text{lborel}$
using *Mg'*
apply (*intro nn_integral_suminf[symmetric]*)
apply (*rule borel_measurable_times_ennreal, simp add: mult.commute set_borel_measurable_def*)
apply (*rule borel_measurable_indicator, subst sets_lborel*)
apply (*simp_all split: split_indicator add: derivg_nonneg*)
done
also have $(\lambda x \ i. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * \text{indicator } (\{a..b\} \cap g - 'f \ i) \ x) =$
 $(\lambda x \ i. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * \text{indicator } (g - 'f \ i) \ x)$
by (*intro ext*) (*simp split: split_indicator*)
also have $(\int^+ x. (\sum i. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * \text{indicator } (g - 'f \ i) \ x) \ \partial \text{lborel}) =$
 $\int^+ x. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * (\sum i. \text{indicator } (g - 'f \ i) \ x) \ \partial \text{lborel}$
by (*intro nn_integral_cong*) (*auto split: split_indicator simp: derivg_nonneg*)
also from *union* **have** $(\lambda x. \sum i. \text{indicator } (g - 'f \ i) \ x :: \text{ennreal}) = (\lambda x. \text{indicator } (\bigcup i. g - 'f \ i) \ x)$
by (*intro ext suminf_indicator*) (*auto simp: disjoint_family_on_def*)
also have $(\int^+ x. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * \dots x \ \partial \text{lborel}) =$
 $(\int^+ x. \text{indicator } (\bigcup i. f \ i) \ (g \ x) * \text{ennreal } (g' \ x) * \text{indicator } \{a..b\} \ x \ \partial \text{lborel})$
by (*intro nn_integral_cong*) (*simp split: split_indicator*)
finally show *?case* .
qed

next

case (*mult f c*)
note $Mf[\text{measurable}] = \langle f \in \text{borel\_measurable borel} \rangle$
let *?I* $= \text{indicator } \{a..b\}$
have $(\lambda x. f \ (g \ x * ?I \ x) * \text{ennreal } (g' \ x * ?I \ x)) \in \text{borel\_measurable borel}$ **using** *Mg Mg'*
by (*intro borel_measurable_times_ennreal measurable_compose[OF _ Mf]*)
(simp_all add: mult.commute set_borel_measurable_def)
also have $(\lambda x. f \ (g \ x * ?I \ x) * \text{ennreal } (g' \ x * ?I \ x)) = (\lambda x. f \ (g \ x) * \text{ennreal } (g' \ x) * ?I \ x)$
by (*intro ext*) (*simp split: split_indicator*)
finally have $Mf': (\lambda x. f \ (g \ x) * \text{ennreal } (g' \ x) * ?I \ x) \in \text{borel\_measurable borel}$

```

.
  with mult show ?case
    by (subst (1 2 3) mult_ac, subst (1 2) nn_integral_cmult) (simp_all add:
mult_ac)

next
  case (add f2 f1)
    let ?I = indicator {a..b}
    {
      fix f :: real  $\Rightarrow$  ennreal assume Mf:  $f \in \text{borel\_measurable borel}$ 
      have  $(\lambda x. f (g x * ?I x) * \text{ennreal } (g' x * ?I x)) \in \text{borel\_measurable borel}$ 
    using Mg Mg'
      by (intro borel_measurable_times_ennreal measurable_compose[OF _ Mf])
        (simp_all add: mult.commute set_borel_measurable_def)
      also have  $(\lambda x. f (g x * ?I x) * \text{ennreal } (g' x * ?I x)) = (\lambda x. f (g x) * \text{ennreal } (g' x) * ?I x)$ 
      by (intro ext) (simp split: split_indicator)
      finally have  $(\lambda x. f (g x) * \text{ennreal } (g' x) * ?I x) \in \text{borel\_measurable borel}$  .
    } note Mf' = this[OF f1  $\in \text{borel\_measurable borel}$ ] this[OF f2  $\in \text{borel\_measurable borel}$ ]

    have  $(\int^+ x. (f1 x + f2 x) * \text{indicator } \{g a..g b\} x \, \partial \text{lborel}) =$ 
       $(\int^+ x. f1 x * \text{indicator } \{g a..g b\} x + f2 x * \text{indicator } \{g a..g b\} x$ 
 $\partial \text{lborel})$ 
    by (intro nn_integral_cong) (simp split: split_indicator)
    also from add have ... =  $(\int^+ x. f1 (g x) * \text{ennreal } (g' x) * \text{indicator } \{a..b\}$ 
 $x \, \partial \text{lborel}) +$ 
       $(\int^+ x. f2 (g x) * \text{ennreal } (g' x) * \text{indicator } \{a..b\} x$ 
 $\partial \text{lborel})$ 
    by (simp_all add: nn_integral_add)
    also from add have ... =  $(\int^+ x. f1 (g x) * \text{ennreal } (g' x) * \text{indicator } \{a..b\}$ 
 $x +$ 
       $f2 (g x) * \text{ennreal } (g' x) * \text{indicator } \{a..b\} x \, \partial \text{lborel})$ 
    by (intro nn_integral_add[symmetric])
      (auto simp add: Mf' derivg_nonneg split: split_indicator)
    also have ... =  $\int^+ x. (f1 (g x) + f2 (g x)) * \text{ennreal } (g' x) * \text{indicator } \{a..b\}$ 
 $x \, \partial \text{lborel}$ 
    by (intro nn_integral_cong) (simp split: split_indicator add: distrib_right)
    finally show ?case .

next
  case (sup F)
    {
      fix i
      let ?I = indicator {a..b}
      have  $(\lambda x. F i (g x * ?I x) * \text{ennreal } (g' x * ?I x)) \in \text{borel\_measurable borel}$ 
    using Mg Mg'
      by (rule_tac borel_measurable_times_ennreal, rule_tac measurable_compose[OF
      _ sup.hyps(i)])

```

```

      (simp_all add: mult.commute set_borel_measurable_def)
    also have  $(\lambda x. F\ i\ (g\ x * ?I\ x) * ennreal\ (g'\ x * ?I\ x)) = (\lambda x. F\ i\ (g\ x) * ennreal\ (g'\ x) * ?I\ x)$ 
    by (intro ext) (simp split: split_indicator)
    finally have  $\dots \in \text{borel\_measurable borel}$  .
  } note  $Mf' = \text{this}$ 

  have  $(\int^{+x}. (SUP\ i. F\ i\ x) * indicator\ \{g\ a..g\ b\}\ x\ \partial lborel) = \int^{+x}. (SUP\ i. F\ i\ x * indicator\ \{g\ a..g\ b\}\ x)\ \partial lborel$ 
  by (intro nn_integral_cong) (simp split: split_indicator)
  also from sup have  $\dots = (SUP\ i. \int^{+x}. F\ i\ x * indicator\ \{g\ a..g\ b\}\ x\ \partial lborel)$ 
  by (intro nn_integral_monotone_convergence_SUP)
  (auto simp: incseq_def le_fun_def split: split_indicator)
  also from sup have  $\dots = (SUP\ i. \int^{+x}. F\ i\ (g\ x) * ennreal\ (g'\ x) * indicator\ \{a..b\}\ x\ \partial lborel)$ 
  by simp
  also from sup have  $\dots = \int^{+x}. (SUP\ i. F\ i\ (g\ x) * ennreal\ (g'\ x) * indicator\ \{a..b\}\ x)\ \partial lborel$ 
  by (intro nn_integral_monotone_convergence_SUP[symmetric])
  (auto simp: incseq_def le_fun_def derivg_nonneg Mf' split: split_indicator intro!: mult_right_mono)
  also from sup have  $\dots = \int^{+x}. (SUP\ i. F\ i\ (g\ x)) * ennreal\ (g'\ x) * indicator\ \{a..b\}\ x\ \partial lborel$ 
  by (subst mult.assoc, subst mult.commute, subst SUP_mult_left_ennreal)
  (auto split: split_indicator simp: derivg_nonneg mult_ac)
  finally show ?case by (simp add: image_comp)
qed
qed

```

theorem nn_integral_substitution:

```

  fixes  $f :: \text{real} \Rightarrow \text{real}$ 
  assumes  $Mf[\text{measurable}]: \text{set\_borel\_measurable borel}\ \{g\ a..g\ b\}\ f$ 
  assumes  $\text{derivg}: \bigwedge x. x \in \{a..b\} \implies (g\ \text{has\_real\_derivative}\ g'\ x)\ (at\ x)$ 
  assumes  $\text{contg'}: \text{continuous\_on}\ \{a..b\}\ g'$ 
  assumes  $\text{derivg\_nonneg}: \bigwedge x. x \in \{a..b\} \implies g'\ x \geq 0$ 
  assumes  $a \leq b$ 
  shows  $(\int^{+x}. f\ x * indicator\ \{g\ a..g\ b\}\ x\ \partial lborel) = (\int^{+x}. f\ (g\ x) * g'\ x * indicator\ \{a..b\}\ x\ \partial lborel)$ 
proof (cases  $a = b$ )
  assume  $a \neq b$ 
  with  $\langle a \leq b \rangle$  have  $a < b$  by auto
  let  $?f' = \lambda x. f\ x * indicator\ \{g\ a..g\ b\}\ x$ 

  from derivg derivg_nonneg have  $\text{monog}: \bigwedge x\ y. a \leq x \implies x \leq y \implies y \leq b \implies g\ x \leq g\ y$ 
  by (rule derivg_nonneg_imp_mono) simp_all
  have  $\text{bounds}: \bigwedge x. x \geq a \implies x \leq b \implies g\ x \geq g\ a \bigwedge x. x \geq a \implies x \leq b \implies g\ x \leq g\ b$ 
  by (auto intro: monog)

```

```

from derivg_nonneg have nonneg:
   $\bigwedge f x. x \geq a \implies x \leq b \implies g' x \neq 0 \implies f x * \text{ennreal } (g' x) \geq 0 \implies f x \geq 0$ 
by (force simp: field_simps)
have nonneg':  $\bigwedge x. a \leq x \implies x \leq b \implies \neg 0 \leq f (g x) \implies 0 \leq f (g x) * g' x$ 
 $\implies g' x = 0$ 
by (metis atLeastAtMost_iff derivg_nonneg eq_iff mult_eq_0_iff mult_le_0_iff)

have  $(\int^+ x. f x * \text{indicator } \{g a..g b\} x \, \partial \text{lborel}) =$ 
 $(\int^+ x. \text{ennreal } (?f' x) * \text{indicator } \{g a..g b\} x \, \partial \text{lborel})$ 
by (intro nn_integral_cong)
  (auto split: split_indicator split_max simp: zero_ennreal.rep_eq ennreal_neg)
also have ... =  $\int^+ x. ?f' (g x) * \text{ennreal } (g' x) * \text{indicator } \{a..b\} x \, \partial \text{lborel}$ 
using Mf
by (subst nn_integral_substitution_aux[OF _ _ derivg contg' derivg_nonneg
  <a < b>])
  (auto simp add: mult.commute set_borel_measurable_def)
also have ... =  $\int^+ x. f (g x) * \text{ennreal } (g' x) * \text{indicator } \{a..b\} x \, \partial \text{lborel}$ 
by (intro nn_integral_cong) (auto split: split_indicator simp: max_def dest:
  bounds)
also have ... =  $\int^+ x. \text{ennreal } (f (g x) * g' x * \text{indicator } \{a..b\} x) \, \partial \text{lborel}$ 
by (intro nn_integral_cong) (auto simp: mult.commute derivg_nonneg en-
  nreal_mult' split: split_indicator)
finally show ?thesis .
qed auto

```

theorem integral_substitution:

```

assumes integrable: set_integrable lborel  $\{g a..g b\} f$ 
assumes derivg:  $\bigwedge x. x \in \{a..b\} \implies (g \text{ has\_real\_derivative } g' x) \text{ (at } x)$ 
assumes contg': continuous_on  $\{a..b\} g'$ 
assumes derivg_nonneg:  $\bigwedge x. x \in \{a..b\} \implies g' x \geq 0$ 
assumes a ≤ b
shows set_integrable lborel  $\{a..b\} (\lambda x. f (g x) * g' x)$ 
and  $(\text{LBINT } x. f x * \text{indicator } \{g a..g b\} x) = (\text{LBINT } x. f (g x) * g' x * \text{indicator } \{a..b\} x)$ 

```

proof—

```

from derivg have contg: continuous_on  $\{a..b\} g$  by (rule has_real_derivative_imp_continuous_on)
with contg' have Mg: set_borel_measurable borel  $\{a..b\} g$ 
and Mg': set_borel_measurable borel  $\{a..b\} g'$ 
by (simp_all only: set_measurable_continuous_on_ivl)
from derivg derivg_nonneg have monog:  $\bigwedge x y. a \leq x \implies x \leq y \implies y \leq b \implies$ 
 $g x \leq g y$ 
by (rule deriv_nonneg_imp_mono) simp_all

```

```

have  $(\lambda x. \text{ennreal } (f x) * \text{indicator } \{g a..g b\} x) =$ 
 $(\lambda x. \text{ennreal } (f x * \text{indicator } \{g a..g b\} x))$ 
by (intro ext) (simp split: split_indicator)
with integrable have M1:  $(\lambda x. f x * \text{indicator } \{g a..g b\} x) \in \text{borel\_measurable borel}$ 

```

```

    by (force simp: mult.commute set_integrable_def)
    from integrable have M2:  $(\lambda x. -f x * \text{indicator } \{g \text{ a..} g \text{ b}\} x) \in \text{borel\_measurable borel}$ 
    by (force simp: mult.commute set_integrable_def)

    have LBINT  $x. (f x :: \text{real}) * \text{indicator } \{g \text{ a..} g \text{ b}\} x =$ 
      enn2real  $(\int^+ x. \text{ennreal } (f x) * \text{indicator } \{g \text{ a..} g \text{ b}\} x \partial \text{lborel}) -$ 
      enn2real  $(\int^+ x. \text{ennreal } (-f x) * \text{indicator } \{g \text{ a..} g \text{ b}\} x \partial \text{lborel})$  using
    integrable
      unfolding set_integrable_def
      by (subst real_lebesgue_integral_def) (simp_all add: nn_integral_set_ennreal
    mult.commute)
    also have *:  $(\int^+ x. \text{ennreal } (f x) * \text{indicator } \{g \text{ a..} g \text{ b}\} x \partial \text{lborel}) =$ 
       $(\int^+ x. \text{ennreal } (f x * \text{indicator } \{g \text{ a..} g \text{ b}\} x) \partial \text{lborel})$ 
    by (intro nn_integral_cong) (simp split: split_indicator)
    also from M1 * have A:  $(\int^+ x. \text{ennreal } (f x * \text{indicator } \{g \text{ a..} g \text{ b}\} x) \partial \text{lborel}) =$ 
       $(\int^+ x. \text{ennreal } (f (g x) * g' x * \text{indicator } \{a..b\} x) \partial \text{lborel})$ 
    by (subst nn_integral_substitution[OF derivg contg' derivg_nonneg  $\langle a \leq b \rangle$ ])
      (auto simp: nn_integral_set_ennreal mult.commute set_borel_measurable_def)
    also have **:  $(\int^+ x. \text{ennreal } (-f x) * \text{indicator } \{g \text{ a..} g \text{ b}\} x \partial \text{lborel}) =$ 
       $(\int^+ x. \text{ennreal } (-f x) * \text{indicator } \{g \text{ a..} g \text{ b}\} x \partial \text{lborel})$ 
    by (intro nn_integral_cong) (simp split: split_indicator)
    also from M2 ** have B:  $(\int^+ x. \text{ennreal } (-f x) * \text{indicator } \{g \text{ a..} g \text{ b}\} x \partial \text{lborel}) =$ 
       $(\int^+ x. \text{ennreal } (-f (g x)) * g' x * \text{indicator } \{a..b\} x \partial \text{lborel})$ 
    by (subst nn_integral_substitution[OF derivg contg' derivg_nonneg  $\langle a \leq b \rangle$ ])
      (auto simp: nn_integral_set_ennreal mult.commute set_borel_measurable_def)

    also {
      from integrable have Mf:  $\text{set\_borel\_measurable borel } \{g \text{ a..} g \text{ b}\} f$ 
      unfolding set_borel_measurable_def set_integrable_def by simp
      from measurable_compose Mg Mf Mg' borel_measurable_times
      have  $(\lambda x. f (g x) * \text{indicator } \{a..b\} x) * \text{indicator } \{g \text{ a..} g \text{ b}\} (g x * \text{indicator } \{a..b\} x) *$ 
         $(g' x * \text{indicator } \{a..b\} x) \in \text{borel\_measurable borel}$  (is ?f  $\in$  _)
      by (simp add: mult.commute set_borel_measurable_def)
      also have ?f =  $(\lambda x. f (g x) * g' x * \text{indicator } \{a..b\} x)$ 
      using monog by (intro ext) (auto split: split_indicator)
      finally show  $\text{set\_integrable lborel } \{a..b\} (\lambda x. f (g x) * g' x)$ 
      using A B integrable unfolding real_integrable_def set_integrable_def
      by (simp_all add: nn_integral_set_ennreal mult.commute)
    } note integrable' = this

    have enn2real  $(\int^+ x. \text{ennreal } (f (g x) * g' x * \text{indicator } \{a..b\} x) \partial \text{lborel}) -$ 
      enn2real  $(\int^+ x. \text{ennreal } (-f (g x) * g' x * \text{indicator } \{a..b\} x) \partial \text{lborel}) =$ 
       $(\text{LBINT } x. f (g x) * g' x * \text{indicator } \{a..b\} x)$ 
    using integrable' unfolding set_integrable_def

```

```

    by (subst real_lebesgue_integral_def) (simp_all add: field_simps)
  finally show (LBINT x. f x * indicator {g a..g b} x) =
    (LBINT x. f (g x) * g' x * indicator {a..b} x) .
qed

theorem interval_integral_substitution:
  assumes integrable: set_integrable lborel {g a..g b} f
  assumes derivg:  $\bigwedge x. x \in \{a..b\} \implies (g \text{ has\_real\_derivative } g' x) \text{ (at } x)$ 
  assumes contg': continuous_on {a..b} g'
  assumes derivg_nonneg:  $\bigwedge x. x \in \{a..b\} \implies g' x \geq 0$ 
  assumes a ≤ b
  shows set_integrable lborel {a..b} ( $\lambda x. f (g x) * g' x$ )
    and (LBINT x=g a..g b. f x) = (LBINT x=a..b. f (g x) * g' x)
  apply (rule integral_substitution[OF assms], simp, simp)
  apply (subst (1 2) interval_integral_Icc, fact)
  apply (rule deriv_nonneg_imp_mono[OF derivg derivg_nonneg], simp, simp,
fact)
  using integral_substitution(2)[OF assms]
  apply (simp add: mult.commute set_lebesgue_integral_def)
  done

lemma set_borel_integrable_singleton[simp]: set_integrable lborel {x} (f :: real  $\Rightarrow$ 
real)
  unfolding set_integrable_def
  by (subst integrable_discrete_difference[where X={x} and g= $\lambda \_. 0$ ]) auto

end

```

6.25 The Volume of an n -Dimensional Ball

```

theory Ball_Volume
  imports Gamma_Function Lebesgue_Integral_Substitution
begin

```

We define the volume of the unit ball in terms of the Gamma function. Note that the dimension need not be an integer; we also allow fractional dimensions, although we do not use this case or prove anything about it for now.

```

definition unit_ball_vol :: real  $\Rightarrow$  real where
  unit_ball_vol n = pi powr (n / 2) / Gamma (n / 2 + 1)

```

```

lemma unit_ball_vol_pos [simp]:  $n \geq 0 \implies \text{unit\_ball\_vol } n > 0$ 
  by (force simp: unit_ball_vol_def intro: divide_nonneg_pos)

```

```

lemma unit_ball_vol_nonneg [simp]:  $n \geq 0 \implies \text{unit\_ball\_vol } n \geq 0$ 
  by (simp add: dual_order.strict_implies_order)

```

We first need the value of the following integral, which is at the core of

computing the measure of an $n + 1$ -dimensional ball in terms of the measure of an n -dimensional one.

lemma *emeasure_cball_aux_integral*:

$$(\int^+ x. \text{indicator } \{-1..1\} x * \text{sqrt } (1 - x^2) \wedge n \partial \text{lborel}) = \text{ennreal } (\text{Beta } (1 / 2) (\text{real } n / 2 + 1))$$

proof –

$$\text{have } ((\lambda t. t \text{ powr } (-1 / 2) * (1 - t) \text{ powr } (\text{real } n / 2)) \text{ has\_integral } \text{Beta } (1 / 2) (\text{real } n / 2 + 1)) \{0..1\}$$

using *has_integral_Beta_real*[of $1/2 \ n \ / \ 2 + 1$] **by** *simp*

from *nn_integral_has_integral_lebesgue*[OF *this*] **have**

$$\text{ennreal } (\text{Beta } (1 / 2) (\text{real } n / 2 + 1)) =$$

$$\text{nn\_integral lborel } (\lambda t. \text{ennreal } (t \text{ powr } (-1 / 2) * (1 - t) \text{ powr } (\text{real } n / 2) * \text{indicator } \{0..1\} t))$$

by (*simp add: mult_ac ennreal_mult' ennreal_indicator*)

$$\text{also have } \dots = (\int^+ x. \text{ennreal } (x^2 \text{ powr } (-1 / 2) * (1 - x^2) \text{ powr } (\text{real } n / 2) * (2 * x) * \text{indicator } \{0..1\} x) \partial \text{lborel})$$

by (*subst nn_integral_substitution*[**where** $g = \lambda x. x^2$ **and** $g' = \lambda x. 2 * x$])
(*auto intro!: derivative_eq_intros continuous_intros simp: set_borel_measurable_def*)

$$\text{also have } \dots = (\int^+ x. 2 * \text{ennreal } ((1 - x^2) \text{ powr } (\text{real } n / 2) * \text{indicator } \{0..1\} x) \partial \text{lborel})$$

by (*intro nn_integral_cong_AE AE_I*[of $\_ \{0\}$])

$$(\text{auto simp: indicator_def powr_minus powr_half_sqrt field_split_simps ennreal_mult'})$$

$$\text{also have } \dots = (\int^+ x. \text{ennreal } ((1 - x^2) \text{ powr } (\text{real } n / 2) * \text{indicator } \{0..1\} x) \partial \text{lborel}) +$$

$$(\int^+ x. \text{ennreal } ((1 - x^2) \text{ powr } (\text{real } n / 2) * \text{indicator } \{0..1\} x) \partial \text{lborel})$$

(*is $\_ = ?I + \_$*) **by** (*simp add: mult_2 nn_integral_add*)

$$\text{also have } ?I = (\int^+ x. \text{ennreal } ((1 - x^2) \text{ powr } (\text{real } n / 2) * \text{indicator } \{-1..0\} x) \partial \text{lborel})$$

by (*subst nn_integral_real_affine*[of $\_ -1 \ 0$])

(*auto simp: indicator_def intro!: nn_integral_cong*)

hence $?I + ?I = \dots + ?I$ **by** *simp*

$$\text{also have } \dots = (\int^+ x. \text{ennreal } ((1 - x^2) \text{ powr } (\text{real } n / 2) * \text{indicator } \{-1..0\} x + \text{indicator } \{0..1\} x) \partial \text{lborel})$$

by (*subst nn_integral_add* [*symmetric*]) (*auto simp: algebra_simps*)

$$\text{also have } \dots = (\int^+ x. \text{ennreal } ((1 - x^2) \text{ powr } (\text{real } n / 2) * \text{indicator } \{-1..1\} x) \partial \text{lborel})$$

by (*intro nn_integral_cong_AE AE_I*[of $\_ \{0\}$]) (*auto simp: indicator_def*)

$$\text{also have } \dots = (\int^+ x. \text{ennreal } (\text{indicator } \{-1..1\} x * \text{sqrt } (1 - x^2) \wedge n) \partial \text{lborel})$$

by (*intro nn_integral_cong_AE AE_I*[of $\_ \{1, -1\}$])

(*auto simp: powr_half_sqrt* [*symmetric*] *indicator_def abs_square_le_1*

abs_square_eq_1 powr_def exp_of_nat_mult [*symmetric*] *emeasure_lborel_countable*)

finally show *?thesis* ..

qed

lemma *real_sqrt_le_iff'*: $x \geq 0 \implies y \geq 0 \implies \text{sqrt } x \leq y \iff x \leq y^2$
using *real_le_sqrt sqrt_le_D* **by** *blast*

Isabelle's type system makes it very difficult to do an induction over the dimension of a Euclidean space type, because the type would change in the inductive step. To avoid this problem, we instead formulate the problem in a more concrete way by unfolding the definition of the Euclidean norm.

lemma *emeasure_cball_aux*:
assumes *finite A r > 0*
shows $\text{emeasure } (Pi_M \ A \ (\lambda_. \text{lborel}))$
 $(\{f. \text{sqrt } (\sum_{i \in A. (f \ i)^2} \leq r)\} \cap \text{space } (Pi_M \ A \ (\lambda_. \text{lborel}))) =$
 $\text{ennreal } (\text{unit\_ball\_vol } (\text{real } (\text{card } A)) * r^{\text{card } A})$
using *assms*
proof (*induction arbitrary: r*)
case (*empty r*)
thus *?case*
by (*simp add: unit_ball_vol_def space_PiM*)
next
case (*insert i A r*)
interpret *product_sigma_finite* $\lambda_. \text{lborel}$
by *standard*
have $\text{emeasure } (Pi_M \ (\text{insert } i \ A) \ (\lambda_. \text{lborel}))$
 $(\{f. \text{sqrt } (\sum_{i \in \text{insert } i \ A. (f \ i)^2} \leq r)\} \cap \text{space } (Pi_M \ (\text{insert } i \ A) \ (\lambda_.$
 $\text{lborel}))) =$
 $\text{nn\_integral } (Pi_M \ (\text{insert } i \ A) \ (\lambda_. \text{lborel}))$
 $(\text{indicator } (\{f. \text{sqrt } (\sum_{i \in \text{insert } i \ A. (f \ i)^2} \leq r)\} \cap$
 $\text{space } (Pi_M \ (\text{insert } i \ A) \ (\lambda_. \text{lborel}))))$
by (*subst nn_integral_indicator*) *auto*
also have $\dots = (\int^+ y. \int^+ x. \text{indicator } (\{f. \text{sqrt } ((f \ i)^2 + (\sum_{i \in A. (f \ i)^2})) \leq$
 $r\} \cap$
 $\text{space } (Pi_M \ (\text{insert } i \ A) \ (\lambda_. \text{lborel}))) (x(i := y))$
 $\partial Pi_M \ A \ (\lambda_. \text{lborel}) \ \partial \text{lborel})$
using *insert.premis insert.hyps* **by** (*subst product_nn_integral_insert_rev*) *auto*
also have $\dots = (\int^+ (y::\text{real}). \int^+ x. \text{indicator } \{-r..r\} \ y * \text{indicator } (\{f. \text{sqrt}$
 $((\sum_{i \in A. (f \ i)^2}) \leq$
 $\text{sqrt } (r^2 - y^2)\} \cap \text{space } (Pi_M \ A \ (\lambda_. \text{lborel}))) \ x \ \partial Pi_M \ A \ (\lambda_.$
 $\text{lborel}) \ \partial \text{lborel})$
proof (*intro nn_integral_cong, goal_cases*)
case (*1 y f*)
have $*$: $y \in \{-r..r\}$ **if** $y^2 + c \leq r^2 \ c \geq 0$ **for** c
proof $-$
have $y^2 \leq y^2 + c$ **using** *that* **by** *simp*
also have $\dots \leq r^2$ **by** *fact*
finally show *?thesis*
using $\langle r > 0 \rangle$ **by** (*simp add: power2_le_iff_abs_le abs_if_split: if_splits*)
qed
have $(\sum_{x \in A. (if \ x = i \ then \ y \ else \ f \ x)^2}) = (\sum_{x \in A. (f \ x)^2})$
using *insert.hyps* **by** (*intro sum.cong*) *auto*
thus *?case* **using** $1 \ \langle r > 0 \rangle$

```

    by (auto simp: sum_nonneg real_sqrt_le_iff' indicator_def PiE_def space_PiM
dest!: *)
  qed
  also have ... = ( $\int^+ (y::real). \text{indicator } \{-r..r\} y * (\int^+ x. \text{indicator } (\{f. \text{sqrt} ((\sum_{i \in A}. (f i)^2)) \leq \text{sqrt } (r^2 - y^2)\} \cap \text{space } (Pi_M A (\lambda_. \text{lborel}))) x$ 
 $\partial Pi_M A (\lambda_. \text{lborel})) \partial \text{lborel}$ ) by (subst nn_integral_cmult) auto
  also have ... = ( $\int^+ (y::real). \text{indicator } \{-r..r\} y * \text{emeasure } (Pi_M A (\lambda_. \text{lborel})) (\{f. \text{sqrt} ((\sum_{i \in A}. (f i)^2)) \leq \text{sqrt } (r^2 - y^2)\} \cap \text{space } (Pi_M A (\lambda_. \text{lborel})) \partial \text{lborel}))$ 
 $\partial \text{lborel}$ )
  using ⟨finite A⟩ by (intro nn_integral_cong, subst nn_integral_indicator) auto
  also have ... = ( $\int^+ (y::real). \text{indicator } \{-r..r\} y * \text{ennreal } (\text{unit\_ball\_vol } (\text{real } (\text{card } A))) * (\text{sqrt } (r^2 - y^2))^{\text{card } A} \partial \text{lborel}$ )
  proof (intro nn_integral_cong_AE, goal_cases)
    case 1
    have AE y in lborel.  $y \notin \{-r, r\}$ 
    by (intro AE_not_in countable_imp_null_set_lborel) auto
    thus ?case
    proof eventually_elim
      case (elim y)
      show ?case
      proof (cases  $y \in \{-r <..< r\}$ )
        case True
        hence  $y^2 < r^2$  by (subst real_sqrt_less_iff [symmetric]) auto
        thus ?thesis by (subst insert.IH) (auto)
      qed (insert elim, auto)
    qed
  qed
  also have ... =  $\text{ennreal } (\text{unit\_ball\_vol } (\text{real } (\text{card } A))) * (\int^+ (y::real). \text{indicator } \{-r..r\} y * (\text{sqrt } (r^2 - y^2))^{\text{card } A} \partial \text{lborel})$ 
  by (subst nn_integral_cmult [symmetric])
  (auto simp: mult_ac ennreal_mult' [symmetric] indicator_def intro!: nn_integral_cong)
  also have ( $\int^+ (y::real). \text{indicator } \{-r..r\} y * (\text{sqrt } (r^2 - y^2))^{\text{card } A} \partial \text{lborel}$ ) =
  ( $\int^+ (y::real). r^{\text{card } A} * \text{indicator } \{-1..1\} y * (\text{sqrt } (1 - y^2))^{\text{card } A}$ 
 $\partial (\text{distr lborel borel } ((*) (1/r))))$  using ⟨r > 0⟩
  by (subst nn_integral_distr)
  (auto simp: indicator_def field_simps real_sqrt_divide intro!: nn_integral_cong)
  also have ... = ( $\int^+ x. \text{ennreal } (r^{\text{Suc } (\text{card } A)}) * (\text{indicator } \{-1..1\} x * \text{sqrt } (1 - x^2))^{\text{card } A} \partial \text{lborel}$ ) using ⟨r > 0⟩
  by (subst lborel_distr_mult) (auto simp: nn_integral_density ennreal_mult' [symmetric] mult_ac)
  also have ... =  $\text{ennreal } (r^{\text{Suc } (\text{card } A)}) * (\int^+ x. \text{indicator } \{-1..1\} x * \text{sqrt } (1 - x^2))^{\text{card } A} \partial \text{lborel}$ 

```

```

    by (subst nn_integral_cmult) auto
  also note emeasure_cball_aux_integral
  also have ennreal (unit_ball_vol (real (card A))) * (ennreal (r ^ Suc (card A))
  *
    ennreal (Beta (1/2) (card A / 2 + 1))) =
    ennreal (unit_ball_vol (card A) * Beta (1/2) (card A / 2 + 1) * r ^
  Suc (card A))
  using <r > 0> by (simp add: ennreal_mult' [symmetric] mult_ac)
  also have unit_ball_vol (card A) * Beta (1/2) (card A / 2 + 1) = unit_ball_vol
  (Suc (card A))
  by (auto simp: unit_ball_vol_def Beta_def Gamma_eq_zero_iff field_simps
    Gamma_one_half_real powr_half_sqrt [symmetric] powr_add [symmetric])
  also have Suc (card A) = card (insert i A) using insert.hyps by simp
  finally show ?case .
qed

```

We now get the main theorem very easily by just applying the above lemma.

context

fixes $c :: 'a :: \text{euclidean\_space}$ **and** $r :: \text{real}$

assumes $r: r \geq 0$

begin

theorem *emeasure_cball*:

$\text{emeasure lborel } (\text{cball } c \ r) = \text{ennreal } (\text{unit\_ball\_vol } (\text{DIM}('a)) * r ^ \text{DIM}('a))$

proof (cases $r = 0$)

case *False*

with r **have** $r: r > 0$ **by** *simp*

have $(\text{lborel} :: 'a \text{ measure}) =$

$\text{distr } (\text{Pi}_M \text{ Basis } (\lambda_. \text{lborel})) \text{ borel } (\lambda f. \sum_{b \in \text{Basis}. f \ b *_{\mathbb{R}} \ b)$

by (rule *lborel_eq*)

also have $\text{emeasure} \dots (\text{cball } 0 \ r) =$

$\text{emeasure } (\text{Pi}_M \text{ Basis } (\lambda_. \text{lborel}))$

$(\{y. \text{dist } 0 (\sum_{b \in \text{Basis}. y \ b *_{\mathbb{R}} \ b :: 'a) \leq r\} \cap \text{space } (\text{Pi}_M \text{ Basis } (\lambda_.$

$\text{lborel}))$

by (subst *emeasure_distr*) (auto simp: *cball_def*)

also have $\{f. \text{dist } 0 (\sum_{b \in \text{Basis}. f \ b *_{\mathbb{R}} \ b :: 'a) \leq r\} = \{f. \text{sqrt } (\sum_{i \in \text{Basis}. (f \ i)^2) \leq r\}$

by (subst *euclidean_dist_l2*) (auto simp: *L2_set_def*)

also have $\text{emeasure } (\text{Pi}_M \text{ Basis } (\lambda_. \text{lborel})) (\dots \cap \text{space } (\text{Pi}_M \text{ Basis } (\lambda_.$

$\text{lborel})) =$

$\text{ennreal } (\text{unit\_ball\_vol } (\text{real } \text{DIM}('a)) * r ^ \text{DIM}('a))$

using r **by** (subst *emeasure_cball_aux*) *simp_all*

also have $\text{emeasure lborel } (\text{cball } 0 \ r :: 'a \text{ set}) =$

$\text{emeasure } (\text{distr lborel borel } (\lambda x. c + x)) (\text{cball } c \ r)$

by (subst *emeasure_distr*) (auto simp: *cball_def* *dist_norm* *norm_minus_commute*)

also have $\text{distr lborel borel } (\lambda x. c + x) = \text{lborel}$

using *lborel_affine*[of 1 c] **by** (simp add: *density_1*)

finally show *?thesis* .

qed *auto*

corollary *content_cball*:

content (*cball* *c* *r*) = *unit_ball_vol* (*DIM*('a)) * *r* ^ *DIM*('a)
by (*simp* *add*: *measure_def* *emeasure_cball* *r*)

corollary *emeasure_ball*:

emeasure *lborel* (*ball* *c* *r*) = *ennreal* (*unit_ball_vol* (*DIM*('a)) * *r* ^ *DIM*('a))

proof –

from *negligible_sphere*[*of* *c* *r*] **have** *sphere* *c* *r* ∈ *null_sets* *lborel*

by (*auto* *simp*: *null_sets_completion_iff_negligible_iff_null_sets_negligible_convex_frontier*)

hence *emeasure* *lborel* (*ball* *c* *r* ∪ *sphere* *c* *r* :: 'a set) = *emeasure* *lborel* (*ball* *c*

r :: 'a set)

by (*intro* *emeasure_Un_null_set*) *auto*

also have *ball* *c* *r* ∪ *sphere* *c* *r* = (*cball* *c* *r* :: 'a set) **by** *auto*

also have *emeasure* *lborel* ... = *ennreal* (*unit_ball_vol* (*real* *DIM*('a)) * *r* ^ *DIM*('a))

by (*rule* *emeasure_cball*)

finally show ?thesis ..

qed

corollary *content_ball*:

content (*ball* *c* *r*) = *unit_ball_vol* (*DIM*('a)) * *r* ^ *DIM*('a)

by (*simp* *add*: *measure_def* *r* *emeasure_ball*)

end

Lastly, we now prove some nicer explicit formulas for the volume of the unit balls in the cases of even and odd integer dimensions.

lemma *unit_ball_vol_even*:

unit_ball_vol (*real* (*2* * *n*)) = *pi* ^ *n* / *fact* *n*

by (*simp* *add*: *unit_ball_vol_def* *add_ac* *powr_realpow* *Gamma_fact*)

lemma *unit_ball_vol_odd'*:

unit_ball_vol (*real* (*2* * *n* + *1*)) = *pi* ^ *n* / *pochhammer* (*1* / *2*) (*Suc* *n*)

and *unit_ball_vol_odd*:

unit_ball_vol (*real* (*2* * *n* + *1*)) =
 (*2* ^ (*2* * *Suc* *n*) * *fact* (*Suc* *n*)) / *fact* (*2* * *Suc* *n*) * *pi* ^ *n*

proof –

have *unit_ball_vol* (*real* (*2* * *n* + *1*)) =

pi *powr* (*real* *n* + *1* / *2*) / *Gamma* (*1* / *2* + *real* (*Suc* *n*))

by (*simp* *add*: *unit_ball_vol_def* *field_simps*)

also have *pochhammer* (*1* / *2*) (*Suc* *n*) = *Gamma* (*1* / *2* + *real* (*Suc* *n*)) / *Gamma* (*1* / *2*)

by (*intro* *pochhammer_Gamma*) *auto*

hence *Gamma* (*1* / *2* + *real* (*Suc* *n*)) = *sqrt* *pi* * *pochhammer* (*1* / *2*) (*Suc* *n*)

by (*simp* *add*: *Gamma_one_half_real*)

also have *pi* *powr* (*real* *n* + *1* / *2*) / ... = *pi* ^ *n* / *pochhammer* (*1* / *2*) (*Suc* *n*)

by (*simp* *add*: *powr_add* *powr_half_sqrt* *powr_realpow*)

```

finally show unit_ball_vol (real (2 * n + 1)) = ... .
also have pochhammer (1 / 2 :: real) (Suc n) =
  fact (2 * Suc n) / (2 ^ (2 * Suc n) * fact (Suc n))
  using fact_double[of Suc n, where ?'a = real] by (simp add: divide_simps
mult_ac)
also have pi ^ n / ... = (2 ^ (2 * Suc n) * fact (Suc n)) / fact (2 * Suc n) *
pi ^ n
  by simp
finally show unit_ball_vol (real (2 * n + 1)) = ... .
qed

```

```

lemma unit_ball_vol_numeral:
  unit_ball_vol (numeral (Num.Bit0 n)) = pi ^ numeral n / fact (numeral n) (is
?th1)
  unit_ball_vol (numeral (Num.Bit1 n)) = 2 ^ (2 * Suc (numeral n)) * fact (Suc
(numeral n)) /
  fact (2 * Suc (numeral n)) * pi ^ numeral n (is ?th2)
proof -
  have numeral (Num.Bit0 n) = (2 * numeral n :: nat)
    by (simp only: numeral_Bit0 mult_2 ring_distrib)
  also have unit_ball_vol ... = pi ^ numeral n / fact (numeral n)
    by (rule unit_ball_vol_even)
  finally show ?th1 by simp
next
  have numeral (Num.Bit1 n) = (2 * numeral n + 1 :: nat)
    by (simp only: numeral_Bit1 mult_2)
  also have unit_ball_vol ... = 2 ^ (2 * Suc (numeral n)) * fact (Suc (numeral
n)) /
    fact (2 * Suc (numeral n)) * pi ^ numeral n
    by (rule unit_ball_vol_odd)
  finally show ?th2 by simp
qed

```

lemmas eval_unit_ball_vol = unit_ball_vol_numeral fact_numeral

Just for fun, we compute the volume of unit balls for a few dimensions.

```

lemma unit_ball_vol_0 [simp]: unit_ball_vol 0 = 1
  using unit_ball_vol_even[of 0] by simp

```

```

lemma unit_ball_vol_1 [simp]: unit_ball_vol 1 = 2
  using unit_ball_vol_odd[of 0] by simp

```

corollary

```

  unit_ball_vol_2: unit_ball_vol 2 = pi
  and unit_ball_vol_3: unit_ball_vol 3 = 4 / 3 * pi
  and unit_ball_vol_4: unit_ball_vol 4 = pi^2 / 2
  and unit_ball_vol_5: unit_ball_vol 5 = 8 / 15 * pi^2
by (simp_all add: eval_unit_ball_vol)

```

corollary *circle_area*:

$r \geq 0 \implies \text{content } (\text{ball } c \ r :: (\text{real}^2) \text{ set}) = r^2 * \pi$
by (*simp add: content_ball unit_ball_vol_2*)

corollary *sphere_volume*:

$r \geq 0 \implies \text{content } (\text{ball } c \ r :: (\text{real}^3) \text{ set}) = 4 / 3 * r^3 * \pi$
by (*simp add: content_ball unit_ball_vol_3*)

Useful equivalent forms

corollary *content_ball_eq_0_iff* [*simp*]: $\text{content } (\text{ball } c \ r) = 0 \longleftrightarrow r \leq 0$

proof –

have $r > 0 \implies \text{content } (\text{ball } c \ r) > 0$
by (*simp add: content_ball unit_ball_vol_def*)
then show *?thesis*
by (*fastforce simp: ball_empty*)

qed

corollary *content_ball_gt_0_iff* [*simp*]: $0 < \text{content } (\text{ball } z \ r) \longleftrightarrow 0 < r$

by (*auto simp: zero_less_measure_iff*)

corollary *content_cball_eq_0_iff* [*simp*]: $\text{content } (\text{cball } c \ r) = 0 \longleftrightarrow r \leq 0$

proof (*cases r = 0*)

case *False*

moreover have $r > 0 \implies \text{content } (\text{cball } c \ r) > 0$
by (*simp add: content_cball unit_ball_vol_def*)
ultimately show *?thesis*
by *fastforce*

qed *auto*

corollary *content_cball_gt_0_iff* [*simp*]: $0 < \text{content } (\text{cball } z \ r) \longleftrightarrow 0 < r$

by (*auto simp: zero_less_measure_iff*)

end

6.26 Integral Test for Summability

theory *Integral_Test*

imports *Henstock_Kurzweil_Integration*

begin

The integral test for summability. We show here that for a decreasing non-negative function, the infinite sum over that function evaluated at the natural numbers converges iff the corresponding integral converges.

As a useful side result, we also provide some results on the difference between the integral and the partial sum. (This is useful e.g. for the definition of the Euler-Mascheroni constant)

locale *antimono_fun_sum_integral_diff* =

fixes $f :: \text{real} \Rightarrow \text{real}$

```

    assumes dec:  $\bigwedge x y. x \geq 0 \implies x \leq y \implies f x \geq f y$ 
    assumes nonneg:  $\bigwedge x. x \geq 0 \implies f x \geq 0$ 
    assumes cont: continuous_on {0..} f
begin

definition sum_integral_diff_series n =  $(\sum k \leq n. f (of\_nat k)) - (integral \{0..of\_nat n\} f)$ 

lemma sum_integral_diff_series_nonneg:
  sum_integral_diff_series n  $\geq 0$ 
proof -
  note int = integrable_continuous_real[OF continuous_on_subset[OF cont]]
  let ?int =  $\lambda a b. integral \{of\_nat a..of\_nat b\} f$ 
  have -sum_integral_diff_series n = ?int 0 n -  $(\sum k \leq n. f (of\_nat k))$ 
    by (simp add: sum_integral_diff_series_def)
  also have ?int 0 n =  $(\sum k < n. ?int k (Suc k))$ 
  proof (induction n)
    case (Suc n)
    have ?int 0 (Suc n) = ?int 0 n + ?int n (Suc n)
      by (intro integral_combine[symmetric] int) simp_all
    with Suc show ?case by simp
  qed simp_all
  also have ...  $\leq (\sum k < n. integral \{of\_nat k..of\_nat (Suc k)\} (\lambda\_::real. f (of\_nat k)))$ 
    by (intro sum_mono integral_le int) (auto intro: dec)
  also have ... =  $(\sum k < n. f (of\_nat k))$  by simp
  also have ... -  $(\sum k \leq n. f (of\_nat k)) = -(\sum k \in \{..n\} - \{..<n\}. f (of\_nat k))$ 
    by (subst sum_diff) auto
  also have ...  $\leq 0$  by (auto intro!: sum_nonneg nonneg)
  finally show sum_integral_diff_series n  $\geq 0$  by simp
qed

lemma sum_integral_diff_series_antimono:
  assumes m  $\leq$  n
  shows sum_integral_diff_series m  $\geq$  sum_integral_diff_series n
proof -
  let ?int =  $\lambda a b. integral \{of\_nat a..of\_nat b\} f$ 
  note int = integrable_continuous_real[OF continuous_on_subset[OF cont]]
  have d_mono: sum_integral_diff_series (Suc n)  $\leq$  sum_integral_diff_series n
for n
proof -
  fix n :: nat
  have sum_integral_diff_series (Suc n) - sum_integral_diff_series n =
     $f (of\_nat (Suc n)) + (?int 0 n - ?int 0 (Suc n))$ 
    unfolding sum_integral_diff_series_def by (simp add: algebra_simps)
  also have ?int 0 n - ?int 0 (Suc n) =  $-?int n (Suc n)$ 
    by (subst integral_combine [symmetric, of of_nat 0 of_nat n of_nat (Suc n)])
    (auto intro!: int simp: algebra_simps)

```

```

    also have ?int n (Suc n) ≥ integral {of_nat n..of_nat (Suc n)} (λ_::real. f
(of_nat (Suc n)))
    by (intro integral_le int) (auto intro: dec)
    hence f (of_nat (Suc n)) + -?int n (Suc n) ≤ 0 by (simp add: algebra_simps)
    finally show sum_integral_diff_series (Suc n) ≤ sum_integral_diff_series n
by simp
qed
with assms show ?thesis
by (induction rule: inc_induct) (auto intro: order.trans[OF d_mono])
qed

```

```

lemma sum_integral_diff_series_Bseq: Bseq sum_integral_diff_series
proof -
  from sum_integral_diff_series_nonneg and sum_integral_diff_series_antimono
  have norm (sum_integral_diff_series n) ≤ sum_integral_diff_series 0 for n
by simp
  thus Bseq sum_integral_diff_series by (rule BseqI')
qed

```

```

lemma sum_integral_diff_series_monoseq: monoseq sum_integral_diff_series
using sum_integral_diff_series_antimono unfolding monoseq_def by blast

```

```

lemma sum_integral_diff_series_convergent: convergent sum_integral_diff_series
using sum_integral_diff_series_Bseq sum_integral_diff_series_monoseq
by (blast intro!: Bseq_monoseq_convergent)

```

theorem integral_test:

```

  summable (λn. f (of_nat n)) ↔ convergent (λn. integral {0..of_nat n} f)
proof -
  have summable (λn. f (of_nat n)) ↔ convergent (λn. ∑ k≤n. f (of_nat k))
  by (simp add: summable_iff_convergent')
  also have ... ↔ convergent (λn. integral {0..of_nat n} f)
proof
  assume convergent (λn. ∑ k≤n. f (of_nat k))
  from convergent_diff[OF this sum_integral_diff_series_convergent]
  show convergent (λn. integral {0..of_nat n} f)
  unfolding sum_integral_diff_series_def by simp
next
  assume convergent (λn. integral {0..of_nat n} f)
  from convergent_add[OF this sum_integral_diff_series_convergent]
  show convergent (λn. ∑ k≤n. f (of_nat k)) unfolding sum_integral_diff_series_def
by simp
qed
finally show ?thesis by simp
qed

```

end

end

6.27 Continuity of the indefinite integral; improper integral theorem

```
theory Improper_Integral
  imports Equivalence_Lebesgue_Henstock_Integration
begin
```

6.27.1 Equiintegrability

The definition here only really makes sense for an elementary set. We just use compact intervals in applications below.

```
definition equiintegrable_on (infixr equiintegrable'_on 46)
  where F equiintegrable_on I  $\equiv$ 
    ( $\forall f \in F. f \text{ integrable\_on } I$ )  $\wedge$ 
    ( $\forall e > 0. \exists \gamma. \text{gauge } \gamma \wedge$ 
      ( $\forall f \mathcal{D}. f \in F \wedge \mathcal{D} \text{ tagged\_division\_of } I \wedge \gamma \text{ fine } \mathcal{D}$ 
         $\longrightarrow \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f x) - \text{integral } I f)$ 
         $< e$ ))
```

```
lemma equiintegrable_on_integrable:
   $\llbracket F \text{ equiintegrable\_on } I; f \in F \rrbracket \implies f \text{ integrable\_on } I$ 
  using equiintegrable_on_def by metis
```

```
lemma equiintegrable_on_sing [simp]:
   $\{f\} \text{ equiintegrable\_on } \text{cbox } a \ b \longleftrightarrow f \text{ integrable\_on } \text{cbox } a \ b$ 
  by (simp add: equiintegrable_on_def has_integral_integral has_integral_integrable_on_def)
```

```
lemma equiintegrable_on_subset:  $\llbracket F \text{ equiintegrable\_on } I; G \subseteq F \rrbracket \implies G \text{ equiintegrable\_on } I$ 
  unfolding equiintegrable_on_def Ball_def
  by (erule conj_forward imp_forward all_forward ex_forward | blast)+
```

```
lemma equiintegrable_on_Un:
  assumes F equiintegrable_on I G equiintegrable_on I
  shows (F  $\cup$  G) equiintegrable_on I
  unfolding equiintegrable_on_def
proof (intro conjI impI allI)
  show  $\forall f \in F \cup G. f \text{ integrable\_on } I$ 
  using assms unfolding equiintegrable_on_def by blast
  show  $\exists \gamma. \text{gauge } \gamma \wedge$ 
    ( $\forall f \mathcal{D}. f \in F \cup G \wedge$ 
       $\mathcal{D} \text{ tagged\_division\_of } I \wedge \gamma \text{ fine } \mathcal{D} \longrightarrow$ 
       $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f x) - \text{integral } I f) < \varepsilon$ )
    if  $\varepsilon > 0$  for  $\varepsilon$ 
  proof -
    obtain  $\gamma 1$  where gauge  $\gamma 1$ 
    and  $\gamma 1$ :  $\bigwedge f \mathcal{D}. f \in F \wedge \mathcal{D} \text{ tagged\_division\_of } I \wedge \gamma 1 \text{ fine } \mathcal{D}$ 
```

```

       $\implies \text{norm } ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) < \varepsilon$ 
    using assms  $\langle \varepsilon > 0 \rangle$  unfolding equiintegrable_on_def by auto
  obtain  $\gamma 2$  where gauge  $\gamma 2$ 
  and  $\gamma 2$ :  $\bigwedge f \mathcal{D}. f \in G \wedge \mathcal{D} \text{ tagged\_division\_of } I \wedge \gamma 2 \text{ fine } \mathcal{D}$ 
       $\implies \text{norm } ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) < \varepsilon$ 
    using assms  $\langle \varepsilon > 0 \rangle$  unfolding equiintegrable_on_def by auto
  have gauge  $(\lambda x. \gamma 1 x \cap \gamma 2 x)$ 
    using  $\langle \text{gauge } \gamma 1 \rangle \langle \text{gauge } \gamma 2 \rangle$  by blast
  moreover have  $\forall f \mathcal{D}. f \in F \cup G \wedge \mathcal{D} \text{ tagged\_division\_of } I \wedge (\lambda x. \gamma 1 x \cap \gamma 2 x) \text{ fine } \mathcal{D} \longrightarrow$ 
       $\text{norm } ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) < \varepsilon$ 
    using  $\gamma 1 \gamma 2$  by (auto simp: fine_Int)
  ultimately show ?thesis
    by (intro exI conjI) assumption+
  qed
qed

```

```

lemma equiintegrable_on_insert:
  assumes  $f \text{ integrable\_on } \text{cbox } a \ b \ F \text{ equiintegrable\_on } \text{cbox } a \ b$ 
  shows  $(\text{insert } f \ F) \text{ equiintegrable\_on } \text{cbox } a \ b$ 
  by (metis assms equiintegrable_on_Un equiintegrable_on_sing insert_is_Un)

```

```

lemma equiintegrable_cmul:
  assumes  $F: F \text{ equiintegrable\_on } I$ 
  shows  $(\bigcup c \in \{-k..k\}. \bigcup f \in F. \{(\lambda x. c *_R f x)\}) \text{ equiintegrable\_on } I$ 
  unfolding equiintegrable_on_def
  proof (intro conjI impI allI ballI)
    show  $f \text{ integrable\_on } I$ 
      if  $f \in (\bigcup c \in \{-k..k\}. \bigcup f \in F. \{(\lambda x. c *_R f x)\})$ 
      for  $f :: 'a \Rightarrow 'b$ 
      using that assms equiintegrable_on_integrable_integrable_cmul by blast
    show  $\exists \gamma. \text{gauge } \gamma \wedge (\forall f \mathcal{D}. f \in (\bigcup c \in \{-k..k\}. \bigcup f \in F. \{(\lambda x. c *_R f x)\}) \wedge \mathcal{D} \text{ tagged\_division\_of } I$ 
       $\wedge \gamma \text{ fine } \mathcal{D} \longrightarrow \text{norm } ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) <$ 
       $\varepsilon)$ 
      if  $\varepsilon > 0$  for  $\varepsilon$ 
    proof -
      obtain  $\gamma$  where gauge  $\gamma$ 
      and  $\gamma$ :  $\bigwedge f \mathcal{D}. \llbracket f \in F; \mathcal{D} \text{ tagged\_division\_of } I; \gamma \text{ fine } \mathcal{D} \rrbracket$ 
           $\implies \text{norm } ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) < \varepsilon /$ 
           $(|k| + 1)$ 
      using assms  $\langle \varepsilon > 0 \rangle$  unfolding equiintegrable_on_def
      by (metis add commute add.right_neutral add_strict_mono divide_pos_pos
        norm_eq_zero real_norm_def zero_less_norm_iff zero_less_one)
      moreover have  $\text{norm } ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R c *_R (f x)) - \text{integral } I$ 
         $(\lambda x. c *_R f x)) < \varepsilon$ 
        if  $c: c \in \{-k..k\}$ 

```

```

    and  $f \in F$   $\mathcal{D}$  tagged_division_of  $I$   $\gamma$  fine  $\mathcal{D}$ 
  for  $\mathcal{D}$   $c$   $f$ 
  proof -
    have norm  $((\sum x \in \mathcal{D}. \text{case } x \text{ of } (x, K) \Rightarrow \text{content } K *_R c *_R f x) - \text{integral } I (\lambda x. c *_R f x))$ 
      =  $|c| * \text{norm } ((\sum x \in \mathcal{D}. \text{case } x \text{ of } (x, K) \Rightarrow \text{content } K *_R f x) - \text{integral } I f)$ 
    by (simp add: algebra_simps scale_sum_right case_prod_unfold flip: norm_scaleR)
    also have  $\dots \leq (|k| + 1) * \text{norm } ((\sum x \in \mathcal{D}. \text{case } x \text{ of } (x, K) \Rightarrow \text{content } K *_R f x) - \text{integral } I f)$ 
    using  $c$  by (auto simp: mult_right_mono)
    also have  $\dots < (|k| + 1) * (\varepsilon / (|k| + 1))$ 
    by (rule mult_strict_left_mono) (use  $\gamma$  less_eq_real_def that in auto)
    also have  $\dots = \varepsilon$ 
    by auto
    finally show ?thesis .
  qed
  ultimately show ?thesis
    by (rule_tac  $x=\gamma$  in exI) auto
  qed
qed

```

lemma *equiintegrable_add*:

```

  assumes  $F$ :  $F$  equiintegrable_on  $I$  and  $G$ :  $G$  equiintegrable_on  $I$ 
  shows  $(\bigcup f \in F. \bigcup g \in G. \{(\lambda x. f x + g x)\})$  equiintegrable_on  $I$ 
  unfolding equiintegrable_on_def
  proof (intro conjI impI allI ballI)
    show  $f$  integrable_on  $I$ 
    if  $f \in (\bigcup f \in F. \bigcup g \in G. \{(\lambda x. f x + g x)\})$  for  $f$ 
    using that equiintegrable_on_integrable assms by (auto intro: integrable_add)
    show  $\exists \gamma. \text{gauge } \gamma \wedge (\forall f \mathcal{D}. f \in (\bigcup f \in F. \bigcup g \in G. \{(\lambda x. f x + g x)\}) \wedge \mathcal{D} \text{ tagged\_division\_of } I$ 
       $\wedge \gamma \text{ fine } \mathcal{D} \longrightarrow \text{norm } ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) < \varepsilon)$ 
    if  $\varepsilon > 0$  for  $\varepsilon$ 
    proof -
      obtain  $\gamma 1$  where gauge  $\gamma 1$ 
      and  $\gamma 1$ :  $\bigwedge f \mathcal{D}. \llbracket f \in F; \mathcal{D} \text{ tagged\_division\_of } I; \gamma 1 \text{ fine } \mathcal{D} \rrbracket$ 
         $\implies \text{norm } ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) < \varepsilon / 2$ 
      using assms  $\langle \varepsilon > 0 \rangle$  unfolding equiintegrable_on_def by (meson half_gt_zero_iff)
      obtain  $\gamma 2$  where gauge  $\gamma 2$ 
      and  $\gamma 2$ :  $\bigwedge g \mathcal{D}. \llbracket g \in G; \mathcal{D} \text{ tagged\_division\_of } I; \gamma 2 \text{ fine } \mathcal{D} \rrbracket$ 
         $\implies \text{norm } ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R g x) - \text{integral } I g) < \varepsilon / 2$ 
      using assms  $\langle \varepsilon > 0 \rangle$  unfolding equiintegrable_on_def by (meson half_gt_zero_iff)
      have gauge  $(\lambda x. \gamma 1 x \cap \gamma 2 x)$ 
      using  $\langle \text{gauge } \gamma 1 \rangle \langle \text{gauge } \gamma 2 \rangle$  by blast
      moreover have norm  $((\sum (x, K) \in \mathcal{D}. \text{content } K *_R h x) - \text{integral } I h) < \varepsilon$ 

```

```

if  $h$ :  $h \in (\bigcup f \in F. \bigcup g \in G. \{\lambda x. f\ x + g\ x\})$ 
  and  $\mathcal{D}$ :  $\mathcal{D}$  tagged_division_of  $I$  and  $\text{fine}$ :  $(\lambda x. \gamma 1\ x \cap \gamma 2\ x)$  fine  $\mathcal{D}$ 
for  $h\ \mathcal{D}$ 
proof -
  obtain  $f\ g$  where  $f \in F\ g \in G$  and  $\text{heq}$ :  $h = (\lambda x. f\ x + g\ x)$ 
  using  $h$  by blast
  then have  $\text{int}$ :  $f$  integrable_on  $I\ g$  integrable_on  $I$ 
  using  $F\ G$  equiintegrable_on_def by blast+
  have  $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R h\ x) - \text{integral } I\ h)$ 
     $= \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f\ x + \text{content } K *_R g\ x) - (\text{integral } I\ f + \text{integral } I\ g))$ 
  by (simp add: heq algebra_simps integral_add int)
  also have  $\dots = \text{norm } (((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f\ x) - \text{integral } I\ f) + (\sum (x,K) \in \mathcal{D}. \text{content } K *_R g\ x) - \text{integral } I\ g))$ 
  by (simp add: sum.distrib algebra_simps case_prod_unfold)
  also have  $\dots \leq \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f\ x) - \text{integral } I\ f) + \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R g\ x) - \text{integral } I\ g)$ 
  by (metis (mono_tags) add_diff_eq norm_triangle_ineq)
  also have  $\dots < \varepsilon/2 + \varepsilon/2$ 
  using  $\gamma 1\ [OF\ \langle f \in F \rangle\ \mathcal{D}]\ \gamma 2\ [OF\ \langle g \in G \rangle\ \mathcal{D}]\ \text{fine}$  by (simp add: fine_Int)
  finally show ?thesis by simp
qed
ultimately show ?thesis
by meson
qed

```

```

lemma equiintegrable_minus:
  assumes  $F$  equiintegrable_on  $I$ 
  shows  $(\bigcup f \in F. \{\lambda x. - f\ x\})$  equiintegrable_on  $I$ 
  by (force intro: equiintegrable_on_subset [OF equiintegrable_cmul [OF assms, of 1]])

```

```

lemma equiintegrable_diff:
  assumes  $F$ :  $F$  equiintegrable_on  $I$  and  $G$ :  $G$  equiintegrable_on  $I$ 
  shows  $(\bigcup f \in F. \bigcup g \in G. \{\lambda x. f\ x - g\ x\})$  equiintegrable_on  $I$ 
  by (rule equiintegrable_on_subset [OF equiintegrable_add [OF F equiintegrable_minus [OF G]]] auto)

```

```

lemma equiintegrable_sum:
  fixes  $F :: ('a::euclidean\_space \Rightarrow 'b::euclidean\_space)$  set
  assumes  $F$  equiintegrable_on cbox a b
  shows  $(\bigcup I \in \text{Collect finite}. \bigcup c \in \{c. (\forall i \in I. c\ i \geq 0) \wedge \text{sum } c\ I = 1\}. \bigcup f \in I \rightarrow F. \{\lambda x. \text{sum } (\lambda i::'j. c\ i *_R f\ i\ x)\ I\})$  equiintegrable_on cbox a b
  (is ?G equiintegrable_on _)
  unfolding equiintegrable_on_def
  proof (intro conjI impI allI ballI)

```

```

show  $f$  integrable_on cbox  $a$   $b$  if  $f \in ?G$  for  $f$ 
using that assms by (auto simp: equiintegrable_on_def intro!: integrable_sum
integrable_cmul)
show  $\exists \gamma. \text{gauge } \gamma$ 
 $\wedge (\forall g \mathcal{D}. g \in ?G \wedge \mathcal{D} \text{ tagged\_division\_of cbox } a \ b \wedge \gamma \text{ fine } \mathcal{D}$ 
 $\longrightarrow \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \ x) - \text{integral } (\text{cbox } a \ b) \ g)$ 
 $< \varepsilon)$ 
if  $\varepsilon > 0$  for  $\varepsilon$ 
proof -
obtain  $\gamma$  where gauge  $\gamma$ 
and  $\gamma: \bigwedge f \mathcal{D}. \llbracket f \in F; \mathcal{D} \text{ tagged\_division\_of cbox } a \ b; \gamma \text{ fine } \mathcal{D} \rrbracket$ 
 $\implies \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \ x) - \text{integral } (\text{cbox } a \ b)$ 
 $f) < \varepsilon / 2$ 
using assms  $\langle \varepsilon > 0 \rangle$  unfolding equiintegrable_on_def by (meson half_gt_zero_iff)
moreover have  $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \ x) - \text{integral } (\text{cbox } a$ 
 $b) \ g) < \varepsilon$ 
if  $g: g \in ?G$ 
and  $\mathcal{D}: \mathcal{D} \text{ tagged\_division\_of cbox } a \ b$ 
and fine:  $\gamma \text{ fine } \mathcal{D}$ 
for  $\mathcal{D} \ g$ 
proof -
obtain  $I \ c \ f$  where finite  $I$  and  $0: \bigwedge i::'j. i \in I \implies 0 \leq c \ i$ 
and  $1: \text{sum } c \ I = 1$  and  $f: f \in I \rightarrow F$  and geq:  $g = (\lambda x. \sum_{i \in I}. c \ i *_R f \ i \ x)$ 
using  $g$  by auto
have fi_int:  $f \ i \text{ integrable\_on cbox } a \ b$  if  $i \in I$  for  $i$ 
by (metis Pi_iff assms equiintegrable_on_def f that)
have *:  $\text{integral } (\text{cbox } a \ b) (\lambda x. c \ i *_R f \ i \ x) = (\sum (x, K) \in \mathcal{D}. \text{integral } K (\lambda x.$ 
 $c \ i *_R f \ i \ x))$ 
if  $i \in I$  for  $i$ 
proof -
have  $f \ i \text{ integrable\_on cbox } a \ b$ 
by (metis Pi_iff assms equiintegrable_on_def f that)
then show ?thesis
by (intro  $\mathcal{D} \text{ integrable\_cmul integral\_combine\_tagged\_division\_topdown}$ )
qed
have finite  $\mathcal{D}$ 
using  $\mathcal{D}$  by blast
have swap:  $(\sum (x,K) \in \mathcal{D}. \text{content } K *_R (\sum_{i \in I}. c \ i *_R f \ i \ x))$ 
 $= (\sum_{i \in I}. c \ i *_R (\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \ i \ x))$ 
by (simp add: scale_sum_right case_prod_unfold algebra_simps) (rule
sum.swap)
have  $\text{norm } ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R g \ x) - \text{integral } (\text{cbox } a \ b) \ g)$ 
 $= \text{norm } ((\sum_{i \in I}. c \ i *_R ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \ i \ x) - \text{integral}$ 
 $(\text{cbox } a \ b) (f \ i))))$ 
unfolding geq swap
by (simp add: scaleR_right.sum algebra_simps integral_sum fi_int inte-
grable_cmul <finite I> sum_subtractf flip: sum_diff)
also have  $\dots \leq (\sum_{i \in I}. c \ i * \varepsilon / 2)$ 

```

```

proof (rule sum_norm_le)
  show norm (c i *R (( $\sum$  (xa, K) ∈  $\mathcal{D}$ . content K *R f i xa) - integral (cbox
a b) (f i))) ≤ c i * ε / 2
  if i ∈ I for i
  proof -
    have norm (( $\sum$  (x, K) ∈  $\mathcal{D}$ . content K *R f i x) - integral (cbox a b) (f i))
≤ ε/2
    using γ [OF _  $\mathcal{D}$  fine, of f i] funcset_mem [OF f] that by auto
    then show ?thesis
    using that by (auto simp: 0 mult.assoc intro: mult_left_mono)
  qed
qed
also have ... < ε
  using 1 <ε> 0 by (simp add: flip: sum_divide_distrib sum_distrib_right)
  finally show ?thesis .
qed
ultimately show ?thesis
  by (rule_tac x=γ in exI) auto
qed
qed

```

```

corollary equiintegrable_sum_real:
  fixes F :: (real ⇒ 'b::euclidean_space) set
  assumes F equiintegrable_on {a..b}
  shows ( $\bigcup I \in \text{Collect finite. } \bigcup c \in \{c. (\forall i \in I. c\ i \geq 0) \wedge \text{sum } c\ I = 1\}.$ 
 $\bigcup f \in I \rightarrow F. \{(\lambda x. \text{sum } (\lambda i. c\ i *_{\mathbb{R}} f\ i\ x)\ I)\}$ )
equiintegrable_on {a..b}
  using equiintegrable_sum [of F a b] assms by auto

```

Basic combining theorems for the interval of integration.

```

lemma equiintegrable_on_null [simp]:
  content(cbox a b) = 0 ⇒ F equiintegrable_on cbox a b
  unfolding equiintegrable_on_def
  by (metis diff_zero gauge_trivial integrable_on_null integral_null norm_zero
sum_content_null)

```

Main limit theorem for an equiintegrable sequence.

```

theorem equiintegrable_limit:
  fixes g :: 'a :: euclidean_space ⇒ 'b :: banach
  assumes feq: range f equiintegrable_on cbox a b
  and to_g:  $\bigwedge x. x \in \text{cbox } a\ b \Rightarrow (\lambda n. f\ n\ x) \longrightarrow g\ x$ 
  shows g integrable_on cbox a b  $\wedge (\lambda n. \text{integral } (\text{cbox } a\ b) (f\ n)) \longrightarrow \text{integral}$ 
(cbox a b) g
proof -
  have Cauchy (λn. integral(cbox a b) (f n))
proof (clarsimp simp add: Cauchy_def)
  fix e::real
  assume 0 < e
  then have e3: 0 < e/3

```

```

    by simp
  then obtain  $\gamma$  where gauge  $\gamma$ 
    and  $\gamma: \bigwedge n. \mathcal{D}. \llbracket \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \text{ } b; \gamma \text{ fine } \mathcal{D} \rrbracket$ 
       $\implies \text{norm}((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \text{ } n \text{ } x) - \text{integral } (\text{cbox } a \text{ } b) (f \text{ } n)) < e/3$ 
    using feq unfolding equiintegrable_on_def
    by (meson image_eqI iso_tuple_UNIV_I)
  obtain  $\mathcal{D}$  where  $\mathcal{D}: \mathcal{D} \text{ tagged\_division\_of } (\text{cbox } a \text{ } b)$  and  $\gamma \text{ fine } \mathcal{D}$  finite  $\mathcal{D}$ 
    by (meson  $\langle \text{gauge } \gamma \rangle$  fine_division_exists tagged_division_of_finite)
  with  $\gamma$  have  $\delta T: \bigwedge n. \text{dist } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \text{ } n \text{ } x)) (\text{integral } (\text{cbox } a \text{ } b) (f \text{ } n)) < e/3$ 
    by (force simp: dist_norm)
  have  $(\lambda n. \sum (x,K) \in \mathcal{D}. \text{content } K *_R f \text{ } n \text{ } x) \longrightarrow (\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \text{ } x)$ 
    using  $\mathcal{D}$  to_g by (auto intro!: tendsto_sum tendsto_scaleR)
  then have Cauchy  $(\lambda n. \sum (x,K) \in \mathcal{D}. \text{content } K *_R f \text{ } n \text{ } x)$ 
    by (meson convergent_eq_Cauchy)
  with  $e/3$  obtain  $M$  where
     $M: \bigwedge m \ n. \llbracket m \geq M; n \geq M \rrbracket \implies \text{dist } (\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \text{ } m \text{ } x) (\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \text{ } n \text{ } x) < e/3$ 
    unfolding Cauchy_def by blast
  have  $\bigwedge m \ n. \llbracket m \geq M; n \geq M; \text{dist } (\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \text{ } m \text{ } x) (\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \text{ } n \text{ } x) < e/3 \rrbracket$ 
     $\implies \text{dist } (\text{integral } (\text{cbox } a \text{ } b) (f \text{ } m)) (\text{integral } (\text{cbox } a \text{ } b) (f \text{ } n)) < e$ 
    by (metis  $\delta T$  dist_commute dist_triangle_third [OF _ _  $\delta T$ ])
  then show  $\exists M. \forall m \geq M. \forall n \geq M. \text{dist } (\text{integral } (\text{cbox } a \text{ } b) (f \text{ } m)) (\text{integral } (\text{cbox } a \text{ } b) (f \text{ } n)) < e$ 
    using  $M$  by auto
  qed
  then obtain  $L$  where  $L: (\lambda n. \text{integral } (\text{cbox } a \text{ } b) (f \text{ } n)) \longrightarrow L$ 
    by (meson convergent_eq_Cauchy)
  have  $(g \text{ has\_integral } L) (\text{cbox } a \text{ } b)$ 
  proof (clarsimp simp: has_integral)
    fix  $e::\text{real}$  assume  $0 < e$ 
    then have  $e/2: 0 < e/2$ 
      by simp
    then obtain  $\gamma$  where gauge  $\gamma$ 
      and  $\gamma: \bigwedge n. \mathcal{D}. \llbracket \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \text{ } b; \gamma \text{ fine } \mathcal{D} \rrbracket$ 
         $\implies \text{norm}((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \text{ } n \text{ } x) - \text{integral } (\text{cbox } a \text{ } b) (f \text{ } n)) < e/2$ 
      using feq unfolding equiintegrable_on_def
      by (meson image_eqI iso_tuple_UNIV_I)
    moreover
      have  $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \text{ } x) - L) < e$ 
        if  $\mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \text{ } b \ \gamma \text{ fine } \mathcal{D}$  for  $\mathcal{D}$ 
      proof -
        have  $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \text{ } x) - L) \leq e/2$ 

```

```

proof (rule Lim_norm_ubound)
  show  $(\lambda n. (\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \ n \ x) - \text{integral } (\text{cbox } a \ b) (f \ n))$ 
 $\longrightarrow (\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \ x) - L$ 
  using to_g that L
  by (intro tendsto_diff tendsto_sum) (auto simp: tag_in_interval tendsto_scaleR)
  show  $\forall_F n$  in sequentially.
     $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \ n \ x) - \text{integral } (\text{cbox } a \ b) (f \ n)) \leq e/2$ 
  by (intro eventuallyI less_imp_le  $\gamma$  that)
  qed auto
  with  $\langle 0 < e \rangle$  show ?thesis
  by linarith
qed
ultimately
show  $\exists \gamma. \text{gauge } \gamma \wedge$ 
 $(\forall \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b \wedge \gamma \text{ fine } \mathcal{D} \longrightarrow$ 
 $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R g \ x) - L) < e)$ 
  by meson
qed
with L show ?thesis
by (simp add:  $\langle \lambda n. \text{integral } (\text{cbox } a \ b) (f \ n) \rangle \longrightarrow L$ ) has_integral_integrable_integral
qed

```

lemma *equiintegrable_reflect:*

```

assumes F equiintegrable_on cbox a b
shows  $(\lambda f. f \circ \text{uminus}) \text{ ' } F \text{ equiintegrable\_on cbox } (-b) \ (-a)$ 
proof -
  have  $\S: \exists \gamma. \text{gauge } \gamma \wedge$ 
 $(\forall f \ \mathcal{D}. f \in (\lambda f. f \circ \text{uminus}) \text{ ' } F \wedge \mathcal{D} \text{ tagged\_division\_of cbox } (-b) \ (-a) \wedge \gamma \text{ fine } \mathcal{D} \longrightarrow$ 
 $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \ x) - \text{integral } (\text{cbox } (-b) \ (-a)) \ f) < e)$ 
  if gauge  $\gamma$  and
 $\gamma: \bigwedge f \ \mathcal{D}. \llbracket f \in F; \mathcal{D} \text{ tagged\_division\_of cbox } a \ b; \gamma \text{ fine } \mathcal{D} \rrbracket \implies$ 
 $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f \ x) - \text{integral } (\text{cbox } a \ b) \ f)$ 
 $< e$  for  $e \ \gamma$ 
  proof (intro exI, safe)
    show gauge  $(\lambda x. \text{uminus ' } \gamma \ (-x))$ 
    by (metis  $\langle \text{gauge } \gamma \rangle$  gauge_reflect)
    show  $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R (f \circ \text{uminus}) \ x) - \text{integral } (\text{cbox } (-b) \ (-a)) \ (f \circ \text{uminus})) < e$ 
    if  $f \in F$  and tag:  $\mathcal{D}$  tagged_division_of cbox (-b) (-a)
    and fine:  $(\lambda x. \text{uminus ' } \gamma \ (-x))$  fine  $\mathcal{D}$  for  $f \ \mathcal{D}$ 
  proof -
    have 1:  $(\lambda(x,K). (-x, \text{uminus ' } K)) \text{ ' } \mathcal{D} \text{ tagged\_partial\_division\_of cbox } a \ b$ 
    if  $\mathcal{D}$  tagged_partial_division_of cbox (-b) (-a)
    proof -

```



```

    have  $-y \in \text{cbox } a \ b$ 
    if  $\bigwedge x \ K. (x, K) \in \mathcal{D} \implies x \in K \wedge K \subseteq \text{cbox } (-b) \ (-a) \wedge (\exists a \ b. K =$ 
 $\text{cbox } a \ b)$ 
    (x, Y)  $\in \mathcal{D}$   $y \in Y$  for x Y y
  proof -
    have  $y \in \text{uminus } ' \text{cbox } a \ b$ 
    using that by auto
    then show  $-y \in \text{cbox } a \ b$ 
    by force
  qed
  with that show ?thesis
  by (fastforce simp: tagged_partial_division_of_def interior_negations
image_iff)
  qed
  have 2:  $\exists K. (\exists x. (x, K) \in (\lambda(x, K). (-x, \text{uminus } 'K)) ' \mathcal{D}) \wedge x \in K$ 
    if  $\bigcup \{K. \exists x. (x, K) \in \mathcal{D}\} = \text{cbox } (-b) \ (-a)$   $x \in \text{cbox } a \ b$  for x
  proof -
    have  $xm: x \in \text{uminus } ' \bigcup \{A. \exists a. (a, A) \in \mathcal{D}\}$ 
    by (simp add: that)
    then obtain a X where  $-x \in X$   $(a, X) \in \mathcal{D}$ 
    by auto
    then show ?thesis
    by (metis (no_types, lifting) add.inverse_inverse image_iff pair_imageI)
  qed
  have 3:  $\bigwedge x \ X \ y. [\mathcal{D} \text{ tagged\_partial\_division\_of } \text{cbox } (-b) \ (-a); (x, X) \in$ 
 $\mathcal{D}; y \in X] \implies -y \in \text{cbox } a \ b$ 
  by (metis (no_types, lifting) equation_minus_iff imageE subsetD tagged_partial_division_ofD(3)
uminus_interval_vector)
  have tag':  $(\lambda(x, K). (-x, \text{uminus } 'K)) ' \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b$ 
    using tag by (auto simp: tagged_division_of_def dest: 1 2 3)
  have fine':  $\gamma \text{ fine } (\lambda(x, K). (-x, \text{uminus } 'K)) ' \mathcal{D}$ 
    using fine by (fastforce simp: fine_def)
  have inj:  $\text{inj\_on } (\lambda(x, K). (-x, \text{uminus } 'K)) \ \mathcal{D}$ 
    unfolding inj_on_def by force
  have eq:  $\text{content } (\text{uminus } 'I) = \text{content } I$ 
    if  $I: (x, I) \in \mathcal{D}$  and fnz:  $f(-x) \neq 0$  for x I
  proof -
    obtain a b where  $I = \text{cbox } a \ b$ 
    using tag I that by (force simp: tagged_division_of_def tagged_partial_division_of_def)
    then show ?thesis
    using content_image_affinity_cbox [of -1 0] by auto
  qed
  have  $(\sum (x, K) \in (\lambda(x, K). (-x, \text{uminus } 'K)) ' \mathcal{D}. \text{content } K *_{\mathbb{R}} f x) =$ 
 $(\sum (x, K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f (-x))$ 
  by (auto simp add: eq sum.reindex [OF inj] intro!: sum.cong)
  then show ?thesis
  using  $\gamma [OF \langle f \in F \rangle \text{ tag' fine}] \text{integral\_reflect}$ 
  by (metis (mono_tags, lifting) Henstock_Kurzweil_Integration.integral_cong
comp_apply split_def sum.cong)

```

```

      qed
    qed
  show ?thesis
  using assms
  apply (auto simp: equiintegrable_on_def)
  subgoal for f
    by (metis (mono_tags, lifting) comp_apply integrable_eq integrable_reflect)
  using § by fastforce
qed

```

6.27.2 Subinterval restrictions for equiintegrable families

First, some technical lemmas about minimizing a "flat" part of a sum over a division.

lemma *lemma0*:

```

  assumes  $i \in \text{Basis}$ 
    shows  $\text{content } (\text{cbox } u \ v) / (\text{interval\_upperbound } (\text{cbox } u \ v) \cdot i - \text{interval\_lowerbound } (\text{cbox } u \ v) \cdot i) =$ 
      (if  $\text{content } (\text{cbox } u \ v) = 0$  then 0
       else  $\prod j \in \text{Basis} - \{i\}. \text{interval\_upperbound } (\text{cbox } u \ v) \cdot j - \text{interval\_lowerbound } (\text{cbox } u \ v) \cdot j$ )
  proof (cases  $\text{content } (\text{cbox } u \ v) = 0$ )
    case True
      then show ?thesis by simp
  next
    case False
      then show ?thesis
        using prod_subset_diff [of  $\{i\} \ \text{Basis}$ ] assms
        by (force simp: content_cbox_if divide_simps split: if_split_asm)
  qed

```

lemma *content_division_lemma1*:

```

  assumes  $\text{div: } \mathcal{D} \text{ division\_of } S \text{ and } S: S \subseteq \text{cbox } a \ b \text{ and } i: i \in \text{Basis}$ 
    and  $\text{mt: } \bigwedge K. K \in \mathcal{D} \implies \text{content } K \neq 0$ 
    and  $\text{disj: } (\forall K \in \mathcal{D}. K \cap \{x. x \cdot i = a \cdot i\} \neq \{\}) \vee (\forall K \in \mathcal{D}. K \cap \{x. x \cdot i = b \cdot i\} \neq \{\})$ 
    shows  $(b \cdot i - a \cdot i) * (\sum_{K \in \mathcal{D}} \text{content } K / (\text{interval\_upperbound } K \cdot i - \text{interval\_lowerbound } K \cdot i))$ 
       $\leq \text{content}(\text{cbox } a \ b) \quad (\text{is } ?\text{lhs} \leq ?\text{rhs})$ 
  proof -
    have finite  $\mathcal{D}$ 
      using div by blast
    define extend where
       $\text{extend} \equiv \lambda K. \text{cbox } (\sum j \in \text{Basis}. \text{if } j = i \text{ then } (a \cdot i) *_R i \text{ else } (\text{interval\_lowerbound } K \cdot j) *_R j)$ 
       $(\sum j \in \text{Basis}. \text{if } j = i \text{ then } (b \cdot i) *_R i \text{ else } (\text{interval\_upperbound } K \cdot j) *_R j)$ 
    have div_subset_cbox:  $\bigwedge K. K \in \mathcal{D} \implies K \subseteq \text{cbox } a \ b$ 

```

```

    using S div by auto
  have  $\bigwedge K. K \in \mathcal{D} \implies K \neq \{\}$ 
    using div by blast
  have extend_cbox:  $\bigwedge K. K \in \mathcal{D} \implies \exists a b. \text{extend } K = \text{cbox } a b$ 
    using extend_def by blast
  have extend:  $\text{extend } K \neq \{\} \implies \text{extend } K \subseteq \text{cbox } a b$  if  $K: K \in \mathcal{D}$  for  $K$ 
  proof -
    obtain u v where  $K: K = \text{cbox } u v$   $K \neq \{\}$   $K \subseteq \text{cbox } a b$ 
      using K cbox_division_memE [OF _ div] by (meson div_subset_cbox)
    with i show  $\text{extend } K \subseteq \text{cbox } a b$ 
      by (auto simp: extend_def subset_box box_ne_empty)
    have  $a \cdot i \leq b \cdot i$ 
      using K by (metis bot.extremum_uniqueI box_ne_empty(1) i)
    with K show  $\text{extend } K \neq \{\}$ 
      by (simp add: extend_def i box_ne_empty)
  qed
  have int_extend_disjoint:
     $\text{interior}(\text{extend } K1) \cap \text{interior}(\text{extend } K2) = \{\}$  if  $K: K1 \in \mathcal{D} K2 \in \mathcal{D} K1 \neq K2$  for  $K1 K2$ 
  proof -
    obtain u v where  $K1: K1 = \text{cbox } u v$   $K1 \neq \{\}$   $K1 \subseteq \text{cbox } a b$ 
      using K cbox_division_memE [OF _ div] by (meson div_subset_cbox)
    obtain w z where  $K2: K2 = \text{cbox } w z$   $K2 \neq \{\}$   $K2 \subseteq \text{cbox } a b$ 
      using K cbox_division_memE [OF _ div] by (meson div_subset_cbox)
    have cboxes:  $\text{cbox } u v \in \mathcal{D} \text{cbox } w z \in \mathcal{D} \text{cbox } u v \neq \text{cbox } w z$ 
      using K1 K2 that by auto
    with div have  $\text{interior}(\text{cbox } u v) \cap \text{interior}(\text{cbox } w z) = \{\}$ 
      by blast
    moreover
    have  $\exists x. x \in \text{box } u v \wedge x \in \text{box } w z$ 
      if  $x \in \text{interior}(\text{extend } K1) \implies x \in \text{interior}(\text{extend } K2)$  for  $x$ 
    proof -
      have  $a \cdot i < x \cdot i \wedge x \cdot i < b \cdot i$ 
        and ux:  $\bigwedge k. k \in \text{Basis} - \{i\} \implies u \cdot k < x \cdot k$ 
        and xv:  $\bigwedge k. k \in \text{Basis} - \{i\} \implies x \cdot k < v \cdot k$ 
        and wx:  $\bigwedge k. k \in \text{Basis} - \{i\} \implies w \cdot k < x \cdot k$ 
        and xz:  $\bigwedge k. k \in \text{Basis} - \{i\} \implies x \cdot k < z \cdot k$ 
          using that K1 K2 i by (auto simp: extend_def box_ne_empty mem_box)
      have  $\text{box } u v \neq \{\} \text{box } w z \neq \{\}$ 
        using cboxes interior_cbox by (auto simp: content_eq_0_interior dest: mt)
      then obtain q s
        where q:  $\bigwedge k. k \in \text{Basis} \implies w \cdot k < q \cdot k \wedge q \cdot k < z \cdot k$ 
        and s:  $\bigwedge k. k \in \text{Basis} \implies u \cdot k < s \cdot k \wedge s \cdot k < v \cdot k$ 
        by (meson all_not_in_conv mem_box(1))
      show ?thesis using disj
    proof
      assume  $\forall K \in \mathcal{D}. K \cap \{x. x \cdot i = a \cdot i\} \neq \{\}$ 
      then have wva:  $(\text{cbox } u v) \cap \{x. x \cdot i = a \cdot i\} \neq \{\}$ 
        and wza:  $(\text{cbox } w z) \cap \{x. x \cdot i = a \cdot i\} \neq \{\}$ 

```

```

    using cboxes by (auto simp: content_eq_0_interior)
    then obtain r t where  $r \cdot i = a \cdot i$  and  $r: \bigwedge k. k \in \text{Basis} \implies w \cdot k \leq r \cdot k \wedge r \cdot k \leq z \cdot k$ 
        and  $t \cdot i = a \cdot i$  and  $t: \bigwedge k. k \in \text{Basis} \implies u \cdot k \leq t \cdot k \wedge t \cdot k \leq v \cdot k$ 
    by (fastforce simp: mem_box)
    have  $u: u \cdot i < q \cdot i$ 
    using  $i$  K2(1) K2(3)  $\langle t \cdot i = a \cdot i \rangle q \ s \ t \ [OF \ i]$  by (force simp: subset_box)
    have  $w: w \cdot i < s \cdot i$ 
    using  $i$  K1(1) K1(3)  $\langle r \cdot i = a \cdot i \rangle s \ r \ [OF \ i]$  by (force simp: subset_box)
    define  $\xi$  where  $\xi \equiv (\sum j \in \text{Basis}. \text{if } j = i \text{ then } \min (q \cdot i) (s \cdot i) *_R i \text{ else } (x \cdot j) *_R j)$ 
    have [simp]:  $\xi \cdot j = (\text{if } j = i \text{ then } \min (q \cdot j) (s \cdot j) \text{ else } x \cdot j)$  if  $j \in \text{Basis}$ 
  for  $j$ 
    unfolding  $\xi\_def$ 
    by (intro sum_if_inner that  $\langle i \in \text{Basis} \rangle$ )
  show ?thesis
  proof (intro exI conjI)
    have  $\min (q \cdot i) (s \cdot i) < v \cdot i$ 
    using  $i$  s by fastforce
    with  $\langle i \in \text{Basis} \rangle s \ u \ u \ x \ v$ 
    show  $\xi \in \text{box } u \ v$ 
    by (force simp: mem_box)
    have  $\min (q \cdot i) (s \cdot i) < z \cdot i$ 
    using  $i$  q by force
    with  $\langle i \in \text{Basis} \rangle q \ w \ w \ x \ z$ 
    show  $\xi \in \text{box } w \ z$ 
    by (force simp: mem_box)
  qed
next
  assume  $\forall K \in \mathcal{D}. K \cap \{x. x \cdot i = b \cdot i\} \neq \{\}$ 
  then have  $u \wedge v: (\text{cbox } u \ v) \cap \{x. x \cdot i = b \cdot i\} \neq \{\}$ 
    and  $w \wedge z: (\text{cbox } w \ z) \cap \{x. x \cdot i = b \cdot i\} \neq \{\}$ 
    using cboxes by (auto simp: content_eq_0_interior)
  then obtain r t where  $r \cdot i = b \cdot i$  and  $r: \bigwedge k. k \in \text{Basis} \implies w \cdot k \leq r \cdot k \wedge r \cdot k \leq z \cdot k$ 
    and  $t \cdot i = b \cdot i$  and  $t: \bigwedge k. k \in \text{Basis} \implies u \cdot k \leq t \cdot k \wedge t \cdot k \leq v \cdot k$ 
  by (fastforce simp: mem_box)
  have  $z: s \cdot i < z \cdot i$ 
  using K1(1) K1(3)  $\langle r \cdot i = b \cdot i \rangle r \ [OF \ i] \ i \ s$  by (force simp: subset_box)
  have  $v: q \cdot i < v \cdot i$ 
  using K2(1) K2(3)  $\langle t \cdot i = b \cdot i \rangle t \ [OF \ i] \ i \ q$  by (force simp: subset_box)
  define  $\xi$  where  $\xi \equiv (\sum j \in \text{Basis}. \text{if } j = i \text{ then } \max (q \cdot i) (s \cdot i) *_R i \text{ else } (x \cdot j) *_R j)$ 
  have [simp]:  $\xi \cdot j = (\text{if } j = i \text{ then } \max (q \cdot j) (s \cdot j) \text{ else } x \cdot j)$  if  $j \in \text{Basis}$ 
  for  $j$ 
    unfolding  $\xi\_def$ 
    by (intro sum_if_inner that  $\langle i \in \text{Basis} \rangle$ )

```

```

show ?thesis
proof (intro exI conjI)
  show  $\xi \in \text{box } u \ v$ 
    using  $\langle i \in \text{Basis} \rangle s$  by (force simp: mem_box ux v xv)
  show  $\xi \in \text{box } w \ z$ 
    using  $\langle i \in \text{Basis} \rangle q$  by (force simp: mem_box wx xz z)
qed
qed
qed
ultimately show ?thesis by auto
qed
define interv_diff where  $\text{interv\_diff} \equiv \lambda K. \lambda i::'a. \text{interval\_upperbound } K \cdot i$ 
-  $\text{interval\_lowerbound } K \cdot i$ 
have ?lhs =  $(\sum_{K \in \mathcal{D}} (b \cdot i - a \cdot i) * \text{content } K / (\text{interv\_diff } K \ i))$ 
  by (simp add: sum_distrib_left interv_diff_def)
also have ... =  $\text{sum } (\text{content} \circ \text{extend}) \ \mathcal{D}$ 
proof (rule sum.cong [OF refl])
  fix K assume  $K \in \mathcal{D}$ 
  then obtain u v where  $K: K = \text{cbox } u \ v \ \text{cbox } u \ v \neq \{\}$   $K \subseteq \text{cbox } a \ b$ 
    using cbox_division_memE [OF _ div] div_subset_cbox by metis
  then have uv:  $u \cdot i < v \cdot i$ 
    using mt [OF  $\langle K \in \mathcal{D} \rangle$ ]  $\langle i \in \text{Basis} \rangle \text{content\_eq\_0}$  by fastforce
  have insert_i ( $\text{Basis} \cap -\{i\}$ ) = Basis
    using  $\langle i \in \text{Basis} \rangle$  by auto
  then have  $(b \cdot i - a \cdot i) * \text{content } K / (\text{interv\_diff } K \ i)$ 
    =  $(b \cdot i - a \cdot i) * (\prod_{i \in \text{insert } i \ (\text{Basis} \cap -\{i\})} v \cdot i - u \cdot i) /$ 
     $(\text{interv\_diff } (\text{cbox } u \ v) \ i)$ 
    using K box_ne_empty(1) content_cbox by fastforce
  also have ... =  $(\prod_{x \in \text{Basis}. \text{if } x = i \text{ then } b \cdot x - a \cdot x}$ 
     $\text{else } (\text{interval\_upperbound } (\text{cbox } u \ v) - \text{interval\_lowerbound } (\text{cbox}$ 
     $u \ v)) \cdot x)$ 
    using  $\langle i \in \text{Basis} \rangle K \ uv$  by (simp add: prod.If_cases interv_diff_def) (simp
    add: algebra_simps)
  also have ... =  $(\prod_{k \in \text{Basis}.}$ 
     $(\sum_{j \in \text{Basis}. \text{if } j = i \text{ then } (b \cdot i - a \cdot i) *_{\mathbb{R}} i}$ 
     $\text{else } ((\text{interval\_upperbound } (\text{cbox } u \ v) -$ 
     $\text{interval\_lowerbound } (\text{cbox } u \ v)) \cdot j) *_{\mathbb{R}} j) \cdot k)$ 
    using  $\langle i \in \text{Basis} \rangle$  by (subst prod.cong [OF refl sum_if_inner]; simp)
  also have ... =  $(\prod_{k \in \text{Basis}.}$ 
     $(\sum_{j \in \text{Basis}. \text{if } j = i \text{ then } (b \cdot i) *_{\mathbb{R}} i \text{ else } (\text{interval\_upperbound}$ 
     $(\text{cbox } u \ v) \cdot j) *_{\mathbb{R}} j) \cdot k -$ 
     $(\sum_{j \in \text{Basis}. \text{if } j = i \text{ then } (a \cdot i) *_{\mathbb{R}} i \text{ else } (\text{interval\_lowerbound}$ 
     $(\text{cbox } u \ v) \cdot j) *_{\mathbb{R}} j) \cdot k)$ 
    using  $\langle i \in \text{Basis} \rangle$ 
  by (intro prod.cong [OF refl]) (subst sum_if_inner; simp add: algebra_simps)+
  also have ... =  $(\text{content} \circ \text{extend}) \ K$ 
    using  $\langle i \in \text{Basis} \rangle K \ \text{box\_ne\_empty } \langle K \in \mathcal{D} \rangle \text{extend}(1)$ 
    by (auto simp add: extend_def content_cbox_if)
  finally show  $(b \cdot i - a \cdot i) * \text{content } K / (\text{interv\_diff } K \ i) = (\text{content} \circ$ 

```

```

extend) K .
qed
also have ... = sum content (extend 'D)
proof -
  have  $\llbracket K1 \in \mathcal{D}; K2 \in \mathcal{D}; K1 \neq K2; \text{extend } K1 = \text{extend } K2 \rrbracket \implies \text{content}$ 
  (extend K1) = 0 for K1 K2
  using int_extend_disjoint [of K1 K2] extend_def by (simp add: con-
  tent_eq_0_interior)
  then show ?thesis
  by (simp add: comm_monoid_add_class.sum_reindex_nontrivial [OF ⟨finite
  D⟩])
qed
also have ... ≤ ?rhs
proof (rule subadditive_content_division)
  show extend 'D division_of  $\bigcup$  (extend 'D)
  using int_extend_disjoint by (auto simp: division_of_def ⟨finite D⟩ extend
  extend_cbox)
  show  $\bigcup$  (extend 'D)  $\subseteq$  cbox a b
  using extend by fastforce
qed
finally show ?thesis .
qed

```

proposition *sum_content_area_over_thin_division:*

```

assumes div: D division_of S and S: S  $\subseteq$  cbox a b and i: i ∈ Basis
  and a · i ≤ c c ≤ b · i
  and nonmt:  $\bigwedge K. K \in \mathcal{D} \implies K \cap \{x. x \cdot i = c\} \neq \{\}$ 
  shows (b · i - a · i) * ( $\sum_{K \in \mathcal{D}. \text{content } K / (\text{interval\_upperbound } K \cdot i -$ 
  interval_lowerbound K · i))
  ≤ 2 * content(cbox a b)
proof (cases content(cbox a b) = 0)
  case True
  have ( $\sum_{K \in \mathcal{D}. \text{content } K / (\text{interval\_upperbound } K \cdot i - \text{interval\_lowerbound}$ 
  K · i)) = 0
  using S div by (force intro!: sum.neutral content_0_subset [OF True])
  then show ?thesis
  by (auto simp: True)
next
  case False
  then have content(cbox a b) > 0
  using zero_less_measure_iff by blast
  then have a · i < b · i if i ∈ Basis for i
  using content_pos_lt_eq that by blast
  have finite D
  using div by blast
  define Dlec where Dlec  $\equiv \{L \in (\lambda L. L \cap \{x. x \cdot i \leq c\}) \text{ 'D. content } L \neq 0\}$ 
  define Dgec where Dgec  $\equiv \{L \in (\lambda L. L \cap \{x. x \cdot i \geq c\}) \text{ 'D. content } L \neq 0\}$ 
  define a' where a'  $\equiv (\sum_{j \in \text{Basis}. (\text{if } j = i \text{ then } c \text{ else } a \cdot j) *_{\mathbb{R}} j)$ 

```

```

define  $b'$  where  $b' \equiv (\sum_{j \in \text{Basis}}. (\text{if } j = i \text{ then } c \text{ else } b \cdot j) *_{\mathcal{R}} j)$ 
define  $\text{interv\_diff}$  where  $\text{interv\_diff} \equiv \lambda K. \lambda i::'a. \text{interval\_upperbound } K \cdot i$ 
 $- \text{interval\_lowerbound } K \cdot i$ 
have  $\text{Dlec\_cbox}$ :  $\bigwedge K. K \in \text{Dlec} \implies \exists a \ b. K = \text{cbox } a \ b$ 
using  $\text{interval\_split } [OF \ i] \text{ div}$  by  $(\text{fastforce simp: Dlec\_def division\_of\_def})$ 
then have  $\text{lec\_is\_cbox}$ :  $\llbracket \text{content } (L \cap \{x. x \cdot i \leq c\}) \neq 0; L \in \mathcal{D} \rrbracket \implies \exists a \ b.$ 
 $L \cap \{x. x \cdot i \leq c\} = \text{cbox } a \ b$  for  $L$ 
using  $\text{Dlec\_def}$  by  $\text{blast}$ 
have  $\text{Dgec\_cbox}$ :  $\bigwedge K. K \in \text{Dgec} \implies \exists a \ b. K = \text{cbox } a \ b$ 
using  $\text{interval\_split } [OF \ i] \text{ div}$  by  $(\text{fastforce simp: Dgec\_def division\_of\_def})$ 
then have  $\text{gec\_is\_cbox}$ :  $\llbracket \text{content } (L \cap \{x. x \cdot i \geq c\}) \neq 0; L \in \mathcal{D} \rrbracket \implies \exists a \ b.$ 
 $L \cap \{x. x \cdot i \geq c\} = \text{cbox } a \ b$  for  $L$ 
using  $\text{Dgec\_def}$  by  $\text{blast}$ 

have  $\text{zero\_left}$ :  $\bigwedge x \ y. \llbracket x \in \mathcal{D}; y \in \mathcal{D}; x \neq y; x \cap \{x. x \cdot i \leq c\} = y \cap \{x. x \cdot i \leq c\} \rrbracket$ 
 $\implies \text{content } (y \cap \{x. x \cdot i \leq c\}) = 0$ 
by  $(\text{metis division\_split\_left\_inj } [OF \ \text{div}] \text{ lec\_is\_cbox content\_eq\_0\_interior})$ 
have  $\text{zero\_right}$ :  $\bigwedge x \ y. \llbracket x \in \mathcal{D}; y \in \mathcal{D}; x \neq y; x \cap \{x. c \leq x \cdot i\} = y \cap \{x. c \leq x \cdot i\} \rrbracket$ 
 $\implies \text{content } (y \cap \{x. c \leq x \cdot i\}) = 0$ 
by  $(\text{metis division\_split\_right\_inj } [OF \ \text{div}] \text{ gec\_is\_cbox content\_eq\_0\_interior})$ 

have  $(b' \cdot i - a \cdot i) * (\sum_{K \in \text{Dlec}}. \text{content } K / \text{interv\_diff } K \ i) \leq \text{content}(\text{cbox } a \ b')$ 
unfolding  $\text{interv\_diff\_def}$ 
proof  $(\text{rule content\_division\_lemma1})$ 
show  $\text{Dlec division\_of } \bigcup \text{Dlec}$ 
unfolding  $\text{division\_of\_def}$ 
proof  $(\text{intro conjI ballI Dlec\_cbox})$ 
show  $\bigwedge K1 \ K2. \llbracket K1 \in \text{Dlec}; K2 \in \text{Dlec} \rrbracket \implies K1 \neq K2 \longrightarrow \text{interior } K1 \cap$ 
 $\text{interior } K2 = \{\}$ 
by  $(\text{clarsimp simp: Dlec\_def})$   $(\text{use div in auto})$ 
qed  $(\text{use } \langle \text{finite } \mathcal{D} \rangle \text{ Dlec\_def in auto})$ 
show  $\bigcup \text{Dlec} \subseteq \text{cbox } a \ b'$ 
using  $\text{Dlec\_def div } S$  by  $(\text{auto simp: } b'\_\text{def division\_of\_def mem\_box})$ 
show  $(\forall K \in \text{Dlec}. K \cap \{x. x \cdot i = a \cdot i\} \neq \{\}) \vee (\forall K \in \text{Dlec}. K \cap \{x. x \cdot i =$ 
 $b' \cdot i\} \neq \{\})$ 
using  $\text{nonmt}$  by  $(\text{fastforce simp: Dlec\_def } b'\_\text{def } i)$ 
qed  $(\text{use } i \text{ Dlec\_def in auto})$ 
moreover
have  $(\sum_{K \in \text{Dlec}}. \text{content } K / (\text{interv\_diff } K \ i)) = (\sum_{K \in (\lambda K. K \cap \{x. x \cdot i \leq c\})} \text{' } \mathcal{D}. \text{content } K / \text{interv\_diff } K \ i)$ 
unfolding  $\text{Dlec\_def}$  using  $\langle \text{finite } \mathcal{D} \rangle$  by  $(\text{auto simp: sum.mono\_neutral\_left})$ 
moreover have  $\dots =$ 
 $(\sum_{K \in \mathcal{D}}. ((\lambda K. \text{content } K / (\text{interv\_diff } K \ i)) \circ ((\lambda K. K \cap \{x. x \cdot i \leq c\}))) \ K)$ 
by  $(\text{simp add: zero\_left sum.reindex\_nontrivial } [OF \ \langle \text{finite } \mathcal{D} \rangle])$ 
moreover have  $(b' \cdot i - a \cdot i) = (c - a \cdot i)$ 

```

```

    by (simp add: b'_def i)
  ultimately
  have lec:  $(c - a \cdot i) * (\sum_{K \in \mathcal{D}} ((\lambda K. \text{content } K / (\text{interv\_diff } K \ i)) \circ ((\lambda K. K \cap \{x. x \cdot i \leq c\}))) K)$ 
     $\leq \text{content}(\text{cbox } a \ b')$ 
  by simp

  have  $(b \cdot i - a' \cdot i) * (\sum_{K \in D_{\text{gec}}} \text{content } K / (\text{interv\_diff } K \ i)) \leq \text{content}(\text{cbox } a' \ b)$ 
  unfolding interv_diff_def
  proof (rule content_division_lemma1)
    show  $D_{\text{gec}} \text{ division\_of } \bigcup D_{\text{gec}}$ 
    unfolding division_of_def
    proof (intro conjI ballI Dgec_cbox)
      show  $\bigwedge K1 \ K2. \llbracket K1 \in D_{\text{gec}}; K2 \in D_{\text{gec}} \rrbracket \implies K1 \neq K2 \longrightarrow \text{interior } K1 \cap \text{interior } K2 = \{\}$ 
      by (clarsimp simp: Dgec_def) (use div in auto)
    qed (use <finite D> Dgec_def in auto)
    show  $\bigcup D_{\text{gec}} \subseteq \text{cbox } a' \ b$ 
    using Dgec_def div S by (auto simp: a'_def division_of_def mem_box)
    show  $(\forall K \in D_{\text{gec}}. K \cap \{x. x \cdot i = a' \cdot i\} \neq \{\}) \vee (\forall K \in D_{\text{gec}}. K \cap \{x. x \cdot i = b \cdot i\} \neq \{\})$ 
    using nonmt by (fastforce simp: Dgec_def a'_def i)
  qed (use i Dgec_def in auto)
  moreover
  have  $(\sum_{K \in D_{\text{gec}}} \text{content } K / (\text{interv\_diff } K \ i)) = (\sum_{K \in (\lambda K. K \cap \{x. c \leq x \cdot i\}) ' \mathcal{D}} \text{content } K / \text{interv\_diff } K \ i)$ 
  unfolding Dgec_def using <finite D> by (auto simp: sum.mono_neutral_left)
  moreover have  $\dots = (\sum_{K \in \mathcal{D}} ((\lambda K. \text{content } K / (\text{interv\_diff } K \ i)) \circ ((\lambda K. K \cap \{x. x \cdot i \geq c\}))) K)$ 
  by (simp add: zero_right sum.reindex_nontrivial [OF <finite D>])
  moreover have  $(b \cdot i - a' \cdot i) = (b \cdot i - c)$ 
  by (simp add: a'_def i)
  ultimately
  have gec:  $(b \cdot i - c) * (\sum_{K \in \mathcal{D}} ((\lambda K. \text{content } K / (\text{interv\_diff } K \ i)) \circ ((\lambda K. K \cap \{x. x \cdot i \geq c\}))) K)$ 
     $\leq \text{content}(\text{cbox } a' \ b)$ 
  by simp

  show ?thesis
  proof (cases  $c = a \cdot i \vee c = b \cdot i$ )
    case True
    then show ?thesis
    proof
      assume c:  $c = a \cdot i$ 
      moreover
      have  $(\sum_{j \in \text{Basis}. (if } j = i \text{ then } a \cdot i \text{ else } a \cdot j) *_R j) = a$ 

```



```

      using euclidean_representation [of a] sum.cong [OF refl, of Basis  $\lambda i$ .  $(a \cdot$ 
 $i) *_{\mathbb{R}} i]$  by presburger
      ultimately have  $a' = a$ 
      by (simp add:  $i \ a'_{\text{def}}$  cong: if_cong)
      then have  $\text{content } (\text{cbox } a' \ b) \leq 2 * \text{content } (\text{cbox } a \ b)$  by simp
      moreover
      have eq:  $(\sum_{K \in \mathcal{D}} \text{content } (K \cap \{x. a \cdot i \leq x \cdot i\}) / \text{interv\_diff } (K \cap \{x.$ 
 $a \cdot i \leq x \cdot i\}) \ i)$ 
      =  $(\sum_{K \in \mathcal{D}} \text{content } K / \text{interv\_diff } K \ i)$ 
      (is  $\text{sum } ?f_{\text{--}} = \text{sum } ?g_{\text{--}}$ )
      proof (rule sum.cong [OF refl])
      fix  $K$  assume  $K \in \mathcal{D}$ 
      then have  $a \cdot i \leq x \cdot i$  if  $x \in K$  for  $x$ 
      by (metis  $S \ \text{UnionI} \ \text{div} \ \text{division\_ofD}(6) \ i \ \text{mem\_box}(2) \ \text{subsetCE} \ \text{that}$ )
      then have  $K \cap \{x. a \cdot i \leq x \cdot i\} = K$ 
      by blast
      then show  $?f \ K = ?g \ K$ 
      by simp
      qed
      ultimately show ?thesis
      using gec  $c \ \text{eq} \ \text{interv\_diff\_def}$  by auto
    next
      assume  $c: c = b \cdot i$ 
      moreover have  $(\sum_{j \in \text{Basis}} (\text{if } j = i \text{ then } b \cdot i \text{ else } b \cdot j) *_{\mathbb{R}} j) = b$ 
      using euclidean_representation [of b] sum.cong [OF refl, of Basis  $\lambda i$ .  $(b \cdot$ 
 $i) *_{\mathbb{R}} i]$  by presburger
      ultimately have  $b' = b$ 
      by (simp add:  $i \ b'_{\text{def}}$  cong: if_cong)
      then have  $\text{content } (\text{cbox } a \ b') \leq 2 * \text{content } (\text{cbox } a \ b)$  by simp
      moreover
      have eq:  $(\sum_{K \in \mathcal{D}} \text{content } (K \cap \{x. x \cdot i \leq b \cdot i\}) / \text{interv\_diff } (K \cap \{x. x$ 
 $\cdot i \leq b \cdot i\}) \ i)$ 
      =  $(\sum_{K \in \mathcal{D}} \text{content } K / \text{interv\_diff } K \ i)$ 
      (is  $\text{sum } ?f_{\text{--}} = \text{sum } ?g_{\text{--}}$ )
      proof (rule sum.cong [OF refl])
      fix  $K$  assume  $K \in \mathcal{D}$ 
      then have  $x \cdot i \leq b \cdot i$  if  $x \in K$  for  $x$ 
      by (metis  $S \ \text{UnionI} \ \text{div} \ \text{division\_ofD}(6) \ i \ \text{mem\_box}(2) \ \text{subsetCE} \ \text{that}$ )
      then have  $K \cap \{x. x \cdot i \leq b \cdot i\} = K$ 
      by blast
      then show  $?f \ K = ?g \ K$ 
      by simp
      qed
      ultimately show ?thesis
      using lec  $c \ \text{eq} \ \text{interv\_diff\_def}$  by auto
    qed
  next
    case False
    have prod_if:  $(\prod_{k \in \text{Basis} \cap - \ \{i\}} f \ k) = (\prod_{k \in \text{Basis}} f \ k) / f \ i$  if  $f \ i \neq$ 

```

```

(0::real) for f
  proof -
    have f i * prod f (Basis ∩ - {i}) = prod f Basis
      using that mk_disjoint_insert [OF i]
    by (metis Int_insert_left_if0 finite_Basis finite_insert le_iff_inf order_refl
prod.insert_subset_Compl_singleton)
    then show ?thesis
      by (metis nonzero_mult_div_cancel_left that)
    qed
    have abc: a • i < c c < b • i
      using False assms by auto
    then have (∑ K ∈ D. ((λK. content K / (interv_diff K i)) ∘ ((λK. K ∩ {x. x
• i ≤ c}))) K)
      ≤ content(cbox a b') / (c - a • i)
      (∑ K ∈ D. ((λK. content K / (interv_diff K i)) ∘ ((λK. K ∩ {x. x • i
≥ c}))) K)
      ≤ content(cbox a' b) / (b • i - c)
      using lec_gec by (simp_all add: field_split_simps)
    moreover
    have (∑ K ∈ D. content K / (interv_diff K i))
      ≤ (∑ K ∈ D. ((λK. content K / (interv_diff K i)) ∘ ((λK. K ∩ {x. x • i
≤ c}))) K) +
      (∑ K ∈ D. ((λK. content K / (interv_diff K i)) ∘ ((λK. K ∩ {x. x • i ≥
c}))) K)
      (is ?lhs ≤ ?rhs)
    proof -
      have ?lhs ≤
        (∑ K ∈ D. ((λK. content K / (interv_diff K i)) ∘ ((λK. K ∩ {x. x • i ≤
c}))) K) +
        ((λK. content K / (interv_diff K i)) ∘ ((λK. K ∩ {x. x • i ≥
c}))) K)
        (is sum ?f _ ≤ sum ?g _)
    proof (rule sum_mono)
      fix K assume K ∈ D
      then obtain u v where uv: K = cbox u v
        using div by blast
      obtain u' v' where uv': cbox u v ∩ {x. x • i ≤ c} = cbox u v'
        cbox u v ∩ {x. c ≤ x • i} = cbox u' v
        ∧ k. k ∈ Basis ⇒ u' • k = (if k = i then max (u • i) c
else u • k)
        ∧ k. k ∈ Basis ⇒ v' • k = (if k = i then min (v • i) c
else v • k)
      using i by (auto simp: interval_split)
      have *: [content (cbox u v') = 0; content (cbox u' v) = 0] ⇒ content (cbox
u v) = 0
        content (cbox u' v) ≠ 0 ⇒ content (cbox u v) ≠ 0
        content (cbox u v') ≠ 0 ⇒ content (cbox u v) ≠ 0
      using i uv uv' by (auto simp: content_eq_0 le_max_iff_disj min_le_iff_disj
split: if_split_asm intro: order_trans)

```

```

    have uniq:  $\bigwedge j. \llbracket j \in \text{Basis}; \neg u \cdot j \leq v \cdot j \rrbracket \implies j = i$ 
      by (metis  $\langle K \in \mathcal{D} \rangle$  box_ne_empty(1) div_division_of_def uv)
    show  $?f K \leq ?g K$ 
      using i uv uv' by (auto simp add: interv_diff_def lemma0 dest: uniq *
intro!: prod_nonneg)
    qed
    also have  $\dots = ?rhs$ 
      by (simp add: sum.distrib)
    finally show ?thesis .
  qed
  moreover have  $\text{content } (\text{cbox } a \ b') / (c - a \cdot i) = \text{content } (\text{cbox } a \ b) / (b \cdot i - a \cdot i)$ 
    using i abc
    apply (simp add: field_simps a'_def b'_def measure_lborel_cbox_eq inner_diff)
    apply (auto simp: if_distrib if_distrib [of  $\lambda f. f \ x$  for  $x$ ] prod.If_cases [of Basis  $\lambda x. x = i$ , simplified] prod_if field_simps)
    done
  moreover have  $\text{content } (\text{cbox } a' \ b) / (b \cdot i - c) = \text{content } (\text{cbox } a \ b) / (b \cdot i - a \cdot i)$ 
    using i abc
    apply (simp add: field_simps a'_def b'_def measure_lborel_cbox_eq inner_diff)
    apply (auto simp: if_distrib prod.If_cases [of Basis  $\lambda x. x = i$ , simplified] prod_if field_simps)
    done
  ultimately
  have  $(\sum_{K \in \mathcal{D}} \text{content } K / (\text{interv\_diff } K \ i)) \leq 2 * \text{content } (\text{cbox } a \ b) / (b \cdot i - a \cdot i)$ 
    by linarith
  then show ?thesis
    using abc interv_diff_def by (simp add: field_split_simps)
  qed
qed

```

proposition *bounded_equiintegral_over_thin_tagged_partial_division:*

```

  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes  $F: F \text{ equiintegrable\_on } \text{cbox } a \ b$  and  $f: f \in F$  and  $0 < \varepsilon$ 
    and  $\text{norm\_f}: \bigwedge h \ x. \llbracket h \in F; x \in \text{cbox } a \ b \rrbracket \implies \text{norm}(h \ x) \leq \text{norm}(f \ x)$ 
  obtains  $\gamma$  where gauge  $\gamma$ 
     $\bigwedge c \ i \ S \ h. \llbracket c \in \text{cbox } a \ b; i \in \text{Basis}; S \text{ tagged\_partial\_division\_of } \text{cbox } a \ b;$ 
       $\gamma \text{ fine } S; h \in F; \bigwedge x \ K. (x, K) \in S \implies (K \cap \{x. x \cdot i = c \cdot i\}$ 
 $\neq \{\}) \rrbracket$ 
       $\implies (\sum (x, K) \in S. \text{norm } (\text{integral } K \ h)) < \varepsilon$ 
  proof (cases  $\text{content}(\text{cbox } a \ b) = 0$ )
    case True
    show ?thesis

```

```

proof
  show gauge ( $\lambda x. \text{ball } x \ 1$ )
    by (simp add: gauge_trivial)
  show ( $\sum (x, K) \in S. \text{norm } (\text{integral } K \ h) < \varepsilon$ )
    if S tagged_partial_division_of cbox a b ( $\lambda x. \text{ball } x \ 1$ ) fine S for S and
h:: 'a  $\Rightarrow$  'b
  proof –
    have ( $\sum (x, K) \in S. \text{norm } (\text{integral } K \ h) = 0$ )
      using that True content_0_subset
      by (fastforce simp: tagged_partial_division_of_def intro: sum.neutral)
    with  $\langle 0 < \varepsilon \rangle$  show ?thesis
      by simp
  qed
qed
next
  case False
  then have contab_gt0: content(cbox a b) > 0
    by (simp add: zero_less_measure_iff)
  then have a_less_b:  $\bigwedge i. i \in \text{Basis} \implies a \cdot i < b \cdot i$ 
    by (auto simp: content_pos_lt_eq)
  obtain  $\gamma 0$  where gauge  $\gamma 0$ 
    and  $\gamma 0: \bigwedge S \ h. \llbracket S \text{ tagged\_partial\_division\_of cbox } a \ b; \gamma 0 \text{ fine } S; h \in F \rrbracket$ 
       $\implies (\sum (x, K) \in S. \text{norm } (\text{content } K *_R h \ x - \text{integral } K$ 
h)) <  $\varepsilon/2$ 
  proof –
    obtain  $\gamma$  where gauge  $\gamma$ 
      and  $\gamma: \bigwedge f \ \mathcal{D}. \llbracket f \in F; \mathcal{D} \text{ tagged\_division\_of cbox } a \ b; \gamma \text{ fine } \mathcal{D} \rrbracket$ 
         $\implies \text{norm } ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R f \ x) - \text{integral}$ 
(cbox a b) f)
           $< \varepsilon / (5 * (\text{Suc } \text{DIM}('b)))$ 
  proof –
    have e5:  $\varepsilon / (5 * (\text{Suc } \text{DIM}('b))) > 0$ 
      using  $\langle \varepsilon > 0 \rangle$  by auto
    then show ?thesis
      using F that by (auto simp: equiintegrable_on_def)
  qed
show ?thesis
proof
  show gauge  $\gamma$ 
    by (rule  $\langle \text{gauge } \gamma \rangle$ )
  show ( $\sum (x, K) \in S. \text{norm } (\text{content } K *_R h \ x - \text{integral } K \ h) < \varepsilon/2$ )
    if S tagged_partial_division_of cbox a b  $\gamma$  fine S h  $\in F$  for S h
  proof –
    have ( $\sum (x, K) \in S. \text{norm } (\text{content } K *_R h \ x - \text{integral } K \ h) \leq 2 * \text{real}$ 
DIM('b) * ( $\varepsilon / (5 * \text{Suc } \text{DIM}('b))$ ))
      proof (rule Henstock_lemma_part2 [of h a b])
        show h integrable_on cbox a b
          using that F equiintegrable_on_def by metis
        show gauge  $\gamma$ 

```

```

      by (rule ‹gauge  $\gamma$ ›)
    qed (use that ‹ $\varepsilon > 0$ ›  $\gamma$  in auto)
  also have ... <  $\varepsilon/2$ 
    using ‹ $\varepsilon > 0$ › by (simp add: divide_simps)
  finally show ?thesis .
qed
qed
qed
define  $\gamma$  where  $\gamma \equiv \lambda x. \gamma 0 \ x \ \cap$ 
  ball  $x \ ((\varepsilon/8 \ / \ (\text{norm}(f \ x) + 1))) * (\text{INF } m \in \text{Basis}. b \cdot m - a$ 
 $\cdot m) / \text{content}(\text{cbox } a \ b))$ 
  define  $\text{interv\_diff}$  where  $\text{interv\_diff} \equiv \lambda K. \lambda i::'a. \text{interval\_upperbound } K \cdot i$ 
 $- \text{interval\_lowerbound } K \cdot i$ 
  have  $8 * \text{content}(\text{cbox } a \ b) + \text{norm}(f \ x) * (8 * \text{content}(\text{cbox } a \ b)) > 0$  for  $x$ 
    by (metis add.right_neutral add_pos_pos contab_gt0 mult_pos_pos mult_zero_left
  norm_eq_zero zero_less_norm_iff zero_less_numeral)
  then have gauge  $(\lambda x. \text{ball } x$ 
     $(\varepsilon * (\text{INF } m \in \text{Basis}. b \cdot m - a \cdot m) / ((8 * \text{norm}(f \ x) + 8) * \text{content}(\text{cbox } a \ b))))$ 
    using ‹ $0 < \text{content}(\text{cbox } a \ b)$ › ‹ $0 < \varepsilon$ › a_less_b
    by (auto simp add: gauge_def field_split_simps add_nonneg_eq_0_iff finite_less_Inf_iff)
  then have gauge  $\gamma$ 
    unfolding  $\gamma\_def$  using ‹gauge  $\gamma 0$ › gauge_Int by auto
  moreover
  have  $(\sum (x,K) \in S. \text{norm}(\text{integral } K \ h)) < \varepsilon$ 
    if  $c \in \text{cbox } a \ b$   $i \in \text{Basis}$  and  $S: S \text{ tagged\_partial\_division\_of } \text{cbox } a \ b$ 
    and  $\gamma \text{ fine } S$   $h \in F$  and  $\text{ne}: \bigwedge x \ K. (x,K) \in S \implies K \cap \{x. x \cdot i = c \cdot i\} \neq \{\}$ 
  proof -
    have  $\text{cbox } c \ b \subseteq \text{cbox } a \ b$ 
      by (meson mem_box(2) order_refl subset_box(1) that(1))
    have finite  $S$ 
      using  $S$  unfolding tagged_partial_division_of_def by blast
    have  $\gamma 0 \text{ fine } S$  and  $\text{fine } S$ :
       $(\lambda x. \text{ball } x \ (\varepsilon * (\text{INF } m \in \text{Basis}. b \cdot m - a \cdot m) / ((8 * \text{norm}(f \ x) + 8) * \text{content}(\text{cbox } a \ b)))) \text{ fine } S$ 
      using ‹ $\gamma \text{ fine } S$ › by (auto simp:  $\gamma\_def$  fine_Int)
    then have  $(\sum (x,K) \in S. \text{norm}(\text{content } K *_R h \ x - \text{integral } K \ h)) < \varepsilon/2$ 
      by (intro  $\gamma 0$  that fineS)
    moreover have  $(\sum (x,K) \in S. \text{norm}(\text{integral } K \ h) - \text{norm}(\text{content } K *_R h \ x - \text{integral } K \ h)) \leq \varepsilon/2$ 
      proof -
        have  $(\sum (x,K) \in S. \text{norm}(\text{integral } K \ h) - \text{norm}(\text{content } K *_R h \ x - \text{integral } K \ h))$ 
           $\leq (\sum (x,K) \in S. \text{norm}(\text{content } K *_R h \ x))$ 
        proof (clarify intro!: sum_mono)
          fix  $x \ K$ 
          assume  $xK: (x,K) \in S$ 

```

```

      have norm (integral K h) - norm (content K *R h x - integral K h) ≤
norm (integral K h - (integral K h - content K *R h x))
      by (metis norm_minus_commute norm_triangle_ineq2)
      also have ... ≤ norm (content K *R h x)
      by simp
      finally show norm (integral K h) - norm (content K *R h x - integral K
h) ≤ norm (content K *R h x) .
    qed
    also have ... ≤ (∑ (x,K) ∈ S. ε/4 * (b · i - a · i) / content (cbox a b) *
content K / interv_diff K i)
    proof (clarify intro!: sum_mono)
      fix x K
      assume xK: (x,K) ∈ S
      then have x: x ∈ cbox a b
      using S unfolding tagged_partial_division_of_def by (meson subset_iff)
      show norm (content K *R h x) ≤ ε/4 * (b · i - a · i) / content (cbox a b)
* content K / interv_diff K i
      proof (cases content K = 0)
        case True
        then show ?thesis by simp
      next
        case False
        then have Kgt0: content K > 0
        using zero_less_measure_iff by blast
        moreover
        obtain u v where uv: K = cbox u v
        using S ⟨(x,K) ∈ S⟩ unfolding tagged_partial_division_of_def by
blast
        then have u_less_v: ∧i. i ∈ Basis ⇒ u · i < v · i
        using content_pos_lt_eq uv Kgt0 by blast
        then have dist_uv: dist u v > 0
        using that by auto
        ultimately have norm (h x) ≤ (ε * (b · i - a · i)) / (4 * content (cbox
a b) * interv_diff K i)
        proof -
          have dist x u < ε * (INF m∈Basis. b · m - a · m) / (4 * (norm (f x)
+ 1) * content (cbox a b)) / 2
          dist x v < ε * (INF m∈Basis. b · m - a · m) / (4 * (norm (f x) +
1) * content (cbox a b)) / 2
          using fineS u_less_v uv xK
          by (force simp: fine_def mem_box field_simps dest!: bspec)+
          moreover have ε * (INF m∈Basis. b · m - a · m) / (4 * (norm (f x)
+ 1) * content (cbox a b)) / 2
          ≤ ε * (b · i - a · i) / (4 * (norm (f x) + 1) * content (cbox a b))
/ 2
          proof (intro mult_left_mono divide_right_mono)
            show (INF m∈Basis. b · m - a · m) ≤ b · i - a · i
            using ⟨i ∈ Basis⟩ by (auto intro!: cInf_le_finite)
          qed (use ⟨0 < ε⟩ in auto)

```

```

ultimately
  have  $\text{dist } x \ u < \varepsilon * (b \cdot i - a \cdot i) / (4 * (\text{norm } (f \ x) + 1) * \text{content } (cbox \ a \ b)) / 2$ 
     $\text{dist } x \ v < \varepsilon * (b \cdot i - a \cdot i) / (4 * (\text{norm } (f \ x) + 1) * \text{content } (cbox \ a \ b)) / 2$ 
  by linarith+
  then have  $\text{duv: dist } u \ v < \varepsilon * (b \cdot i - a \cdot i) / (4 * (\text{norm } (f \ x) + 1) * \text{content } (cbox \ a \ b))$ 
    using  $\text{dist\_triangle\_half\_r}$  by blast
  have  $\text{uvi: } |v \cdot i - u \cdot i| \leq \text{norm } (v - u)$ 
  by (metis  $\text{inner\_commute inner\_diff\_right } \langle i \in \text{Basis} \rangle \text{Basis\_le\_norm}$ )
  have  $\text{norm } (h \ x) \leq \text{norm } (f \ x)$ 
  using  $x$  that by (auto simp:  $\text{norm\_f}$ )
  also have  $\dots < (\text{norm } (f \ x) + 1)$ 
  by simp
  also have  $\dots < \varepsilon * (b \cdot i - a \cdot i) / \text{dist } u \ v / (4 * \text{content } (cbox \ a \ b))$ 
  proof -
    have  $0 < \text{norm } (f \ x) + 1$ 
    by (simp add:  $\text{add.commute add\_pos\_nonneg}$ )
    then show ?thesis
      using  $\text{duv dist\_uv contab\_gt0}$ 
      by (simp only:  $\text{mult\_ac divide\_simps}$ ) auto
  qed
  also have  $\dots = \varepsilon * (b \cdot i - a \cdot i) / \text{norm } (v - u) / (4 * \text{content } (cbox \ a \ b))$ 
    by (simp add:  $\text{dist\_norm norm\_minus\_commute}$ )
  also have  $\dots \leq \varepsilon * (b \cdot i - a \cdot i) / |v \cdot i - u \cdot i| / (4 * \text{content } (cbox \ a \ b))$ 
  proof (intro  $\text{mult\_right\_mono divide\_left\_mono divide\_right\_mono uvi}$ )
    show  $\text{norm } (v - u) * |v \cdot i - u \cdot i| > 0$ 
    using  $\text{u\_less\_v } [OF \ \langle i \in \text{Basis} \rangle]$ 
    by (auto simp:  $\text{less\_eq\_real\_def zero\_less\_mult\_iff}$  that)
    show  $\varepsilon * (b \cdot i - a \cdot i) \geq 0$ 
    using  $\text{a\_less\_b } \langle 0 < \varepsilon \rangle \langle i \in \text{Basis} \rangle$  by force
  qed auto
  also have  $\dots = \varepsilon * (b \cdot i - a \cdot i) / (4 * \text{content } (cbox \ a \ b) * \text{interv\_diff } K \ i)$ 
    using  $\text{uv False that(2) u\_less\_v interv\_diff\_def}$  by fastforce
  finally show ?thesis by simp
qed
with  $\text{Kgt0}$  have  $\text{norm } (\text{content } K *_{\mathbb{R}} h \ x) \leq \text{content } K * ((\varepsilon/4 * (b \cdot i - a \cdot i) / \text{content } (cbox \ a \ b)) / \text{interv\_diff } K \ i)$ 
  using  $\text{mult\_left\_mono}$  by fastforce
  also have  $\dots = \varepsilon/4 * (b \cdot i - a \cdot i) / \text{content } (cbox \ a \ b) * \text{content } K / \text{interv\_diff } K \ i$ 
  by (simp add:  $\text{field\_split\_sims}$ )
  finally show ?thesis .
qed

```

```

qed
also have ... = (∑ K∈snd ‘ S. ε/4 * (b · i - a · i) / content (cbox a b) *
content K / interv_diff K i)
  unfolding interv_diff_def
  apply (rule sum.over_tagged_division_lemma [OF tagged_partial_division_of_Union_self
[OF S]])
  apply (simp add: box_eq_empty(1) content_eq_0)
  done
also have ... = ε/2 * ((b · i - a · i) / (2 * content (cbox a b))) * (∑ K∈snd
‘ S. content K / interv_diff K i)
  by (simp add: interv_diff_def sum_distrib_left mult.assoc)
also have ... ≤ (ε/2) * 1
  proof (rule mult_left_mono)
    have (b · i - a · i) * (∑ K∈snd ‘ S. content K / interv_diff K i) ≤ 2 *
content (cbox a b)
      unfolding interv_diff_def
    proof (rule sum_content_area_over_thin_division)
      show snd ‘ S division_of ∪ (snd ‘ S)
    by (auto intro: S tagged_partial_division_of_Union_self division_of_tagged_division)
      show ∪ (snd ‘ S) ⊆ cbox a b
        using S unfolding tagged_partial_division_of_def by force
      show a · i ≤ c · i c · i ≤ b · i
        using mem_box(2) that by blast+
    qed (use that in auto)
    then show (b · i - a · i) / (2 * content (cbox a b)) * (∑ K∈snd ‘ S.
content K / interv_diff K i) ≤ 1
      by (simp add: contab_gt0)
    qed (use ‹0 < ε› in auto)
    finally show ?thesis by simp
  qed
then have (∑ (x,K) ∈ S. norm (integral K h)) - (∑ (x,K) ∈ S. norm (content
K *R h x - integral K h)) ≤ ε/2
  by (simp add: Groups_Big.sum_subtractf [symmetric])
ultimately show (∑ (x,K) ∈ S. norm (integral K h)) < ε
  by linarith
qed
ultimately show ?thesis using that by auto
qed

```

proposition *equiintegrable_halfspace_restrictions_le:*

fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$

assumes $F: F$ *equiintegrable_on* $cbox\ a\ b$ **and** $f: f \in F$

and $norm\_f: \bigwedge h\ x. \llbracket h \in F; x \in cbox\ a\ b \rrbracket \implies norm(h\ x) \leq norm(f\ x)$

shows $(\bigcup i \in Basis. \bigcup c. \bigcup h \in F. \{(\lambda x. \text{if } x \cdot i \leq c \text{ then } h\ x \text{ else } 0)\})$

equiintegrable_on $cbox\ a\ b$

proof (cases $content(cbox\ a\ b) = 0$)

case *True*


```

    then show ?thesis by simp
next
case False
then have content(cbox a b) > 0
  using zero_less_measure_iff by blast
then have  $a \cdot i < b \cdot i$  if  $i \in \text{Basis}$  for  $i$ 
  using content_pos_lt_eq that by blast
have int_F:  $f$  integrable_on cbox a b if  $f \in F$  for  $f$ 
  using F that by (simp add: equiintegrable_on_def)
let ?CI =  $\lambda K h x. \text{content } K *_R h x - \text{integral } K h$ 
show ?thesis
  unfolding equiintegrable_on_def
proof (intro conjI; clarify)
  show int_lec:  $\llbracket i \in \text{Basis}; h \in F \rrbracket \implies (\lambda x. \text{if } x \cdot i \leq c \text{ then } h x \text{ else } 0)$ 
    integrable_on cbox a b for  $i c h$ 
    using integrable_restrict_Int [of  $\{x. x \cdot i \leq c\}$   $h$ ]
    by (simp add: inf_commute int_F integrable_split(1))
  show  $\exists \gamma. \text{gauge } \gamma \wedge$ 
     $(\forall f T. f \in (\bigcup i \in \text{Basis}. \bigcup c. \bigcup h \in F. \{\lambda x. \text{if } x \cdot i \leq c \text{ then } h x \text{ else } 0\}))$ 
 $\wedge$ 
     $T \text{ tagged\_division\_of cbox a b} \wedge \gamma \text{ fine } T \implies$ 
     $\text{norm } ((\sum (x,K) \in T. \text{content } K *_R f x) - \text{integral } (\text{cbox a b}) f)$ 
 $< \varepsilon)$ 
    if  $\varepsilon > 0$  for  $\varepsilon$ 
  proof -
    obtain  $\gamma 0$  where gauge  $\gamma 0$  and  $\gamma 0$ :
       $\bigwedge c i S h. \llbracket c \in \text{cbox a b}; i \in \text{Basis}; S \text{ tagged\_partial\_division\_of cbox a b};$ 
       $\gamma 0 \text{ fine } S; h \in F; \bigwedge x K. (x,K) \in S \implies (K \cap \{x. x \cdot i = c \cdot$ 
 $i\} \neq \{\}) \rrbracket$ 
       $\implies (\sum (x,K) \in S. \text{norm } (\text{integral } K h)) < \varepsilon/12$ 
    proof (rule bounded_equiintegral_over_thin_tagged_partial_division [OF F
      f, of  $\langle \varepsilon/12 \rangle$ ])
      show  $\bigwedge h x. \llbracket h \in F; x \in \text{cbox a b} \rrbracket \implies \text{norm } (h x) \leq \text{norm } (f x)$ 
        by (auto simp: norm_f)
      qed (use  $\langle \varepsilon > 0 \rangle$  in auto)
      obtain  $\gamma 1$  where gauge  $\gamma 1$ 
        and  $\gamma 1$ :  $\bigwedge h T. \llbracket h \in F; T \text{ tagged\_division\_of cbox a b}; \gamma 1 \text{ fine } T \rrbracket$ 
         $\implies \text{norm } ((\sum (x,K) \in T. \text{content } K *_R h x) - \text{integral}$ 
 $(\text{cbox a b}) h)$ 
         $< \varepsilon / (\gamma * (\text{Suc DIM}('b)))$ 
      proof -
        have e5:  $\varepsilon / (\gamma * (\text{Suc DIM}('b))) > 0$ 
          using  $\langle \varepsilon > 0 \rangle$  by auto
        then show ?thesis
          using F that by (auto simp: equiintegrable_on_def)
      qed
      have h_less3:  $(\sum (x,K) \in T. \text{norm } (?CI K h x)) < \varepsilon/3$ 
        if  $T \text{ tagged\_partial\_division\_of cbox a b} \wedge \gamma 1 \text{ fine } T \wedge h \in F$  for  $T h$ 
      proof -

```

```

have (∑ (x,K) ∈ T. norm (?CI K h x)) ≤ 2 * real DIM('b) * (ε/(7 * Suc
DIM('b)))
proof (rule Henstock_lemma_part2 [of h a b])
  show h integrable_on cbox a b
    using that F equiintegrable_on_def by metis
qed (use that ⟨ε > 0⟩ ⟨gauge γ1⟩ γ1 in auto)
also have ... < ε/3
  using ⟨ε > 0⟩ by (simp add: divide_simps)
finally show ?thesis .
qed
have *: norm ((∑ (x,K) ∈ T. content K *R f x) - integral (cbox a b) f) < ε
  if f: f = (λx. if x · i ≤ c then h x else 0)
  and T: T tagged_division_of cbox a b
  and fine: (λx. γ0 x ∩ γ1 x) fine T and i ∈ Basis h ∈ F for f T i c h
proof (cases a · i ≤ c ∧ c ≤ b · i)
  case True
  have finite T
  using T by blast
  define T' where T' ≡ {(x,K) ∈ T. K ∩ {x. x · i ≤ c} ≠ {}}
  then have T' ⊆ T
  by auto
  then have finite T'
  using ⟨finite T⟩ infinite_super by blast
  have T'_tagged: T' tagged_partial_division_of cbox a b
  by (meson T ⟨T' ⊆ T⟩ tagged_division_of_def tagged_partial_division_subset)
  have fine': γ0 fine T' γ1 fine T'
  using ⟨T' ⊆ T⟩ fine_Int fine_subset fine by blast+
  have int_KK': (∑ (x,K) ∈ T. integral K f) = (∑ (x,K) ∈ T'. integral K f)
  proof (rule sum.mono_neutral_right [OF ⟨finite T⟩ ⟨T' ⊆ T⟩])
    show ∀ i ∈ T - T'. (case i of (x, K) ⇒ integral K f) = 0
    using f ⟨finite T⟩ ⟨T' ⊆ T⟩ integral_restrict_Int [of _ {x. x · i ≤ c} h]
    by (auto simp: T'_def Int_commute)
  qed
  have (∑ (x,K) ∈ T. content K *R f x) = (∑ (x,K) ∈ T'. content K *R f
x)
  proof (rule sum.mono_neutral_right [OF ⟨finite T⟩ ⟨T' ⊆ T⟩])
    show ∀ i ∈ T - T'. (case i of (x, K) ⇒ content K *R f x) = 0
    using T f ⟨finite T⟩ ⟨T' ⊆ T⟩ by (force simp: T'_def)
  qed
  moreover have norm ((∑ (x,K) ∈ T'. content K *R f x) - integral (cbox
a b) f) < ε
  proof -
    have *: norm y < ε if norm x < ε/3 norm(x - y) ≤ 2 * ε/3 for x y::'b
    proof -
      have norm y ≤ norm x + norm(x - y)
      by (metis norm_minus_commute norm_triangle_sub)
      also have ... < ε/3 + 2*ε/3
      using that by linarith
      also have ... = ε

```

```

    by simp
    finally show ?thesis .
qed
have norm  $(\sum (x,K) \in T'. ?CI\ K\ h\ x)$ 
   $\leq (\sum (x,K) \in T'. norm\ (?CI\ K\ h\ x))$ 
  by (simp add: norm_sum_split_def)
also have ...  $< \varepsilon/3$ 
  by (intro h_less3 T'_tagged fine' that)
finally have norm  $(\sum (x,K) \in T'. ?CI\ K\ h\ x) < \varepsilon/3$  .
moreover have integral (cbox a b) f =  $(\sum (x,K) \in T. integral\ K\ f)$ 
using int_lec that by (auto simp: integral_combine_tagged_division_topdown)
moreover have norm  $(\sum (x,K) \in T'. ?CI\ K\ h\ x - ?CI\ K\ f\ x)$ 
   $\leq 2*\varepsilon/3$ 
proof -
  define T'' where  $T'' \equiv \{(x,K) \in T'. \neg (K \subseteq \{x. x \cdot i \leq c\})\}$ 
  then have  $T'' \subseteq T'$ 
    by auto
  then have finite T''
    using ⟨finite T'⟩ infinite_super by blast
  have T''_tagged:  $T''$  tagged_partial_division_of cbox a b
    using T'_tagged ⟨ $T'' \subseteq T'$ ⟩ tagged_partial_division_subset by blast
  have fine'':  $\gamma 0$  fine T''  $\gamma 1$  fine T''
    using ⟨ $T'' \subseteq T'$ ⟩ fine' by (blast intro: fine_subset)+
  have  $(\sum (x,K) \in T'. ?CI\ K\ h\ x - ?CI\ K\ f\ x)$ 
    =  $(\sum (x,K) \in T''. ?CI\ K\ h\ x - ?CI\ K\ f\ x)$ 
  proof (clarify intro!: sum.mono_neutral_right [OF ⟨finite T'⟩ ⟨ $T'' \subseteq$ 
T'⟩])
    fix x K
    assume  $(x,K) \in T' (x,K) \notin T''$ 
    then have  $x \in K\ x \cdot i \leq c\ \{x. x \cdot i \leq c\} \cap K = K$ 
      using T''_def T'_tagged tagged_partial_division_of_def by blast+
    then show  $?CI\ K\ h\ x - ?CI\ K\ f\ x = 0$ 
      using integral_restrict_Int [of _  $\{x. x \cdot i \leq c\}\ h]$  by (auto simp: f)
  qed
moreover have norm  $(\sum (x,K) \in T''. ?CI\ K\ h\ x - ?CI\ K\ f\ x) \leq 2*\varepsilon/3$ 
proof -
  define A where  $A \equiv \{(x,K) \in T''. x \cdot i \leq c\}$ 
  define B where  $B \equiv \{(x,K) \in T''. x \cdot i > c\}$ 
  then have  $A \subseteq T''\ B \subseteq T''$  and disj:  $A \cap B = \{\}$  and T''_eq:  $T''$ 
= A  $\cup$  B
    by (auto simp: A_def B_def)
  then have finite A finite B
    using ⟨finite T''⟩ by (auto intro: finite_subset)
  have A_tagged: A tagged_partial_division_of cbox a b
    using T''_tagged ⟨ $A \subseteq T''$ ⟩ tagged_partial_division_subset by blast
  have fineA:  $\gamma 0$  fine A  $\gamma 1$  fine A
    using ⟨ $A \subseteq T''$ ⟩ fine'' by (blast intro: fine_subset)+
  have B_tagged: B tagged_partial_division_of cbox a b
    using T''_tagged ⟨ $B \subseteq T''$ ⟩ tagged_partial_division_subset by blast

```

```

have fineB:  $\gamma 0$  fine B  $\gamma 1$  fine B
  using  $\langle B \subseteq T'' \rangle$  fine'' by (blast intro: fine_subset)+
have norm  $(\sum (x,K) \in T''. ?CI K h x - ?CI K f x)$ 
   $\leq (\sum (x,K) \in T''. norm (?CI K h x - ?CI K f x))$ 
  by (simp add: norm_sum_split_def)
also have ... =  $(\sum (x,K) \in A. norm (?CI K h x - ?CI K f x)) +$ 
   $(\sum (x,K) \in B. norm (?CI K h x - ?CI K f x))$ 
  by (simp add: sum.union_disjoint T''_eq disj  $\langle finite A \rangle$   $\langle finite B \rangle$ )
also have ... =  $(\sum (x,K) \in A. norm (integral K h - integral K f)) +$ 
   $(\sum (x,K) \in B. norm (?CI K h x + integral K f))$ 
  by (auto simp: A_def B_def f_norm_minus_commute intro!: sum.cong
arg_cong2 [where f = (+)])
also have ...  $\leq (\sum (x,K) \in A. norm (integral K h)) +$ 
   $(\sum (x,K) \in (\lambda(x,K). (x,K \cap \{x. x \cdot i \leq c\}))) ' A. norm$ 
   $(integral K h))$ 
  +  $((\sum (x,K) \in B. norm (?CI K h x)) +$ 
   $(\sum (x,K) \in B. norm (integral K h)) +$ 
   $(\sum (x,K) \in (\lambda(x,K). (x,K \cap \{x. c \leq x \cdot i\}))) ' B. norm$ 
   $(integral K h))$ 
proof (rule add_mono)
  show  $(\sum (x,K) \in A. norm (integral K h - integral K f))$ 
     $\leq (\sum (x,K) \in A. norm (integral K h)) +$ 
     $(\sum (x,K) \in (\lambda(x,K). (x,K \cap \{x. x \cdot i \leq c\}))) ' A.$ 
     $norm (integral K h)$ 
proof (subst sum.reindex_nontrivial [OF  $\langle finite A \rangle$ ], clarsimp)
  fix x K L
  assume  $(x,K) \in A$   $(x,L) \in A$ 
  and int_ne0:  $integral (L \cap \{x. x \cdot i \leq c\}) h \neq 0$ 
  and eq:  $K \cap \{x. x \cdot i \leq c\} = L \cap \{x. x \cdot i \leq c\}$ 
  have False if  $K \neq L$ 
  proof -
    obtain u v where uv:  $L = cbox u v$ 
    using T'_tagged  $\langle (x, L) \in A \rangle$   $\langle A \subseteq T'' \rangle$   $\langle T'' \subseteq T' \rangle$  by (blast
dest: tagged_partial_division_ofD)
    have interior  $(K \cap \{x. x \cdot i \leq c\}) = \{\}$ 
    proof (rule tagged_division_split_left_inj [OF _  $\langle (x,K) \in A \rangle$ 
 $\langle (x,L) \in A \rangle$ ])
      show  $A$  tagged_division_of  $\bigcup (snd ' A)$ 
      using A_tagged tagged_partial_division_of_Union_self by
      auto
      show  $K \cap \{x. x \cdot i \leq c\} = L \cap \{x. x \cdot i \leq c\}$ 
      using eq  $\langle i \in Basis \rangle$  by auto
    qed (use that in auto)
    then show False
  using interval_split [OF  $\langle i \in Basis \rangle$ ] int_ne0 content_eq_0_interior
eq uv by fastforce
  qed
  then show  $K = L$  by blast
next

```

```

    show  $(\sum (x,K) \in A. \text{norm} (\text{integral } K \ h - \text{integral } K \ f))$ 
       $\leq (\sum (x,K) \in A. \text{norm} (\text{integral } K \ h)) +$ 
         $\text{sum} ((\lambda(x,K). \text{norm} (\text{integral } K \ h)) \circ (\lambda(x,K). (x,K \cap \{x.$ 
 $x \cdot i \leq c\}))) A$ 
      using integral_restrict_Int [of _  $\{x. x \cdot i \leq c\} h]$  f
      by (auto simp: Int_commute A_def [symmetric] sum.distrib
[symmetric] intro!: sum_mono norm_triangle_ineq4)
    qed
  next
    show  $(\sum (x,K) \in B. \text{norm} (?CI \ K \ h \ x + \text{integral } K \ f))$ 
       $\leq (\sum (x,K) \in B. \text{norm} (?CI \ K \ h \ x)) + (\sum (x,K) \in B. \text{norm} (\text{integral}$ 
 $K \ h)) +$ 
         $(\sum (x,K) \in (\lambda(x,K). (x,K \cap \{x. c \leq x \cdot i\})) ' B. \text{norm} (\text{integral}$ 
 $K \ h))$ 
    proof (subst sum.reindex_nontrivial [OF  $\langle \text{finite } B \rangle$ ], clarsimp)
      fix x K L
      assume  $(x,K) \in B \ (x,L) \in B$ 
      and int_ne0:  $\text{integral} (L \cap \{x. c \leq x \cdot i\}) \ h \neq 0$ 
      and eq:  $K \cap \{x. c \leq x \cdot i\} = L \cap \{x. c \leq x \cdot i\}$ 
      have False if  $K \neq L$ 
      proof -
        obtain u v where uv:  $L = \text{cbox } u \ v$ 
        using T'_tagged  $\langle (x, L) \in B \rangle \langle B \subseteq T'' \rangle \langle T'' \subseteq T' \rangle$  by (blast
dest: tagged_partial_division_ofD)
        have interior  $(K \cap \{x. c \leq x \cdot i\}) = \{\}$ 
        proof (rule tagged_division_split_right_inj [OF _  $\langle (x,K) \in B \rangle$ 
 $\langle (x,L) \in B \rangle$ ])
          show  $B \text{ tagged\_division\_of } \bigcup (\text{snd } ' B)$ 
          using B_tagged tagged_partial_division_of_Union_self by
auto
          show  $K \cap \{x. c \leq x \cdot i\} = L \cap \{x. c \leq x \cdot i\}$ 
          using eq  $\langle i \in \text{Basis} \rangle$  by auto
        qed (use that in auto)
        then show False
          using interval_split [OF  $\langle i \in \text{Basis} \rangle$ ] int_ne0
            content_eq_0_interior eq uv by fastforce
      qed
    then show  $K = L$  by blast
  next
    show  $(\sum (x,K) \in B. \text{norm} (?CI \ K \ h \ x + \text{integral } K \ f))$ 
       $\leq (\sum (x,K) \in B. \text{norm} (?CI \ K \ h \ x)) +$ 
         $(\sum (x,K) \in B. \text{norm} (\text{integral } K \ h)) + \text{sum} ((\lambda(x,K). \text{norm}$ 
 $(\text{integral } K \ h)) \circ (\lambda(x,K). (x,K \cap \{x. c \leq x \cdot i\}))) B$ 
    proof (clarsimp simp: B_def [symmetric] sum.distrib [symmetric]
intro!: sum_mono)
      fix x K
      assume  $(x,K) \in B$ 
      have *:  $i = i1 + i2 \implies \text{norm}(c + i1) \leq \text{norm } c + \text{norm } i +$ 
 $\text{norm}(i2)$ 

```

```

      for i::'b and c i1 i2
    by (metis add.commute add.left_commute add_diff_cancel_right'
dual_order.refl norm_add_rule_thm norm_triangle_ineq4)
    obtain u v where uv: K = cbox u v
      using T'_tagged ⟨(x,K) ∈ B⟩ ⟨B ⊆ T'⟩ ⟨T' ⊆ T⟩ by (blast
dest: tagged_partial_division_ofD)
    have huv: h integrable_on cbox u v
    proof (rule integrable_on_subcbox)
      show cbox u v ⊆ cbox a b
    using B_tagged ⟨(x,K) ∈ B⟩ uv by (blast dest: tagged_partial_division_ofD)
      show h integrable_on cbox a b
      by (simp add: int_F ⟨h ∈ F⟩)
    qed
    have integral K h = integral K f + integral (K ∩ {x. c ≤ x · i}) h
      using integral_restrict_Int [of _ {x. x · i ≤ c} h] f uv ⟨i ∈
Basis⟩
      by (simp add: Int_commute integral_split [OF huv ⟨i ∈ Basis⟩])
    then show norm (?CI K h x + integral K f)
      ≤ norm (?CI K h x) + norm (integral K h) + norm (integral
(K ∩ {x. c ≤ x · i}) h)
      by (rule *)
    qed
  qed
qed
also have ... ≤ 2*ε/3
proof -
  have overlap: K ∩ {x. x · i = c} ≠ {} if (x,K) ∈ T'' for x K
  proof -
    obtain y y' where y: y' ∈ K c < y' · i y ∈ K y · i ≤ c
      using that T''_def T'_def ⟨(x,K) ∈ T''⟩ by fastforce
    obtain u v where uv: K = cbox u v
  using T''_tagged ⟨(x,K) ∈ T''⟩ by (blast dest: tagged_partial_division_ofD)
    then have connected K
      by (simp add: is_interval_connected)
    then have (∃ z ∈ K. z · i = c)
      using y connected_ivt_component by fastforce
    then show ?thesis
      by fastforce
    qed
  have **: [x < ε/12; y < ε/12; z ≤ ε/2] ⇒ x + y + z ≤ 2 * ε/3 for
x y z
    by auto
  show ?thesis
  proof (rule **)
    have cb_ab: (∑ j ∈ Basis. if j = i then c *R i else (a · j) *R j) ∈
cbox a b
      using ⟨i ∈ Basis⟩ True ⟨∧ i. i ∈ Basis ⇒ a · i < b · i⟩
      by (force simp add: mem_box sum_if_inner [where f = λj. c])
    show (∑ (x,K) ∈ A. norm (integral K h)) < ε/12

```

```

    using ⟨i ∈ Basis⟩ ⟨A ⊆ T''⟩ overlap
  by (force simp add: sum_if_inner [where f = λj. c]
      intro!: γ0 [OF cb_ab ⟨i ∈ Basis⟩ A_tagged_fineA(1) ⟨h ∈ F⟩])
let ?F = λ(x,K). (x, K ∩ {x. x • i ≤ c})
have 1: ?F ' A tagged_partial_division_of_cbox a b
  unfolding tagged_partial_division_of_def
proof (intro conjI strip)
  show ∧x K. (x, K) ∈ ?F ' A ⟹ ∃ a b. K = cbox a b
    using A_tagged_interval_split(1) [OF ⟨i ∈ Basis⟩, of _ _ c]
    by (force dest: tagged_partial_division_ofD(4))
  show ∧x K. (x, K) ∈ ?F ' A ⟹ x ∈ K
using A_def A_tagged by (fastforce dest: tagged_partial_division_ofD)
qed (use A_tagged in ⟨fastforce dest: tagged_partial_division_ofD⟩)+
  have 2: γ0 fine (λ(x,K). (x,K ∩ {x. x • i ≤ c})) ' A
    using fineA(1) fine_def by fastforce
  show (∑ (x,K) ∈ (λ(x,K). (x,K ∩ {x. x • i ≤ c})) ' A. norm (integral
K h)) < ε/12
    using ⟨i ∈ Basis⟩ ⟨A ⊆ T''⟩ overlap
  by (force simp add: sum_if_inner [where f = λj. c]
      intro!: γ0 [OF cb_ab ⟨i ∈ Basis⟩ 1 2 ⟨h ∈ F⟩])
have *: [x < ε/3; y < ε/12; z < ε/12] ⟹ x + y + z ≤ ε/2 for x
y z
  by auto
show (∑ (x,K) ∈ B. norm (?CI K h x)) +
  (∑ (x,K) ∈ B. norm (integral K h)) +
  (∑ (x,K) ∈ (λ(x,K). (x,K ∩ {x. c ≤ x • i})) ' B. norm (integral
K h))
  ≤ ε/2
proof (rule *)
  show (∑ (x,K) ∈ B. norm (?CI K h x)) < ε/3
    by (intro h_less3 B_tagged_fineB that)
  show (∑ (x,K) ∈ B. norm (integral K h)) < ε/12
    using ⟨i ∈ Basis⟩ ⟨B ⊆ T''⟩ overlap
  by (force simp add: sum_if_inner [where f = λj. c]
      intro!: γ0 [OF cb_ab ⟨i ∈ Basis⟩ B_tagged_fineB(1) ⟨h ∈ F⟩])
let ?F = λ(x,K). (x, K ∩ {x. c ≤ x • i})
have 1: ?F ' B tagged_partial_division_of_cbox a b
  unfolding tagged_partial_division_of_def
proof (intro conjI strip)
  show ∧x K. (x, K) ∈ ?F ' B ⟹ ∃ a b. K = cbox a b
    using B_tagged_interval_split(2) [OF ⟨i ∈ Basis⟩, of _ _ c]
    by (force dest: tagged_partial_division_ofD(4))
  show ∧x K. (x, K) ∈ ?F ' B ⟹ x ∈ K
using B_def B_tagged by (fastforce dest: tagged_partial_division_ofD)
qed (use B_tagged in ⟨fastforce dest: tagged_partial_division_ofD⟩)+
  have 2: γ0 fine (λ(x,K). (x,K ∩ {x. c ≤ x • i})) ' B
    using fineB(1) fine_def by fastforce
  show (∑ (x,K) ∈ (λ(x,K). (x,K ∩ {x. c ≤ x • i})) ' B. norm
(integral K h)) < ε/12

```

```

    using ⟨i ∈ Basis⟩ ⟨A ⊆ T'⟩ overlap
    by (force simp add: B_def sum_if_inner [where f = λj. c]
        intro!: γ0 [OF cb_ab ⟨i ∈ Basis⟩ 1 2 ⟨h ∈ F⟩])
  qed
qed
qed
finally show ?thesis .
qed
ultimately show ?thesis by metis
qed
ultimately show ?thesis
  by (simp add: sum_subtractf [symmetric] int_KK' *)
qed
ultimately show ?thesis by metis
next
case False
then consider c < a • i | b • i < c
  by auto
then show ?thesis
proof cases
case 1
then have f0: f x = 0 if x ∈ cbox a b for x
  using that f ⟨i ∈ Basis⟩ mem_box(2) by force
then have int_f0: integral (cbox a b) f = 0
  by (simp add: integral_cong)
have f0_tag: f x = 0 if (x,K) ∈ T for x K
  using T f0 that by (meson tag_in_interval)
then have (∑ (x,K) ∈ T. content K *R f x) = 0
  by (metis (mono_tags, lifting) real_vector.scale_eq_0_iff split_conv
    sum.neutral surj_pair)
then show ?thesis
  using ⟨0 < ε⟩ by (simp add: int_f0)
next
case 2
then have fh: f x = h x if x ∈ cbox a b for x
  using that f ⟨i ∈ Basis⟩ mem_box(2) by force
then have int_f: integral (cbox a b) f = integral (cbox a b) h
  using integral_cong by blast
have fh_tag: f x = h x if (x,K) ∈ T for x K
  using T fh that by (meson tag_in_interval)
then have fh: (∑ (x,K) ∈ T. content K *R f x) = (∑ (x,K) ∈ T. content
  K *R h x)
  by (metis (mono_tags, lifting) split_cong sum.cong)
show ?thesis
  unfolding fh int_f
proof (rule less_trans [OF γ1])
show γ1 fine T
  by (meson fine fine_Int)
show ε / (7 * Suc DIM('b)) < ε

```



```

      using ‹0 < ε› by (force simp: divide_simps)+
    qed (use that in auto)
  qed
  qed
  have gauge (λx. γ0 x ∩ γ1 x)
    by (simp add: ‹gauge γ0› ‹gauge γ1› gauge_Int)
  then show ?thesis
    by (auto intro: *)
  qed
  qed
  qed

corollary equiintegrable_halfspace_restrictions_ge:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes F: F equiintegrable_on cbox a b and f: f ∈ F
  and norm_f: ⋀h x. ‖h ∈ F; x ∈ cbox a b‖ ⇒ norm(h x) ≤ norm(f x)
  shows (⋃ i ∈ Basis. ⋃ c. ⋃ h ∈ F. {(λx. if x • i ≥ c then h x else 0)})
    equiintegrable_on cbox a b
proof -
  have *: (⋃ i ∈ Basis. ⋃ c. ⋃ h ∈ (λf. f ∘ uminus) ‘ F. {(λx. if x • i ≤ c then h x
    else 0)})
    equiintegrable_on cbox (- b) (- a)
  proof (rule equiintegrable_halfspace_restrictions_le)
    show (λf. f ∘ uminus) ‘ F equiintegrable_on cbox (- b) (- a)
      using F equiintegrable_reflect by blast
    show f ∘ uminus ∈ (λf. f ∘ uminus) ‘ F
      using f by auto
    show ⋀h x. ‖h ∈ (λf. f ∘ uminus) ‘ F; x ∈ cbox (- b) (- a)‖ ⇒ norm(h x)
      ≤ norm((f ∘ uminus) x)
      using f unfolding comp_def image_iff
      by (metis (no_types, lifting) equation_minus_iff imageE norm_f umi-
nus_interval_vector)
  qed
  have eq: (λf. f ∘ uminus) ‘
    (⋃ i ∈ Basis. ⋃ c. ⋃ h ∈ F. {(λx. if x • i ≤ c then (h ∘ uminus) x else 0)}) =
    (⋃ i ∈ Basis. ⋃ c. ⋃ h ∈ F. {(λx. if c ≤ x • i then h x else 0)}) (is ?lhs =
    ?rhs)
  proof
    show ?lhs ⊆ ?rhs
      using minus_le_iff by fastforce
    show ?rhs ⊆ ?lhs
      apply clarsimp
      apply (rule_tac x=λx. if c ≤ (-x) • i then h(-x) else 0 in image_eqI)
      using le_minus_iff by fastforce+
  qed
  show ?thesis
    using equiintegrable_reflect [OF *] by (auto simp: eq)
qed

```

corollary *equiintegrable_halfspace_restrictions_lt*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes $F: F \text{ equiintegrable\_on cbox } a \ b$ **and** $f: f \in F$
and $\text{norm\_f}: \bigwedge h \ x. \llbracket h \in F; x \in \text{cbox } a \ b \rrbracket \implies \text{norm}(h \ x) \leq \text{norm}(f \ x)$
shows $(\bigcup i \in \text{Basis}. \bigcup c. \bigcup h \in F. \{(\lambda x. \text{if } x \cdot i < c \text{ then } h \ x \text{ else } 0)\}) \text{ equiintegrable\_on cbox } a \ b$
(is ?G equiintegrable_on cbox } a \ b)
proof –
have $*$: $(\bigcup i \in \text{Basis}. \bigcup c. \bigcup h \in F. \{(\lambda x. \text{if } c \leq x \cdot i \text{ then } h \ x \text{ else } 0)\}) \text{ equiintegrable\_on cbox } a \ b$
using *equiintegrable_halfspace_restrictions_ge* $[OF \ F \ f] \text{ norm\_f}$ **by** *auto*
have $(\lambda x. \text{if } x \cdot i < c \text{ then } h \ x \text{ else } 0) = (\lambda x. h \ x - (\text{if } c \leq x \cdot i \text{ then } h \ x \text{ else } 0))$
if $i \in \text{Basis}$ $h \in F$ **for** $i \ c \ h$
using *that* **by** *force*
then show *?thesis*
by (*blast intro: equiintegrable_on_subset* $[OF \text{ equiintegrable\_diff } [OF \ F \ *]]$)
qed

corollary *equiintegrable_halfspace_restrictions_gt*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes $F: F \text{ equiintegrable\_on cbox } a \ b$ **and** $f: f \in F$
and $\text{norm\_f}: \bigwedge h \ x. \llbracket h \in F; x \in \text{cbox } a \ b \rrbracket \implies \text{norm}(h \ x) \leq \text{norm}(f \ x)$
shows $(\bigcup i \in \text{Basis}. \bigcup c. \bigcup h \in F. \{(\lambda x. \text{if } x \cdot i > c \text{ then } h \ x \text{ else } 0)\}) \text{ equiintegrable\_on cbox } a \ b$
(is ?G equiintegrable_on cbox } a \ b)
proof –
have $*$: $(\bigcup i \in \text{Basis}. \bigcup c. \bigcup h \in F. \{(\lambda x. \text{if } c \geq x \cdot i \text{ then } h \ x \text{ else } 0)\}) \text{ equiintegrable\_on cbox } a \ b$
using *equiintegrable_halfspace_restrictions_le* $[OF \ F \ f] \text{ norm\_f}$ **by** *auto*
have $(\lambda x. \text{if } x \cdot i > c \text{ then } h \ x \text{ else } 0) = (\lambda x. h \ x - (\text{if } c \geq x \cdot i \text{ then } h \ x \text{ else } 0))$
if $i \in \text{Basis}$ $h \in F$ **for** $i \ c \ h$
using *that* **by** *force*
then show *?thesis*
by (*blast intro: equiintegrable_on_subset* $[OF \text{ equiintegrable\_diff } [OF \ F \ *]]$)
qed

proposition *equiintegrable_closed_interval_restrictions*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes $f: f \text{ integrable\_on cbox } a \ b$
shows $(\bigcup c \ d. \{(\lambda x. \text{if } x \in \text{cbox } c \ d \text{ then } f \ x \text{ else } 0)\}) \text{ equiintegrable\_on cbox } a \ b$
proof –
let $?g = \lambda B \ c \ d \ x. \text{if } \forall i \in B. c \cdot i \leq x \cdot i \wedge x \cdot i \leq d \cdot i \text{ then } f \ x \text{ else } 0$
have $*$: $\text{insert } f \ (\bigcup c \ d. \{?g \ B \ c \ d\}) \text{ equiintegrable\_on cbox } a \ b$ **if** $B \subseteq \text{Basis}$ **for** B
proof –
have *finite* B
using *finite_Basis finite_subset* $\langle B \subseteq \text{Basis} \rangle$ **by** *blast*
then show *?thesis* **using** $\langle B \subseteq \text{Basis} \rangle$

```

proof (induction B)
  case empty
  with f show ?case by auto
next
  case (insert i B)
  then have  $i \in \text{Basis } B \subseteq \text{Basis}$ 
  by auto
  have *:  $\text{norm } (h \ x) \leq \text{norm } (f \ x)$ 
  if  $h \in \text{insert } f \ (\bigcup c \ d. \ \{?g \ B \ c \ d\})$   $x \in \text{cbox } a \ b$  for  $h \ x$ 
  using that by auto
  define F where  $F \equiv (\bigcup i \in \text{Basis}. \bigcup \xi. \bigcup h \in \text{insert } f \ (\bigcup i \in \text{Basis}. \bigcup \psi. \bigcup h \in \text{insert } f \ (\bigcup c \ d. \ \{?g \ B \ c \ d\})).$ 
 $\{\lambda x. \text{if } x \cdot i \leq \psi \text{ then } h \ x \text{ else } 0\}).$ 
 $\{\lambda x. \text{if } \xi \leq x \cdot i \text{ then } h \ x \text{ else } 0\})$ 
  show ?case
  proof (rule equiintegrable_on_subset)
    have F equiintegrable_on cbox a b
    unfolding F_def
    proof (rule equiintegrable_halfspace_restrictions_ge)
      show  $\text{insert } f \ (\bigcup i \in \text{Basis}. \bigcup \xi. \bigcup h \in \text{insert } f \ (\bigcup c \ d. \ \{?g \ B \ c \ d\})).$ 
 $\{\lambda x. \text{if } x \cdot i \leq \xi \text{ then } h \ x \text{ else } 0\}$  equiintegrable_on cbox a b
      by (intro * f equiintegrable_on_insert equiintegrable_halfspace_restrictions_le
[OF insert.IH insertI1]  $\langle B \subseteq \text{Basis} \rangle$ )
      show  $\text{norm}(h \ x) \leq \text{norm}(f \ x)$ 
      if  $h \in \text{insert } f \ (\bigcup i \in \text{Basis}. \bigcup \xi. \bigcup h \in \text{insert } f \ (\bigcup c \ d. \ \{?g \ B \ c \ d\})).$   $\{\lambda x.$ 
 $\text{if } x \cdot i \leq \xi \text{ then } h \ x \text{ else } 0\}$ 
 $x \in \text{cbox } a \ b$  for  $h \ x$ 
      using that by auto
    qed auto
    then show  $\text{insert } f \ F$ 
      equiintegrable_on cbox a b
      by (blast intro: f equiintegrable_on_insert)
    show  $\text{insert } f \ (\bigcup c \ d. \ \{\lambda x. \text{if } \forall j \in \text{insert } i \ B. \ c \cdot j \leq x \cdot j \wedge x \cdot j \leq d \cdot j$ 
then  $f \ x \text{ else } 0\})$ 
 $\subseteq \text{insert } f \ F$ 
    using  $\langle i \in \text{Basis} \rangle$ 
    apply clarify
    apply (simp add: F_def)
    apply (drule_tac  $x=i$  in bspec, assumption)
    apply (drule_tac  $x=c \cdot i$  in spec, clarify)
    apply (drule_tac  $x=i$  in bspec, assumption)
    apply (drule_tac  $x=d \cdot i$  in spec)
    apply (clarsimp simp: fun_eq_iff)
    apply (drule_tac  $x=c$  in spec)
    apply (drule_tac  $x=d$  in spec)
    apply (simp split: if_split_asm)
    done
  qed
qed

```

```

qed
show ?thesis
by (rule equiintegrable_on_subset [OF * [OF subset_refl]]) (auto simp: mem_box)
qed

```

6.27.3 Continuity of the indefinite integral

proposition *indefinite_integral_continuous*:

```

fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: euclidean_space
assumes int_f: f integrable_on cbox a b
and c: c  $\in$  cbox a b and d: d  $\in$  cbox a b 0 <  $\varepsilon$ 
obtains  $\delta$  where 0 <  $\delta$ 

$$\bigwedge c' d'. \llbracket c' \in \text{cbox } a \text{ } b; d' \in \text{cbox } a \text{ } b; \text{norm}(c' - c) \leq \delta; \text{norm}(d' - d) \leq \delta \rrbracket$$


$$\implies \text{norm}(\text{integral}(\text{cbox } c' \text{ } d') f - \text{integral}(\text{cbox } c \text{ } d) f) < \varepsilon$$


```

proof –

```

{ assume  $\exists c' d'. c' \in \text{cbox } a \text{ } b \wedge d' \in \text{cbox } a \text{ } b \wedge \text{norm}(c' - c) \leq \delta \wedge \text{norm}(d' - d) \leq \delta \wedge$ 

```

```

 $\text{norm}(\text{integral}(\text{cbox } c' \text{ } d') f - \text{integral}(\text{cbox } c \text{ } d) f) \geq \varepsilon$ 
(is  $\exists c' d'. ?\Phi \text{ } c' \text{ } d' \text{ } \delta$ ) if 0 <  $\delta$  for  $\delta$ 

```

```

then have  $\exists c' d'. ?\Phi \text{ } c' \text{ } d' (1 / \text{Suc } n)$  for  $n$ 

```

```

by simp

```

```

then obtain u v where  $\bigwedge n. ?\Phi (u \text{ } n) (v \text{ } n) (1 / \text{Suc } n)$ 

```

```

by metis

```

```

then have u: u n  $\in$  cbox a b and norm_u: norm(u n - c)  $\leq$  1 / Suc n

```

```

and v: v n  $\in$  cbox a b and norm_v: norm(v n - d)  $\leq$  1 / Suc n

```

```

and  $\varepsilon$ :  $\varepsilon \leq \text{norm}(\text{integral}(\text{cbox } (u \text{ } n) (v \text{ } n)) f - \text{integral}(\text{cbox } c \text{ } d) f)$  for

```

n

```

by blast+

```

```

then have False

```

```

proof –

```

```

have uvn: cbox (u n) (v n)  $\subseteq$  cbox a b for  $n$ 

```

```

by (meson u v mem_box(2) subset_box(1))

```

```

define S where  $S \equiv \bigcup i \in \text{Basis}. \{x. x \cdot i = c \cdot i\} \cup \{x. x \cdot i = d \cdot i\}$ 

```

```

have negligible S

```

```

unfolding S_def by force

```

```

then have int_f': ( $\lambda x. \text{if } x \in S \text{ then } 0 \text{ else } f \text{ } x$ ) integrable_on cbox a b

```

```

by (force intro: integrable_spike assms)

```

```

have get_n:  $\exists n. \forall m \geq n. x \in \text{cbox } (u \text{ } m) (v \text{ } m) \longleftrightarrow x \in \text{cbox } c \text{ } d$  if  $x: x \notin S$ 

```

for x

```

proof –

```

```

define  $\varepsilon$  where  $\varepsilon \equiv \text{Min}((\lambda i. \min |x \cdot i - c \cdot i| |x \cdot i - d \cdot i|) \text{ } \text{Basis})$ 

```

```

have  $\varepsilon > 0$ 

```

```

using  $\langle x \notin S \rangle$  by (auto simp: S_def  $\varepsilon$ _def)

```

```

then obtain n where  $n \neq 0$  and n: 1 / (real n) <  $\varepsilon$ 

```

```

by (metis inverse_eq_divide real_arch_inverse)

```

```

have emin:  $\varepsilon \leq \min |x \cdot i - c \cdot i| |x \cdot i - d \cdot i|$  if  $i \in \text{Basis}$  for  $i$ 

```

```

unfolding  $\varepsilon$ _def

```

```

by (meson Min.coboundedI euclidean_space_class.finite_Basis finite_imageI

```

```

image_iff that)
  have 1 / real (Suc n) < ε
    using n ⟨n ≠ 0⟩ ⟨ε > 0⟩ by (simp add: field_simps)
  have x ∈ cbox (u m) (v m) ⟷ x ∈ cbox c d if m ≥ n for m
  proof -
    have *: [|u - c| ≤ n; |v - d| ≤ n; N < |x - c|; N < |x - d|; n ≤ N]
      ⟹ u ≤ x ∧ x ≤ v ⟷ c ≤ x ∧ x ≤ d for N n u v c d and x::real
    by linarith
    have (u m · i ≤ x · i ∧ x · i ≤ v m · i) = (c · i ≤ x · i ∧ x · i ≤ d · i)
      if i ∈ Basis for i
    proof (rule *)
      show |u m · i - c · i| ≤ 1 / Suc m
        using norm_u [of m]
        by (metis (full_types) order_trans Basis_le_norm inner_commute
inner_diff_right that)
      show |v m · i - d · i| ≤ 1 / real (Suc m)
        using norm_v [of m]
        by (metis (full_types) order_trans Basis_le_norm inner_commute
inner_diff_right that)
      show 1/n < |x · i - c · i| 1/n < |x · i - d · i|
        using n ⟨n ≠ 0⟩ emin [OF ⟨i ∈ Basis⟩]
        by (simp_all add: inverse_eq_divide)
      show 1 / real (Suc m) ≤ 1 / real n
        using ⟨n ≠ 0⟩ ⟨m ≥ n⟩ by (simp add: field_split_simps)
    qed
    then show ?thesis by (simp add: mem_box)
  qed
  then show ?thesis by blast
qed
have 1: range (λn x. if x ∈ cbox (u n) (v n) then if x ∈ S then 0 else f x else
0) equiintegrable_on cbox a b
  by (blast intro: equiintegrable_on_subset [OF equiintegrable_closed_interval_restrictions
[OF int_f]])
have 2: (λn. if x ∈ cbox (u n) (v n) then if x ∈ S then 0 else f x else 0)
  ⟶ (if x ∈ cbox c d then if x ∈ S then 0 else f x else 0) for x
by (fastforce simp: dest: get_n intro: tendsto_eventually_essentiallyI)
have [simp]: cbox c d ∩ cbox a b = cbox c d
  using c d by (force simp: mem_box)
have [simp]: cbox (u n) (v n) ∩ cbox a b = cbox (u n) (v n) for n
  using u v by (fastforce simp: mem_box intro: order.trans)
have ∧y A. y ∈ A - S ⟹ f y = (λx. if x ∈ S then 0 else f x) y
  by simp
then have ∧A. integral A (λx. if x ∈ S then 0 else f (x)) = integral A (λx.
f (x))
  by (blast intro: integral_spike [OF ⟨negligible S⟩])
moreover
obtain N where dist (integral (cbox (u N) (v N)) (λx. if x ∈ S then 0 else f
x))
  (integral (cbox c d) (λx. if x ∈ S then 0 else f x)) < ε

```

```

    using equiintegrable_limit [OF 1 2] ⟨0 < ε⟩ by (force simp: integral_restrict_Int
lim_sequentially)
    ultimately have dist (integral (cbox (u N) (v N)) f) (integral (cbox c d) f)
< ε
    by simp
    then show False
    by (metis dist_norm not_le ε)
qed
}
then show ?thesis
by (meson not_le that)
qed

```

```

corollary indefinite_integral_uniformly_continuous:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: euclidean_space
  assumes f_integrable_on_cbox a b
  shows uniformly_continuous_on (cbox (Pair a a) (Pair b b)) (λy. integral (cbox
(fst y) (snd y)) f)
proof -
  show ?thesis
proof (rule compact_uniformly_continuous, clarsimp simp add: continuous_on_iff)
  fix c d and ε::real
  assume c: c ∈ cbox a b and d: d ∈ cbox a b and 0 < ε
  obtain δ where 0 < δ and δ:
    ∧c' d'. ‖c' ∈ cbox a b; d' ∈ cbox a b; norm(c' - c) ≤ δ; norm(d' - d)
≤ δ‖
    ⇒ norm(integral(cbox c' d') f -
integral(cbox c d) f) < ε
  using indefinite_integral_continuous ⟨0 < ε⟩ assms c d by blast
  show ∃δ > 0. ∀x' ∈ cbox (a, a) (b, b).
    dist x' (c, d) < δ ⟶
    dist (integral (cbox (fst x') (snd x')) f)
(integral (cbox c d) f)
< ε
  using ⟨0 < δ⟩
  by (force simp: dist_norm intro: δ order_trans [OF norm_fst_le] order_trans
[OF norm_snd_le] less_imp_le)
qed auto
qed

```

```

corollary bounded_integrals_over_subintervals:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: euclidean_space
  assumes f_integrable_on_cbox a b
  shows bounded {integral (cbox c d) f | c d. cbox c d ⊆ cbox a b}
proof -
  have bounded ((λy. integral (cbox (fst y) (snd y)) f) ` cbox (a, a) (b, b))
(is bounded ?I)
  by (blast intro: bounded_cbox bounded_uniformly_continuous_image indefi-

```

```

nite_integral_uniformly_continuous [OF assms]
  then obtain B where B > 0 and B:  $\bigwedge x. x \in ?I \implies \text{norm } x \leq B$ 
    by (auto simp: bounded_pos)
  have  $\text{norm } x \leq B$  if  $x = \text{integral } (\text{cbox } c \ d) \ f$   $\text{cbox } c \ d \subseteq \text{cbox } a \ b$  for  $x \ c \ d$ 
  proof (cases  $\text{cbox } c \ d = \{\}$ )
    case True
      with  $\langle 0 < B \rangle$  that show ?thesis by auto
    next
      case False
        then have  $\exists x \in \text{cbox } (a,a) \ (b,b). \text{integral } (\text{cbox } c \ d) \ f = \text{integral } (\text{cbox } (\text{fst } x) \ (\text{snd } x)) \ f$ 
          using that by (metis cbox_Pair_iff_interval_subset_is_interval_is_interval_cbox_prod.sel)
        then show ?thesis
          using B that(1) by blast
    qed
  then show ?thesis
    by (blast intro: boundedI)
qed

```

An existence theorem for "improper" integrals. Hake's theorem implies that if the integrals over subintervals have a limit, the integral exists. We only need to assume that the integrals are bounded, and we get absolute integrability, but we also need a (rather weak) bound assumption on the function.

```

theorem absolutely_integrable_improper:
  fixes f :: 'M::euclidean_space  $\Rightarrow$  'N::euclidean_space
  assumes int_f:  $\bigwedge c \ d. \text{cbox } c \ d \subseteq \text{box } a \ b \implies f \text{ integrable\_on } \text{cbox } c \ d$ 
    and bo: bounded  $\{\text{integral } (\text{cbox } c \ d) \ f \mid c \ d. \text{cbox } c \ d \subseteq \text{box } a \ b\}$ 
    and absi:  $\bigwedge i. i \in \text{Basis} \implies \exists g. g \text{ absolutely\_integrable\_on } \text{cbox } a \ b \wedge$ 
       $((\forall x \in \text{cbox } a \ b. f \ x \cdot i \leq g \ x) \vee (\forall x \in \text{cbox } a \ b. f \ x \cdot i \geq g \ x))$ 
    shows  $f \text{ absolutely\_integrable\_on } \text{cbox } a \ b$ 
  proof (cases  $\text{content}(\text{cbox } a \ b) = 0$ )
    case True
      then show ?thesis
        by auto
    next
      case False
        then have pos:  $\text{content}(\text{cbox } a \ b) > 0$ 
          using zero_less_measure_iff by blast
        show ?thesis
          unfolding absolutely_integrable_componentwise_iff [where  $f = f$ ]
        proof
          fix j :: 'N
          assume j  $\in \text{Basis}$ 
          then obtain g where absint_g:  $g \text{ absolutely\_integrable\_on } \text{cbox } a \ b$ 
            and g:  $(\forall x \in \text{cbox } a \ b. f \ x \cdot j \leq g \ x) \vee (\forall x \in \text{cbox } a \ b. f \ x \cdot j \geq g \ x)$ 
          using absi by blast

```

```

have int_gab: g integrable_on cbox a b
  using absint_g set_lebesgue_integral_eq_integral(1) by blast
define α where α ≡ λk. a + (b - a) /R real k
define β where β ≡ λk. b - (b - a) /R real k
define I where I ≡ λk. cbox (α k) (β k)
have ISuc_box: I (Suc n) ⊆ box a b for n
  using pos_unfolding I_def
  by (intro subset_box_imp) (auto simp: α_def β_def content_pos_lt_eq
algebra_simps)
have ISucSuc: I (Suc n) ⊆ I (Suc (Suc n)) for n
proof -
  have ∧i. i ∈ Basis
    ⇒ a · i / Suc n + b · i / (real n + 2) ≤ b · i / Suc n + a · i /
(real n + 2)
  using pos
  by (simp add: content_pos_lt_eq divide_simps) (auto simp: algebra_simps)
  then show ?thesis
  unfolding I_def
  by (intro subset_box_imp) (auto simp: algebra_simps inverse_eq_divide
α_def β_def)
qed
have getN: ∃ N::nat. ∀ k. k ≥ N ⟶ x ∈ I k
  if x: x ∈ box a b for x
proof -
  define Δ where Δ ≡ (⋃ i ∈ Basis. {((x - a) · i) / ((b - a) · i), (b - x) ·
i / ((b - a) · i)})
  obtain N where N: real N > 1 / Inf Δ
  using reals_Archimedean2 by blast
  moreover have Δ: Inf Δ > 0
  using that by (auto simp: Δ_def finite_less_Inf_iff mem_box algebra_simps
divide_simps)
  ultimately have N > 0
  using of_nat_0_less_iff by fastforce
  show ?thesis
  proof (intro exI impI allI)
    fix k assume N ≤ k
    with ‹0 < N› have k > 0
    by linarith
    have xa_gt: (x - a) · i > ((b - a) · i) / (real k) if i ∈ Basis for i
    proof -
      have *: Inf Δ ≤ ((x - a) · i) / ((b - a) · i)
      unfolding Δ_def using that by (force intro: cInf_le_finite)
      have 1 / Inf Δ ≥ ((b - a) · i) / ((x - a) · i)
      using le_imp_inverse_le [OF * Δ]
      by (simp add: field_simps)
      with N have k > ((b - a) · i) / ((x - a) · i)
      using ‹N ≤ k› by linarith
      with x that show ?thesis
      by (auto simp: mem_box algebra_simps field_split_simps)
    qed
  end

```



```

qed
have bx_gt:  $(b - x) \cdot i > ((b - a) \cdot i) / k$  if  $i \in \text{Basis}$  for  $i$ 
proof -
  have *:  $\text{Inf } \Delta \leq ((b - x) \cdot i) / ((b - a) \cdot i)$ 
    using that unfolding  $\Delta\_def$  by (force intro: cInf_le_finite)
  have  $1 / \text{Inf } \Delta \geq ((b - a) \cdot i) / ((b - x) \cdot i)$ 
    using le_imp_inverse_le [OF *  $\Delta$ ]
    by (simp add: field_simps)
  with N have  $k > ((b - a) \cdot i) / ((b - x) \cdot i)$ 
    using  $\langle N \leq k \rangle$  by linarith
  with x that show ?thesis
    by (auto simp: mem_box algebra_simps field_split_simps)
qed
show  $x \in I$  k
  using that  $\Delta \langle k > 0 \rangle$  unfolding  $I\_def$ 
  by (auto simp:  $\alpha\_def \beta\_def$  mem_box algebra_simps divide_inverse dest:
xa_gt bx_gt)
qed
qed
obtain Bf where Bf:  $\bigwedge c d. \text{cbox } c d \subseteq \text{box } a b \implies \text{norm } (\text{integral } (\text{cbox } c d) f) \leq Bf$ 
  using bo unfolding bounded_iff by blast
obtain Bg where Bg:  $\bigwedge c d. \text{cbox } c d \subseteq \text{cbox } a b \implies |\text{integral } (\text{cbox } c d) g| \leq Bg$ 
  using bounded_integrals_over_subintervals [OF int_gab] unfolding bounded_iff
real_norm_def by blast
show  $(\lambda x. f x \cdot j)$  absolutely_integrable_on cbox a b
  using g
proof — A lot of duplication in the two proofs
  assume fg [rule_format]:  $\forall x \in \text{cbox } a b. f x \cdot j \leq g x$ 
  have  $(\lambda x. (f x \cdot j)) = (\lambda x. g x - (g x - (f x \cdot j)))$ 
    by simp
  moreover have  $(\lambda x. g x - (g x - (f x \cdot j)))$  integrable_on cbox a b
  proof (rule Henstock_Kurzweil_Integration.integrable_diff [OF int_gab])
    define  $\varphi$  where  $\varphi \equiv \lambda k x. \text{if } x \in I \text{ (Suc } k) \text{ then } g x - f x \cdot j \text{ else } 0$ 
    have  $(\lambda x. g x - f x \cdot j)$  integrable_on box a b
    proof (rule monotone_convergence_increasing [of  $\varphi$ , THEN conjunct1])
      have *:  $I \text{ (Suc } k) \cap \text{box } a b = I \text{ (Suc } k)$  for  $k$ 
        using box_subset_cbox ISuc_box by fastforce
      show  $\varphi k$  integrable_on box a b for  $k$ 
    proof -
      have  $I \text{ (Suc } k) \subseteq \text{cbox } a b$ 
        using * box_subset_cbox by blast
      moreover have  $(\lambda m. f m \cdot j)$  integrable_on  $I \text{ (Suc } k)$ 
        by (metis ISuc_box  $I\_def$  int_f integrable_component)
      ultimately have  $(\lambda m. g m - f m \cdot j)$  integrable_on  $I \text{ (Suc } k)$ 
        by (metis Henstock_Kurzweil_Integration.integrable_diff  $I\_def$  int_gab
integrable_on_subcbox)
      then show ?thesis

```

```

      by (simp add: *  $\varphi\_def$  integrable_restrict_Int)
    qed
  show  $\varphi\ k\ x \leq \varphi\ (Suc\ k)\ x$  if  $x \in box\ a\ b$  for  $k\ x$ 
    using ISucSuc box_subset_cbox that by (force simp:  $\varphi\_def$  intro!: fg)
  show  $(\lambda k. \varphi\ k\ x) \longrightarrow g\ x - f\ x \cdot j$  if  $x: x \in box\ a\ b$  for  $x$ 
  proof (rule tendsto_eventually)
    obtain  $N::nat$  where  $N: \bigwedge k. k \geq N \implies x \in I\ k$ 
    using getN [OF x] by blast
    show  $\forall_F k$  in sequentially.  $\varphi\ k\ x = g\ x - f\ x \cdot j$ 
  proof
    fix  $k::nat$  assume  $N \leq k$ 
    have  $x \in I\ (Suc\ k)$ 
    by (metis  $\langle N \leq k \rangle$  le_Suc_eq N)
    then show  $\varphi\ k\ x = g\ x - f\ x \cdot j$ 
    by (simp add:  $\varphi\_def$ )
  qed
qed
qed
have  $|integral\ (box\ a\ b)\ (\lambda x. \text{if } x \in I\ (Suc\ k) \text{ then } g\ x - f\ x \cdot j \text{ else } 0)| \leq$ 
Bg + Bf for  $k$ 
proof -
  have ABK_def [simp]:  $I\ (Suc\ k) \cap box\ a\ b = I\ (Suc\ k)$ 
  using ISuc_box by (simp add: Int_absorb2)
  have int_fI:  $f$  integrable_on  $I\ (Suc\ k)$ 
  using ISuc_box I_def int_f by auto
  moreover
  have  $|integral\ (I\ (Suc\ k))\ (\lambda x. f\ x \cdot j)| \leq norm\ (integral\ (I\ (Suc\ k))\ f)$ 
  by (simp add: Basis_le_norm int_fI  $\langle j \in Basis \rangle$ )
  with ISuc_box ABK_def have  $|integral\ (I\ (Suc\ k))\ (\lambda x. f\ x \cdot j)| \leq Bf$ 
  by (metis Bf I_def  $\langle j \in Basis \rangle$  int_fI integral_component_eq
norm_bound_Basis_le)
  ultimately
  have  $|integral\ (I\ (Suc\ k))\ g - integral\ (I\ (Suc\ k))\ (\lambda x. f\ x \cdot j)| \leq Bg$ 
+ Bf
    using * box_subset_cbox unfolding I_def
    by (blast intro: Bg add_mono order_trans [OF abs_triangle_ineq4])
  moreover have  $g$  integrable_on  $I\ (Suc\ k)$ 
  by (metis ISuc_box I_def int_gab integrable_on_open_interval
integrable_on_subcbox)
  moreover have  $(\lambda x. f\ x \cdot j)$  integrable_on  $I\ (Suc\ k)$ 
  using int_fI by (simp add: integrable_component)
  ultimately show ?thesis
  by (simp add: integral_restrict_Int integral_diff)
qed
then show bounded (range  $(\lambda k. integral\ (box\ a\ b)\ (\varphi\ k))$ )
  by (auto simp add: bounded_iff  $\varphi\_def$ )
qed
then show  $(\lambda x. g\ x - f\ x \cdot j)$  integrable_on cbox  $a\ b$ 
  by (simp add: integrable_on_open_interval)
qed

```

```

ultimately have  $(\lambda x. f x \cdot j)$  integrable_on cbox a b
  by auto
then show ?thesis
  using absolutely_integrable_component_ubound [OF _ absint_g] fg by force
next
assume gf [rule_format]:  $\forall x \in \text{cbox } a \ b. g x \leq f x \cdot j$ 
have  $(\lambda x. (f x \cdot j)) = (\lambda x. ((f x \cdot j) - g x) + g x)$ 
  by simp
moreover have  $(\lambda x. (f x \cdot j - g x) + g x)$  integrable_on cbox a b
proof (rule Henstock_Kurzweil_Integration.integrable_add [OF _ int_gab])
  let ? $\varphi = \lambda k x. \text{if } x \in I(\text{Suc } k) \text{ then } f x \cdot j - g x \text{ else } 0$ 
  have  $(\lambda x. f x \cdot j - g x)$  integrable_on box a b
  proof (rule monotone_convergence_increasing [of ? $\varphi$ , THEN conjunct1])
    have *:  $I(\text{Suc } k) \cap \text{box } a \ b = I(\text{Suc } k)$  for k
    using box_subset_cbox ISuc_box by fastforce
    show ? $\varphi$  k integrable_on box a b for k
  proof (simp add: integrable_restrict_Int integral_restrict_Int *)
    show  $(\lambda x. f x \cdot j - g x)$  integrable_on I (Suc k)
    by (metis ISuc_box Henstock_Kurzweil_Integration.integrable_diff I_def
int_f int_gab integrable_component integrable_on_open_interval integrable_on_subcbox)
  qed
  show ? $\varphi$  k  $x \leq ?\varphi(\text{Suc } k) x$  if  $x \in \text{box } a \ b$  for k x
    using ISucSuc box_subset_cbox that by (force simp: I_def intro!: gf)
  show  $(\lambda k. ?\varphi k x) \longrightarrow f x \cdot j - g x$  if  $x: x \in \text{box } a \ b$  for x
  proof (rule tendsto_eventually)
    obtain N::nat where N:  $\bigwedge k. k \geq N \implies x \in I k$ 
    using getN [OF x] by blast
    then show  $\forall_F k$  in sequentially.  $?\varphi k x = f x \cdot j - g x$ 
    by (metis (no_types, lifting) eventually_at_top_linorder I le_Suc_eq)
  qed
  have |integral (box a b)
    ( $\lambda x. \text{if } x \in I(\text{Suc } k) \text{ then } f x \cdot j - g x \text{ else } 0$ )|  $\leq Bf + Bg$  for k
  proof -
    define ABK where  $ABK \equiv \text{cbox } (a + (b - a) /_R (1 + \text{real } k)) (b -$ 
 $(b - a) /_R (1 + \text{real } k))$ 
    have ABK_eq [simp]:  $ABK \cap \text{box } a \ b = ABK$ 
    using * I_def  $\alpha$ _def  $\beta$ _def ABK_def by auto
    have int_fI:  $f$  integrable_on ABK
    unfolding ABK_def
    using ISuc_box I_def  $\alpha$ _def  $\beta$ _def int_f by force
    then have  $(\lambda x. f x \cdot j)$  integrable_on ABK
    by (simp add: integrable_component)
    moreover have  $g$  integrable_on ABK
    by (metis ABK_def ABK_eq IntE box_subset_cbox int_gab inte-
grable_on_subcbox subset_eq)
    moreover
    have |integral ABK  $(\lambda x. f x \cdot j)$ |  $\leq \text{norm } (\text{integral ABK } f)$ 
    by (simp add: Basis_le_norm int_fI  $\langle j \in \text{Basis} \rangle$ )
    then have |integral ABK  $(\lambda x. f x \cdot j)$ |  $\leq Bf$ 

```

```

      by (metis ABK_eq ABK_def Bf IntE dual_order.trans subset_eq)
    ultimately show ?thesis
      using * box_subset_cbox
      apply (simp add: integral_restrict_Int integral_diff ABK_def I_def
 $\alpha\_def$   $\beta\_def$ )
      by (blast intro: Bg add_mono order_trans [OF abs_triangle_ineq4])
    qed
    then show bounded (range ( $\lambda k. \text{integral } (\text{box } a \ b) \ (\varphi \ k)$ ))
      by (auto simp add: bounded_iff)
    qed
    then show ( $\lambda x. f \ x \cdot j - g \ x$ ) integrable_on cbox a b
      by (simp add: integrable_on_open_interval)
    qed
    ultimately have ( $\lambda x. f \ x \cdot j$ ) integrable_on cbox a b
      by auto
    then show ?thesis
      using absint_g absolutely_integrable_absolutely_integrable_lbound gf by
blast
    qed
  qed
qed

```

6.27.4 Second mean value theorem and corollaries

```

lemma level_approx:
  fixes f :: real  $\Rightarrow$  real and n::nat
  assumes f:  $\bigwedge x. x \in S \implies 0 \leq f \ x \wedge f \ x \leq 1$  and  $x \in S \ n \neq 0$ 
  shows  $|f \ x - (\sum k = \text{Suc } 0..n. \text{if } k / n \leq f \ x \text{ then inverse } n \text{ else } 0)| < \text{inverse } n$ 
    (is ?lhs < _)
proof -
  have  $n * f \ x \geq 0$ 
    using assms by auto
  then obtain m::nat where m:  $\text{floor}(n * f \ x) = \text{int } m$ 
    using nonneg_int_cases zero_le_floor by blast
  then have kn:  $\text{real } k / \text{real } n \leq f \ x \iff k \leq m$  for k
    using  $\langle n \neq 0 \rangle$  by (simp add: field_split_simps) linarith
  then have  $\text{Suc } n / \text{real } n \leq f \ x \iff \text{Suc } n \leq m$ 
    by blast
  have  $\text{real } n * f \ x \leq \text{real } n$ 
    by (simp add:  $\langle x \in S \rangle f \text{ mult\_left\_le}$ )
  then have  $m \leq n$ 
    using m by linarith
  have ?lhs =  $|f \ x - (\sum k \in \{\text{Suc } 0..n\} \cap \{..m\}. \text{inverse } n)|$ 
    by (subst sum.inter_restrict) (auto simp: kn)
  also have  $\dots < \text{inverse } n$ 
    using  $\langle m \leq n \rangle \langle n \neq 0 \rangle m$ 
    by (simp add: min_absorb2 field_split_simps) linarith
  finally show ?thesis .
qed

```

```

lemma SMVT_lemma2:
  fixes f :: real  $\Rightarrow$  real
  assumes f: f integrable_on {a..b}
  and g:  $\bigwedge x y. x \leq y \implies g x \leq g y$ 
  shows  $(\bigcup y::real. \{\lambda x. \text{if } g x \geq y \text{ then } f x \text{ else } 0\})$  equiintegrable_on {a..b}
proof -
  have ffab: {f} equiintegrable_on {a..b}
  by (metis equiintegrable_on_sing f interval_cbox)
  then have ff: {f} equiintegrable_on (cbox a b)
  by simp
  have ge:  $(\bigcup c. \{\lambda x. \text{if } x \geq c \text{ then } f x \text{ else } 0\})$  equiintegrable_on {a..b}
  using equiintegrable_halfspace_restrictions_ge [OF ff] by auto
  have gt:  $(\bigcup c. \{\lambda x. \text{if } x > c \text{ then } f x \text{ else } 0\})$  equiintegrable_on {a..b}
  using equiintegrable_halfspace_restrictions_gt [OF ff] by auto
  have 0:  $\{(\lambda x. 0)\}$  equiintegrable_on {a..b}
  by (metis box_real(2) equiintegrable_on_sing integrable_0)
  have †:  $(\lambda x. \text{if } g x \geq y \text{ then } f x \text{ else } 0) \in \{(\lambda x. 0), f\} \cup (\bigcup z. \{\lambda x. \text{if } z < x \text{ then } f x \text{ else } 0\}) \cup (\bigcup z. \{\lambda x. \text{if } z \leq x \text{ then } f x \text{ else } 0\})$ 
  for y
proof (cases  $(\forall x. g x \geq y) \vee (\forall x. \neg (g x \geq y))$ )
  let ? $\mu$  = Inf {x. g x  $\geq$  y}
  case False
  have lower: ? $\mu \leq x$  if g x  $\geq$  y for x
  proof (rule cInf_lower)
    show x  $\in$  {x. y  $\leq$  g x}
    using False by (auto simp: that)
    show bdd_below {x. y  $\leq$  g x}
    by (metis False bdd_belowI dual_order.trans g linear mem_Collect_eq)
  qed
  have greatest: ? $\mu \geq z$  if  $(\bigwedge x. g x \geq y \implies z \leq x)$  for z
  by (metis False cInf_greatest empty_iff mem_Collect_eq that)
  show ?thesis
proof (cases g ? $\mu \geq y$ )
  case True
  then obtain  $\zeta$  where  $\zeta: \bigwedge x. g x \geq y \longleftrightarrow x \geq \zeta$ 
  by (metis g lower order.trans) — in fact y is Inf {x. y  $\leq$  g x}
  then show ?thesis
  by (force simp:  $\zeta$ )
next
  case False
  have  $(y \leq g x) \longleftrightarrow (? \mu < x)$  for x
  proof
    show ? $\mu < x$  if y  $\leq$  g x
    using that False less_eq_real_def lower by blast
    show y  $\leq$  g x if ? $\mu < x$ 
    by (metis g greatest le_less_trans that less_le_trans linear not_less)
  qed
qed

```

```

    then obtain  $\zeta$  where  $\zeta: \bigwedge x. g\ x \geq y \longleftrightarrow x > \zeta$  ..
    then show ?thesis
    by (force simp:  $\zeta$ )
  qed
qed auto
show ?thesis
  using † by (simp add: UN_subset_iff equiintegrable_on_subset [OF equiintegrable_on_Un [OF gt equiintegrable_on_Un [OF ge equiintegrable_on_Un [OF ffab 0]]]])
qed

lemma SMVT_lemma4:
  fixes  $f :: \text{real} \Rightarrow \text{real}$ 
  assumes  $f: f \text{ integrable\_on } \{a..b\}$ 
    and  $a \leq b$ 
    and  $g: \bigwedge x\ y. x \leq y \implies g\ x \leq g\ y$ 
    and  $01: \bigwedge x. \llbracket a \leq x; x \leq b \rrbracket \implies 0 \leq g\ x \wedge g\ x \leq 1$ 
  obtains  $c$  where  $a \leq c \leq b$   $((\lambda x. g\ x *_{\mathbb{R}} f\ x) \text{ has\_integral } \text{integral } \{c..b\}\ f)$ 
  { $a..b$ }
proof -
  have connected  $((\lambda x. \text{integral } \{x..b\}\ f) ' \{a..b\})$ 
  by (simp add: f_indefinite_integral_continuous_1' connected_continuous_image)
  moreover have compact  $((\lambda x. \text{integral } \{x..b\}\ f) ' \{a..b\})$ 
  by (simp add: compact_continuous_image f_indefinite_integral_continuous_1')
  ultimately obtain  $m\ M$  where  $\text{int\_fab}: (\lambda x. \text{integral } \{x..b\}\ f) ' \{a..b\} = \{m..M\}$ 
  using connected_compact_interval_1 by meson
  have  $\exists c. c \in \{a..b\} \wedge$ 
     $\text{integral } \{c..b\}\ f =$ 
     $\text{integral } \{a..b\}\ (\lambda x. (\sum k = 1..n. \text{if } g\ x \geq \text{real } k / \text{real } n \text{ then inverse } n$ 
 $*_{\mathbb{R}} f\ x \text{ else } 0))$  for  $n$ 
  proof (cases  $n=0$ )
    case True
    then show ?thesis
    using  $\langle a \leq b \rangle$  by auto
  next
    case False
    have  $(\bigcup c::\text{real}. \{\lambda x. \text{if } g\ x \geq c \text{ then } f\ x \text{ else } 0\}) \text{ equiintegrable\_on } \{a..b\}$ 
    using SMVT_lemma2 [OF  $f\ g$ ] .
    then have  $\text{int}: (\lambda x. \text{if } g\ x \geq c \text{ then } f\ x \text{ else } 0) \text{ integrable\_on } \{a..b\}$  for  $c$ 
    by (simp add: equiintegrable_on_def)
    have  $\text{int}': (\lambda x. \text{if } g\ x \geq c \text{ then } u * f\ x \text{ else } 0) \text{ integrable\_on } \{a..b\}$  for  $c\ u$ 
    proof -
      have  $(\lambda x. \text{if } g\ x \geq c \text{ then } u * f\ x \text{ else } 0) = (\lambda x. u * (\text{if } g\ x \geq c \text{ then } f\ x \text{ else } 0))$ 
      by (force simp: if_distrib)
    then show ?thesis
    using integrable_on_cmult_left [OF  $\text{int}$ ] by simp

```

```

qed
have  $\exists d. d \in \{a..b\} \wedge \text{integral } \{a..b\} (\lambda x. \text{if } g\ x \geq y \text{ then } f\ x \text{ else } 0) = \text{integral } \{d..b\} f$  for  $y$ 
proof -
  let  $?X = \{x. g\ x \geq y\}$ 
  have *:  $\exists a. ?X = \{a.. \} \vee ?X = \{a < ..\}$ 
  if 1:  $?X \neq \{\}$  and 2:  $?X \neq UNIV$ 
  proof -
    let  $? \mu = \text{Inf } \{x. g\ x \geq y\}$ 
    have lower:  $? \mu \leq x$  if  $g\ x \geq y$  for  $x$ 
    proof (rule cInf_lower)
      show  $x \in \{x. y \leq g\ x\}$ 
      using 1 2 by (auto simp: that)
      show bdd_below  $\{x. y \leq g\ x\}$ 
      unfolding bdd_below_def
      by (metis 2 UNIV_eq_I dual_order.trans g_less_eq_real_def mem_Collect_eq not_le)
    qed
    have greatest:  $? \mu \geq z$  if  $\bigwedge x. g\ x \geq y \implies z \leq x$  for  $z$ 
    by (metis cInf_greatest mem_Collect_eq that 1)
    show ?thesis
    proof (cases  $g\ ? \mu \geq y$ )
      case True
      then obtain  $\zeta$  where  $\zeta: \bigwedge x. g\ x \geq y \longleftrightarrow x \geq \zeta$ 
      by (metis g_lower_order.trans) — in fact  $y$  is  $\text{Inf } \{x. y \leq g\ x\}$ 
      then show ?thesis
      by (force simp:  $\zeta$ )
    next
      case False
      have  $(y \leq g\ x) = (? \mu < x)$  for  $x$ 
      proof
        show  $? \mu < x$  if  $y \leq g\ x$ 
        using that False less_eq_real_def lower by blast
        show  $y \leq g\ x$  if  $? \mu < x$ 
        by (metis g_greatest le_less_trans that less_le_trans linear not_less)
      qed
      then obtain  $\zeta$  where  $\zeta: \bigwedge x. g\ x \geq y \longleftrightarrow x > \zeta$  ..
      then show ?thesis
      by (force simp:  $\zeta$ )
    qed
  qed
qed
then consider  $?X = \{\} \mid ?X = UNIV \mid (\text{intv } d) \text{ where } ?X = \{d.. \} \vee ?X = \{d < ..\}$ 
by metis
then have  $\exists d. d \in \{a..b\} \wedge \text{integral } \{a..b\} (\lambda x. \text{if } x \in ?X \text{ then } f\ x \text{ else } 0) = \text{integral } \{d..b\} f$ 
proof cases
  case (intv d)
  show ?thesis

```

```

proof (cases  $d < a$ )
  case True
    with intv have  $\text{integral } \{a..b\} (\lambda x. \text{if } y \leq g \ x \text{ then } f \ x \text{ else } 0) = \text{integral}$ 
 $\{a..b\} \ f$ 
      by (intro Henstock_Kurzweil_Integration.integral_cong) force
      then show ?thesis
        by (rule_tac  $x=a$  in exI) (simp add:  $\langle a \leq b \rangle$ )
    next
      case False
      show ?thesis
      proof (cases  $b < d$ )
        case True
          have  $\text{integral } \{a..b\} (\lambda x. \text{if } x \in \{x. y \leq g \ x\} \text{ then } f \ x \text{ else } 0) = \text{integral}$ 
 $\{a..b\} (\lambda x. 0)$ 
            by (rule Henstock_Kurzweil_Integration.integral_cong) (use intv True
in fastforce)
            then show ?thesis
              using  $\langle a \leq b \rangle$  by auto
          next
            case False
            with  $\langle \neg d < a \rangle$  have  $\text{eq: } \{d..\} \cap \{a..b\} = \{d..b\} \ \{d<..\} \cap \{a..b\} =$ 
 $\{d<..b\}$ 
              by force+
              moreover have  $\text{integral } \{d<..b\} \ f = \text{integral } \{d..b\} \ f$ 
                by (rule integral_spike_set [OF empty_imp_negligible negligible_subset
OF negligible_sing [of d]]) auto
                ultimately
                have  $\text{integral } \{a..b\} (\lambda x. \text{if } x \in \{x. y \leq g \ x\} \text{ then } f \ x \text{ else } 0) = \text{integral}$ 
 $\{d..b\} \ f$ 
                  unfolding integral_restrict_Int using intv by presburger
                  moreover have  $d \in \{a..b\}$ 
                    using  $\langle \neg d < a \rangle \ \langle a \leq b \rangle$  False by auto
                    ultimately show ?thesis
                      by auto
                  qed
                qed
              qed (use  $\langle a \leq b \rangle$  in auto)
            then show ?thesis
              by auto
          qed
        then have  $\forall k. \exists d. d \in \{a..b\} \wedge \text{integral } \{a..b\} (\lambda x. \text{if } \text{real } k / \text{real } n \leq g \ x$ 
 $\text{then } f \ x \text{ else } 0) = \text{integral } \{d..b\} \ f$ 
          by meson
          then obtain d where  $dab: \bigwedge k. d \ k \in \{a..b\}$ 
            and  $deg: \bigwedge k::\text{nat}. \text{integral } \{a..b\} (\lambda x. \text{if } k/n \leq g \ x \text{ then } f \ x \text{ else } 0) = \text{integral}$ 
 $\{d \ k..b\} \ f$ 
              by metis
              have  $(\sum k = 1..n. \text{integral } \{a..b\} (\lambda x. \text{if } \text{real } k / \text{real } n \leq g \ x \text{ then } f \ x \text{ else } 0))$ 
 $/_{\mathbb{R}} n \in \{m..M\}$ 

```



```

    unfolding scaleR_right.sum
  proof (intro conjI allI impI convex [THEN iffD1, rule_format])
    show integral {a..b} (λx a. if real k / real n ≤ g x then f x else 0) ∈ {m..M}
  for k
    by (metis (no_types, lifting) deq image_eqI int_fab dab)
  qed (use ⟨n ≠ 0⟩ in auto)
  then have ∃ c. c ∈ {a..b} ∧
    integral {c..b} f = inverse n *R (∑ k = 1..n. integral {a..b} (λx. if g
x ≥ real k / real n then f x else 0))
    by (metis (no_types, lifting) int_fab imageE)
  then show ?thesis
    by (simp add: sum_distrib_left if_distrib integral_sum int' flip: integral_mult_right
cong: if_cong)
  qed
  then obtain c where cab: ∧ n. c n ∈ {a..b}
    and c: ∧ n. integral {c n..b} f = integral {a..b} (λx. (∑ k = 1..n. if g x ≥ real
k / real n then f x /R n else 0))
    by metis
  obtain d and σ :: nat ⇒ nat
    where d ∈ {a..b} and σ: strict_mono σ and d: (c ∘ σ) ⟶ d and non0:
∧ n. σ n ≥ Suc 0
  proof -
    have compact {a..b}
      by auto
    with cab obtain d and s0
      where d ∈ {a..b} and s0: strict_mono s0 and tends: (c ∘ s0) ⟶ d
    unfolding compact_def
    using that by blast
  show thesis
  proof
    show d ∈ {a..b}
      by fact
    show strict_mono (s0 ∘ Suc)
      using s0 by (auto simp: strict_mono_def)
    show (c ∘ (s0 ∘ Suc)) ⟶ d
      by (metis tends LIMSEQ_subseq_LIMSEQ Suc_less_eq comp_assoc
strict_mono_def)
    show ∧ n. (s0 ∘ Suc) n ≥ Suc 0
      by (metis comp_apply le0 not_less_eq_eq old.nat.exhaust s0 seq_suble)
  qed
  qed
  define φ where φ ≡ λ n x. ∑ k = Suc 0..σ n. if k / (σ n) ≤ g x then f x /R (σ
n) else 0
  define ψ where ψ ≡ λ n x. ∑ k = Suc 0..σ n. if k / (σ n) ≤ g x then inverse (σ
n) else 0
  have **: (λ x. g x *R f x) integrable_on cbox a b ∧
    (λ n. integral (cbox a b) (φ n)) ⟶ integral (cbox a b) (λ x. g x *R f x)
  proof (rule equiintegrable_limit)
    have †: ((λ n. λ x. (∑ k = Suc 0..n. if k / n ≤ g x then inverse n *R f x else

```

```

0)) ' {Suc 0..} } equiintegrable_on {a..b}
proof -
  have *: ( $\bigcup c :: \text{real}. \{ \lambda x. \text{if } g \ x \geq c \text{ then } f \ x \text{ else } 0 \}$ ) equiintegrable_on {a..b}
    using SMVT_lemma2 [OF f g] .
  show ?thesis
    apply (rule equiintegrable_on_subset [OF equiintegrable_sum_real [OF *]],
clarify)
    apply (rule_tac a={Suc 0..n} in UN_I, force)
    apply (rule_tac a= $\lambda k. \text{inverse } n$  in UN_I, auto)
    apply (rule_tac x= $\lambda k \ x. \text{if } \text{real } k / \text{real } n \leq g \ x \text{ then } f \ x \text{ else } 0$  in bexI)
    apply (force intro: sum.cong)+
  done
qed
show range  $\varphi$  equiintegrable_on cbox a b
  unfolding  $\varphi\_def$ 
  by (auto simp: non0 intro: equiintegrable_on_subset [OF †])
show ( $\lambda n. \varphi \ n \ x$ )  $\longrightarrow$   $g \ x *_R f \ x$ 
  if x:  $x \in \text{cbox } a \ b$  for x
proof -
  have eq:  $\varphi \ n \ x = \psi \ n \ x *_R f \ x$  for n
    by (auto simp:  $\varphi\_def \ \psi\_def \ \text{sum\_distrib\_right} \ \text{if\_distrib} \ \text{intro: sum.cong}$ )
  show ?thesis
    unfolding eq
  proof (rule tendsto_scaleR [OF _ tendsto_const])
    show ( $\lambda n. \psi \ n \ x$ )  $\longrightarrow$   $g \ x$ 
      unfolding lim_sequentially dist_real_def
    proof (intro allI impI)
      fix e :: real
      assume e > 0
      then obtain N where  $N \neq 0 \ 0 < \text{inverse } (\text{real } N)$  and  $N: \text{inverse } (\text{real } N) < e$ 
        using real_arch_inverse by metis
      moreover have  $|\psi \ n \ x - g \ x| < \text{inverse } (\text{real } N)$  if  $n \geq N$  for n
        proof -
          have  $|g \ x - \psi \ n \ x| < \text{inverse } (\text{real } (\sigma \ n))$ 
            unfolding  $\psi\_def$ 
          proof (rule level_approx [of {a..b} g])
            show  $\sigma \ n \neq 0$ 
              by (metis Suc_n_not_le_n non0)
          qed
          qed (use x 01 non0 in auto)
          also have  $\dots \leq \text{inverse } N$ 
            using seq_suble [OF  $\sigma$ ]  $\langle N \neq 0 \rangle$  non0 that by (auto intro: order_trans
simp: field_split_simps)
          finally show ?thesis
            by linarith
        qed
      qed
      ultimately show  $\exists N. \forall n \geq N. |\psi \ n \ x - g \ x| < e$ 
        using less_trans by blast
    qed
  qed

```

```

      qed
    qed
  qed
  show thesis
  proof
    show  $a \leq d \wedge d \leq b$ 
      using  $\langle d \in \{a..b\} \rangle$  atLeastAtMost_iff by blast+
    show  $((\lambda x. g \ x *_{\mathbb{R}} f \ x) \text{ has\_integral } \text{integral } \{d..b\} \ f) \ \{a..b\}$ 
      unfolding has_integral_iff
    proof
      show  $(\lambda x. g \ x *_{\mathbb{R}} f \ x) \text{ integrable\_on } \{a..b\}$ 
        using ** by simp
      show  $\text{integral } \{a..b\} \ (\lambda x. g \ x *_{\mathbb{R}} f \ x) = \text{integral } \{d..b\} \ f$ 
        proof (rule tendsto_unique)
          show  $(\lambda n. \text{integral } \{c(\sigma \ n)..b\} \ f) \longrightarrow \text{integral } \{a..b\} \ (\lambda x. g \ x *_{\mathbb{R}} f \ x)$ 
            using ** by (simp add: c  $\varphi$ _def)
          have continuous  $(\text{at } d \text{ within } \{a..b\}) \ (\lambda x. \text{integral } \{x..b\} \ f)$ 
            using indefinite_integral_continuous_1' [OF f]  $\langle d \in \{a..b\} \rangle$ 
            by (simp add: continuous_on_eq_continuous_within)
          then show  $(\lambda n. \text{integral } \{c(\sigma \ n)..b\} \ f) \longrightarrow \text{integral } \{d..b\} \ f$ 
            using d cab unfolding o_def
            by (simp add: continuous_within_sequentially o_def)
        qed auto
      qed
    qed
  qed
  qed

theorem second_mean_value_theorem_full:
  fixes f :: real  $\Rightarrow$  real
  assumes f: f integrable_on {a..b} and a  $\leq$  b
  and g:  $\bigwedge x \ y. \llbracket a \leq x; x \leq y; y \leq b \rrbracket \Longrightarrow g \ x \leq g \ y$ 
  obtains c where  $c \in \{a..b\}$ 
  and  $((\lambda x. g \ x * f \ x) \text{ has\_integral } (g \ a * \text{integral } \{a..c\} \ f + g \ b * \text{integral } \{c..b\} \ f)) \ \{a..b\}$ 
proof -
  have gab:  $g \ a \leq g \ b$ 
    using  $\langle a \leq b \rangle$  g by blast
  then consider  $g \ a < g \ b \mid g \ a = g \ b$ 
    by linarith
  then show thesis
  proof cases
    case 1
      define h where  $h \equiv \lambda x. \text{if } x < a \text{ then } 0 \text{ else if } b < x \text{ then } 1$ 
        else  $(g \ x - g \ a) / (g \ b - g \ a)$ 
      obtain c where  $a \leq c \wedge c \leq b$  and c:  $((\lambda x. h \ x *_{\mathbb{R}} f \ x) \text{ has\_integral } \text{integral } \{c..b\} \ f) \ \{a..b\}$ 
      proof (rule SMVT_lemma4 [OF f  $\langle a \leq b \rangle$ , of h])
        show  $h \ x \leq h \ y \wedge 0 \leq h \ x \wedge h \ x \leq 1$  if  $x \leq y$  for  $x \ y$ 

```

```

      using that gab by (auto simp: divide_simps g h_def)
    qed
  show ?thesis
  proof
    show  $c \in \{a..b\}$ 
    using  $\langle a \leq c \rangle \langle c \leq b \rangle$  by auto
    have I:  $((\lambda x. g x * f x - g a * f x) \text{ has\_integral } (g b - g a) * \text{integral } \{c..b\} f) \{a..b\}$ 
    proof (subst has_integral_cong)
      show  $g x * f x - g a * f x = (g b - g a) * h x *_R f x$ 
      if  $x \in \{a..b\}$  for x
      using 1 that by (simp add: h_def field_split_simps)
      show  $((\lambda x. (g b - g a) * h x *_R f x) \text{ has\_integral } (g b - g a) * \text{integral } \{c..b\} f) \{a..b\}$ 
      using has_integral_mult_right [OF c, of  $g b - g a$ ] .
    qed
    have II:  $((\lambda x. g a * f x) \text{ has\_integral } g a * \text{integral } \{a..b\} f) \{a..b\}$ 
    using has_integral_mult_right [where  $c = g a$ , OF integrable_integral [OF f]] .
    have  $((\lambda x. g x * f x) \text{ has\_integral } (g b - g a) * \text{integral } \{c..b\} f + g a * \text{integral } \{a..b\} f) \{a..b\}$ 
    using has_integral_add [OF I II] by simp
    then show  $((\lambda x. g x * f x) \text{ has\_integral } g a * \text{integral } \{a..c\} f + g b * \text{integral } \{c..b\} f) \{a..b\}$ 
    by (simp add: algebra_simps flip: integral_combine [OF  $\langle a \leq c \rangle \langle c \leq b \rangle$  f])
  qed
next
case 2
show ?thesis
proof
  show  $a \in \{a..b\}$ 
  by (simp add:  $\langle a \leq b \rangle$ )
  have  $((\lambda x. g x * f x) \text{ has\_integral } g a * \text{integral } \{a..b\} f) \{a..b\}$ 
  proof (rule has_integral_eq)
    show  $((\lambda x. g a * f x) \text{ has\_integral } g a * \text{integral } \{a..b\} f) \{a..b\}$ 
    using f has_integral_mult_right by blast
    show  $g a * f x = g x * f x$ 
    if  $x \in \{a..b\}$  for x
    by (metis atLeastAtMost_iff g less_eq_real_def not_le that 2)
  qed
  then show  $((\lambda x. g x * f x) \text{ has\_integral } g a * \text{integral } \{a..a\} f + g b * \text{integral } \{a..b\} f) \{a..b\}$ 
  by (simp add: 2)
qed
qed
qed

```

corollary second_mean_value_theorem:

```

fixes  $f :: \text{real} \Rightarrow \text{real}$ 
assumes  $f: f \text{ integrable\_on } \{a..b\} \text{ and } a \leq b$ 
and  $g: \bigwedge x y. \llbracket a \leq x; x \leq y; y \leq b \rrbracket \Longrightarrow g\ x \leq g\ y$ 
obtains  $c$  where  $c \in \{a..b\}$ 
 $\text{integral } \{a..b\} (\lambda x. g\ x * f\ x) = g\ a * \text{integral } \{a..c\} f + g\ b * \text{integral } \{c..b\} f$ 
using second_mean_value_theorem_full [where  $g=g$ , OF assms]
by (metis (full_types) integral_unique)

end

```

6.28 Continuous Extensions of Functions

```

theory Continuous_Extension
imports Starlike
begin

```

6.28.1 Partitions of unity subordinate to locally finite open coverings

A difference from HOL Light: all summations over infinite sets equal zero, so the "support" must be made explicit in the summation below!

```

proposition subordinate_partition_of_unity:
fixes  $S :: 'a::\text{metric\_space set}$ 
assumes  $S \subseteq \bigcup \mathcal{C}$  and  $\text{opC}: \bigwedge T. T \in \mathcal{C} \Longrightarrow \text{open } T$ 
and  $\text{fin}: \bigwedge x. x \in S \Longrightarrow \exists V. \text{open } V \wedge x \in V \wedge \text{finite } \{U \in \mathcal{C}. U \cap V \neq \{\}\}$ 
obtains  $F :: ['a \text{ set}, 'a] \Rightarrow \text{real}$ 
where  $\bigwedge U. U \in \mathcal{C} \Longrightarrow \text{continuous\_on } S (F\ U) \wedge (\forall x \in S. 0 \leq F\ U\ x)$ 
and  $\bigwedge x U. \llbracket U \in \mathcal{C}; x \in S; x \notin U \rrbracket \Longrightarrow F\ U\ x = 0$ 
and  $\bigwedge x. x \in S \Longrightarrow \text{supp\_sum } (\lambda W. F\ W\ x)\ \mathcal{C} = 1$ 
and  $\bigwedge x. x \in S \Longrightarrow \exists V. \text{open } V \wedge x \in V \wedge \text{finite } \{U \in \mathcal{C}. \exists x \in V. F\ U\ x \neq 0\}$ 
proof (cases  $\exists W. W \in \mathcal{C} \wedge S \subseteq W$ )
case True
then obtain  $W$  where  $W \in \mathcal{C} \ S \subseteq W$  by metis
then show ?thesis
by (rule_tac  $F = \lambda V x. \text{if } V = W \text{ then } 1 \text{ else } 0$  in that) (auto simp: supp_sum_def support_on_def)
next
case False
have nonneg:  $0 \leq \text{supp\_sum } (\lambda V. \text{setdist } \{x\} (S - V))\ \mathcal{C}$  for  $x$ 
by (simp add: supp_sum_def sum_nonneg)
have sd_pos:  $0 < \text{setdist } \{x\} (S - V)$  if  $V \in \mathcal{C} \ x \in S \ x \in V$  for  $V$ 
proof -
have closedin (top_of_set  $S$ )  $(S - V)$ 
by (simp add: Diff_Diff_Int closedin_def opC openin_open_Int  $\langle V \in \mathcal{C} \rangle$ )
with that False setdist_pos_le [of  $\{x\}$   $S - V$ ]
show ?thesis

```

```

    using setdist_gt_0_closedin by fastforce
qed
have ss_pos:  $0 < \text{supp\_sum } (\lambda V. \text{setdist } \{x\} (S - V)) \mathcal{C}$  if  $x \in S$  for  $x$ 
proof -
  obtain  $U$  where  $U \in \mathcal{C}$   $x \in U$  using  $\langle x \in S \rangle \langle S \subseteq \bigcup \mathcal{C} \rangle$ 
  by blast
  obtain  $V$  where open  $V$   $x \in V$  finite  $\{U \in \mathcal{C}. U \cap V \neq \{\}\}$ 
  using  $\langle x \in S \rangle$  fin by blast
  then have *: finite  $\{A \in \mathcal{C}. \neg S \subseteq A \wedge x \notin \text{closure } (S - A)\}$ 
  using closure_def that by (blast intro: rev_finite_subset)
  have  $x \notin \text{closure } (S - U)$ 
  using  $\langle U \in \mathcal{C} \rangle \langle x \in U \rangle$  opC open_Int_closure_eq_empty by fastforce
  then show ?thesis
  apply (simp add: setdist_eq_0_sing_1 supp_sum_def support_on_def)
  apply (rule ordered_comm_monoid_add_class.sum_pos2 [OF *, of U])
  using  $\langle U \in \mathcal{C} \rangle \langle x \in U \rangle$  False
  apply (auto simp: sd_pos that)
  done
qed
define  $F$  where
 $F \equiv \lambda W x. \text{if } x \in S \text{ then } \text{setdist } \{x\} (S - W) / \text{supp\_sum } (\lambda V. \text{setdist } \{x\} (S - V)) \mathcal{C} \text{ else } 0$ 
show ?thesis
proof (rule_tac  $F = F$  in that)
  have continuous_on  $S$  ( $F$   $U$ ) if  $U \in \mathcal{C}$  for  $U$ 
  proof -
    have *: continuous_on  $S$  ( $\lambda x. \text{supp\_sum } (\lambda V. \text{setdist } \{x\} (S - V)) \mathcal{C}$ )
    proof (clarsimp simp add: continuous_on_eq_continuous_within)
      fix  $x$  assume  $x \in S$ 
      then obtain  $X$  where open  $X$  and  $x: x \in S \cap X$  and finX: finite  $\{U \in \mathcal{C}. U \cap X \neq \{\}\}$ 
      using assms by blast
      then have OSX: openin (top_of_set  $S$ ) ( $S \cap X$ ) by blast
      have sumeq:  $\bigwedge x. x \in S \cap X \implies$ 
         $(\sum V \mid V \in \mathcal{C} \wedge V \cap X \neq \{\}. \text{setdist } \{x\} (S - V))$ 
         $= \text{supp\_sum } (\lambda V. \text{setdist } \{x\} (S - V)) \mathcal{C}$ 
      apply (simp add: supp_sum_def)
      apply (rule sum.mono_neutral_right [OF finX])
      apply (auto simp: setdist_eq_0_sing_1 support_on_def subset_iff)
      apply (meson DiffI closure_subset disjoint_iff_not_equal subsetCE)
      done
      show continuous (at  $x$  within  $S$ ) ( $\lambda x. \text{supp\_sum } (\lambda V. \text{setdist } \{x\} (S - V)) \mathcal{C}$ )
      apply (rule continuous_transform_within_openin
        [where  $f = \lambda x. (\text{sum } (\lambda V. \text{setdist } \{x\} (S - V))) \{V \in \mathcal{C}. V \cap X \neq \{\}\}$ ])
        and  $S = S \cap X$ )
      apply (rule continuous_intros continuous_at_setdist continuous_at_imp_continuous_at_within
        OSX  $x$ ) +

```

```

      apply (simp add: sumeq)
    done
  qed
  show ?thesis
    apply (simp add: F_def)
    apply (rule continuous_intros *)+
    using ss_pos apply force
  done
  qed
  moreover have  $\llbracket U \in \mathcal{C}; x \in S \rrbracket \implies 0 \leq F \ U \ x$  for  $U \ x$ 
    using nonneg [of x] by (simp add: F_def field_split_simps)
  ultimately show  $\bigwedge U. U \in \mathcal{C} \implies \text{continuous\_on } S \ (F \ U) \wedge (\forall x \in S. 0 \leq F$ 
 $U \ x)$ 
    by metis
  next
  show  $\bigwedge x \ U. \llbracket U \in \mathcal{C}; x \in S; x \notin U \rrbracket \implies F \ U \ x = 0$ 
    by (simp add: setdist_eq_0_sing_1 closure_def F_def)
  next
  show  $\text{supp\_sum } (\lambda W. F \ W \ x) \ \mathcal{C} = 1$  if  $x \in S$  for  $x$ 
    using that ss_pos [OF that]
    by (simp add: F_def field_split_simps supp_sum_divide_distrib [symmetric])
  next
  show  $\exists V. \text{open } V \wedge x \in V \wedge \text{finite } \{U \in \mathcal{C}. \exists x \in V. F \ U \ x \neq 0\}$  if  $x \in S$ 
for  $x$ 
    using fin [OF that] that
    by (fastforce simp: setdist_eq_0_sing_1 closure_def F_def elim!: rev_finite_subset)
  qed
  qed

```

6.28.2 Urysohn's Lemma for Euclidean Spaces

For Euclidean spaces the proof is easy using distances.

lemma *Urysohn_both_ne*:

```

  assumes  $US: \text{closedin } (\text{top\_of\_set } U) \ S$ 
    and  $UT: \text{closedin } (\text{top\_of\_set } U) \ T$ 
    and  $S \cap T = \{\} \ S \neq \{\} \ T \neq \{\} \ a \neq b$ 
  obtains  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{real\_normed\_vector}$ 
    where  $\text{continuous\_on } U \ f$ 
       $\bigwedge x. x \in U \implies f \ x \in \text{closed\_segment } a \ b$ 
       $\bigwedge x. x \in U \implies (f \ x = a \longleftrightarrow x \in S)$ 
       $\bigwedge x. x \in U \implies (f \ x = b \longleftrightarrow x \in T)$ 

```

proof –

```

  have  $S0: \bigwedge x. x \in U \implies \text{setdist } \{x\} \ S = 0 \longleftrightarrow x \in S$ 
    using  $\langle S \neq \{\} \rangle \ US \ \text{setdist\_eq\_0\_closedin}$  by auto
  have  $T0: \bigwedge x. x \in U \implies \text{setdist } \{x\} \ T = 0 \longleftrightarrow x \in T$ 
    using  $\langle T \neq \{\} \rangle \ UT \ \text{setdist\_eq\_0\_closedin}$  by auto
  have  $sdpos: 0 < \text{setdist } \{x\} \ S + \text{setdist } \{x\} \ T$  if  $x \in U$  for  $x$ 
  proof –
    have  $\neg (\text{setdist } \{x\} \ S = 0 \wedge \text{setdist } \{x\} \ T = 0)$ 

```

```

    using assms by (metis IntI empty_iff setdist_eq_0_closedin that)
  then show ?thesis
    by (metis add.left_neutral add.right_neutral add_pos_pos linorder_neqE_linordered_idom
not_le setdist_pos_le)
  qed
  define f where  $f \equiv \lambda x. a + (\text{setdist } \{x\} S / (\text{setdist } \{x\} S + \text{setdist } \{x\} T))$ 
*_R (b - a)
  show ?thesis
  proof (rule_tac f = f in that)
    show continuous_on U f
      using sdpos unfolding f_def
      by (intro continuous_intros | force)+
    show  $f x \in \text{closed\_segment } a b$  if  $x \in U$  for  $x$ 
      unfolding f_def
      apply (simp add: closed_segment_def)
      apply (rule_tac  $x = (\text{setdist } \{x\} S / (\text{setdist } \{x\} S + \text{setdist } \{x\} T))$  in exI)
      using sdpos that apply (simp add: algebra_simps)
      done
    show  $\bigwedge x. x \in U \implies (f x = a \longleftrightarrow x \in S)$ 
      using S0  $\langle a \neq b \rangle$  f_def sdpos by force
    show  $(f x = b \longleftrightarrow x \in T)$  if  $x \in U$  for  $x$ 
    proof -
      have  $f x = b \longleftrightarrow (\text{setdist } \{x\} S / (\text{setdist } \{x\} S + \text{setdist } \{x\} T)) = 1$ 
        unfolding f_def
        apply (rule iffI)
        apply (metis  $\langle a \neq b \rangle$  add_diff_cancel_left' eq_iff_diff_eq_0 pth_1
real_vector.scale_right_imp_eq, force)
      done
      also have  $\dots \longleftrightarrow \text{setdist } \{x\} T = 0 \wedge \text{setdist } \{x\} S \neq 0$ 
        using sdpos that
        by (simp add: field_split_simps) linarith
      also have  $\dots \longleftrightarrow x \in T$ 
        using  $\langle S \neq \{\} \rangle \langle T \neq \{\} \rangle \langle S \cap T = \{\} \rangle$  that
        by (force simp: S0 T0)
      finally show ?thesis .
    qed
  qed
qed

```

proposition *Urysohn_local_strong*:

```

  assumes US: closedin (top_of_set U) S
    and UT: closedin (top_of_set U) T
    and S ∩ T = {} a ≠ b
  obtains f :: 'a::euclidean_space ⇒ 'b::euclidean_space
    where continuous_on U f
       $\bigwedge x. x \in U \implies f x \in \text{closed\_segment } a b$ 
       $\bigwedge x. x \in U \implies (f x = a \longleftrightarrow x \in S)$ 
       $\bigwedge x. x \in U \implies (f x = b \longleftrightarrow x \in T)$ 
  proof (cases S = {})

```



```

case True show ?thesis
proof (cases T = {})
case True show ?thesis
proof (rule_tac f =  $\lambda x. \text{midpoint } a \ b$  in that)
show continuous_on U ( $\lambda x. \text{midpoint } a \ b$ )
by (intro continuous_intros)
show  $\text{midpoint } a \ b \in \text{closed\_segment } a \ b$ 
using csegment_midpoint_subset by blast
show ( $\text{midpoint } a \ b = a$ ) = ( $x \in S$ ) for x
using  $\langle S = \{\} \rangle \langle a \neq b \rangle$  by simp
show ( $\text{midpoint } a \ b = b$ ) = ( $x \in T$ ) for x
using  $\langle T = \{\} \rangle \langle a \neq b \rangle$  by simp
qed
next
case False
show ?thesis
proof (cases T = U)
case True with  $\langle S = \{\} \rangle \langle a \neq b \rangle$  show ?thesis
by (rule_tac f =  $\lambda x. b$  in that) (auto)
next
case False
with UT_closedin_subset obtain c where  $c: c \in U \ c \notin T$ 
by fastforce
obtain f where f: continuous_on U f
 $\wedge x. x \in U \implies f \ x \in \text{closed\_segment } (\text{midpoint } a \ b) \ b$ 
 $\wedge x. x \in U \implies (f \ x = \text{midpoint } a \ b \longleftrightarrow x = c)$ 
 $\wedge x. x \in U \implies (f \ x = b \longleftrightarrow x \in T)$ 
apply (rule Urysohn_both_ne [of U {c} T midpoint a b b])
using  $c \langle T \neq \{\} \rangle$  assms apply simp_all
done
show ?thesis
apply (rule_tac f=f in that)
using  $\langle S = \{\} \rangle \langle T \neq \{\} \rangle f \text{ csegment\_midpoint\_subset notin\_segment\_midpoint}$ 
[OF  $\langle a \neq b \rangle$ ]
apply force+
done
qed
qed
next
case False
show ?thesis
proof (cases T = {})
case True show ?thesis
proof (cases S = U)
case True with  $\langle T = \{\} \rangle \langle a \neq b \rangle$  show ?thesis
by (rule_tac f =  $\lambda x. a$  in that) (auto)
next
case False
with US_closedin_subset obtain c where  $c: c \in U \ c \notin S$ 

```

```

    by fastforce
  obtain f where f: continuous_on U f
     $\bigwedge x. x \in U \implies f x \in \text{closed\_segment } a \text{ (midpoint } a \text{ } b)$ 
     $\bigwedge x. x \in U \implies (f x = \text{midpoint } a \text{ } b \longleftrightarrow x = c)$ 
     $\bigwedge x. x \in U \implies (f x = a \longleftrightarrow x \in S)$ 
  apply (rule Urysohn_both_ne [of U S {c} a midpoint a b])
  using c  $\langle S \neq \{\} \rangle$  assms apply simp_all
  apply (metis midpoint_eq_endpoint)
  done
show ?thesis
  apply (rule_tac f=f in that)
  using  $\langle S \neq \{\} \rangle \langle T = \{\} \rangle f \langle a \neq b \rangle$ 
  apply simp_all
  apply (metis (no_types) closed_segment_commute csegment_midpoint_subset
midpoint_sym subset_iff)
  apply (metis closed_segment_commute midpoint_sym notin_segment_midpoint)
  done
qed
next
case False
show ?thesis
  using Urysohn_both_ne [OF US UT  $\langle S \cap T = \{\} \rangle \langle S \neq \{\} \rangle \langle T \neq \{\} \rangle \langle a \neq b \rangle$ ] that
  by blast
qed
qed

lemma Urysohn_local:
  assumes US: closedin (top_of_set U) S
    and UT: closedin (top_of_set U) T
    and S  $\cap T = \{\}$ 
  obtains f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  where continuous_on U f
     $\bigwedge x. x \in U \implies f x \in \text{closed\_segment } a \text{ } b$ 
     $\bigwedge x. x \in S \implies f x = a$ 
     $\bigwedge x. x \in T \implies f x = b$ 
proof (cases a = b)
case True then show ?thesis
  by (rule_tac f =  $\lambda x. b$  in that) (auto)
next
case False
  then show ?thesis
    apply (rule Urysohn_local_strong [OF assms])
    apply (erule that, assumption)
    apply (meson US closedin_singleton closedin_trans)
    apply (meson UT closedin_singleton closedin_trans)
    done
qed

```

```

lemma Urysohn_strong:
  assumes US: closed S
    and UT: closed T
    and  $S \cap T = \{\}$   $a \neq b$ 
  obtains  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
    where continuous_on UNIV f
       $\bigwedge x. f\ x \in \text{closed\_segment } a\ b$ 
       $\bigwedge x. f\ x = a \longleftrightarrow x \in S$ 
       $\bigwedge x. f\ x = b \longleftrightarrow x \in T$ 
using assms by (auto intro: Urysohn_local_strong [of UNIV S T])

```

```

proposition Urysohn:
  assumes US: closed S
    and UT: closed T
    and  $S \cap T = \{\}$ 
  obtains  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
    where continuous_on UNIV f
       $\bigwedge x. f\ x \in \text{closed\_segment } a\ b$ 
       $\bigwedge x. x \in S \implies f\ x = a$ 
       $\bigwedge x. x \in T \implies f\ x = b$ 
using assms by (auto intro: Urysohn_local [of UNIV S T a b])

```

6.28.3 Dugundji's Extension Theorem and Tietze Variants

See [3].

```

lemma convex_supp_sum:
  assumes convex S and  $1: \text{supp\_sum } u\ I = 1$ 
    and  $\bigwedge i. i \in I \implies 0 \leq u\ i \wedge (u\ i = 0 \vee f\ i \in S)$ 
  shows  $\text{supp\_sum } (\lambda i. u\ i *_{\mathbb{R}} f\ i)\ I \in S$ 
proof –
  have fin: finite  $\{i \in I. u\ i \neq 0\}$ 
    using  $1\ \text{sum.infinite}$  by (force simp: supp_sum_def support_on_def)
  then have  $\text{supp\_sum } (\lambda i. u\ i *_{\mathbb{R}} f\ i)\ I = \text{sum } (\lambda i. u\ i *_{\mathbb{R}} f\ i)\ \{i \in I. u\ i \neq 0\}$ 
    by (force intro: sum.mono_neutral_left simp: supp_sum_def support_on_def)
  also have  $\dots \in S$ 
    using  $1\ \text{assms}$  by (force simp: supp_sum_def support_on_def intro: convex_sum [OF fin  $\langle \text{convex } S \rangle$ ])
  finally show ?thesis .
qed

```

```

theorem Dugundji:
  fixes  $f :: 'a::\{\text{metric\_space}, \text{second\_countable\_topology}\} \Rightarrow 'b::\text{real\_inner}$ 
  assumes convex C  $C \neq \{\}$ 
    and cloin: closedin (top_of_set U) S
    and contf: continuous_on S f and  $f\ 'S \subseteq C$ 
  obtains g where continuous_on U g  $g\ 'U \subseteq C$ 
     $\bigwedge x. x \in S \implies g\ x = f\ x$ 
proof (cases S = {})
  case True then show thesis

```

```

    apply (rule_tac g= $\lambda x. \text{SOME } y. y \in C$  in that)
    apply (rule continuous_intros)
    apply (meson all_not_in_conv  $\langle C \neq \{\} \rangle$  image_subsetI someI_ex, simp)
  done
next
case False
then have sd_pos:  $\bigwedge x. \llbracket x \in U; x \notin S \rrbracket \implies 0 < \text{setdist } \{x\} S$ 
  using setdist_eq_0_closedin [OF cloin] le_less setdist_pos_le by fastforce
define  $\mathcal{B}$  where  $\mathcal{B} = \{\text{ball } x (\text{setdist } \{x\} S / 2) \mid x. x \in U - S\}$ 
have [simp]:  $\bigwedge T. T \in \mathcal{B} \implies \text{open } T$ 
  by (auto simp:  $\mathcal{B}$ _def)
have USS:  $U - S \subseteq \bigcup \mathcal{B}$ 
  by (auto simp: sd_pos  $\mathcal{B}$ _def)
obtain  $\mathcal{C}$  where USSub:  $U - S \subseteq \bigcup \mathcal{C}$ 
  and nbrhd:  $\bigwedge U. U \in \mathcal{C} \implies \text{open } U \wedge (\exists T. T \in \mathcal{B} \wedge U \subseteq T)$ 
  and fin:  $\bigwedge x. x \in U - S \implies \exists V. \text{open } V \wedge x \in V \wedge \text{finite } \{U. U \in \mathcal{C} \wedge U$ 
 $\cap V \neq \{\}\}$ 
  by (rule paracompact [OF USS]) auto
have  $\exists v a. v \in U \wedge v \notin S \wedge a \in S \wedge$ 
 $T \subseteq \text{ball } v (\text{setdist } \{v\} S / 2) \wedge$ 
 $\text{dist } v a \leq 2 * \text{setdist } \{v\} S$  if  $T \in \mathcal{C}$  for  $T$ 
proof -
  obtain  $v$  where  $v: T \subseteq \text{ball } v (\text{setdist } \{v\} S / 2) \wedge v \in U \wedge v \notin S$ 
    using  $\langle T \in \mathcal{C} \rangle$  nbrhd by (force simp:  $\mathcal{B}$ _def)
  then obtain  $a$  where  $a \in S \wedge \text{dist } v a < 2 * \text{setdist } \{v\} S$ 
    using setdist_ltE [of  $\{v\} S 2 * \text{setdist } \{v\} S$ ]
    using False sd_pos by force
  with  $v$  show ?thesis
    apply (rule_tac  $x=v$  in exI)
    apply (rule_tac  $x=a$  in exI, auto)
  done
qed
then obtain  $\mathcal{V} \mathcal{A}$  where
  VA:  $\bigwedge T. T \in \mathcal{C} \implies \mathcal{V} T \in U \wedge \mathcal{V} T \notin S \wedge \mathcal{A} T \in S \wedge$ 
 $T \subseteq \text{ball } (\mathcal{V} T) (\text{setdist } \{\mathcal{V} T\} S / 2) \wedge$ 
 $\text{dist } (\mathcal{V} T) (\mathcal{A} T) \leq 2 * \text{setdist } \{\mathcal{V} T\} S$ 
  by metis
have sdle:  $\text{setdist } \{\mathcal{V} T\} S \leq 2 * \text{setdist } \{v\} S$  if  $T \in \mathcal{C} \wedge v \in T$  for  $T v$ 
  using setdist_Lipschitz [of  $\mathcal{V} T S v$ ] VA [OF  $\langle T \in \mathcal{C} \rangle$ ]  $\langle v \in T \rangle$  by auto
have d6:  $\text{dist } a (\mathcal{A} T) \leq 6 * \text{dist } a v$  if  $T \in \mathcal{C} \wedge v \in T \wedge a \in S$  for  $T v a$ 
proof -
  have  $\text{dist } (\mathcal{V} T) v < \text{setdist } \{\mathcal{V} T\} S / 2$ 
    using that VA mem_ball by blast
  also have  $\dots \leq \text{setdist } \{v\} S$ 
    using sdle [OF  $\langle T \in \mathcal{C} \rangle \langle v \in T \rangle$ ] by simp
  also have  $vS: \text{setdist } \{v\} S \leq \text{dist } a v$ 
    by (simp add: setdist_le_dist setdist_sym  $\langle a \in S \rangle$ )
  finally have VTV:  $\text{dist } (\mathcal{V} T) v < \text{dist } a v$  .
  have VTS:  $\text{setdist } \{\mathcal{V} T\} S \leq 2 * \text{dist } a v$ 

```

```

    using sdle that vS by force
  have  $\text{dist } a (\mathcal{A} \ T) \leq \text{dist } a \ v + \text{dist } v (\mathcal{V} \ T) + \text{dist } (\mathcal{V} \ T) (\mathcal{A} \ T)$ 
    by (metis add.commute add_le_cancel_left dist_commute dist_triangle2
dist_triangle_le)
  also have  $\dots \leq \text{dist } a \ v + \text{dist } a \ v + \text{dist } (\mathcal{V} \ T) (\mathcal{A} \ T)$ 
    using VTV by (simp add: dist_commute)
  also have  $\dots \leq 2 * \text{dist } a \ v + 2 * \text{setdist } \{\mathcal{V} \ T\} \ S$ 
    using VA [OF ‹ $T \in \mathcal{C}$ ›] by auto
  finally show ?thesis
    using VTS by linarith
qed
obtain  $H :: ['a \ \text{set}, 'a] \Rightarrow \text{real}$ 
  where Hcont:  $\bigwedge Z. Z \in \mathcal{C} \implies \text{continuous\_on } (U - S) (H \ Z)$ 
    and Hge0:  $\bigwedge Z \ x. \llbracket Z \in \mathcal{C}; x \in U - S \rrbracket \implies 0 \leq H \ Z \ x$ 
    and Heq0:  $\bigwedge x \ Z. \llbracket Z \in \mathcal{C}; x \in U - S; x \notin Z \rrbracket \implies H \ Z \ x = 0$ 
    and H1:  $\bigwedge x. x \in U - S \implies \text{supp\_sum } (\lambda W. H \ W \ x) \ \mathcal{C} = 1$ 
    and Hfin:  $\bigwedge x. x \in U - S \implies \exists V. \text{open } V \wedge x \in V \wedge \text{finite } \{U \in \mathcal{C}. \exists x \in V. H \ U \ x \neq 0\}$ 
  apply (rule subordinate_partition_of_unity [OF USsub _ fin])
  using nbrhd by auto
  define g where  $g \equiv \lambda x. \text{if } x \in S \text{ then } f \ x \text{ else } \text{supp\_sum } (\lambda T. H \ T \ x *_{\mathbb{R}} f(\mathcal{A} \ T)) \ \mathcal{C}$ 
  show ?thesis
  proof (rule that)
    show continuous_on U g
  proof (clarsimp simp: continuous_on_eq_continuous_within)
    fix a assume  $a \in U$ 
    show continuous (at a within U) g
  proof (cases  $a \in S$ )
    case True show ?thesis
  proof (clarsimp simp add: continuous_within_topological)
    fix W
    assume open W  $g \ a \in W$ 
    then obtain e where  $0 < e$  and  $e: \text{ball } (f \ a) \ e \subseteq W$ 
      using openE True g_def by auto
    have continuous (at a within S) f
      using True contf continuous_on_eq_continuous_within by blast
    then obtain d where  $0 < d$ 
      and  $d: \bigwedge x. \llbracket x \in S; \text{dist } x \ a < d \rrbracket \implies \text{dist } (f \ x) (f \ a) < e$ 
      using continuous_within_eps_delta ‹ $0 < e$ › by force
    have  $g \ y \in \text{ball } (f \ a) \ e$  if  $y \in U$  and  $y: y \in \text{ball } a \ (d / 6)$  for y
  proof (cases  $y \in S$ )
    case True
    then have  $\text{dist } (f \ a) (f \ y) < e$ 
  by (metis ball_divide_subset_numeral dist_commute in_mono mem_ball
y d)
    then show ?thesis
      by (simp add: True g_def)
  next

```

```

case False
have *: dist (f (A T)) (f a) < e if T ∈ C H T y ≠ 0 for T
proof -
  have y ∈ T
  using Heq0 that False ⟨y ∈ U⟩ by blast
  have dist (A T) a < d
  using d6 [OF ⟨T ∈ C⟩ ⟨y ∈ T⟩ ⟨a ∈ S⟩] y
  by (simp add: dist_commute mult.commute)
  then show ?thesis
  using VA [OF ⟨T ∈ C⟩] by (auto simp: d)
qed
have supp_sum (λT. H T y *R f (A T)) C ∈ ball (f a) e
  apply (rule convex_supp_sum [OF convex_ball])
  apply (simp_all add: False H1 Hge0 ⟨y ∈ U⟩)
  by (metis dist_commute *)
then show ?thesis
  by (simp add: False g_def)
qed
then show ∃ A. open A ∧ a ∈ A ∧ (∀ y ∈ U. y ∈ A → g y ∈ W)
  apply (rule_tac x = ball a (d / 6) in exI)
  using e ⟨0 < d⟩ by fastforce
qed
next
case False
obtain N where N: open N a ∈ N
  and finN: finite {U ∈ C. ∃ a ∈ N. H U a ≠ 0}
  using Hfin False ⟨a ∈ U⟩ by auto
have oUS: openin (top_of_set U) (U - S)
  using cloin by (simp add: openin_diff)
have HcontU: continuous (at a within U) (H T) if T ∈ C for T
  using Hcont [OF ⟨T ∈ C⟩] False ⟨a ∈ U⟩ ⟨T ∈ C⟩
  apply (simp add: continuous_on_eq_continuous_within continuous_within)
  apply (rule Lim_transform_within_set)
  using oUS
  apply (force simp: eventually_at openin_contains_ball dist_commute
dest!: bspec)+
  done
show ?thesis
proof (rule continuous_transform_within_openin [OF _ oUS])
  show continuous (at a within U) (λx. supp_sum (λT. H T x *R f (A T))
C)
  proof (rule continuous_transform_within_openin)
    show continuous (at a within U)
      (λx. ∑ T ∈ {U ∈ C. ∃ x ∈ N. H U x ≠ 0}. H T x *R f (A T))
    by (force intro: continuous_intros HcontU)+
  next
    show openin (top_of_set U) ((U - S) ∩ N)
    using N oUS openin_trans by blast
  next

```

```

      show  $a \in (U - S) \cap N$  using False  $\langle a \in U \rangle N$  by blast
    next
      show  $\bigwedge x. x \in (U - S) \cap N \implies$ 
         $(\sum T \in \{U \in \mathcal{C}. \exists x \in N. H \ U \ x \neq 0\}. H \ T \ x *_R f (\mathcal{A} \ T))$ 
         $= \text{supp\_sum } (\lambda T. H \ T \ x *_R f (\mathcal{A} \ T)) \ \mathcal{C}$ 
      by (auto simp: supp_sum_def support_on_def
        intro: sum.mono_neutral_right [OF finN])
    qed
  next
    show  $a \in U - S$  using False  $\langle a \in U \rangle$  by blast
  next
    show  $\bigwedge x. x \in U - S \implies \text{supp\_sum } (\lambda T. H \ T \ x *_R f (\mathcal{A} \ T)) \ \mathcal{C} = g \ x$ 
    by (simp add: g_def)
  qed
qed
qed
qed
show  $g \restriction U \subseteq C$ 
using  $\langle f \restriction S \subseteq C \rangle \ VA$ 
by (fastforce simp: g_def Hge0 intro!: convex_supp_sum [OF  $\langle \text{convex } C \rangle$ ])
H1)
show  $\bigwedge x. x \in S \implies g \ x = f \ x$ 
by (simp add: g_def)
qed
qed

```

corollary *Tietze*:

```

fixes  $f :: 'a :: \{\text{metric\_space}, \text{second\_countable\_topology}\} \Rightarrow 'b :: \text{real\_inner}$ 
assumes continuous_on  $S \ f$ 
and closedin (top_of_set  $U$ )  $S$ 
and  $0 \leq B$ 
and  $\bigwedge x. x \in S \implies \text{norm}(f \ x) \leq B$ 
obtains  $g$  where continuous_on  $U \ g$   $\bigwedge x. x \in S \implies g \ x = f \ x$ 
 $\bigwedge x. x \in U \implies \text{norm}(g \ x) \leq B$ 
using assms by (auto simp: image_subset_iff intro: Dugundji [of cball 0 B U S f])

```

corollary *Tietze_closed_interval*:

```

fixes  $f :: 'a :: \{\text{metric\_space}, \text{second\_countable\_topology}\} \Rightarrow 'b :: \text{euclidean\_space}$ 
assumes continuous_on  $S \ f$ 
and closedin (top_of_set  $U$ )  $S$ 
and cbox  $a \ b \neq \{\}$ 
and  $\bigwedge x. x \in S \implies f \ x \in \text{cbox } a \ b$ 
obtains  $g$  where continuous_on  $U \ g$   $\bigwedge x. x \in S \implies g \ x = f \ x$ 
 $\bigwedge x. x \in U \implies g \ x \in \text{cbox } a \ b$ 
apply (rule Dugundji [of cbox a b U S f])
using assms by auto

```

corollary *Tietze_closed_interval_1*:

```

fixes  $f :: 'a::\{\text{metric\_space}, \text{second\_countable\_topology}\} \Rightarrow \text{real}$ 
assumes  $\text{continuous\_on } S f$ 
and  $\text{closedin } (\text{top\_of\_set } U) S$ 
and  $a \leq b$ 
and  $\bigwedge x. x \in S \implies f x \in \text{cbox } a b$ 
obtains  $g$  where  $\text{continuous\_on } U g \bigwedge x. x \in S \implies g x = f x$ 
 $\bigwedge x. x \in U \implies g x \in \text{cbox } a b$ 
apply ( $\text{rule Dugundji [of cbox } a b U S f]$ )
using  $\text{assms by (auto simp: image\_subset\_iff)}$ 

```

corollary *Tietze_open_interval:*

```

fixes  $f :: 'a::\{\text{metric\_space}, \text{second\_countable\_topology}\} \Rightarrow 'b::\text{euclidean\_space}$ 
assumes  $\text{continuous\_on } S f$ 
and  $\text{closedin } (\text{top\_of\_set } U) S$ 
and  $\text{box } a b \neq \{\}$ 
and  $\bigwedge x. x \in S \implies f x \in \text{box } a b$ 
obtains  $g$  where  $\text{continuous\_on } U g \bigwedge x. x \in S \implies g x = f x$ 
 $\bigwedge x. x \in U \implies g x \in \text{box } a b$ 
apply ( $\text{rule Dugundji [of box } a b U S f]$ )
using  $\text{assms by auto}$ 

```

corollary *Tietze_open_interval_1:*

```

fixes  $f :: 'a::\{\text{metric\_space}, \text{second\_countable\_topology}\} \Rightarrow \text{real}$ 
assumes  $\text{continuous\_on } S f$ 
and  $\text{closedin } (\text{top\_of\_set } U) S$ 
and  $a < b$ 
and  $\text{no: } \bigwedge x. x \in S \implies f x \in \text{box } a b$ 
obtains  $g$  where  $\text{continuous\_on } U g \bigwedge x. x \in S \implies g x = f x$ 
 $\bigwedge x. x \in U \implies g x \in \text{box } a b$ 
apply ( $\text{rule Dugundji [of box } a b U S f]$ )
using  $\text{assms by (auto simp: image\_subset\_iff)}$ 

```

corollary *Tietze_unbounded:*

```

fixes  $f :: 'a::\{\text{metric\_space}, \text{second\_countable\_topology}\} \Rightarrow 'b::\text{real\_inner}$ 
assumes  $\text{continuous\_on } S f$ 
and  $\text{closedin } (\text{top\_of\_set } U) S$ 
obtains  $g$  where  $\text{continuous\_on } U g \bigwedge x. x \in S \implies g x = f x$ 
apply ( $\text{rule Dugundji [of UNIV } U S f]$ )
using  $\text{assms by auto}$ 

```

end

6.29 Equivalence Between Classical Borel Measurability and HOL Light's

theory *Equivalence_Measurable_On_Borel*

imports *Equivalence_Lebesgue_Henstock_Integration Improper_Integral Continuous_Extension*

begin

abbreviation $\text{sym\_diff} :: 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$ **where**
 $\text{sym\_diff } A \ B \equiv ((A - B) \cup (B - A))$

6.29.1 Austin's Lemma

lemma *Austin_Lemma*:
fixes $\mathcal{D} :: 'a::\text{euclidean\_space} \text{ set set}$
assumes *finite* $\mathcal{D}$ **and** $\mathcal{D}$: $\bigwedge D. D \in \mathcal{D} \implies \exists k \ a \ b. D = \text{cbox } a \ b \wedge (\forall i \in \text{Basis}. b \cdot i - a \cdot i = k)$
obtains $\mathcal{C}$ **where** $\mathcal{C} \subseteq \mathcal{D}$ *pairwise disjoint* $\mathcal{C}$
 $\text{measure lebesgue } (\bigcup \mathcal{C}) \geq \text{measure lebesgue } (\bigcup \mathcal{D}) / 3^{\wedge}(\text{DIM}('a))$
using *assms*
proof (*induction card* $\mathcal{D}$ *arbitrary*: $\mathcal{D}$ *thesis rule*: *less_induct*)
case *less*
show *?case*
proof (*cases* $\mathcal{D} = \{\}$)
case *True*
then show *thesis*
using *less* **by** *auto*
next
case *False*
then have $\text{Max } (\text{Sigma\_Algebra.measure lebesgue } ' \mathcal{D}) \in \text{Sigma\_Algebra.measure lebesgue } ' \mathcal{D}$
using *Max_in_finite_imageI* $\langle \text{finite } \mathcal{D} \rangle$ **by** *blast*
then obtain D **where** $D \in \mathcal{D}$ **and** $\text{measure lebesgue } D = \text{Max } (\text{measure lebesgue } ' \mathcal{D})$
by *auto*
then have $D: \bigwedge C. C \in \mathcal{D} \implies \text{measure lebesgue } C \leq \text{measure lebesgue } D$
by (*simp add*: $\langle \text{finite } \mathcal{D} \rangle$)
let $\mathcal{E} = \{C. C \in \mathcal{D} - \{D\} \wedge \text{disjnt } C \ D\}$
obtain $\mathcal{D}'$ **where** $\mathcal{D}'_{\text{sub}}: \mathcal{D}' \subseteq \mathcal{E}$ **and** $\mathcal{D}'_{\text{dis}}: \text{pairwise disjoint } \mathcal{D}'$
and $\mathcal{D}'_{\text{m}}: \text{measure lebesgue } (\bigcup \mathcal{D}') \geq \text{measure lebesgue } (\bigcup \mathcal{E}) / 3^{\wedge}(\text{DIM}('a))$
proof (*rule less.hyps*)
have $*$: $\mathcal{E} \subseteq \mathcal{D}$
using $\langle D \in \mathcal{D} \rangle$ **by** *auto*
then show $\text{card } \mathcal{E} < \text{card } \mathcal{D}$ *finite* $\mathcal{E}$
by (*auto simp*: $\langle \text{finite } \mathcal{D} \rangle$ *psubset_card_mono*)
show $\exists k \ a \ b. D = \text{cbox } a \ b \wedge (\forall i \in \text{Basis}. b \cdot i - a \cdot i = k)$ **if** $D \in \mathcal{E}$ **for** D
using *less.prem* $s(3)$ **that** **by** *auto*
qed
then have $[\text{simp}]: \bigcup \mathcal{D}' - D = \bigcup \mathcal{D}'$
by (*auto simp*: *disjnt_iff*)
show *?thesis*
proof (*rule less.prem* s)
show $\text{insert } D \ \mathcal{D}' \subseteq \mathcal{D}$
using $\mathcal{D}'_{\text{sub}} \ \langle D \in \mathcal{D} \rangle$ **by** *blast*

```

show disjoint (insert D  $\mathcal{D}'$ )
  using  $\mathcal{D}'dis \mathcal{D}'sub$  by (fastforce simp add: pairwise_def disjnt_sym)
  obtain  $a3 \ b3$  where  $m3$ : content (cbox  $a3 \ b3$ ) =  $3 \wedge DIM('a) * measure$ 
lebesgue D
  and  $sub3$ :  $\bigwedge C. \llbracket C \in \mathcal{D}; \neg disjnt \ C \ D \rrbracket \implies C \subseteq cbox \ a3 \ b3$ 
proof -
  obtain  $k \ a \ b$  where  $ab$ :  $D = cbox \ a \ b$  and  $k$ :  $\bigwedge i. i \in Basis \implies b \cdot i - a \cdot i$ 
= k
  using less.premis  $\langle D \in \mathcal{D} \rangle$  by meson
  then have  $eqk$ :  $\bigwedge i. i \in Basis \implies a \cdot i \leq b \cdot i \iff k \geq 0$ 
  by force
  show thesis
  proof
    let  $?a = (a + b) /_R 2 - (3/2) *_R (b - a)$ 
    let  $?b = (a + b) /_R 2 + (3/2) *_R (b - a)$ 
    have  $eq$ :  $(\prod i \in Basis. b \cdot i * 3 - a \cdot i * 3) = (\prod i \in Basis. b \cdot i - a \cdot i) *$ 
 $3 \wedge DIM('a)$ 
    by (simp add: comm_monoid_mult_class.prod.distrib flip: left_diff_distrib
inner_diff_left)
    show content (cbox  $?a \ ?b$ ) =  $3 \wedge DIM('a) * measure \ lebesgue \ D$ 
    by (simp add: content_cbox_if box_eq_empty algebra_simps eq  $ab \ k$ )
    show  $C \subseteq cbox \ ?a \ ?b$  if  $C \in \mathcal{D}$  and  $CD$ :  $\neg disjnt \ C \ D$  for  $C$ 
    proof -
      obtain  $k' \ a' \ b'$  where  $ab'$ :  $C = cbox \ a' \ b'$  and  $k'$ :  $\bigwedge i. i \in Basis \implies$ 
 $b' \cdot i - a' \cdot i = k'$ 
      using less.premis  $\langle C \in \mathcal{D} \rangle$  by meson
      then have  $eqk'$ :  $\bigwedge i. i \in Basis \implies a' \cdot i \leq b' \cdot i \iff k' \geq 0$ 
      by force
      show ?thesis
      proof (clarsimp simp add: disjoint_interval disjnt_def  $ab \ ab' \ not\_less$ 
subset_box algebra_simps)
        show  $a \cdot i * 2 \leq a' \cdot i + b \cdot i \wedge a \cdot i + b' \cdot i \leq b \cdot i * 2$ 
        if * [rule_format]:  $\forall j \in Basis. a' \cdot j \leq b' \cdot j$  and  $i \in Basis$  for  $i$ 
        proof -
          have  $a' \cdot i \leq b' \cdot i \wedge a \cdot i \leq b \cdot i \wedge a \cdot i \leq b' \cdot i \wedge a' \cdot i \leq b \cdot i$ 
          using  $\langle i \in Basis \rangle \ CD$  by (simp_all add: disjoint_interval disjnt_def
 $ab \ ab' \ not\_less$ )
          then show ?thesis
          using  $D \ [OF \ \langle C \in \mathcal{D} \rangle] \ \langle i \in Basis \rangle$ 
          apply (simp add:  $ab \ ab' \ k \ k' \ eqk \ eqk' \ content\_cbox\_cases$ )
          using  $k \ k'$  by fastforce
        qed
      qed
    qed
  qed
  have  $Dlm$ :  $\bigwedge D. D \in \mathcal{D} \implies D \in lmeasurable$ 
  using less.premis(3) by blast
  have measure lebesgue  $(\bigcup \mathcal{D}) \leq measure \ lebesgue \ (cbox \ a3 \ b3 \cup (\bigcup \mathcal{D} - cbox$ 

```

```

a3 b3))
  proof (rule measure_mono_fmeasurable)
    show  $\bigcup \mathcal{D} \in \text{sets lebesgue}$ 
      using  $\mathcal{D} \text{lm } \langle \text{finite } \mathcal{D} \rangle$  by blast
    show  $\text{cbox } a3 \ b3 \cup (\bigcup \mathcal{D} - \text{cbox } a3 \ b3) \in \text{lmeasurable}$ 
      by (simp add:  $\mathcal{D} \text{lm fmeasurable.Un fmeasurable.finite\_Union less.prem}(2)$ 
subset_eq)
    qed auto
    also have  $\dots = \text{content } (\text{cbox } a3 \ b3) + \text{measure lebesgue } (\bigcup \mathcal{D} - \text{cbox } a3 \ b3)$ 
      by (simp add:  $\mathcal{D} \text{lm fmeasurable.finite\_Union less.prem}(2)$  measure_Un2
subsetI)
    also have  $\dots \leq (\text{measure lebesgue } D + \text{measure lebesgue } (\bigcup \mathcal{D}')) * 3 ^ \wedge \text{DIM}('a)$ 
      proof -
        have  $(\bigcup \mathcal{D} - \text{cbox } a3 \ b3) \subseteq \bigcup ?\mathcal{E}$ 
          using sub3 by fastforce
        then have  $\text{measure lebesgue } (\bigcup \mathcal{D} - \text{cbox } a3 \ b3) \leq \text{measure lebesgue } (\bigcup ?\mathcal{E})$ 
          proof (rule measure_mono_fmeasurable)
            show  $\bigcup \mathcal{D} - \text{cbox } a3 \ b3 \in \text{sets lebesgue}$ 
              by (simp add:  $\mathcal{D} \text{lm fmeasurableD less.prem}(2)$  sets.Diff sets.finite\_Union
subsetI)
            show  $\bigcup \{C \in \mathcal{D} - \{D\}. \text{disjnt } C \ D\} \in \text{lmeasurable}$ 
              using  $\mathcal{D} \text{lm less.prem}(2)$  by auto
          qed
        then have  $\text{measure lebesgue } (\bigcup \mathcal{D} - \text{cbox } a3 \ b3) / 3 ^ \wedge \text{DIM}('a) \leq \text{measure lebesgue } (\bigcup \mathcal{D}')$ 
          using  $\mathcal{D}' \text{m}$  by (simp add: field_split_simps)
        then show ?thesis
          by (simp add: m3 field_simps)
      qed
    also have  $\dots \leq \text{measure lebesgue } (\bigcup (\text{insert } D \ \mathcal{D}')) * 3 ^ \wedge \text{DIM}('a)$ 
      proof (simp add:  $\mathcal{D} \text{lm } \langle D \in \mathcal{D} \rangle$ )
        show  $\text{measure lebesgue } D + \text{measure lebesgue } (\bigcup \mathcal{D}') \leq \text{measure lebesgue } (D \cup \bigcup \mathcal{D}')$ 
          proof (subst measure_Un2)
            show  $\bigcup \mathcal{D}' \in \text{lmeasurable}$ 
              by (meson  $\mathcal{D} \text{lm } \langle \text{insert } D \ \mathcal{D}' \subseteq \mathcal{D} \rangle$  fmeasurable.finite\_Union less.prem}(2)
finite_subset subset_eq subset_insertI)
            show  $\text{measure lebesgue } D + \text{measure lebesgue } (\bigcup \mathcal{D}') \leq \text{measure lebesgue } D + \text{measure lebesgue } (\bigcup \mathcal{D}' - D)$ 
              using  $\langle \text{insert } D \ \mathcal{D}' \subseteq \mathcal{D} \rangle$  infinite_super less.prem}(2) by force
          qed
        qed (simp add:  $\mathcal{D} \text{lm } \langle D \in \mathcal{D} \rangle$ )
      qed
    finally show  $\text{measure lebesgue } (\bigcup \mathcal{D}) / 3 ^ \wedge \text{DIM}('a) \leq \text{measure lebesgue } (\bigcup (\text{insert } D \ \mathcal{D}'))$ 
      by (simp add: field_split_simps)
    qed
  qed

```

qed

6.29.2 A differentiability-like property of the indefinite integral.

proposition *integrable_ccontinuous_explicit:*

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$

assumes $\bigwedge a b::'a. f \text{ integrable\_on } \text{cbox } a \ b$

obtains N **where**

negligible N

$\bigwedge x e. \llbracket x \notin N; 0 < e \rrbracket \implies$

$\exists d > 0. \forall h. 0 < h \wedge h < d \longrightarrow$

$\text{norm}(\text{integral } (\text{cbox } x \ (x + h *_{\mathbb{R}} \text{One})) \ f \ /_{\mathbb{R}} h^{\wedge \text{DIM}('a)} - f$

$x) < e$

proof –

define BOX **where** $BOX \equiv \lambda h. \lambda x::'a. \text{cbox } x \ (x + h *_{\mathbb{R}} \text{One})$

define $BOX2$ **where** $BOX2 \equiv \lambda h. \lambda x::'a. \text{cbox } (x - h *_{\mathbb{R}} \text{One}) \ (x + h *_{\mathbb{R}} \text{One})$

define i **where** $i \equiv \lambda h x. \text{integral } (BOX \ h \ x) \ f \ /_{\mathbb{R}} h^{\wedge \text{DIM}('a)}$

define Ψ **where** $\Psi \equiv \lambda x r. \forall d > 0. \exists h. 0 < h \wedge h < d \wedge r \leq \text{norm}(i \ h \ x - f \ x)$

let $?N = \{x. \exists e > 0. \Psi \ x \ e\}$

have $\exists N. \text{negligible } N \wedge (\forall x e. x \notin N \wedge 0 < e \longrightarrow \neg \Psi \ x \ e)$

proof (*rule exI ; intro conjI allI impI*)

let $?M = \bigcup n. \{x. \Psi \ x \ (\text{inverse}(\text{real } n + 1))\}$

have *negligible* $(\{x. \Psi \ x \ \mu\} \cap \text{cbox } a \ b)$

if $\mu > 0$ **for** $a \ b \ \mu$

proof (*cases negligible(cbox a b)*)

case *True*

then show *?thesis*

by (*simp add: negligible_Int*)

next

case *False*

then have $\text{box } a \ b \neq \{\}$

by (*simp add: negligible_interval*)

then have $ab: \bigwedge i. i \in \text{Basis} \implies a \cdot i < b \cdot i$

by (*simp add: box_ne_empty*)

show *?thesis*

unfolding *negligible_outer_le*

proof (*intro allI impI*)

fix $e::\text{real}$

let $?ee = (e * \mu) / 2 / 6^{\wedge (\text{DIM}('a))}$

assume $e > 0$

then have *gt0: ?ee > 0*

using $\langle \mu > 0 \rangle$ **by** *auto*

have $f': f \text{ integrable\_on } \text{cbox } (a - \text{One}) \ (b + \text{One})$

using *assms* **by** *blast*

obtain γ **where** *gauge* γ

and $\gamma: \bigwedge p. \llbracket p \text{ tagged\_partial\_division\_of } (\text{cbox } (a - \text{One}) \ (b + \text{One})) ;$

$\gamma \text{ fine } p \rrbracket$

$\implies (\sum (x, k) \in p. \text{norm } (\text{content } k *_{\mathbb{R}} f \ x - \text{integral } k \ f)) < ?ee$

```

    using Henstock_lemma [OF f' gt0] that by auto
  let ?E = {x. x ∈ cbox a b ∧ Ψ x μ}
  have ∃ h>0. BOX h x ⊆ γ x ∧
    BOX h x ⊆ cbox (a - One) (b + One) ∧ μ ≤ norm (i h x - f x)
    if x ∈ cbox a b Ψ x μ for x
  proof -
    obtain d where d > 0 and d: ball x d ⊆ γ x
    using gaugeD [OF ⟨gauge γ⟩, of x] openE by blast
    then obtain h where 0 < h h < 1 and hless: h < d / real DIM('a)
      and mule: μ ≤ norm (i h x - f x)
    using ⟨Ψ x μ⟩ [unfolded Ψ_def, rule_format, of min 1 (d / DIM('a))]
    by auto
    show ?thesis
  proof (intro exI conjI)
    show 0 < h μ ≤ norm (i h x - f x) by fact+
    have BOX h x ⊆ ball x d
    proof (clarsimp simp: BOX_def mem_box dist_norm algebra_simps)
      fix y
      assume ∀ i∈Basis. x · i ≤ y · i ∧ y · i ≤ h + x · i
      then have lt: |(x - y) · i| < d / real DIM('a) if i ∈ Basis for i
        using hless that by (force simp: inner_diff_left)
      have norm (x - y) ≤ (∑ i∈Basis. |(x - y) · i|)
        using norm_le_l1 by blast
      also have ... < d
        using sum_bounded_above_strict [of Basis λi. |(x - y) · i| d /
DIM('a), OF lt]
        by auto
      finally show norm (x - y) < d .
    qed
    with d show BOX h x ⊆ γ x
    by blast
    show BOX h x ⊆ cbox (a - One) (b + One)
    using that ⟨h < 1⟩
    by (force simp: BOX_def mem_box algebra_simps intro: subset_box_imp)
  qed
  qed
  then obtain η where h0: ∧x. x ∈ ?E ⇒ η x > 0
  and BOX_γ: ∧x. x ∈ ?E ⇒ BOX (η x) x ⊆ γ x
  and ∧x. x ∈ ?E ⇒ BOX (η x) x ⊆ cbox (a - One) (b + One) ∧ μ ≤
norm (i (η x) x - f x)
  by simp metis
  then have BOX_cbox: ∧x. x ∈ ?E ⇒ BOX (η x) x ⊆ cbox (a - One) (b
+ One)
    and μ_le: ∧x. x ∈ ?E ⇒ μ ≤ norm (i (η x) x - f x)
  by blast+
  define γ' where γ' ≡ λx. if x ∈ cbox a b ∧ Ψ x μ then ball x (η x) else γ x
  have gauge γ'
    using ⟨gauge γ⟩ by (auto simp: h0 gauge_def γ'_def)
  obtain D where countable D

```

```

and  $\mathcal{D}$ :  $\bigcup \mathcal{D} \subseteq \text{cbox } a \ b$ 
 $\bigwedge K. K \in \mathcal{D} \implies \text{interior } K \neq \{\}$   $\wedge (\exists c \ d. K = \text{cbox } c \ d)$ 
and  $D_{\text{covered}}$ :  $\bigwedge K. K \in \mathcal{D} \implies \exists x. x \in \text{cbox } a \ b \wedge \Psi \ x \ \mu \wedge x \in K \wedge K$ 
 $\subseteq \gamma' \ x$ 
and  $\text{subUD}$ :  $?E \subseteq \bigcup \mathcal{D}$ 
by (rule covering_lemma [of ?E a b  $\gamma'$ ]) (simp_all add: Bex_def  $\langle \text{box } a \ b$ 
 $\neq \{\} \rangle$   $\langle \text{gauge } \gamma' \rangle$ )
then have  $\mathcal{D} \subseteq \text{sets lebesgue}$ 
by fastforce
show  $\exists T. \{x. \Psi \ x \ \mu\} \cap \text{cbox } a \ b \subseteq T \wedge T \in \text{lmeasurable} \wedge \text{measure lebesgue}$ 
 $T \leq e$ 
proof (intro exI conjI)
show  $\{x. \Psi \ x \ \mu\} \cap \text{cbox } a \ b \subseteq \bigcup \mathcal{D}$ 
apply auto
using subUD by auto
have  $mUE$ :  $\text{measure lebesgue } (\bigcup \mathcal{E}) \leq \text{measure lebesgue } (\text{cbox } a \ b)$ 
if  $\mathcal{E} \subseteq \mathcal{D}$  finite  $\mathcal{E}$  for  $\mathcal{E}$ 
proof (rule measure_mono_fmeasurable)
show  $\bigcup \mathcal{E} \subseteq \text{cbox } a \ b$ 
using  $\mathcal{D}(1)$  that(1) by blast
show  $\bigcup \mathcal{E} \in \text{sets lebesgue}$ 
by (metis  $\mathcal{D}(2)$  fmeasurable.finite_Union fmeasurableD lmeasurable_cbox
subset_eq that)
qed auto
then show  $\bigcup \mathcal{D} \in \text{lmeasurable}$ 
by (metis  $\mathcal{D}(2)$   $\langle \text{countable } \mathcal{D} \rangle$  fmeasurable_Union_bound lmeasur-
able_cbox)
then have leab:  $\text{measure lebesgue } (\bigcup \mathcal{D}) \leq \text{measure lebesgue } (\text{cbox } a \ b)$ 
by (meson  $\mathcal{D}(1)$  fmeasurableD lmeasurable_cbox measure_mono_fmeasurable)
obtain  $\mathcal{F}$  where  $\mathcal{F} \subseteq \mathcal{D}$  finite  $\mathcal{F}$ 
and  $\mathcal{F}$ :  $\text{measure lebesgue } (\bigcup \mathcal{D}) \leq 2 * \text{measure lebesgue } (\bigcup \mathcal{F})$ 
proof (cases  $\text{measure lebesgue } (\bigcup \mathcal{D}) = 0$ )
case True
then show ?thesis
by (force intro: that [where  $\mathcal{F} = \{\}$ ])
next
case False
obtain  $\mathcal{F}$  where  $\mathcal{F} \subseteq \mathcal{D}$  finite  $\mathcal{F}$ 
and  $\mathcal{F}$ :  $\text{measure lebesgue } (\bigcup \mathcal{D}) / 2 < \text{measure lebesgue } (\bigcup \mathcal{F})$ 
proof (rule measure_countable_Union_approachable [of  $\mathcal{D}$  measure
lebesgue  $(\bigcup \mathcal{D}) / 2$  content (cbox a b)])
show countable  $\mathcal{D}$ 
by fact
show  $0 < \text{measure lebesgue } (\bigcup \mathcal{D}) / 2$ 
using False by (simp add: zero_less_measure_iff)
show  $D_{lm}$ :  $D \in \text{lmeasurable}$  if  $D \in \mathcal{D}$  for  $D$ 
using  $\mathcal{D}(2)$  that by blast
show  $\text{measure lebesgue } (\bigcup \mathcal{F}) \leq \text{content } (\text{cbox } a \ b)$ 
if  $\mathcal{F} \subseteq \mathcal{D}$  finite  $\mathcal{F}$  for  $\mathcal{F}$ 

```

```

proof -
  have measure lebesgue ( $\bigcup \mathcal{F}$ )  $\leq$  measure lebesgue ( $\bigcup \mathcal{D}$ )
  proof (rule measure_mono_fmeasurable)
    show  $\bigcup \mathcal{F} \subseteq \bigcup \mathcal{D}$ 
    by (simp add: Sup_subset_mono  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$ )
    show  $\bigcup \mathcal{F} \in \text{sets lebesgue}$ 
    by (meson Dlm fmeasurableD sets.finite_Union subset_eq that)
    show  $\bigcup \mathcal{D} \in \text{lmeasurable}$ 
    by fact
  qed
  also have  $\dots \leq$  measure lebesgue (cbox a b)
  proof (rule measure_mono_fmeasurable)
    show  $\bigcup \mathcal{D} \in \text{sets lebesgue}$ 
    by (simp add:  $\langle \bigcup \mathcal{D} \in \text{lmeasurable} \rangle$  fmeasurableD)
  qed (auto simp: D(1))
  finally show ?thesis
  by simp
qed
qed auto
then show ?thesis
using that by auto
qed
obtain tag where tag_in_E:  $\bigwedge D. D \in \mathcal{D} \implies \text{tag } D \in ?E$ 
and tag_in_self:  $\bigwedge D. D \in \mathcal{D} \implies \text{tag } D \in D$ 
and tag_sub:  $\bigwedge D. D \in \mathcal{D} \implies D \subseteq \gamma'(\text{tag } D)$ 
using Dcovered by simp metis
then have sub_ball_tag:  $\bigwedge D. D \in \mathcal{D} \implies D \subseteq \text{ball}(\text{tag } D) (\eta(\text{tag } D))$ 
by (simp add:  $\gamma'_\text{def}$ )
define  $\Phi$  where  $\Phi \equiv \lambda D. \text{BOX}(\eta(\text{tag } D))(\text{tag } D)$ 
define  $\Phi 2$  where  $\Phi 2 \equiv \lambda D. \text{BOX2}(\eta(\text{tag } D))(\text{tag } D)$ 
obtain  $\mathcal{C}$  where  $\mathcal{C} \subseteq \Phi 2 \text{ ' } \mathcal{F}$  pairwise disjoint  $\mathcal{C}$ 
measure lebesgue ( $\bigcup \mathcal{C}$ )  $\geq$  measure lebesgue ( $\bigcup (\Phi 2 \text{ ' } \mathcal{F})$ ) /  $3 \wedge (\text{DIM}('a))$ 
proof (rule Austin_Lemma)
  show finite ( $\Phi 2 \text{ ' } \mathcal{F}$ )
  using  $\langle \text{finite } \mathcal{F} \rangle$  by blast
  have  $\exists k \ a \ b. \Phi 2 \ D = \text{cbox } a \ b \wedge (\forall i \in \text{Basis}. b \cdot i - a \cdot i = k)$  if  $D \in \mathcal{F}$ 
for D
  apply (rule_tac x=2 *  $\eta(\text{tag } D)$  in exI)
  apply (rule_tac x=tag D -  $\eta(\text{tag } D)$  *R One in exI)
  apply (rule_tac x=tag D +  $\eta(\text{tag } D)$  *R One in exI)
  using that
  apply (auto simp:  $\Phi 2_\text{def}$  BOX2_def algebra_simps)
  done
  then show  $\bigwedge D. D \in \Phi 2 \text{ ' } \mathcal{F} \implies \exists k \ a \ b. D = \text{cbox } a \ b \wedge (\forall i \in \text{Basis}. b \cdot i - a \cdot i = k)$ 
  by blast
qed auto
then obtain  $\mathcal{G}$  where  $\mathcal{G} \subseteq \mathcal{F}$  and disj: pairwise disjoint ( $\Phi 2 \text{ ' } \mathcal{G}$ )
and measure lebesgue ( $\bigcup (\Phi 2 \text{ ' } \mathcal{G})$ )  $\geq$  measure lebesgue ( $\bigcup (\Phi 2 \text{ ' } \mathcal{F})$ ) / 3

```

```

 $\wedge (DIM('a))$ 
  unfolding  $\Phi2\_def$  subset_image_iff
  by (meson empty_subsetI equals0D pairwise_imageI)
moreover
have measure_lebesgue  $(\bigcup (\Phi2 \text{ ' } \mathcal{G})) * 3 \wedge DIM('a) \leq e/2$ 
proof -
  have finite  $\mathcal{G}$ 
  using  $\langle \text{finite } \mathcal{F} \rangle \langle \mathcal{G} \subseteq \mathcal{F} \rangle$  infinite_super by blast
  have BOX2_m:  $\bigwedge x. x \in \text{tag } \text{' } \mathcal{G} \implies \text{BOX2 } (\eta x) x \in \text{lmeasurable}$ 
  by (auto simp: BOX2_def)
  have BOX_m:  $\bigwedge x. x \in \text{tag } \text{' } \mathcal{G} \implies \text{BOX } (\eta x) x \in \text{lmeasurable}$ 
  by (auto simp: BOX_def)
  have BOX_sub:  $\text{BOX } (\eta x) x \subseteq \text{BOX2 } (\eta x) x$  for  $x$ 
  by (auto simp: BOX_def BOX2_def subset_box algebra_simps)
  have DISJ2:  $\text{BOX2 } (\eta (\text{tag } X)) (\text{tag } X) \cap \text{BOX2 } (\eta (\text{tag } Y)) (\text{tag } Y)$ 
= {
  if  $X \in \mathcal{G} \ Y \in \mathcal{G} \ \text{tag } X \neq \text{tag } Y$  for  $X \ Y$ 
  proof -
    obtain  $i$  where  $i: i \in \text{Basis tag } X \cdot i \neq \text{tag } Y \cdot i$ 
    using  $\langle \text{tag } X \neq \text{tag } Y \rangle$  by (auto simp: euclidean_eq_iff [of tag X])
    have XY:  $X \in \mathcal{D} \ Y \in \mathcal{D}$ 
    using  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle \langle \mathcal{G} \subseteq \mathcal{F} \rangle$  that by auto
    then have  $0 \leq \eta (\text{tag } X) \ 0 \leq \eta (\text{tag } Y)$ 
    by (meson h0 le_cases not_le tag_in_E)+
    with XY  $i$  have  $\text{BOX2 } (\eta (\text{tag } X)) (\text{tag } X) \neq \text{BOX2 } (\eta (\text{tag } Y)) (\text{tag } Y)$ 
Y)
    unfolding eq_iff
    by (fastforce simp add: BOX2_def subset_box algebra_simps)
    then show ?thesis
    using disj that by (auto simp: pairwise_def disjnt_def  $\Phi2\_def$ )
  qed
  then have BOX2_disj: pairwise  $(\lambda x y. \text{negligible } (\text{BOX2 } (\eta x) x \cap \text{BOX2 } (\eta y) y)) (\text{tag } \text{' } \mathcal{G})$ 
  by (simp add: pairwise_imageI)
  then have BOX_disj: pairwise  $(\lambda x y. \text{negligible } (\text{BOX } (\eta x) x \cap \text{BOX } (\eta y) y)) (\text{tag } \text{' } \mathcal{G})$ 
  proof (rule pairwise_mono)
    show negligible  $(\text{BOX } (\eta x) x \cap \text{BOX } (\eta y) y)$ 
    if negligible  $(\text{BOX2 } (\eta x) x \cap \text{BOX2 } (\eta y) y)$  for  $x \ y$ 
    by (metis (no_types, opaque_lifting) that Int_mono negligible_subset BOX_sub)
  qed auto
  have eq:  $\bigwedge \text{box}. (\lambda D. \text{box } (\eta (\text{tag } D)) (\text{tag } D)) \text{' } \mathcal{G} = (\lambda t. \text{box } (\eta t) t) \text{' } \mathcal{G}$ 
  by (simp add: image_comp)
  have measure_lebesgue  $(\text{BOX2 } (\eta t) t) * 3 \wedge DIM('a)$ 
  = measure_lebesgue  $(\text{BOX } (\eta t) t) * (2*3) \wedge DIM('a)$ 
  if  $t \in \text{tag } \text{' } \mathcal{G}$  for  $t$ 

```



```

proof –
  have content (cbox ( $t - \eta \, t *_{\mathbb{R}} \text{One}$ ) ( $t + \eta \, t *_{\mathbb{R}} \text{One}$ ))
    = content (cbox  $t$  ( $t + \eta \, t *_{\mathbb{R}} \text{One}$ )) *  $2^{\wedge \text{DIM}('a)}$ )
using that by (simp add: algebra_simps content_cbox_if_box_eq_empty)
  then show ?thesis
    by (simp add: BOX2_def BOX_def flip: power_mult_distrib)
qed
  then have measure lebesgue ( $\bigcup (\Phi 2 \, ' \mathcal{G})$ ) *  $3^{\wedge \text{DIM}('a)}$  = measure
lebesgue ( $\bigcup (\Phi \, ' \mathcal{G})$ ) *  $6^{\wedge \text{DIM}('a)}$ 
  unfolding Phi_def Phi2_def eq
  by (simp add: measure_negligible_finite_Union_image
     $\langle \text{finite } \mathcal{G} \rangle$  BOX2_m BOX_m BOX2_disj BOX_disj sum_distrib_right
    del: UN_simps)
  also have ...  $\leq e/2$ 
proof –
  have  $\mu * \text{measure } \text{lebesgue} (\bigcup D \in \mathcal{G}. \Phi \, D) \leq \mu * (\sum D \in \Phi \, \mathcal{G}. \text{measure}$ 
lebesgue  $D)$ 
  using  $\langle \mu > 0 \rangle$   $\langle \text{finite } \mathcal{G} \rangle$  by (force simp: BOX_m Phi_def fmeasurableD
intro: measure_Union_le)
  also have ... =  $(\sum D \in \Phi \, \mathcal{G}. \text{measure } \text{lebesgue } D * \mu)$ 
  by (metis mult.commute sum_distrib_right)
  also have ...  $\leq (\sum (x, K) \in (\lambda D. (\text{tag } D, \Phi \, D)) \, ' \mathcal{G}. \text{norm } (\text{content}$ 
 $K *_{\mathbb{R}} f \, x - \text{integral } K \, f))$ 
  proof (rule sum_le_included; clarify?)
    fix  $D$ 
    assume  $D \in \mathcal{G}$ 
    then have  $\eta (\text{tag } D) > 0$ 
    using  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$   $\langle \mathcal{G} \subseteq \mathcal{F} \rangle$  h0 tag_in_E by auto
    then have  $m_{\Phi} \text{measure } \text{lebesgue} (\Phi \, D) > 0$ 
    by (simp add: Phi_def BOX_def algebra_simps)
    have  $\mu \leq \text{norm } (i (\eta (\text{tag } D))) (\text{tag } D) - f (\text{tag } D)$ 
    using  $\mu_{\text{le}} \langle D \in \mathcal{G} \rangle$   $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$   $\langle \mathcal{G} \subseteq \mathcal{F} \rangle$  tag_in_E by auto
    also have ... =  $\text{norm } ((\text{content } (\Phi \, D) *_{\mathbb{R}} f (\text{tag } D) - \text{integral } (\Phi \, D)$ 
 $f) /_{\mathbb{R}} \text{measure } \text{lebesgue} (\Phi \, D))$ 
    using  $m_{\Phi}$ 
    unfolding i_def Phi_def BOX_def
  by (simp add: algebra_simps content_cbox_plus norm_minus_commute)
  finally have  $\text{measure } \text{lebesgue} (\Phi \, D) * \mu \leq \text{norm } (\text{content } (\Phi \, D) *_{\mathbb{R}}$ 
 $f (\text{tag } D) - \text{integral } (\Phi \, D) \, f)$ 
  using  $m_{\Phi}$  by simp (simp add: field_simps)
  then show  $\exists y \in (\lambda D. (\text{tag } D, \Phi \, D)) \, ' \mathcal{G}.$ 
 $\text{snd } y = \Phi \, D \wedge \text{measure } \text{lebesgue} (\Phi \, D) * \mu \leq (\text{case } y \text{ of } (x, k)$ 
 $\Rightarrow \text{norm } (\text{content } k *_{\mathbb{R}} f \, x - \text{integral } k \, f))$ 
  using  $\langle D \in \mathcal{G} \rangle$  by auto
qed (use  $\langle \text{finite } \mathcal{G} \rangle$  in auto)
also have ...  $< ?ee$ 
proof (rule  $\gamma$ )
  show  $(\lambda D. (\text{tag } D, \Phi \, D)) \, ' \mathcal{G}$  tagged_partial_division_of cbox ( $a -$ 
 $\text{One}$ ) ( $b + \text{One}$ )

```

```

    unfolding tagged_partial_division_of_def
  proof (intro conjI allI impI ; clarify ?)
    show tag D ∈ Φ D
    if D ∈ G for D
    using that ⟨F ⊆ D⟩ ⟨G ⊆ F⟩ h0 tag_in_E
      by (auto simp: Φ_def BOX_def mem_box algebra_simps
eucl_less_le_not_le in_mono)
    show y ∈ cbox (a - One) (b + One) if D ∈ G y ∈ Φ D for D y
      using that BOX_cbox Φ_def ⟨F ⊆ D⟩ ⟨G ⊆ F⟩ tag_in_E by
blast

    show tag D = tag E ∧ Φ D = Φ E
      if D ∈ G E ∈ G and ne: interior (Φ D) ∩ interior (Φ E) ≠ {}
for D E
    proof -
      have BOX2 (η (tag D)) (tag D) ∩ BOX2 (η (tag E)) (tag E) =
{} ∨ tag E = tag D
      using DISJ2 ⟨D ∈ G⟩ ⟨E ∈ G⟩ by force
      then have BOX (η (tag D)) (tag D) ∩ BOX (η (tag E)) (tag E)
= {} ∨ tag E = tag D
      using BOX_sub by blast
      then show tag D = tag E ∧ Φ D = Φ E
      by (metis Φ_def interior_Int interior_empty ne)
    qed
    qed (use ⟨finite G⟩ Φ_def BOX_def in auto)
    show γ fine (λD. (tag D, Φ D)) ‘G
      unfolding fine_def Φ_def using BOX_γ ⟨F ⊆ D⟩ ⟨G ⊆ F⟩
tag_in_E by blast
    qed
    finally show ?thesis
      using ⟨μ > 0⟩ by (auto simp: field_split_simps)
    qed
    finally show ?thesis .
    qed
  moreover
  have measure_lebesgue (⋃ F) ≤ measure_lebesgue (⋃ (Φ2‘F))
  proof (rule measure_mono_fmeasurable)
    have D ⊆ ball (tag D) (η(tag D)) if D ∈ F for D
    using ⟨F ⊆ D⟩ sub_ball_tag that by blast
    moreover have ball (tag D) (η(tag D)) ⊆ BOX2 (η (tag D)) (tag D) if
D ∈ F for D
    proof (clarsimp simp: Φ2_def BOX2_def mem_box algebra_simps
dist_norm)
      fix x and i::'a
      assume norm (tag D - x) < η (tag D) and i ∈ Basis
      then have |tag D • i - x • i| ≤ η (tag D)
      by (metis eucl_less_le_not_le inner_commute inner_diff_right
norm_bound_Basis_le)
      then show tag D • i ≤ x • i + η (tag D) ∧ x • i ≤ η (tag D) + tag D
• i

```

```

      by (simp add: abs_diff_le_iff)
    qed
  ultimately show  $\bigcup \mathcal{F} \subseteq \bigcup (\Phi 2' \mathcal{F})$ 
    by (force simp:  $\Phi 2\_def$ )
  show  $\bigcup \mathcal{F} \in \text{sets lebesgue}$ 
    using  $\langle \text{finite } \mathcal{F} \rangle \langle \mathcal{D} \subseteq \text{sets lebesgue} \rangle \langle \mathcal{F} \subseteq \mathcal{D} \rangle$  by blast
  show  $\bigcup (\Phi 2' \mathcal{F}) \in \text{lmeasurable}$ 
    unfolding  $\Phi 2\_def$   $BOX2\_def$  using  $\langle \text{finite } \mathcal{F} \rangle$  by blast
  qed
  ultimately
  have  $\text{measure lebesgue } (\bigcup \mathcal{F}) \leq e/2$ 
    by (auto simp: field_split_simps)
  then show  $\text{measure lebesgue } (\bigcup \mathcal{D}) \leq e$ 
    using  $\mathcal{F}$  by linarith
  qed
  qed
  qed
  then have  $\bigwedge j. \text{negligible } \{x. \Psi x (\text{inverse}(\text{real } j + 1))\}$ 
    using negligible_on_intervals
    by (metis (full_types) inverse_positive_iff_positive le_add_same_cancel1
linorder_not_le nat_le_real_less not_add_less1 of_nat_0)
  then have negligible ?M
    by auto
  moreover have  $?N \subseteq ?M$ 
  proof (clarsimp simp: dist_norm)
    fix y e
    assume  $0 < e$ 
    and ye [rule_format]:  $\Psi y e$ 
    then obtain k where  $0 < k$   $\text{inverse}(\text{real } k + 1) < e$ 
    by (metis One_nat_def add.commute less_add_same_cancel2 less_imp_inverse_less
less_trans neq0_conv of_nat_1 of_nat_Suc reals_Archimedean zero_less_one)
    with ye show  $\exists n. \Psi y (\text{inverse}(\text{real } n + 1))$ 
    apply (rule_tac x=k in exI)
    unfolding  $\Psi\_def$ 
    by (force intro: less_le_trans)
  qed
  ultimately show negligible ?N
    by (blast intro: negligible_subset)
  show  $\neg \Psi x e$  if  $x \notin ?N \wedge 0 < e$  for  $x e$ 
    using that by blast
  qed
  with that show ?thesis
    unfolding i_def BOX_def  $\Psi\_def$  by (fastforce simp add: not_le)
  qed

```

6.29.3 HOL Light measurability

definition $\text{measurable\_on} :: ('a::\text{euclidean\_space} \Rightarrow 'b::\text{real\_normed\_vector}) \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$

```

(infixr measurable'_on 46)
where f measurable_on S ≡
  ∃ N g. negligible N ∧
    (∀ n. continuous_on UNIV (g n)) ∧
    (∀ x. x ∉ N ⟶ (λn. g n x) ⟶ (if x ∈ S then f x else 0))

```

```

lemma measurable_on_UNIV:
  (λx. if x ∈ S then f x else 0) measurable_on UNIV ⟷ f measurable_on S
by (auto simp: measurable_on_def)

```

```

lemma measurable_on_spike_set:
  assumes f: f measurable_on S and neg: negligible ((S - T) ∪ (T - S))
  shows f measurable_on T
proof -
  obtain N and F
  where N: negligible N
    and conF: ⋀n. continuous_on UNIV (F n)
    and tendsF: ⋀x. x ∉ N ⟹ (λn. F n x) ⟶ (if x ∈ S then f x else 0)
  using f by (auto simp: measurable_on_def)
show ?thesis
  unfolding measurable_on_def
proof (intro exI conjI allI impI)
  show continuous_on UNIV (λx. F n x) for n
    by (intro conF continuous_intros)
  show negligible (N ∪ (S - T) ∪ (T - S))
    by (metis (full_types) N neg negligible_Un_eq)
  show (λn. F n x) ⟶ (if x ∈ T then f x else 0)
    if x ∉ (N ∪ (S - T) ∪ (T - S)) for x
    using that tendsF [of x] by auto
qed
qed

```

Various common equivalent forms of function measurability.

```

lemma measurable_on_0 [simp]: (λx. 0) measurable_on S
  unfolding measurable_on_def
proof (intro exI conjI allI impI)
  show (λn. 0) ⟶ (if x ∈ S then 0::'b else 0) for x
    by force
qed auto

```

```

lemma measurable_on_scaleR_const:
  assumes f: f measurable_on S
  shows (λx. c *R f x) measurable_on S
proof -
  obtain NF and F
  where NF: negligible NF
    and conF: ⋀n. continuous_on UNIV (F n)
    and tendsF: ⋀x. x ∉ NF ⟹ (λn. F n x) ⟶ (if x ∈ S then f x else 0)
  using f by (auto simp: measurable_on_def)

```

```

show ?thesis
  unfolding measurable_on_def
proof (intro exI conjI allI impI)
  show continuous_on UNIV  $(\lambda x. c *_{\mathbb{R}} F \, n \, x)$  for  $n$ 
    by (intro conF continuous_intros)
  show  $(\lambda n. c *_{\mathbb{R}} F \, n \, x) \longrightarrow (\text{if } x \in S \text{ then } c *_{\mathbb{R}} f \, x \text{ else } 0)$ 
    if  $x \notin NF$  for  $x$ 
    using tendsto_scaleR [OF tendsto_const tendsF, of  $x$ ] that by auto
  qed (auto simp: NF)
qed

```

```

lemma measurable_on_cmul:
  fixes  $c :: \text{real}$ 
  assumes  $f$  measurable_on  $S$ 
  shows  $(\lambda x. c * f \, x)$  measurable_on  $S$ 
  using measurable_on_scaleR_const [OF assms] by simp

```

```

lemma measurable_on_cdivide:
  fixes  $c :: \text{real}$ 
  assumes  $f$  measurable_on  $S$ 
  shows  $(\lambda x. f \, x / c)$  measurable_on  $S$ 
proof (cases  $c=0$ )
  case False
  then show ?thesis
    using measurable_on_cmul [of  $f \, S \, 1/c$ ]
    by (simp add: assms)
qed auto

```

```

lemma measurable_on_minus:
   $f$  measurable_on  $S \implies (\lambda x. -(f \, x))$  measurable_on  $S$ 
  using measurable_on_scaleR_const [of  $f \, S \, -1$ ] by auto

```

```

lemma continuous_imp_measurable_on:
  continuous_on UNIV  $f \implies f$  measurable_on UNIV
  unfolding measurable_on_def
  apply (rule_tac  $x=\{\}$  in exI)
  apply (rule_tac  $x=\lambda n. f$  in exI, auto)
  done

```

```

proposition integrable_subintervals_imp_measurable:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes  $\bigwedge a \, b. f$  integrable_on cbox  $a \, b$ 
  shows  $f$  measurable_on UNIV
proof –
  define BOX where  $BOX \equiv \lambda h. \lambda x::'a. \text{cbox } x \, (x + h *_{\mathbb{R}} \text{One})$ 
  define  $i$  where  $i \equiv \lambda h \, x. \text{integral } (BOX \, h \, x) \, f /_{\mathbb{R}} h \wedge DIM('a)$ 

```

```

obtain  $N$  where negligible  $N$ 
and  $k$ :  $\bigwedge x \in e. \llbracket x \notin N; 0 < e \rrbracket$ 
 $\implies \exists d > 0. \forall h. 0 < h \wedge h < d \implies$ 
 $\text{norm}(\text{integral}(\text{cbox } x (x + h *_R \text{One})) f /_R h \wedge \text{DIM}('a) - f x)$ 
 $< e$ 
using integrable_ccontinuous_explicit assms by blast
show ?thesis
unfolding measurable_on_def
proof (intro exI conjI allI impI)
show continuous_on UNIV  $((\lambda n x. i(\text{inverse}(\text{Suc } n)) x) n)$  for  $n$ 
proof (clarsimp simp: continuous_on_iff)
show  $\exists d > 0. \forall x'. \text{dist } x' x < d \implies \text{dist}(i(\text{inverse}(1 + \text{real } n)) x') (i$ 
 $(\text{inverse}(1 + \text{real } n)) x) < e$ 
if  $0 < e$ 
for  $x \in e$ 
proof –
let  $?e = e / (1 + \text{real } n) \wedge \text{DIM}('a)$ 
have  $?e > 0$ 
using  $\langle e > 0 \rangle$  by auto
moreover have  $x \in \text{cbox } (x - 2 *_R \text{One}) (x + 2 *_R \text{One})$ 
by (simp add: mem_box inner_diff_left inner_left_distrib)
moreover have  $x + \text{One} /_R \text{real}(\text{Suc } n) \in \text{cbox } (x - 2 *_R \text{One}) (x + 2$ 
 $*_R \text{One})$ 
by (auto simp: mem_box inner_diff_left inner_left_distrib field_simps)
ultimately obtain  $\delta > 0$ 
and  $\delta$ :  $\bigwedge c' d'. \llbracket c' \in \text{cbox } (x - 2 *_R \text{One}) (x + 2 *_R \text{One});$ 
 $d' \in \text{cbox } (x - 2 *_R \text{One}) (x + 2 *_R \text{One});$ 
 $\text{norm}(c' - x) \leq \delta; \text{norm}(d' - (x + \text{One} /_R \text{Suc } n)) \leq \delta \rrbracket$ 
 $\implies \text{norm}(\text{integral}(\text{cbox } c' d') f - \text{integral}(\text{cbox } x (x + \text{One}$ 
 $/_R \text{Suc } n)) f) < ?e$ 
by (blast intro: indefinite_integral_continuous [of f _ _ x] assms)
show ?thesis
proof (intro exI impI conjI allI)
show  $\min \delta 1 > 0$ 
using  $\langle \delta > 0 \rangle$  by auto
show  $\text{dist}(i(\text{inverse}(1 + \text{real } n)) y) (i(\text{inverse}(1 + \text{real } n)) x) < e$ 
if  $\text{dist } y x < \min \delta 1$  for  $y$ 
proof –
have  $\text{no}: \text{norm}(y - x) < 1$ 
using that by (auto simp: dist_norm)
have  $\text{le1}: \text{inverse}(1 + \text{real } n) \leq 1$ 
by (auto simp: field_split_simps)
have  $\text{norm}(\text{integral}(\text{cbox } y (y + \text{One} /_R \text{real}(\text{Suc } n))) f$ 
 $- \text{integral}(\text{cbox } x (x + \text{One} /_R \text{real}(\text{Suc } n))) f)$ 
 $< e / (1 + \text{real } n) \wedge \text{DIM}('a)$ 
proof (rule  $\delta$ )
show  $y \in \text{cbox } (x - 2 *_R \text{One}) (x + 2 *_R \text{One})$ 
using no by (auto simp: mem_box algebra_simps dest: Basis_le_norm
 $[of\_ y - x])$ 

```

```

    show  $y + \text{One} /_R \text{real} (\text{Suc } n) \in \text{cbox } (x - 2 *_R \text{One}) (x + 2 *_R \text{One})$ 
  proof (simp add: dist_norm mem_box algebra_simps, intro ballI conjI)
    fix  $i :: 'a$ 
    assume  $i \in \text{Basis}$ 
    then have  $1: |y \cdot i - x \cdot i| < 1$ 
    by (metis inner_commute inner_diff_right no_norm_bound_Basis_lt)
    moreover have  $\dots < (2 + \text{inverse } (1 + \text{real } n)) \ 1 \leq 2 - \text{inverse}$ 
    (1 + real n)
    by (auto simp: field_simps)
    ultimately show  $x \cdot i \leq y \cdot i + (2 + \text{inverse } (1 + \text{real } n))$ 
     $y \cdot i + \text{inverse } (1 + \text{real } n) \leq x \cdot i + 2$ 
    by linarith+
  qed
  show  $\text{norm } (y - x) \leq \delta \text{ norm } (y + \text{One} /_R \text{real} (\text{Suc } n) - (x + \text{One} /_R \text{real} (\text{Suc } n))) \leq \delta$ 
  using that by (auto simp: dist_norm)
  qed
  then show ?thesis
  using that by (simp add: dist_norm i_def BOX_def flip: scaleR_diff_right)
  (simp add: field_simps)
  qed
  qed
  qed
  show negligible N
  by (simp add: negligible N)
  show  $(\lambda n. i (\text{inverse } (\text{Suc } n)) x) \longrightarrow (\text{if } x \in \text{UNIV} \text{ then } f x \text{ else } 0)$ 
  if  $x \notin N$  for  $x$ 
  unfolding lim_sequentially
  proof clarsimp
    show  $\exists \text{no. } \forall n \geq \text{no. } \text{dist } (i (\text{inverse } (1 + \text{real } n)) x) (f x) < e$ 
    if  $0 < e$  for  $e$ 
    proof -
      obtain  $d$  where  $d > 0$ 
      and  $d: \bigwedge h. [0 < h; h < d] \implies$ 
       $\text{norm } (\text{integral } (\text{cbox } x (x + h *_R \text{One})) f /_R h \wedge \text{DIM}('a) - f x) < e$ 
      using  $k [of x e] \langle x \notin N \rangle \langle 0 < e \rangle$  by blast
      then obtain  $M$  where  $M: M \neq 0 \ 0 < \text{inverse } (\text{real } M) \text{ inverse } (\text{real } M)$ 
      < d
      using real_arch_invD by auto
    show ?thesis
  proof (intro exI allI impI)
    show  $\text{dist } (i (\text{inverse } (1 + \text{real } n)) x) (f x) < e$ 
    if  $M \leq n$  for  $n$ 
    proof -
      have  $0 < \text{inverse } (1 + \text{real } n) \text{ inverse } (1 + \text{real } n) \leq \text{inverse } M$ 
      using that  $\langle M \neq 0 \rangle$  by auto
    show ?thesis
    using that M

```

```

      apply (simp add: i_def BOX_def dist_norm)
      apply (blast intro: le_less_trans * d)
    done
  qed
  qed
  qed
  qed
  qed
  qed

```

6.29.4 Composing continuous and measurable functions; a few variants

```

lemma measurable_on_compose_continuous:
  assumes f: f measurable_on UNIV and g: continuous_on UNIV g
  shows (g ∘ f) measurable_on UNIV
proof -
  obtain N and F
  where negligible N
    and conF:  $\bigwedge n. \text{continuous\_on UNIV } (F\ n)$ 
    and tendsF:  $\bigwedge x. x \notin N \implies (\lambda n. F\ n\ x) \longrightarrow f\ x$ 
  using f by (auto simp: measurable_on_def)
  show ?thesis
    unfolding measurable_on_def
  proof (intro exI conjI allI impI)
    show negligible N
      by fact
    show continuous_on UNIV (g ∘ (F n)) for n
      using conF continuous_on_compose continuous_on_subset g by blast
    show  $(\lambda n. (g \circ F\ n)\ x) \longrightarrow (if\ x \in UNIV\ then\ (g \circ f)\ x\ else\ 0)$ 
      if  $x \notin N$  for  $x :: 'a$ 
      using that g tendsF by (auto simp: continuous_on_def intro: tendsto_compose)
    qed
  qed

```

```

lemma measurable_on_compose_continuous_0:
  assumes f: f measurable_on S and g: continuous_on UNIV g and g 0 = 0
  shows (g ∘ f) measurable_on S
proof -
  have f':  $(\lambda x. if\ x \in S\ then\ f\ x\ else\ 0)$  measurable_on UNIV
    using f measurable_on_UNIV by blast
  show ?thesis
    using measurable_on_compose_continuous [OF f' g]
    by (simp add: measurable_on_UNIV o_def if_distrib ⟨g 0 = 0⟩ cong: if_cong)
  qed

```

```

lemma measurable_on_compose_continuous_box:
  assumes fm: f measurable_on UNIV and fab:  $\bigwedge x. f\ x \in box\ a\ b$ 

```



```

    and contg: continuous_on (box a b) g
  shows (g ∘ f) measurable_on UNIV
proof -
  have ∃γ. (∀ n. continuous_on UNIV (γ n)) ∧ (∀ x. x ∉ N ⟶ (λ n. γ n x)
    ⟶ g (f x))
  if negligible N
    and conth [rule_format]: ∀ n. continuous_on UNIV (λ x. h n x)
    and tends [rule_format]: ∀ x. x ∉ N ⟶ (λ n. h n x) ⟶ f x
  for N and h :: nat ⇒ 'a ⇒ 'b
proof -
  define ϑ where ϑ ≡ λ n x. (∑ i ∈ Basis. (max (a · i + (b · i - a · i) / real (n + 2))
    (min ((h n x) · i
      (b · i - (b · i - a · i) / real (n + 2))))) *R i)
  have aibi: ⋀ i. i ∈ Basis ⟹ a · i < b · i
  using box_ne_empty(2) fab by auto
  then have *: ⋀ i n. i ∈ Basis ⟹ a · i + real n * (a · i) < b · i + real n * (b
    · i)
  by (meson add_mono_thms_linordered_field(3) less_eq_real_def mult_left_mono
    of_nat_0_le_iff)
  show ?thesis
  proof (intro exI conjI allI impI)
    show continuous_on UNIV (g ∘ (ϑ n)) for n :: nat
    unfolding ϑ_def
    apply (intro continuous_on_compose2 [OF contg] continuous_intros conth)
    apply (auto simp: aibi * mem_box less_max_iff_disj min_less_iff_disj
      field_split_simps)
    done
    show (λ n. (g ∘ ϑ n) x) ⟶ g (f x)
    if x ∉ N for x
    unfolding o_def
  proof (rule isCont_tendsto_compose [where g=g])
    show isCont g (f x)
    using contg fab continuous_on_eq_continuous_at by blast
    have (λ n. ϑ n x) ⟶ (∑ i ∈ Basis. max (a · i) (min (f x · i) (b · i)) *R
      i)
    unfolding ϑ_def
  proof (intro tendsto_intros ⟨x ∉ N⟩ tends)
    fix i :: 'b
    assume i ∈ Basis
    have a: (λ n. a · i + (b · i - a · i) / real n) ⟶ a · i + 0
    by (intro tendsto_add lim_const_over_n tendsto_const)
    show (λ n. a · i + (b · i - a · i) / real (n + 2)) ⟶ a · i
    using LIMSEQ_ignore_initial_segment [where k=2, OF a] by simp
    have b: (λ n. b · i - (b · i - a · i) / (real n)) ⟶ b · i - 0
    by (intro tendsto_diff lim_const_over_n tendsto_const)
    show (λ n. b · i - (b · i - a · i) / real (n + 2)) ⟶ b · i
    using LIMSEQ_ignore_initial_segment [where k=2, OF b] by simp
  qed
  also have (∑ i ∈ Basis. max (a · i) (min (f x · i) (b · i)) *R i) = (∑ i ∈ Basis.

```

```

(f x · i) *R i)
  using fab by (auto simp add: mem_box intro: sum.cong)
  also have ... = f x
    using euclidean_representation by blast
  finally show (λn. ∅ n x) ⟶ f x .
qed
qed
qed
then show ?thesis
  using fm by (auto simp: measurable_on_def)
qed

```

lemma *measurable_on_Pair*:

assumes *f*: *f* measurable_on *S* **and** *g*: *g* measurable_on *S*
shows (λ*x*. (*f* *x*, *g* *x*)) measurable_on *S*

proof –

obtain *NF* **and** *F*

where *NF*: negligible *NF*

and *conF*: $\bigwedge n. \text{continuous\_on } UNIV (F\ n)$

and *tendsF*: $\bigwedge x. x \notin NF \implies (\lambda n. F\ n\ x) \longrightarrow (\text{if } x \in S \text{ then } f\ x \text{ else } 0)$

using *f* **by** (auto simp: measurable_on_def)

obtain *NG* **and** *G*

where *NG*: negligible *NG*

and *conG*: $\bigwedge n. \text{continuous\_on } UNIV (G\ n)$

and *tendsG*: $\bigwedge x. x \notin NG \implies (\lambda n. G\ n\ x) \longrightarrow (\text{if } x \in S \text{ then } g\ x \text{ else } 0)$

using *g* **by** (auto simp: measurable_on_def)

show ?thesis

unfolding measurable_on_def

proof (intro exI conjI allI impI)

show negligible (*NF* ∪ *NG*)

by (simp add: *NF* *NG*)

show continuous_on *UNIV* (λ*x*. (*F* *n* *x*, *G* *n* *x*)) **for** *n*

using *conF* *conG* continuous_on_Pair **by** blast

show (λ*n*. (*F* *n* *x*, *G* *n* *x*)) ⟶ (if *x* ∈ *S* then (*f* *x*, *g* *x*) else 0)

if *x* ∉ *NF* ∪ *NG* **for** *x*

using tendssto_Pair [OF *tendsF* *tendsG*, of *x* *x*] **that** **unfolding** zero_prod_def

by (simp add: split: if_split_asm)

qed

qed

lemma *measurable_on_combine*:

assumes *f*: *f* measurable_on *S* **and** *g*: *g* measurable_on *S*

and *h*: continuous_on *UNIV* (λ*x*. *h* (fst *x*) (snd *x*)) **and** *h* 0 0 = 0

shows (λ*x*. *h* (*f* *x*) (*g* *x*)) measurable_on *S*

proof –

have *: (λ*x*. *h* (*f* *x*) (*g* *x*)) = (λ*x*. *h* (fst *x*) (snd *x*)) ∘ (λ*x*. (*f* *x*, *g* *x*))

by auto

show ?thesis

unfolding * **by** (auto simp: measurable_on_compose_continuous_0 measur-

able_on_Pair assms)
qed

lemma measurable_on_add:
 assumes $f: f \text{ measurable\_on } S$ and $g: g \text{ measurable\_on } S$
 shows $(\lambda x. f\ x + g\ x) \text{ measurable\_on } S$
 by (intro continuous_intros measurable_on_combine [OF assms]) auto

lemma measurable_on_diff:
 assumes $f: f \text{ measurable\_on } S$ and $g: g \text{ measurable\_on } S$
 shows $(\lambda x. f\ x - g\ x) \text{ measurable\_on } S$
 by (intro continuous_intros measurable_on_combine [OF assms]) auto

lemma measurable_on_scaleR:
 assumes $f: f \text{ measurable\_on } S$ and $g: g \text{ measurable\_on } S$
 shows $(\lambda x. f\ x *_{\mathbb{R}} g\ x) \text{ measurable\_on } S$
 by (intro continuous_intros measurable_on_combine [OF assms]) auto

lemma measurable_on_sum:
 assumes finite I $\bigwedge i. i \in I \implies f\ i \text{ measurable\_on } S$
 shows $(\lambda x. \text{sum } (\lambda i. f\ i\ x) I) \text{ measurable\_on } S$
 using assms by (induction I) (auto simp: measurable_on_add)

lemma measurable_on_spike:
 assumes $f: f \text{ measurable\_on } T$ and negligible S and $gf: \bigwedge x. x \in T - S \implies g\ x = f\ x$
 shows $g \text{ measurable\_on } T$
proof –
 obtain NF and F
 where $NF: \text{negligible } NF$
 and $\text{con}F: \bigwedge n. \text{continuous\_on } UNIV\ (F\ n)$
 and $\text{tends}F: \bigwedge x. x \notin NF \implies (\lambda n. F\ n\ x) \longrightarrow (\text{if } x \in T \text{ then } f\ x \text{ else } 0)$
 using f by (auto simp: measurable_on_def)
 show ?thesis
 unfolding measurable_on_def
proof (intro exI conjI allI impI)
 show negligible $(NF \cup S)$
 by (simp add: $NF \llbracket \text{negligible } S \rrbracket$)
 show $\bigwedge x. x \notin NF \cup S \implies (\lambda n. F\ n\ x) \longrightarrow (\text{if } x \in T \text{ then } g\ x \text{ else } 0)$
 by (metis (full_types) Diff_iff Un_iff $gf \text{ tends } F$)
 qed (auto simp: $\text{con}F$)
qed

proposition indicator_measurable_on:
 assumes $S \in \text{sets lebesgue}$
 shows $\text{indicat\_real } S \text{ measurable\_on } UNIV$
proof –
 { fix $n::\text{nat}$
 let $?\varepsilon = (1::\text{real}) / (2 * 2^n)$

```

have  $\varepsilon$ :  $?\varepsilon > 0$ 
  by auto
  obtain  $T$  where closed  $T$   $T \subseteq S$   $S - T \in \text{lmeasurable}$  and  $ST$ : emeasure
lebesgue  $(S - T) < ?\varepsilon$ 
    by (meson  $\varepsilon$  assms sets_lebesgue_inner_closed)
  obtain  $U$  where open  $U$   $S \subseteq U$   $(U - S) \in \text{lmeasurable}$  and  $US$ : emeasure
lebesgue  $(U - S) < ?\varepsilon$ 
    by (meson  $\varepsilon$  assms sets_lebesgue_outer_open)
  have  $eq$ :  $-T \cap U = (S - T) \cup (U - S)$ 
    using  $\langle T \subseteq S \rangle \langle S \subseteq U \rangle$  by auto
  have emeasure lebesgue  $((S - T) \cup (U - S)) \leq \text{emeasure } \text{lebesgue } (S - T) +$ 
emeasure lebesgue  $(U - S)$ 
    using  $\langle S - T \in \text{lmeasurable} \rangle \langle U - S \in \text{lmeasurable} \rangle$  emeasure_subadditive
by blast
  also have  $\dots < ?\varepsilon + ?\varepsilon$ 
    using  $ST$   $US$  add_mono_enreal by metis
  finally have  $le$ : emeasure lebesgue  $(-T \cap U) < \text{ennreal } (1 / 2^n)$ 
    by (simp add:  $eq$ )
  have  $1$ : continuous_on  $(T \cup -U)$  (indicat_real  $S$ )
    unfolding indicator_def of_bool_def
  proof (rule continuous_on_cases [OF  $\langle \text{closed } T \rangle$ ])
    show closed  $(-U)$ 
      using  $\langle \text{open } U \rangle$  by blast
    show continuous_on  $T$   $(\lambda x. 1::\text{real})$  continuous_on  $(-U)$   $(\lambda x. 0::\text{real})$ 
      by (auto simp: continuous_on)
    show  $\forall x. x \in T \wedge x \notin S \vee x \in -U \wedge x \in S \longrightarrow (1::\text{real}) = 0$ 
      using  $\langle T \subseteq S \rangle \langle S \subseteq U \rangle$  by auto
    qed
  have  $2$ : closedin (top_of_set  $UNIV$ )  $(T \cup -U)$ 
    using  $\langle \text{closed } T \rangle \langle \text{open } U \rangle$  by auto
  obtain  $g$  where continuous_on  $UNIV$   $g \wedge x. x \in T \cup -U \implies g \ x = \text{indicat\_real } S \ x$ 
norm( $g \ x$ )  $\leq 1$ 
    by (rule Tietze [OF  $1$   $2$ , of  $1$ ]) auto
  with  $le$  have  $\exists g \ E. \text{continuous\_on } UNIV \ g \wedge (\forall x \in -E. g \ x = \text{indicat\_real } S \ x) \wedge$ 
 $(\forall x. \text{norm}(g \ x) \leq 1) \wedge E \in \text{sets } \text{lebesgue} \wedge \text{emeasure } \text{lebesgue}$ 
 $E < \text{ennreal } (1 / 2^n)$ 
    apply (rule_tac  $x=g$  in  $exI$ )
    apply (rule_tac  $x=-T \cap U$  in  $exI$ )
    using  $\langle S - T \in \text{lmeasurable} \rangle \langle U - S \in \text{lmeasurable} \rangle$   $eq$  by auto
  }
then obtain  $g \ E$  where cont:  $\bigwedge n. \text{continuous\_on } UNIV \ (g \ n)$ 
and  $geq$ :  $\bigwedge n \ x. x \in -E \ n \implies g \ n \ x = \text{indicat\_real } S \ x$ 
and  $ng1$ :  $\bigwedge n \ x. \text{norm}(g \ n \ x) \leq 1$ 
and  $Eset$ :  $\bigwedge n. E \ n \in \text{sets } \text{lebesgue}$ 
and  $Em$ :  $\bigwedge n. \text{emeasure } \text{lebesgue } (E \ n) < \text{ennreal } (1 / 2^n)$ 
by metis
have null: limsup  $E \in \text{null\_sets } \text{lebesgue}$ 
proof (rule borel_cantelli_limsup1 [OF  $Eset$ ])

```

```

show emeasure lebesgue (E n) < ∞ for n
  by (metis Em infinity_ennreal_def order.asym top.not_eq_extremum)
show summable (λn. measure lebesgue (E n))
proof (rule summable_comparison_test' [OF summable_geometric, of 1/2 0])
  show norm (measure lebesgue (E n)) ≤ (1/2) ^ n for n
    using Em [of n] by (simp add: measure_def enn2real_leI power_one_over)
qed auto
qed
have tends: (λn. g n x) ⟶ indicat_real S x if x ∉ limsup E for x
proof -
  have ∀_F n in sequentially. x ∈ - E n
    using that by (simp add: mem_limsup_iff not_frequently)
  then show ?thesis
    unfolding tendsto_iff dist_real_def
    by (simp add: eventually_mono geq)
qed
show ?thesis
  unfolding measurable_on_def
proof (intro exI conjI allI impI)
  show negligible (limsup E)
    using negligible_iff_null_sets null by blast
  show continuous_on UNIV (g n) for n
    using cont by blast
qed (use tends in auto)
qed

lemma measurable_on_restrict:
  assumes f: f measurable_on UNIV and S: S ∈ sets lebesgue
  shows (λx. if x ∈ S then f x else 0) measurable_on UNIV
proof -
  have indicat_real S measurable_on UNIV
    by (simp add: S indicator_measurable_on)
  then show ?thesis
    using measurable_on_scaleR [OF _ f, of indicat_real S]
    by (simp add: indicator_scaleR_eq_if)
qed

lemma measurable_on_const_UNIV: (λx. k) measurable_on UNIV
  by (simp add: continuous_imp_measurable_on)

lemma measurable_on_const [simp]: S ∈ sets lebesgue ⟹ (λx. k) measurable_on S
  using measurable_on_UNIV measurable_on_const_UNIV measurable_on_restrict
by blast

lemma simple_function_indicator_representation_real:
  fixes f :: 'a ⇒ real
  assumes f: simple_function M f and x: x ∈ space M and nn: ⋀x. f x ≥ 0
  shows f x = (∑ y ∈ f ' space M. y * indicator (f - {y} ∩ space M) x)

```

proof –
have f' : *simple_function* M ($\text{ennreal} \circ f$)
by (*simp add: f*)
have $*$: $f\ x =$
 enn2real
 $(\sum y \in \text{ennreal} \text{ ' } f \text{ ' } \text{space } M.$
 $y * \text{indicator } ((\text{ennreal} \circ f) - \{y\} \cap \text{space } M) \ x)$
using *arg_cong [OF simple_function_indicator_representation [OF f' x], of*
enn2real, simplified nn o_def] nn
unfolding *o_def image_comp*
by (*metis enn2real_ennreal*)
have $\text{enn2real } (\sum y \in \text{ennreal} \text{ ' } f \text{ ' } \text{space } M. \text{if } \text{ennreal } (f\ x) = y \wedge x \in \text{space } M$
then y else 0)
 $= \text{sum } (\text{enn2real} \circ (\lambda y. \text{if } \text{ennreal } (f\ x) = y \wedge x \in \text{space } M \text{ then } y \text{ else } 0))$
 $(\text{ennreal} \text{ ' } f \text{ ' } \text{space } M)$
by (*rule enn2real_sum*) *auto*
also have $\dots = \text{sum } (\text{enn2real} \circ (\lambda y. \text{if } \text{ennreal } (f\ x) = y \wedge x \in \text{space } M \text{ then}$
 $y \text{ else } 0)) \circ \text{ennreal}$
 $(f \text{ ' } \text{space } M)$
by (*rule sum.reindex*) (*use nn in <auto simp: inj_on_def intro: sum.cong>*)
also have $\dots = (\sum y \in f \text{ ' } \text{space } M. \text{if } f\ x = y \wedge x \in \text{space } M \text{ then } y \text{ else } 0)$
using *nn*
by (*auto simp: inj_on_def intro: sum.cong*)
finally show *?thesis*
by (*subst **) (*simp add: enn2real_sum indicator_def of_bool_def if_distrib*
cong: if_cong)
qed

lemma *simple_function_induct_real*
 $[\text{consumes } 1, \text{case\_names } \text{cong set mult add, induct set: simple\_function}]:$
fixes $u :: 'a \Rightarrow \text{real}$
assumes u : *simple_function* $M\ u$
assumes *cong*: $\bigwedge f\ g. \text{simple\_function } M\ f \Longrightarrow \text{simple\_function } M\ g \Longrightarrow (AE\ x$
in } M. f\ x = g\ x) \Longrightarrow P\ f \Longrightarrow P\ g
assumes *set*: $\bigwedge A. A \in \text{sets } M \Longrightarrow P\ (\text{indicator } A)$
assumes *mult*: $\bigwedge u\ c. P\ u \Longrightarrow P\ (\lambda x. c * u\ x)$
assumes *add*: $\bigwedge u\ v. P\ u \Longrightarrow P\ v \Longrightarrow P\ (\lambda x. u\ x + v\ x)$
and *nn*: $\bigwedge x. u\ x \geq 0$
shows $P\ u$
proof (*rule cong*)
from AE_space **show** $AE\ x \text{ in } M. (\sum y \in u \text{ ' } \text{space } M. y * \text{indicator } (u - \{y\}$
 $\cap \text{space } M) \ x) = u\ x$
proof *eventually_elim*
fix x **assume** $x: x \in \text{space } M$
from *simple_function_indicator_representation_real[OF u x] nn*
show $(\sum y \in u \text{ ' } \text{space } M. y * \text{indicator } (u - \{y\} \cap \text{space } M) \ x) = u\ x$
by *metis*
qed
next

```

from  $u$  have  $\text{finite } (u \text{ ' space } M)$ 
  unfolding  $\text{simple\_function\_def}$  by  $\text{auto}$ 
then show  $P (\lambda x. \sum_{y \in u \text{ ' space } M}. y * \text{indicator } (u \text{ - ' } \{y\} \cap \text{space } M) x)$ 
proof induct
  case empty
  then show  $?case$ 
    using  $\text{set[of \{\}]}$  by  $(\text{simp add: indicator\_def[abs\_def]})$ 
  next
  case  $(\text{insert } a \ F)$ 
  have  $\text{eq: } \sum \{y. u \ x = y \wedge (y = a \vee y \in F) \wedge x \in \text{space } M\}$ 
     $= (\text{if } u \ x = a \wedge x \in \text{space } M \text{ then } a \text{ else } 0) + \sum \{y. u \ x = y \wedge y \in F$ 
 $\wedge x \in \text{space } M\}$  for  $x$ 
  proof  $(\text{cases } x \in \text{space } M)$ 
    case True
    have  $*$ :  $\{y. u \ x = y \wedge (y = a \vee y \in F)\} = \{y. u \ x = a \wedge y = a\} \cup \{y. u \ x$ 
 $= y \wedge y \in F\}$ 
    by  $\text{auto}$ 
    show  $?thesis$ 
    using  $\text{insert by (simp add: * True)}$ 
  qed auto
  have  $a$ :  $P (\lambda x. a * \text{indicator } (u \text{ - ' } \{a\} \cap \text{space } M) x)$ 
  proof  $(\text{intro mult set})$ 
    show  $u \text{ - ' } \{a\} \cap \text{space } M \in \text{sets } M$ 
    using  $u$  by  $\text{auto}$ 
  qed
  show  $?case$ 
    using  $nn \text{ insert } a$ 
    by  $(\text{simp add: eq indicator\_times\_eq\_if [where } f = \lambda x. a] \text{ add})$ 
  qed
next
  show  $\text{simple\_function } M (\lambda x. (\sum_{y \in u \text{ ' space } M}. y * \text{indicator } (u \text{ - ' } \{y\} \cap$ 
 $\text{space } M) x))$ 
  apply  $(\text{subst simple\_function\_cong})$ 
  apply  $(\text{rule simple\_function\_indicator\_representation\_real[symmetric]})$ 
  apply  $(\text{auto intro: } u \ nn)$ 
  done
qed fact

proposition  $\text{simple\_function\_measurable\_on\_UNIV}$ :
  fixes  $f :: 'a :: \text{euclidean\_space} \Rightarrow \text{real}$ 
  assumes  $f$ :  $\text{simple\_function lebesgue } f$  and  $nn$ :  $\bigwedge x. f \ x \geq 0$ 
  shows  $f \text{ measurable\_on UNIV}$ 
  using  $f$ 
proof  $(\text{induction } f)$ 
  case  $(\text{cong } f \ g)$ 
  then obtain  $N$  where  $\text{negligible } N \ \{x. g \ x \neq f \ x\} \subseteq N$ 
    by  $(\text{auto simp: eventually\_ae\_filter\_negligible eq\_commute})$ 
  then show  $?case$ 
    by  $(\text{blast intro: measurable\_on\_spike cong})$ 

```

```

next
  case (set S)
  then show ?case
    by (simp add: indicator_measurable_on)
next
  case (mult u c)
  then show ?case
    by (simp add: measurable_on_cmul)
  case (add u v)
  then show ?case
    by (simp add: measurable_on_add)
qed (auto simp: nn)

lemma simple_function_lebesgue_if:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes f: simple_function lebesgue f and S:  $S \in \text{sets lebesgue}$ 
  shows simple_function lebesgue  $(\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0)$ 
proof -
  have ffin: finite (range f) and fsets:  $\forall x. f - \{f\ x\} \in \text{sets lebesgue}$ 
    using f by (auto simp: simple_function_def)
  have finite (f ' S)
    by (meson finite_subset subset_image_iff ffin top_greatest)
  moreover have finite  $((\lambda x. 0::\text{real}) ' T)$  for  $T :: 'a \text{ set}$ 
    by (auto simp: image_def)
  moreover have if_sets:  $(\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) - \{f\ a\} \in \text{sets lebesgue}$ 
for a
proof -
  have *:  $(\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) - \{f\ a\}$ 
    =  $(\text{if } f\ a = 0 \text{ then } -S \cup f - \{f\ a\} \text{ else } (f - \{f\ a\}) \cap S)$ 
    by (auto simp: split_if_asm)
  show ?thesis
    unfolding * by (metis Compl_in_sets_lebesgue S sets.Int sets.Un fsets)
qed
moreover have  $(\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) - \{0\} \in \text{sets lebesgue}$ 
proof (cases  $0 \in \text{range } f$ )
  case True
  then show ?thesis
    by (metis (no_types, lifting) if_sets rangeE)
next
  case False
  then have  $(\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) - \{0\} = -S$ 
    by auto
  then show ?thesis
    by (simp add: Compl_in_sets_lebesgue S)
qed
ultimately show ?thesis
  by (auto simp: simple_function_def)
qed

```


corollary *simple_function_measurable_on*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow \text{real}$
assumes f : *simple_function_lebesgue* f **and** nn : $\bigwedge x. f\ x \geq 0$ **and** S : $S \in \text{sets}$ *lebesgue*
shows f *measurable_on* S
by (*simp add: measurable_on_UNIV [symmetric, off]*) S *simple_function_lebesgue_if* nn *simple_function_measurable_on_UNIV*)

lemma
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{ordered\_euclidean\_space}$
assumes f : f *measurable_on* S **and** g : g *measurable_on* S
shows *measurable_on_sup*: $(\lambda x. \text{sup } (f\ x) (g\ x))$ *measurable_on* S
and *measurable_on_inf*: $(\lambda x. \text{inf } (f\ x) (g\ x))$ *measurable_on* S
proof –
obtain NF **and** F
where NF : *negligible* NF
and $conF$: $\bigwedge n. \text{continuous\_on } UNIV\ (F\ n)$
and $tendsF$: $\bigwedge x. x \notin NF \implies (\lambda n. F\ n\ x) \longrightarrow (\text{if } x \in S \text{ then } f\ x \text{ else } 0)$
using f **by** (*auto simp: measurable_on_def*)
obtain NG **and** G
where NG : *negligible* NG
and $conG$: $\bigwedge n. \text{continuous\_on } UNIV\ (G\ n)$
and $tendsG$: $\bigwedge x. x \notin NG \implies (\lambda n. G\ n\ x) \longrightarrow (\text{if } x \in S \text{ then } g\ x \text{ else } 0)$
using g **by** (*auto simp: measurable_on_def*)
show $(\lambda x. \text{sup } (f\ x) (g\ x))$ *measurable_on* S
unfolding *measurable_on_def*
proof (*intro exI conjI allI impI*)
show *continuous_on* $UNIV\ (\lambda x. \text{sup } (F\ n\ x) (G\ n\ x))$ **for** n
unfolding *sup_max eucl_sup* **by** (*intro conF conG continuous_intros*)
show $(\lambda n. \text{sup } (F\ n\ x) (G\ n\ x)) \longrightarrow (\text{if } x \in S \text{ then } \text{sup } (f\ x) (g\ x) \text{ else } 0)$
if $x \notin NF \cup NG$ **for** x
using *tendsto_sup* [$OF\ tendsF\ tendsG$, of $x\ x$] **that** **by** *auto*
qed (*simp add: NF NG*)
show $(\lambda x. \text{inf } (f\ x) (g\ x))$ *measurable_on* S
unfolding *measurable_on_def*
proof (*intro exI conjI allI impI*)
show *continuous_on* $UNIV\ (\lambda x. \text{inf } (F\ n\ x) (G\ n\ x))$ **for** n
unfolding *inf_min eucl_inf* **by** (*intro conF conG continuous_intros*)
show $(\lambda n. \text{inf } (F\ n\ x) (G\ n\ x)) \longrightarrow (\text{if } x \in S \text{ then } \text{inf } (f\ x) (g\ x) \text{ else } 0)$
if $x \notin NF \cup NG$ **for** x
using *tendsto_inf* [$OF\ tendsF\ tendsG$, of $x\ x$] **that** **by** *auto*
qed (*simp add: NF NG*)
qed

proposition *measurable_on_componentwise_UNIV*:
 f *measurable_on* $UNIV \iff (\forall i \in \text{Basis}. (\lambda x. (f\ x \cdot i) *_R i))$ *measurable_on* $UNIV$)
(is ?lhs = ?rhs)
proof
assume L : ?lhs

```

show ?rhs
proof
  fix i::'b
  assume i ∈ Basis
  have cont: continuous_on UNIV (λx. (x · i) *R i)
    by (intro continuous_intros)
  show (λx. (f x · i) *R i) measurable_on UNIV
    using measurable_on_compose_continuous [OF L cont]
    by (simp add: o_def)
qed
next
assume ?rhs
then have ∃ N g. negligible N ∧
  (∀ n. continuous_on UNIV (g n)) ∧
  (∀ x. x ∉ N ⟶ (λn. g n x) ⟶ (f x · i) *R i)
  if i ∈ Basis for i
  by (simp add: measurable_on_def that)
then obtain N g where N: ⋀ i. i ∈ Basis ⟹ negligible (N i)
  and cont: ⋀ i n. i ∈ Basis ⟹ continuous_on UNIV (g i n)
  and tends: ⋀ i x. [i ∈ Basis; x ∉ N i] ⟹ (λn. g i n x) ⟶ (f x · i) *R i
  by metis
show ?lhs
  unfolding measurable_on_def
proof (intro exI conjI allI impI)
  show negligible (⋃ i ∈ Basis. N i)
    using N eucl.finite_Basis by blast
  show continuous_on UNIV (λx. (∑ i ∈ Basis. g i n x)) for n
    by (intro continuous_intros cont)
next
fix x
assume x ∉ (⋃ i ∈ Basis. N i)
then have ⋀ i. i ∈ Basis ⟹ x ∉ N i
  by auto
then have (λn. (∑ i ∈ Basis. g i n x)) ⟶ (∑ i ∈ Basis. (f x · i) *R i)
  by (intro tends_tendsto_intros)
then show (λn. (∑ i ∈ Basis. g i n x)) ⟶ (if x ∈ UNIV then f x else 0)
  by (simp add: euclidean_representation)
qed
qed

corollary measurable_on_componentwise:
  f measurable_on S ⟷ (∀ i ∈ Basis. (λx. (f x · i) *R i) measurable_on S)
apply (subst measurable_on_UNIV [symmetric])
apply (subst measurable_on_componentwise_UNIV)
apply (simp add: measurable_on_UNIV if_distrib [of λx. inner x _] if_distrib
  [of λx. scaleR x _] cong: if_cong)
done

```

lemma *borel_measurable_implies_simple_function_sequence_real*:

fixes $u :: 'a \Rightarrow \text{real}$
assumes $u[\text{measurable}]$: $u \in \text{borel\_measurable } M$ **and** nn : $\bigwedge x. u\ x \geq 0$
shows $\exists f. \text{incseq } f \wedge (\forall i. \text{simple\_function } M\ (f\ i)) \wedge (\forall x. \text{bdd\_above } (\text{range } (\lambda i. f\ i\ x))) \wedge$
 $(\forall i\ x. 0 \leq f\ i\ x) \wedge u = (\text{SUP } i. f\ i)$

proof –

define f **where** $[\text{abs\_def}]$:

$f\ i\ x = \text{real\_of\_int } (\text{floor } ((\min i\ (u\ x)) * 2^i)) / 2^i$ **for** $i\ x$

have $[\text{simp}]$: $0 \leq f\ i\ x$ **for** $i\ x$

by (*auto simp: f_def intro!: divide_nonneg_nonneg mult_nonneg_nonneg nn*)

have $*$: $2^n * \text{real\_of\_int } x = \text{real\_of\_int } (2^n * x)$ **for** $n\ x$

by *simp*

have $\text{real\_of\_int } \lfloor \text{real } i * 2^i \rfloor = \text{real\_of\_int } \lfloor i * 2^i \rfloor$ **for** i

by (*intro arg_cong[where f=real_of_int] simp*)

then have $[\text{simp}]$: $\text{real\_of\_int } \lfloor \text{real } i * 2^i \rfloor = i * 2^i$ **for** i

unfolding *floor_of_nat* **by** *simp*

have *bdd*: *bdd_above* (*range* $(\lambda i. f\ i\ x)$) **for** x

by (*rule bdd_aboveI [where M = u x] (auto simp: f_def field_simps min_def)*)

have *incseq* f

proof (*intro monoI le_funI*)

fix $m\ n :: \text{nat}$ **and** x **assume** $m \leq n$

moreover

{ fix $d :: \text{nat}$

have $\lfloor 2^d :: \text{real} \rfloor * \lfloor 2^m * (\min (\text{of\_nat } m)\ (u\ x)) \rfloor \leq \lfloor 2^d * (2^m * (\min (\text{of\_nat } m)\ (u\ x))) \rfloor$

by (*rule le_mult_floor (auto simp: nn)*)

also have $\dots \leq \lfloor 2^d * (2^m * (\min (\text{of\_nat } d + \text{of\_nat } m)\ (u\ x))) \rfloor$

by (*intro floor_mono mult_mono min.mono*)

(*auto simp: nn min_less_iff_disj of_nat_less_top*)

finally have $f\ m\ x \leq f\ (m + d)\ x$

unfolding *f_def*

by (*auto simp: field_simps power_add * simp del: of_int_mult*) }

ultimately show $f\ m\ x \leq f\ n\ x$

by (*auto simp: le_iff_add*)

qed

then have *inc_f*: *incseq* $(\lambda i. f\ i\ x)$ **for** x

by (*auto simp: incseq_def le_fun_def*)

moreover

have *simple_function* $M\ (f\ i)$ **for** i

proof (*rule simple_function_borel_measurable*)

have $\lfloor (\min (\text{of\_nat } i)\ (u\ x)) * 2^i \rfloor \leq \lfloor \text{int } i * 2^i \rfloor$ **for** x

by (*auto split: split_min intro!: floor_mono*)

```

then have f i ' space M  $\subseteq$  ( $\lambda n. \text{real\_of\_int } n / 2^i$ ) '  $\{0 \dots \text{of\_nat } i * 2^i\}$ 
  unfolding floor_of_int by (auto simp: f_def nn intro!: imageI)
then show finite (f i ' space M)
  by (rule finite_subset) auto
show f i  $\in$  borel_measurable M
  unfolding f_def enn2real_def by measurable
qed
moreover
{ fix x
  have (SUP i. (f i x)) = u x
  proof -
    obtain n where u x  $\leq$  of_nat n using real_arch_simple by auto
    then have min_eq_r:  $\forall_F i$  in sequentially. min (real i) (u x) = u x
      by (auto simp: eventually_sequentially intro!: exI[of _ n] split: split_min)
    have ( $\lambda i. \text{real\_of\_int } \lfloor \min (\text{real } i) (u x) * 2^i \rfloor / 2^i$ )  $\longrightarrow$  u x
    proof (rule tendsto_sandwich)
      show ( $\lambda n. u x - (1/2)^n$ )  $\longrightarrow$  u x
        by (auto intro!: tendsto_eq_intros LIMSEQ_power_zero)
      show  $\forall_F n$  in sequentially.  $\text{real\_of\_int } \lfloor \min (\text{real } n) (u x) * 2^n \rfloor / 2^n$ 
         $n \leq u x$ 
        using min_eq_r by eventually_elim (auto simp: field_simps)
        have *:  $u x * (2^n * 2^n) \leq 2^n + 2^n * \text{real\_of\_int } \lfloor u x * 2^n \rfloor$  for
        n
        using real_of_int_floor_ge_diff_one[of u x * 2^n, THEN mult_left_mono,
        of 2^n]
        by (auto simp: field_simps)
        show  $\forall_F n$  in sequentially.  $u x - (1/2)^n \leq \text{real\_of\_int } \lfloor \min (\text{real } n) (u x) * 2^n \rfloor / 2^n$ 
        using min_eq_r by eventually_elim (insert *, auto simp: field_simps)
      qed auto
    then have ( $\lambda i. (f i x)$ )  $\longrightarrow$  u x
      by (simp add: f_def)
    from LIMSEQ_unique LIMSEQ_incseq_SUP [OF bdd_inc_f] this
    show ?thesis
      by blast
    qed }
ultimately show ?thesis
  by (intro exI [of _  $\lambda i x. f i x$ ]) (auto simp: incseq_f bdd_image_comp)
qed

```

lemma *homeomorphic_open_interval_UNIV*:

fixes a b:: real

assumes a < b

shows {a<..**<**b} *homeomorphic* (UNIV::real set)

proof -

have {a<..**<**b} = ball ((b+a) / 2) ((b-a) / 2)

using *assms*

by (auto simp: dist_real_def abs_if field_split_simps split: if_split_asm)

```

then show ?thesis
  by (simp add: homeomorphic_ball_UNIV assms)
qed

proposition homeomorphic_box_UNIV:
  fixes a b:: 'a::euclidean_space
  assumes box a b  $\neq \{\}$ 
  shows box a b homeomorphic (UNIV::'a set)
proof -
  have  $\{a \cdot i <..<b \cdot i\}$  homeomorphic (UNIV::real set) if  $i \in \text{Basis}$  for  $i$ 
    using assms box_ne_empty that by (blast intro: homeomorphic_open_interval_UNIV)
  then have  $\exists f g. (\forall x. a \cdot i < x \wedge x < b \cdot i \longrightarrow g (f x) = x) \wedge$ 
     $(\forall y. a \cdot i < g y \wedge g y < b \cdot i \wedge f(g y) = y) \wedge$ 
    continuous_on  $\{a \cdot i <..<b \cdot i\}$  f  $\wedge$ 
    continuous_on (UNIV::real set) g
    if  $i \in \text{Basis}$  for  $i$ 
    using that by (auto simp: homeomorphic_minimal mem_box Ball_def)
  then obtain f g where gf:  $\bigwedge i x. \llbracket i \in \text{Basis}; a \cdot i < x; x < b \cdot i \rrbracket \Longrightarrow g i (f i$ 
 $x) = x$ 
    and fg:  $\bigwedge i y. i \in \text{Basis} \Longrightarrow a \cdot i < g i y \wedge g i y < b \cdot i \wedge f i (g i y)$ 
 $= y$ 
    and contf:  $\bigwedge i. i \in \text{Basis} \Longrightarrow \text{continuous\_on } \{a \cdot i <..<b \cdot i\} (f i)$ 
    and contg:  $\bigwedge i. i \in \text{Basis} \Longrightarrow \text{continuous\_on } (\text{UNIV::real set}) (g i)$ 
  bymetis
  define F where  $F \equiv \lambda x. \sum_{i \in \text{Basis}. (f i (x \cdot i)) *_R i}$ 
  define G where  $G \equiv \lambda x. \sum_{i \in \text{Basis}. (g i (x \cdot i)) *_R i}$ 
  show ?thesis
    unfolding homeomorphic_minimal
  proof (intro exI conjI ballI)
    show  $G y \in \text{box } a b$  for  $y$ 
      using fg by (simp add: G_def mem_box)
    show  $G (F x) = x$  if  $x \in \text{box } a b$  for  $x$ 
      using that by (simp add: F_def G_def gf mem_box euclidean_representation)
    show  $F (G y) = y$  for  $y$ 
      by (simp add: F_def G_def fg mem_box euclidean_representation)
    show continuous_on (box a b) F
      unfolding F_def
    proof (intro continuous_intros continuous_on_compose2 [OF contf continu-
ous_on_inner])
      show  $(\lambda x. x \cdot i) ' \text{box } a b \subseteq \{a \cdot i <..<b \cdot i\}$  if  $i \in \text{Basis}$  for  $i$ 
        using that by (auto simp: mem_box)
    qed
    show continuous_on UNIV G
      unfolding G_def
      by (intro continuous_intros continuous_on_compose2 [OF contg continu-
ous_on_inner]) auto
    qed auto
  qed

```

lemma *diff_null_sets_lebesgue*: $\llbracket N \in \text{null\_sets } (\text{lebesgue\_on } S); X - N \in \text{sets } (\text{lebesgue\_on } S); N \subseteq X \rrbracket$
 $\implies X \in \text{sets } (\text{lebesgue\_on } S)$
by (*metis Int_Diff_Un inf.commute inf.orderE null_setsD2 sets.Un*)

lemma *borel_measurable_diff_null*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes $N: N \in \text{null\_sets } (\text{lebesgue\_on } S)$ **and** $S: S \in \text{sets } \text{lebesgue}$
shows $f \in \text{borel\_measurable } (\text{lebesgue\_on } (S - N)) \longleftrightarrow f \in \text{borel\_measurable } (\text{lebesgue\_on } S)$
unfolding *in_borel_measurable_space_lebesgue_on_sets_restrict_UNIV*
proof (*intro ball_cong iffI*)
show $f - ' T \cap S \in \text{sets } (\text{lebesgue\_on } S)$
if $f - ' T \cap (S - N) \in \text{sets } (\text{lebesgue\_on } (S - N))$ **for** T
proof -
have $N \cap S = N$
by (*metis N S inf.orderE null_sets_restrict_space*)
moreover have $N \cap S \in \text{sets } \text{lebesgue}$
by (*metis N S inf.orderE null_setsD2 null_sets_restrict_space*)
moreover have $f - ' T \cap S \cap (f - ' T \cap N) \in \text{sets } \text{lebesgue}$
by (*metis N S completion.complete inf.absorb2 inf_le2 inf_mono null_sets_restrict_space*)
ultimately show *?thesis*
by (*metis Diff_Int_distrib Int_Diff_Un S inf_le2 sets.Diff sets.Un sets_restrict_space_iff space_lebesgue_on_space_restrict_space that*)
qed
show $f - ' T \cap (S - N) \in \text{sets } (\text{lebesgue\_on } (S - N))$
if $f - ' T \cap S \in \text{sets } (\text{lebesgue\_on } S)$ **for** T
proof -
have $(S - N) \cap f - ' T = (S - N) \cap (f - ' T \cap S)$
by *blast*
then have $(S - N) \cap f - ' T \in \text{sets.restricted\_space } \text{lebesgue } (S - N)$
by (*metis S image_iff sets.Int_space_eq2 sets_restrict_space_iff that*)
then show *?thesis*
by (*simp add: inf.commute sets_restrict_space*)
qed
qed *auto*

lemma *lebesgue_measurable_diff_null*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes $N \in \text{null\_sets } \text{lebesgue}$
shows $f \in \text{borel\_measurable } (\text{lebesgue\_on } (-N)) \longleftrightarrow f \in \text{borel\_measurable } \text{lebesgue}$
by (*simp add: Compl_eq_Diff_UNIV assms borel_measurable_diff_null lebesgue_on_UNIV_eq*)

proposition *measurable_on_imp_borel_measurable_lebesgue_UNIV*:

```

fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
assumes  $f \text{ measurable\_on } UNIV$ 
shows  $f \in \text{borel\_measurable lebesgue}$ 
proof –
  obtain  $N$  and  $F$ 
    where  $NF$ : negligible  $N$ 
      and  $conF$ :  $\bigwedge n. \text{continuous\_on } UNIV (F\ n)$ 
      and  $tendsF$ :  $\bigwedge x. x \notin N \implies (\lambda n. F\ n\ x) \longrightarrow f\ x$ 
    using assms by (auto simp: measurable_on_def)
  obtain  $N$  where  $N \in \text{null\_sets lebesgue}$   $f \in \text{borel\_measurable (lebesgue\_on (} - N))$ 
proof
  show  $f \in \text{borel\_measurable (lebesgue\_on (} - N))$ 
proof (rule borel\_measurable LIMSEQ\_metric)
  show  $F\ i \in \text{borel\_measurable (lebesgue\_on (} - N))$  for  $i$ 
    by (meson Compl_in_sets lebesgue NF conF continuous_imp_measurable_on_sets lebesgue
continuous_on_subset negligible_imp_sets subset_UNIV)
  show  $(\lambda i. F\ i\ x) \longrightarrow f\ x$  if  $x \in \text{space (lebesgue\_on (} - N))$  for  $x$ 
    using that
    by (simp add: tendsF)
qed
show  $N \in \text{null\_sets lebesgue}$ 
  using  $NF$  negligible_iff_null_sets by blast
qed
then show ?thesis
  using lebesgue_measurable_diff_null by blast
qed

corollary measurable_on_imp_borel_measurable_lebesgue:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
assumes  $f \text{ measurable\_on } S$  and  $S$ :  $S \in \text{sets lebesgue}$ 
shows  $f \in \text{borel\_measurable (lebesgue\_on } S)$ 
proof –
  have  $(\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) \text{ measurable\_on } UNIV$ 
    using assms(1) measurable_on_UNIV by blast
  then show ?thesis
    by (simp add: borel\_measurable_if_D measurable_on_imp_borel_measurable_lebesgue_UNIV)
qed

proposition measurable_on_limit:
  fixes  $f :: \text{nat} \Rightarrow 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
assumes  $f$ :  $\bigwedge n. f\ n \text{ measurable\_on } S$  and  $N$ : negligible  $N$ 
    and  $lim$ :  $\bigwedge x. x \in S - N \implies (\lambda n. f\ n\ x) \longrightarrow g\ x$ 
shows  $g \text{ measurable\_on } S$ 
proof –
  have  $box\ (0::'b)$  One homeomorphic (UNIV::'b set)
    by (simp add: homeomorphic_box_UNIV)
  then obtain  $h\ h'$ :  $'b \Rightarrow 'b$  where  $hh'$ :  $\bigwedge x. x \in box\ 0\ One \implies h\ (h'\ x) = x$ 

```

```

    and h'im: h' ' box 0 One = UNIV
    and conth: continuous_on UNIV h
    and conth': continuous_on (box 0 One) h'
    and h'h:  $\bigwedge y. h' (h y) = y$ 
    and rangeh: range h = box 0 One
  by (auto simp: homeomorphic_def homeomorphism_def)
have norm y  $\leq$  DIM('b) if y: y  $\in$  box 0 One for y::'b
proof -
  have y01:  $0 < y \cdot i \ y \cdot i < 1$  if i  $\in$  Basis for i
    using that y by (auto simp: mem_box)
  have norm y  $\leq$  ( $\sum i \in \text{Basis}. |y \cdot i|$ )
    using norm_le_l1 by blast
  also have ...  $\leq$  ( $\sum i::'b \in \text{Basis}. 1$ )
  proof (rule sum_mono)
    show  $|y \cdot i| \leq 1$  if i  $\in$  Basis for i
      using y01 that by fastforce
  qed
  also have ...  $\leq$  DIM('b)
    by auto
  finally show ?thesis .
qed
then have norm_le: norm(h y)  $\leq$  DIM('b) for y
  by (metis UNIV_I image_eqI rangeh)
have (h'  $\circ$  (h  $\circ$  ( $\lambda x. \text{if } x \in S \text{ then } g x \text{ else } 0$ )))) measurable_on UNIV
proof (rule measurable_on_compose_continuous_box)
  let ? $\chi$  = h  $\circ$  ( $\lambda x. \text{if } x \in S \text{ then } g x \text{ else } 0$ )
  let ?f =  $\lambda n. h \circ (\lambda x. \text{if } x \in S \text{ then } f n x \text{ else } 0)$ 
  show ? $\chi$  measurable_on UNIV
  proof (rule integrable_subintervals_imp_measurable)
    show ? $\chi$  integrable_on cbox a b for a b
    proof (rule integrable_spike_set)
      show ? $\chi$  integrable_on (cbox a b - N)
      proof (rule dominated_convergence_integrable)
        show const: ( $\lambda x. \text{DIM}('b)$ ) integrable_on cbox a b - N
        by (simp add: N has_integral_iff integrable_const integrable_negligible
integrable_setdiff negligible_diff)
        show norm ((h  $\circ$  ( $\lambda x. \text{if } x \in S \text{ then } g x \text{ else } 0$ )) x)  $\leq$  DIM('b) if x  $\in$  cbox
a b - N for x
          using that norm_le by (simp add: o_def)
        show ( $\lambda k. ?f k x$ )  $\longrightarrow$  ? $\chi$  x if x  $\in$  cbox a b - N for x
          using that lim [of x] conth
        by (auto simp: continuous_on_def intro: tendsto_compose)
        show (?f n) absolutely_integrable_on cbox a b - N for n
        proof (rule measurable_bounded_by_integrable_imp_absolutely_integrable)
          show ?f n  $\in$  borel_measurable (lebesgue_on (cbox a b - N))
          proof (rule measurable_on_imp_borel_measurable_lebesgue [OF mea-
surable_on_spike_set])
            show ?f n measurable_on cbox a b
              unfolding measurable_on_UNIV [symmetric, of _ cbox a b]

```



```

proof (rule measurable_on_restrict)
  have f': ( $\lambda x.$  if  $x \in S$  then  $f\ n\ x$  else  $0$ ) measurable_on UNIV
  by (simp add: f measurable_on_UNIV)
  show ?f n measurable_on UNIV
  using measurable_on_compose_continuous [OF f' conth] by auto
qed auto
show negligible (sym_diff (cbox a b) (cbox a b - N))
  by (auto intro: negligible_subset [OF N])
show cbox a b - N  $\in$  sets lebesgue
  by (simp add: N negligible_imp_sets sets.Diff)
qed
show cbox a b - N  $\in$  sets lebesgue
  by (simp add: N negligible_imp_sets sets.Diff)
show norm (?f n x)  $\leq$  DIM('b)
  if  $x \in$  cbox a b - N for x
  using that local.norm_le by simp
qed (auto simp: const)
qed
show negligible { $x \in$  cbox a b - N - cbox a b. ? $\chi\ x \neq 0$ }
  by (auto simp: empty_imp_negligible)
have { $x \in$  cbox a b - (cbox a b - N). ? $\chi\ x \neq 0$ }  $\subseteq$  N
  by auto
then show negligible { $x \in$  cbox a b - (cbox a b - N). ? $\chi\ x \neq 0$ }
  using N negligible_subset by blast
qed
qed
show ? $\chi\ x \in$  box 0 One for x
  using rangeh by auto
show continuous_on (box 0 One) h'
  by (rule conth')
qed
then show ?thesis
  by (simp add: o_def h'h measurable_on_UNIV)
qed

```

```

lemma measurable_on_if_simple_function_limit:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  shows  $\llbracket \bigwedge n. g\ n$  measurable_on UNIV;  $\bigwedge n.$  finite (range (g n));  $\bigwedge x. (\lambda n. g\ n\ x) \longrightarrow f\ x \rrbracket$ 
   $\implies$  f measurable_on UNIV
  by (force intro: measurable_on_limit [where N={}])

```

```

lemma lebesgue_measurable_imp_measurable_on_nnreal_UNIV:
  fixes u :: 'a::euclidean_space  $\Rightarrow$  real
  assumes u:  $u \in$  borel_measurable lebesgue and nn:  $\bigwedge x. u\ x \geq 0$ 
  shows u measurable_on UNIV
proof -

```

```

obtain  $f$  where  $incseq\ f$  and  $f: \forall i. simple\_function\ lebesgue\ (f\ i)$ 
and  $bdd: \bigwedge x. bdd\_above\ (range\ (\lambda i. f\ i\ x))$ 
and  $nnf: \bigwedge i\ x. 0 \leq f\ i\ x$  and  $*: u = (SUP\ i. f\ i)$ 
using  $borel\_measurable\_implies\_simple\_function\_sequence\_real\ nn\ u$  by  $metis$ 
show  $?thesis$ 
unfolding  $*$ 
proof ( $rule\ measurable\_on\_if\_simple\_function\_limit\ [of\ concl: Sup\ (range\ f)]$ )
show  $(f\ i)\ measurable\_on\ UNIV$  for  $i$ 
by ( $simp\ add: f\ nnf\ simple\_function\_measurable\_on\_UNIV$ )
show  $finite\ (range\ (f\ i))$  for  $i$ 
by ( $metis\ f\ simple\_function\_def\ space\_borel\ space\_completion\ space\_lborel$ )
show  $(\lambda i. f\ i\ x) \longrightarrow Sup\ (range\ f)\ x$  for  $x$ 
proof –
have  $incseq\ (\lambda i. f\ i\ x)$ 
using  $\langle incseq\ f \rangle$  apply ( $auto\ simp: incseq\_def$ )
by ( $simp\ add: le\_funD$ )
then show  $?thesis$ 
by ( $metis\ SUP\_apply\ bdd\ LIMSEQ\_incseq\_SUP$ )
qed
qed
qed

```

```

lemma  $lebesgue\_measurable\_imp\_measurable\_on\_nnreal$ :
fixes  $u :: 'a::euclidean\_space \Rightarrow real$ 
assumes  $u \in borel\_measurable\ lebesgue\ \bigwedge x. u\ x \geq 0$   $S \in sets\ lebesgue$ 
shows  $u\ measurable\_on\ S$ 
unfolding  $measurable\_on\_UNIV$  [ $symmetric, of\ u$ ]
using  $assms$ 
by ( $auto\ intro: lebesgue\_measurable\_imp\_measurable\_on\_nnreal\_UNIV$ )

```

```

lemma  $lebesgue\_measurable\_imp\_measurable\_on\_real$ :
fixes  $u :: 'a::euclidean\_space \Rightarrow real$ 
assumes  $u: u \in borel\_measurable\ lebesgue$  and  $S: S \in sets\ lebesgue$ 
shows  $u\ measurable\_on\ S$ 
proof –
let  $?f = \lambda x. |u\ x| + u\ x$ 
let  $?g = \lambda x. |u\ x| - u\ x$ 
have  $?f\ measurable\_on\ S\ ?g\ measurable\_on\ S$ 
using  $S\ u$  by ( $auto\ intro: lebesgue\_measurable\_imp\_measurable\_on\_nnreal$ )
then have  $(\lambda x. (?f\ x - ?g\ x) / 2)\ measurable\_on\ S$ 
using  $measurable\_on\_cdivide\ measurable\_on\_diff$  by  $blast$ 
then show  $?thesis$ 
by  $auto$ 
qed

```

```

proposition  $lebesgue\_measurable\_imp\_measurable\_on$ :
fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
assumes  $f: f \in borel\_measurable\ lebesgue$  and  $S: S \in sets\ lebesgue$ 

```

```

shows  $f$  measurable_on  $S$ 
unfolding measurable_on_componentwise [of  $f$ ]
proof
  fix  $i :: 'b$ 
  assume  $i \in \text{Basis}$ 
  have  $(\lambda x. (f\ x \cdot i)) \in \text{borel\_measurable lebesgue}$ 
    using  $\langle i \in \text{Basis} \rangle \text{borel\_measurable\_euclidean\_space } f$  by blast
  then have  $(\lambda x. (f\ x \cdot i)) \text{ measurable\_on } S$ 
    using  $S \text{ lebesgue\_measurable\_imp\_measurable\_on\_real}$  by blast
  then show  $(\lambda x. (f\ x \cdot i) *_{\mathbb{R}} i) \text{ measurable\_on } S$ 
    by (intro measurable_on_scaleR measurable_on_const  $S$ )
qed

proposition measurable_on_iff_borel_measurable:
  fixes  $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$ 
  assumes  $S \in \text{sets lebesgue}$ 
  shows  $f \text{ measurable\_on } S \iff f \in \text{borel\_measurable (lebesgue\_on } S)$  (is ?lhs =
  ?rhs)
proof
  show  $f \in \text{borel\_measurable (lebesgue\_on } S)$ 
    if  $f \text{ measurable\_on } S$ 
    using that by (simp add: assms measurable_on_imp_borel_measurable_lebesgue)
next
  assume  $f \in \text{borel\_measurable (lebesgue\_on } S)$ 
  then have  $(\lambda a. \text{if } a \in S \text{ then } f\ a \text{ else } 0) \text{ measurable\_on UNIV}$ 
    by (simp add: assms borel_measurable_if_lebesgue_measurable_imp_measurable_on)
  then show  $f \text{ measurable\_on } S$ 
    using measurable_on_UNIV by blast
qed

```

6.29.5 Measurability on generalisations of the binary product

```

lemma measurable_on_bilinear:
  fixes  $h :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space} \Rightarrow 'c :: \text{euclidean\_space}$ 
  assumes  $h: \text{bilinear } h$  and  $f: f \text{ measurable\_on } S$  and  $g: g \text{ measurable\_on } S$ 
  shows  $(\lambda x. h\ (f\ x)\ (g\ x)) \text{ measurable\_on } S$ 
proof (rule measurable_on_combine [where  $h = h$ ])
  show continuous_on UNIV  $(\lambda x. h\ (fst\ x)\ (snd\ x))$ 
    by (simp add: bilinear_continuous_on_compose [OF continuous_on_fst continuous_on_snd  $h$ ])
  show  $h\ 0\ 0 = 0$ 
    by (simp add: bilinear_lzero  $h$ )
qed (auto intro: assms)

```

```

lemma borel_measurable_bilinear:
  fixes  $h :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space} \Rightarrow 'c :: \text{euclidean\_space}$ 
  assumes bilinear  $h$   $f \in \text{borel\_measurable (lebesgue\_on } S)$   $g \in \text{borel\_measurable (lebesgue\_on } S)$ 

```

```

    and S: S ∈ sets lebesgue
  shows (λx. h (f x) (g x)) ∈ borel_measurable (lebesgue_on S)
  using assms measurable_on_bilinear [of h f S g]
  by (simp flip: measurable_on_iff_borel_measurable)

lemma absolutely_integrable_bounded_measurable_product:
  fixes h :: 'a::euclidean_space ⇒ 'b::euclidean_space ⇒ 'c::euclidean_space
  assumes bilinear h and f: f ∈ borel_measurable (lebesgue_on S) S ∈ sets
  lebesgue
  and bou: bounded (f ' S) and g: g absolutely_integrable_on S
  shows (λx. h (f x) (g x)) absolutely_integrable_on S
proof -
  obtain B where B > 0 and B: ∧x y. norm (h x y) ≤ B * norm x * norm y
  using bilinear_bounded_pos ⟨bilinear h⟩ by blast
  obtain C where C > 0 and C: ∧x. x ∈ S ⇒ norm (f x) ≤ C
  using bounded_pos by (metis bou imageI)
  show ?thesis
proof (rule measurable_bounded_by_integrable_imp_absolutely_integrable [OF
  _ ⟨S ∈ sets lebesgue⟩])
  show norm (h (f x) (g x)) ≤ B * C * norm(g x) if x ∈ S for x
  by (meson less_le mult_left_mono mult_right_mono norm_ge_zero or-
  der_trans that ⟨B > 0⟩ B C)
  show (λx. h (f x) (g x)) ∈ borel_measurable (lebesgue_on S)
  using ⟨bilinear h⟩ f g
  by (blast intro: borel_measurable_bilinear dest: absolutely_integrable_measurable)
  show (λx. B * C * norm(g x)) integrable_on S
  using ⟨0 < B⟩ ⟨0 < C⟩ absolutely_integrable_on_def g by auto
qed
qed

lemma absolutely_integrable_bounded_measurable_product_real:
  fixes f :: real ⇒ real
  assumes f ∈ borel_measurable (lebesgue_on S) S ∈ sets lebesgue
  and bounded (f ' S) and g absolutely_integrable_on S
  shows (λx. f x * g x) absolutely_integrable_on S
  using absolutely_integrable_bounded_measurable_product bilinear_times assms
  by blast

lemma borel_measurable_AE:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes f ∈ borel_measurable lebesgue and ae: AE x in lebesgue. f x = g x
  shows g ∈ borel_measurable lebesgue
proof -
  obtain N where N: N ∈ null_sets lebesgue ∧x. x ∉ N ⇒ f x = g x
  using ae unfolding completion.AE_iff_null_sets by auto
  have f measurable_on UNIV
  by (simp add: assms lebesgue_measurable_imp_measurable_on)
  then have g measurable_on UNIV

```

```

    by (metis Diff_iff N measurable_on_spike negligible_iff_null_sets)
  then show ?thesis
    using measurable_on_imp_borel_measurable_lebesgue_UNIV by blast
qed

lemma has_bochner_integral_combine:
  fixes f :: real  $\Rightarrow$  'a::euclidean_space
  assumes a  $\leq$  c c  $\leq$  b
    and ac: has_bochner_integral (lebesgue_on {a..c}) f i
    and cb: has_bochner_integral (lebesgue_on {c..b}) f j
  shows has_bochner_integral (lebesgue_on {a..b}) f (i + j)
proof -
  have i: has_bochner_integral lebesgue ( $\lambda x$ . indicator {a..c} x *R f x) i
    and j: has_bochner_integral lebesgue ( $\lambda x$ . indicator {c..b} x *R f x) j
    using assms by (auto simp: has_bochner_integral_restrict_space)
  have AE: AE x in lebesgue. indicat_real {a..c} x *R f x + indicat_real {c..b} x
    *R f x = indicat_real {a..b} x *R f x
  proof (rule AE_I')
    have eq: indicat_real {a..c} x *R f x + indicat_real {c..b} x *R f x = indicat_real {a..b} x *R f x if x  $\neq$  c for x
      using assms that by (auto simp: indicator_def)
    then show {x  $\in$  space lebesgue. indicat_real {a..c} x *R f x + indicat_real {c..b} x *R f x  $\neq$  indicat_real {a..b} x *R f x}  $\subseteq$  {c}
      by auto
    qed auto
  have has_bochner_integral lebesgue ( $\lambda x$ . indicator {a..b} x *R f x) (i + j)
  proof (rule has_bochner_integralI_AE [OF has_bochner_integral_add [OF i j] _ AE])
    have eq: indicat_real {a..c} x *R f x + indicat_real {c..b} x *R f x = indicat_real {a..b} x *R f x if x  $\neq$  c for x
      using assms that by (auto simp: indicator_def)
    show ( $\lambda x$ . indicat_real {a..b} x *R f x)  $\in$  borel_measurable lebesgue
    proof (rule borel_measurable_AE [OF borel_measurable_add AE])
      show ( $\lambda x$ . indicator {a..c} x *R f x)  $\in$  borel_measurable lebesgue
        ( $\lambda x$ . indicator {c..b} x *R f x)  $\in$  borel_measurable lebesgue
      using i j by auto
    qed
  qed
qed
then show ?thesis
  by (simp add: has_bochner_integral_restrict_space)
qed

```

```

lemma integrable_combine:
  fixes f :: real  $\Rightarrow$  'a::euclidean_space
  assumes integrable (lebesgue_on {a..c}) f integrable (lebesgue_on {c..b}) f
    and a  $\leq$  c c  $\leq$  b
  shows integrable (lebesgue_on {a..b}) f
  using assms has_bochner_integral_combine has_bochner_integral_iff by blast

```

lemma *integral_combine*:

fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$
assumes $f :: \text{integrable} (\text{lebesgue\_on} \{a..b\}) f$ **and** $a \leq c \leq b$
shows $\text{integral}^L (\text{lebesgue\_on} \{a..b\}) f = \text{integral}^L (\text{lebesgue\_on} \{a..c\}) f +$
 $\text{integral}^L (\text{lebesgue\_on} \{c..b\}) f$
proof –
have $i :: \text{has\_bochner\_integral} (\text{lebesgue\_on} \{a..c\}) f (\text{integral}^L (\text{lebesgue\_on} \{a..c\})$
 $f)$
using *integrable_subinterval* $\langle c \leq b \rangle$ f *has_bochner_integral_iff* **by** *fastforce*
have $j :: \text{has\_bochner\_integral} (\text{lebesgue\_on} \{c..b\}) f (\text{integral}^L (\text{lebesgue\_on} \{c..b\})$
 $f)$
using *integrable_subinterval* $\langle a \leq c \rangle$ f *has_bochner_integral_iff* **by** *fastforce*
show *?thesis*
by (*meson* $\langle a \leq c \rangle \langle c \leq b \rangle$ *has_bochner_integral_combine* *has_bochner_integral_iff*
 $i j$)
qed

lemma *has_bochner_integral_null* [*intro*]:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes $N \in \text{null\_sets } \text{lebesgue}$
shows $\text{has\_bochner\_integral} (\text{lebesgue\_on } N) f 0$
unfolding *has_bochner_integral_iff* — strange that the proof's so long
proof
show *integrable* (*lebesgue_on* N) f
proof (*subst integrable_restrict_space*)
show $N \cap \text{space } \text{lebesgue} \in \text{sets } \text{lebesgue}$
using *assms* **by** *force*
show *integrable* *lebesgue* ($\lambda x. \text{indicat\_real } N x *_{\mathbb{R}} f x$)
proof (*rule integrable_cong_AE_imp*)
show *integrable* *lebesgue* ($\lambda x. 0$)
by *simp*
show $AE x \text{ in } \text{lebesgue}. 0 = \text{indicat\_real } N x *_{\mathbb{R}} f x$
using *assms*
by (*simp add: indicator_def completion.null_sets_iff AE_eventually_mono*)
show ($\lambda x. \text{indicat\_real } N x *_{\mathbb{R}} f x$) $\in \text{borel\_measurable } \text{lebesgue}$
by (*auto intro: borel_measurable_AE [OF _ *]*)
qed
qed
show $\text{integral}^L (\text{lebesgue\_on } N) f = 0$
proof (*rule integral_eq_zero_AE*)
show $AE x \text{ in } \text{lebesgue\_on } N. f x = 0$
by (*rule AE_I' [where N=N]*) (*auto simp: assms null_setsD2 null_sets_restrict_space*)
qed
qed

lemma *has_bochner_integral_null_eq*[*simp*]:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes $N \in \text{null\_sets } \text{lebesgue}$
shows $\text{has\_bochner\_integral} (\text{lebesgue\_on } N) f i \longleftrightarrow i = 0$

```

    using assms has_bochner_integral_eq by blast
end

```

6.30 Embedding Measure Spaces with a Function

```

theory Embed_Measure
imports Binary_Product_Measure
begin

```

Given a measure space on some carrier set Ω and a function f , we can define a push-forward measure on the carrier set $f(\Omega)$ whose σ -algebra is the one generated by mapping f over the original sigma algebra.

This is useful e.g. when f is injective, i.e. it is some kind of “tagging” function. For instance, suppose we have some algebraic datatype of values with various constructors, including a constructor *RealVal* for real numbers. Then *embed_measure* allows us to lift a measure on real numbers to the appropriate subset of that algebraic datatype.

definition *embed_measure* :: *'a measure* $\Rightarrow$ (*'a* $\Rightarrow$ *'b*) $\Rightarrow$ *'b measure* **where**
embed_measure *M f* = *measure_of* (*f* ‘ *space M*) {*f* ‘ *A* | *A. A* $\in$ *sets M*}
($\lambda A.$ *emeasure M* (*f* – *A* $\cap$ *space M*))

lemma *space_embed_measure*: *space* (*embed_measure M f*) = *f* ‘ *space M*
unfolding *embed_measure_def*
by (*subst space_measure_of*) (*auto dest: sets.sets_into_space*)

lemma *sets_embed_measure'*:
assumes *inj*: *inj_on f* (*space M*)
shows *sets* (*embed_measure M f*) = {*f* ‘ *A* | *A. A* $\in$ *sets M*}
unfolding *embed_measure_def*

proof (*intro sigma_algebra.sets_measure_of_eq sigma_algebra_iff2* [*THEN iffD2*]
conjI allI ballI impI)

fix *s* **assume** *s* $\in$ {*f* ‘ *A* | *A. A* $\in$ *sets M*}
then obtain *s'* **where** *s'_props*: *s* = *f* ‘ *s'* *s'* $\in$ *sets M* **by** *auto*
hence *f* ‘ *space M* – *s* = *f* ‘ (*space M* – *s'*) **using** *inj*
by (*auto dest: inj_onD sets.sets_into_space*)
also have ... $\in$ {*f* ‘ *A* | *A. A* $\in$ *sets M*} **using** *s'_props* **by** *auto*
finally show *f* ‘ *space M* – *s* $\in$ {*f* ‘ *A* | *A. A* $\in$ *sets M*} .

next

fix *A* :: *nat* $\Rightarrow$ _ **assume** *range A* $\subseteq$ {*f* ‘ *A* | *A. A* $\in$ *sets M*}
then obtain *A'* **where** *A'*: $\bigwedge i. A\ i = f\ 'A'\ i \wedge i. A'\ i \in sets\ M$
by (*auto simp: subset_eq choice_iff*)
then have ($\bigcup x. f\ 'A'\ x$) = *f* ‘ ($\bigcup x. A'\ x$) **by** *blast*
with *A'* **show** ($\bigcup i. A\ i$) $\in$ {*f* ‘ *A* | *A. A* $\in$ *sets M*}
by *simp blast*

qed (*auto dest: sets.sets_into_space*)

lemma *the_inv_into_vimage*:

$inj\_on\ f\ X \implies A \subseteq X \implies the\_inv\_into\ X\ f - 'A \cap (f'X) = f - 'A$
by (*auto simp: the_inv_into_f_f*)

lemma *sets_embed_eq_vimage_algebra*:

assumes *inj_on f (space M)*

shows $sets\ (embed\_measure\ M\ f) = sets\ (vimage\_algebra\ (f'space\ M)\ (the\_inv\_into\ (space\ M)\ f)\ M)$

by (*auto simp: sets_embed_measure'[OF assms] Pi_iff the_inv_into_f_f assms sets_vimage_algebra2 Setcompr_eq_image*

dest: sets.sets_into_space

intro!: image_cong the_inv_into_vimage[symmetric])

lemma *sets_embed_measure*:

assumes *inj: inj f*

shows $sets\ (embed\_measure\ M\ f) = \{f - 'A \mid A. A \in sets\ M\}$

using *assms* **by** (*subst sets_embed_measure' (auto intro!: inj_onI dest: injD)*)

lemma *in_sets_embed_measure*: $A \in sets\ M \implies f - 'A \in sets\ (embed\_measure\ M\ f)$

unfolding *embed_measure_def*

by (*intro in_measure_of (auto dest: sets.sets_into_space)*)

lemma *measurable_embed_measure1*:

assumes $g: (\lambda x. g\ (f\ x)) \in measurable\ M\ N$

shows $g \in measurable\ (embed\_measure\ M\ f)\ N$

unfolding *measurable_def*

proof *safe*

fix *A* **assume** $A \in sets\ N$

with *g* **have** $(\lambda x. g\ (f\ x)) - 'A \cap space\ M \in sets\ M$

by (*rule measurable_sets*)

then **have** $f - '((\lambda x. g\ (f\ x)) - 'A \cap space\ M) \in sets\ (embed\_measure\ M\ f)$

by (*rule in_sets_embed_measure*)

also **have** $f - '((\lambda x. g\ (f\ x)) - 'A \cap space\ M) = g - 'A \cap space\ (embed\_measure\ M\ f)$

by (*auto simp: space_embed_measure*)

finally **show** $g - 'A \cap space\ (embed\_measure\ M\ f) \in sets\ (embed\_measure\ M\ f)$.

qed (*insert measurable_space[OF assms], auto simp: space_embed_measure*)

lemma *measurable_embed_measure2'*:

assumes *inj_on f (space M)*

shows $f \in measurable\ M\ (embed\_measure\ M\ f)$

proof–

{

fix *A* **assume** $A: A \in sets\ M$

also **from** *A* **have** $A = A \cap space\ M$ **by** *auto*

also **have** $\dots = f - 'f - 'A \cap space\ M$ **using** *A assms*

by (*auto dest: inj_onD sets.sets_into_space*)

finally **have** $f - 'f - 'A \cap space\ M \in sets\ M$.


```

}
thus ?thesis using assms unfolding embed_measure_def
  by (intro measurable_measure_of) (auto dest: sets.sets_into_space)
qed

```

```

lemma measurable_embed_measure2:
  assumes [simp]: inj f shows  $f \in \text{measurable } M \text{ (embed\_measure } M f)$ 
  by (auto simp: inj_vimage_image_eq embed_measure_def
    intro!: measurable_measure_of dest: sets.sets_into_space)

```

```

lemma embed_measure_eq_distr':
  assumes inj_on f (space M)
  shows  $\text{embed\_measure } M f = \text{distr } M \text{ (embed\_measure } M f) f$ 
proof–
  have  $\text{distr } M \text{ (embed\_measure } M f) f =$ 
     $\text{measure\_of } (f \text{ ' } \text{space } M) \{f \text{ ' } A \mid A. A \in \text{sets } M\}$ 
     $(\lambda A. \text{emeasure } M (f \text{ - ' } A \cap \text{space } M))$  unfolding distr_def
  by (simp add: space_embed_measure sets_embed_measure'[OF assms])
  also have ... =  $\text{embed\_measure } M f$  unfolding embed_measure_def ..
  finally show ?thesis ..
qed

```

```

lemma embed_measure_eq_distr:
   $\text{inj } f \implies \text{embed\_measure } M f = \text{distr } M \text{ (embed\_measure } M f) f$ 
  by (rule embed_measure_eq_distr') (auto intro!: inj_onI dest: injD)

```

```

lemma nn_integral_embed_measure':
   $\text{inj\_on } f \text{ (space } M) \implies g \in \text{borel\_measurable (embed\_measure } M f) \implies$ 
   $\text{nn\_integral (embed\_measure } M f) g = \text{nn\_integral } M (\lambda x. g (f x))$ 
  apply (subst embed_measure_eq_distr', simp)
  apply (subst nn_integral_distr)
  apply (simp_all add: measurable_embed_measure2')
done

```

```

lemma nn_integral_embed_measure:
   $\text{inj } f \implies g \in \text{borel\_measurable (embed\_measure } M f) \implies$ 
   $\text{nn\_integral (embed\_measure } M f) g = \text{nn\_integral } M (\lambda x. g (f x))$ 
  by(erule nn_integral_embed_measure'[OF subset_inj_on]) simp

```

```

lemma emeasure_embed_measure':
  assumes inj_on f (space M)  $A \in \text{sets (embed\_measure } M f)$ 
  shows  $\text{emeasure (embed\_measure } M f) A = \text{emeasure } M (f \text{ - ' } A \cap \text{space } M)$ 
  by (subst embed_measure_eq_distr'[OF assms(1)])
    (simp add: emeasure_distr[OF measurable_embed_measure2'[OF assms(1)]
assms(2)])

```

```

lemma emeasure_embed_measure:
  assumes inj f  $A \in \text{sets (embed\_measure } M f)$ 
  shows  $\text{emeasure (embed\_measure } M f) A = \text{emeasure } M (f \text{ - ' } A \cap \text{space } M)$ 

```

using *assms* **by** (intro *emeasure_embed_measure'*) (auto intro!: *inj_onI dest: injD*)

lemma *embed_measure_comp*:

assumes [*simp*]: *inj f inj g*

shows *embed_measure (embed_measure M f) g = embed_measure M (g ∘ f)*

proof –

have [*simp*]: *inj (λx. g (f x))* **by** (*subst o_def[symmetric]*) (auto intro: *inj_compose*)

note *measurable_embed_measure2[measurable]*

have *embed_measure (embed_measure M f) g =*

distr M (embed_measure (embed_measure M f) g) (g ∘ f)

by (*subst (1 2) embed_measure_eq_distr*)

(*simp_all add: distr_distr sets_embed_measure cong: distr_cong*)

also have ... = *embed_measure M (g ∘ f)*

by (*subst (3) embed_measure_eq_distr, simp add: o_def, rule distr_cong*)

(auto *simp: sets_embed_measure o_def image_image[symmetric]*)

intro: inj_compose cong: distr_cong)

finally show ?*thesis* .

qed

lemma *sigma_finite_embed_measure*:

assumes *sigma_finite_measure M* **and** *inj: inj f*

shows *sigma_finite_measure (embed_measure M f)*

proof –

from *assms(1)* **interpret** *sigma_finite_measure M* .

from *sigma_finite_countable* **obtain** *A* **where**

A_props: countable A A ⊆ sets M ∪ A = space M ∧ X. X ∈ A ⇒ emeasure M X ≠ ∞ **by** *blast*

from *A_props* **have** *countable (($\cdot$) f' A)* **by** *auto*

moreover

from *inj* **and** *A_props* **have** *(($\cdot$) f' A ⊆ sets (embed_measure M f))*

by (*auto simp: sets_embed_measure*)

moreover

from *A_props* **and** *inj* **have** *∪ (($\cdot$) f' A) = space (embed_measure M f)*

by (*auto simp: space_embed_measure intro!: imageI*)

moreover

from *A_props* **and** *inj* **have** *∀ a ∈ (($\cdot$) f' A. emeasure (embed_measure M f) a ≠ ∞*

by (*intro ballI, subst emeasure_embed_measure*)

(auto *simp: inj_vimage_image_eq intro: in_sets_embed_measure*)

ultimately show ?*thesis* **by** – (*standard, blast*)

qed

lemma *embed_measure_count_space'*:

inj_on f A ⇒ embed_measure (count_space A) f = count_space (f' A)

apply (*subst distr_bij_count_space[of f A f' A, symmetric]*)

apply (*simp add: inj_on_def bij_betw_def*)

apply (*subst embed_measure_eq_distr'*)

apply *simp*

```

apply(auto 4 3 intro!: measure_eqI imageI simp add: sets_embed_measure' subset_image_iff)
apply (subst (1 2) emeasure_distr)
apply (auto simp: space_embed_measure sets_embed_measure')
done

```

```

lemma embed_measure_count_space:
  inj f  $\implies$  embed_measure (count_space A) f = count_space (f`A)
by(rule embed_measure_count_space')(erule subset_inj_on, simp)

```

```

lemma sets_embed_measure_alt:
  inj f  $\implies$  sets (embed_measure M f) = (( $\cdot$ ) f) ` sets M
by (auto simp: sets_embed_measure)

```

```

lemma emeasure_embed_measure_image':
  assumes inj_on f (space M) X  $\in$  sets M
  shows emeasure (embed_measure M f) (f`X) = emeasure M (f -` f`X  $\cap$  space M)
proof-
  from assms have emeasure (embed_measure M f) (f`X) = emeasure M (f -` f`X  $\cap$  space M)
  by (subst emeasure_embed_measure') (auto simp: sets_embed_measure')
  also from assms have f -` f`X  $\cap$  space M = X by (auto dest: inj_onD sets.sets_into_space)
  finally show ?thesis .
qed

```

```

lemma emeasure_embed_measure_image:
  inj f  $\implies$  X  $\in$  sets M  $\implies$  emeasure (embed_measure M f) (f`X) = emeasure M X
by (simp_all add: emeasure_embed_measure in_sets_embed_measure inj_vimage_image_eq)

```

```

lemma embed_measure_eq_iff:
  assumes inj f
  shows embed_measure A f = embed_measure B f  $\longleftrightarrow$  A = B (is ?M = ?N  $\longleftrightarrow$  ?)
proof
  from assms have I: inj (( $\cdot$ ) f) by (auto intro: injI dest: injD)
  assume asm: ?M = ?N
  hence sets (embed_measure A f) = sets (embed_measure B f) by simp
  with assms have sets A = sets B by (simp only: I inj_image_eq_iff sets_embed_measure_alt)
  moreover {
    fix X assume X  $\in$  sets A
    from asm have emeasure ?M (f`X) = emeasure ?N (f`X) by simp
    with X  $\in$  sets A and sets A = sets B and assms
    have emeasure A X = emeasure B X by (simp add: emeasure_embed_measure_image)
  }
  ultimately show A = B by (rule measure_eqI)
qed simp

```

lemma *the_inv_into_in_Pi*: $\text{inj\_on } f \ A \implies \text{the\_inv\_into } A \ f \in f^{-1} A \rightarrow A$
by (*auto simp: the_inv_into_f_f*)

lemma *map_prod_image*: $\text{map\_prod } f \ g^{-1} (A \times B) = (f^{-1} A) \times (g^{-1} B)$
using *map_prod_surj_on[OF refl refl]* .

lemma *map_prod_vimage*: $\text{map\_prod } f \ g^{-1} (A \times B) = (f^{-1} A) \times (g^{-1} B)$
by *auto*

lemma *embed_measure_prod*:

assumes *f*: *inj f* **and** *g*: *inj g* **and** [*simp*]: *sigma_finite_measure M sigma_finite_measure N*

shows $\text{embed\_measure } M \ f \otimes_M \text{embed\_measure } N \ g = \text{embed\_measure } (M \otimes_M N) \ (\lambda(x, y). (f \ x, g \ y))$
(is ?L = _)

unfolding *map_prod_def[symmetric]*

proof (*rule pair_measure_eqI*)

have *fg[simp]*: $\bigwedge A. \text{inj\_on } (\text{map\_prod } f \ g) \ A \bigwedge A. \text{inj\_on } f \ A \bigwedge A. \text{inj\_on } g \ A$
using *f g* **by** (*auto simp: inj_on_def*)

note *complete_lattice_class.Sup_insert[simp del] ccSup_insert[simp del]*
ccSUP_insert[simp del]

show *sets*: $\text{sets } ?L = \text{sets } (\text{embed\_measure } (M \otimes_M N) (\text{map\_prod } f \ g))$

unfolding *map_prod_def[symmetric]*

apply (*simp add: sets_pair_eq_setsfst_snd sets_embed_eq_vimage_algebra cong: vimage_algebra_cong*)

apply (*subst sets_vimage_Sup_eq[where Y=space (M ⊗_M N)]*)

apply (*simp_all add: space_pair_measure[symmetric]*)

apply (*auto simp add: the_inv_into_f_f*

simp del: map_prod_simp

del: prod_fun_imageE) []

apply *auto* []

apply (*subst (1 2 3 4) vimage_algebra_vimage_algebra_eq*)

apply (*simp_all add: the_inv_into_in_Pi Pi_iff[of snd] Pi_iff[of fst] space_pair_measure*)

apply (*simp_all add: Pi_iff[of snd] Pi_iff[of fst] the_inv_into_in_Pi vimage_algebra_vimage_algebra_eq*

space_pair_measure[symmetric] map_prod_image[symmetric])

apply (*intro arg_cong[where f=sets] arg_cong[where f=Sup] arg_cong2[where f=insert] vimage_algebra_cong*)

apply (*auto simp: map_prod_image the_inv_into_f_f*

simp del: map_prod_simp del: prod_fun_imageE)

apply (*simp_all add: the_inv_into_f_f space_pair_measure*)

done

note *measurable_embed_measure2[measurable]*

fix *A B* **assume** *AB*: $A \in \text{sets } (\text{embed\_measure } M \ f) \ B \in \text{sets } (\text{embed\_measure } N \ g)$

moreover **have** $f^{-1} A \times g^{-1} B \cap \text{space } (M \otimes_M N) = (f^{-1} A \cap \text{space } M) \times (g^{-1} B \cap \text{space } N)$

```

  by (auto simp: space_pair_measure)
  ultimately show  $\text{emeasure} (\text{embed\_measure } M f) A * \text{emeasure} (\text{embed\_measure } N g) B =$ 
 $\text{emeasure} (\text{embed\_measure } (M \otimes_M N) (\text{map\_prod } f g)) (A \times B)$ 
  by (simp add: map_prod_vimage_sets[symmetric] emeasure_embed_measure
    sigma_finite_measure.emeasure_pair_measure_Times)
qed (insert assms, simp_all add: sigma_finite_embed_measure)

```

lemma mono_embed_measure:

```

  space  $M = \text{space } M' \implies \text{sets } M \subseteq \text{sets } M' \implies \text{sets} (\text{embed\_measure } M f) \subseteq$ 
 $\text{sets} (\text{embed\_measure } M' f)$ 
  unfolding embed_measure_def
  apply (subst (1 2) sets_measure_of)
  apply (blast dest: sets_sets_into_space)
  apply (blast dest: sets_sets_into_space)
  apply simp
  apply (intro sigma_sets_mono')
  apply safe
  apply (simp add: subset_eq)
  apply metis
done

```

lemma density_embed_measure:

```

  assumes inj:  $\text{inj } f$  and  $Mg[\text{measurable}]: g \in \text{borel\_measurable} (\text{embed\_measure } M f)$ 
  shows  $\text{density} (\text{embed\_measure } M f) g = \text{embed\_measure} (\text{density } M (g \circ f)) f$ 
  (is  $?M1 = ?M2$ )
  proof (rule measure_eqI)
    fix  $X$  assume  $X: X \in \text{sets } ?M1$ 
    from inj have  $Mf[\text{measurable}]: f \in \text{measurable } M (\text{embed\_measure } M f)$ 
    by (rule measurable_embed_measure2)
    from  $Mg$  and  $X$  have  $\text{emeasure } ?M1 X = \int^+ x. g x * \text{indicator } X x \partial \text{embed\_measure } M f$ 
    by (subst emeasure_density) simp_all
    also from  $X$  have  $\dots = \int^+ x. g (f x) * \text{indicator } X (f x) \partial M$ 
    by (subst embed_measure_eq_distr[OF inj], subst nn_integral_distr) auto
    also have  $\dots = \int^+ x. g (f x) * \text{indicator} (f - ' X \cap \text{space } M) x \partial M$ 
    by (intro nn_integral_cong) (auto split: split_indicator)
    also from  $X$  have  $\dots = \text{emeasure} (\text{density } M (g \circ f)) (f - ' X \cap \text{space } M)$ 
    by (subst emeasure_density) (simp_all add: measurable_comp[OF Mf Mg]
      measurable_sets[OF Mf])
    also from  $X$  and inj have  $\dots = \text{emeasure } ?M2 X$ 
    by (subst emeasure_embed_measure) (simp_all add: sets_embed_measure)
    finally show  $\text{emeasure } ?M1 X = \text{emeasure } ?M2 X$  .
  qed (simp_all add: sets_embed_measure inj)

```

lemma density_embed_measure':

```

  assumes inj:  $\text{inj } f$  and inv:  $\bigwedge x. f' (f x) = x$  and  $Mg[\text{measurable}]: g \in \text{borel\_measurable } M$ 

```

shows $\text{density } (\text{embed\_measure } M f) (g \circ f') = \text{embed\_measure } (\text{density } M g) f$
proof–
have $\text{density } (\text{embed\_measure } M f) (g \circ f') = \text{embed\_measure } (\text{density } M (g \circ f' \circ f)) f$
by (*rule density_embed_measure[OF inj]*)
(rule measurable_comp, rule measurable_embed_measure1, subst measurable_cong,
rule inv, rule measurable_ident_sets, simp, rule Mg)
also have $\text{density } M (g \circ f' \circ f) = \text{density } M g$
by (*intro density_cong*) (*subst measurable_cong, simp add: o_def inv, simp_all add: Mg inv*)
finally show *?thesis* .
qed

lemma *inj_on_image_subset_iff*:
assumes *inj_on f C A ⊆ C B ⊆ C*
shows $f' A \subseteq f' B \longleftrightarrow A \subseteq B$
proof (*intro iffI subsetI*)
fix *x* **assume** $A: f' A \subseteq f' B$ **and** $B: x \in A$
from *B* **have** $f x \in f' A$ **by** *blast*
with *A* **have** $f x \in f' B$ **by** *blast*
then obtain *y* **where** $f x = f y$ **and** $y \in B$ **by** *blast*
with *assms* **and** *B* **have** $x = y$ **by** (*auto dest: inj_onD*)
with $\langle y \in B \rangle$ **show** $x \in B$ **by** *simp*
qed *auto*

lemma *AE_embed_measure'*:
assumes *inj: inj_on f (space M)*
shows $(AE x \text{ in } \text{embed\_measure } M f. P x) \longleftrightarrow (AE x \text{ in } M. P (f x))$
proof
let $?M = \text{embed\_measure } M f$
assume $AE x \text{ in } ?M. P x$
then obtain *A* **where** $A\_props: A \in \text{sets } ?M \text{ emeasure } ?M A = 0 \{x \in \text{space } ?M. \neg P x\} \subseteq A$
by (*force elim: AE_E*)
then obtain *A'* **where** $A'_props: A = f' A' A' \in \text{sets } M$ **by** (*auto simp: sets_embed_measure' inj*)
moreover have $\{x \in \text{space } ?M. \neg P x\} = f' \{x \in \text{space } M. \neg P (f x)\}$
by (*auto simp: inj_space_embed_measure*)
from $A\_props(3)$ **have** $\{x \in \text{space } M. \neg P (f x)\} \subseteq A'$
by (*subst (asm) B, subst (asm) A'_props, subst (asm) inj_on_image_subset_iff[OF inj]*)
(insert A'_props, auto dest: sets_sets_into_space)
moreover from $A\_props A'_props$ **have** $\text{emeasure } M A' = 0$
by (*simp add: emeasure_embed_measure_image' inj*)
ultimately show $AE x \text{ in } M. P (f x)$ **by** (*intro AE_I*)
next
let $?M = \text{embed\_measure } M f$

```

  assume AE x in M. P (f x)
  then obtain A where A_props: A ∈ sets M emeasure M A = 0 {x ∈ space M.
¬P (f x)} ⊆ A
  by (force elim: AE_E)
  hence f'A ∈ sets ?M emeasure ?M (f'A) = 0 {x ∈ space ?M. ¬P x} ⊆ f'A
  by (auto simp: space_embed_measure emeasure_embed_measure_image' sets_embed_measure'
inj)
  thus AE x in ?M. P x by (intro AE_I)
qed

```

lemma AE_embed_measure:

```

  assumes inj: inj f
  shows (AE x in embed_measure M f. P x) ⟷ (AE x in M. P (f x))
  using assms by (intro AE_embed_measure') (auto intro!: inj_onI dest: injD)

```

lemma nn_integral_monotone_convergence_SUP_countable:

```

  fixes f :: 'a ⇒ 'b ⇒ ennreal
  assumes nonempty: Y ≠ {}
  and chain: Complete_Partial_Order.chain (≤) (f ' Y)
  and countable: countable B
  shows (∫+ x. (SUP i ∈ Y. f i x) ∂count_space B) = (SUP i ∈ Y. (∫+ x. f i x
∂count_space B))
  (is ?lhs = ?rhs)
proof -
  let ?f = (λi x. f i (from_nat_into B x) * indicator (to_nat_on B ' B) x)
  have ?lhs = ∫+ x. (SUP i ∈ Y. f i (from_nat_into B (to_nat_on B x)))
∂count_space B
  by (rule nn_integral_cong) (simp add: countable)
  also have ... = ∫+ x. (SUP i ∈ Y. f i (from_nat_into B x)) ∂count_space
(to_nat_on B ' B)
  by (simp add: embed_measure_count_space'[symmetric] inj_on_to_nat_on
countable nn_integral_embed_measure' measurable_embed_measure1)
  also have ... = ∫+ x. (SUP i ∈ Y. ?f i x) ∂count_space UNIV
  by (simp add: nn_integral_count_space_indicator ennreal_indicator[symmetric]
SUP_mult_right_ennreal nonempty)
  also have ... = (SUP i ∈ Y. ∫+ x. ?f i x ∂count_space UNIV)
proof (rule nn_integral_monotone_convergence_SUP_nat)
  show Complete_Partial_Order.chain (≤) (?f ' Y)
  by (rule chain_imageI[OF chain, unfolded image_image])(auto intro!: le_funI
split: split_indicator dest: le_funD)
qed fact
  also have ... = (SUP i ∈ Y. ∫+ x. f i (from_nat_into B x) ∂count_space
(to_nat_on B ' B))
  by (simp add: nn_integral_count_space_indicator)
  also have ... = (SUP i ∈ Y. ∫+ x. f i (from_nat_into B (to_nat_on B x))
∂count_space B)
  by (simp add: embed_measure_count_space'[symmetric] inj_on_to_nat_on
countable nn_integral_embed_measure' measurable_embed_measure1)
  also have ... = ?rhs

```

```

    by(intro arg_cong2[where f =  $\lambda A f. \text{Sup } (f \text{ ' } A)$ ] ext nn_integral_cong_AE)(simp_all
add: AE_count_space countable)
    finally show ?thesis .
qed

end

```

6.31 Brouwer's Fixed Point Theorem

```

theory Brouwer_Fixpoint
  imports Homeomorphism Derivative
begin

```

6.31.1 Retractions

```

lemma retract_of_contractible:
  assumes contractible T S retract_of T
  shows contractible S
using assms
apply (clarsimp simp add: retract_of_def contractible_def retraction_def homo-
topic_with)
apply (rule_tac x=r a in exI)
apply (rule_tac x=r  $\circ$  h in exI)
apply (intro conjI continuous_intros continuous_on_compose)
apply (erule continuous_on_subset | force)+
done

```

```

lemma retract_of_path_connected:
   $\llbracket \text{path\_connected } T; S \text{ retract\_of } T \rrbracket \implies \text{path\_connected } S$ 
  by (metis path_connected_continuous_image retract_of_def retraction)

```

```

lemma retract_of_simply_connected:
   $\llbracket \text{simply\_connected } T; S \text{ retract\_of } T \rrbracket \implies \text{simply\_connected } S$ 
  apply (simp add: retract_of_def retraction_def, clarify)
  apply (rule simply_connected_retraction_gen)
  apply (force elim!: continuous_on_subset)+
  done

```

```

lemma retract_of_homotopically_trivial:
  assumes ts: T retract_of S
  and hom:  $\bigwedge f g. \llbracket \text{continuous\_on } U f; f \text{ ' } U \subseteq S; \text{continuous\_on } U g; g \text{ ' } U \subseteq S \rrbracket$ 
            $\implies \text{homotopic\_with\_canon } (\lambda x. \text{True}) \ U \ S \ f \ g$ 
  and continuous_on U f f ' U  $\subseteq$  T
  and continuous_on U g g ' U  $\subseteq$  T
  shows homotopic_with_canon ( $\lambda x. \text{True}$ ) U T f g
proof -
  obtain r where r ' S  $\subseteq$  S continuous_on S r  $\forall x \in S. r \ (r \ x) = r \ x \ T = r \text{ ' } S$ 
  using ts by (auto simp: retract_of_def retraction)

```



```

then obtain k where Retracts S r T k
  unfolding Retracts_def
  by (metis continuous_on_subset dual_order.trans image_iff image_mono)
then show ?thesis
  apply (rule Retracts.homotopically_trivial_retraction_gen)
  using assms
  apply (force simp: hom)+
  done
qed

```

```

lemma retract_of_homotopically_trivial_null:
  assumes ts: T retract_of S
    and hom:  $\bigwedge f. \llbracket \text{continuous\_on } U f; f' U \subseteq S \rrbracket \implies \exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) U S f (\lambda x. c)$ 
    and continuous_on U f f' U  $\subseteq$  T
  obtains c where homotopic_with_canon ( $\lambda x. \text{True}$ ) U T f ( $\lambda x. c$ )
proof -
  obtain r where r' S  $\subseteq$  S continuous_on S r  $\forall x \in S. r (r x) = r x$  T = r' S
    using ts by (auto simp: retract_of_def retraction)
  then obtain k where Retracts S r T k
    unfolding Retracts_def
    by (metis continuous_on_subset dual_order.trans image_iff image_mono)
  then show ?thesis
    apply (rule Retracts.homotopically_trivial_retraction_null_gen)
    apply (rule TrueI refl assms that | assumption)+
    done
qed

```

```

lemma retraction_openin_vimage_iff:
  openin (top_of_set S) (S  $\cap$  r' U)  $\longleftrightarrow$  openin (top_of_set T) U
  if retraction: retraction S T r and U  $\subseteq$  T
  using retraction apply (rule retractionE)
  apply (rule continuous_right_inverse_imp_quotient_map [where g=r])
  using  $\langle U \subseteq T \rangle$  apply (auto elim: continuous_on_subset)
  done

```

```

lemma retract_of_locally_compact:
  fixes S :: 'a :: {heine_borel, real_normed_vector} set
  shows  $\llbracket \text{locally compact } S; T \text{ retract\_of } S \rrbracket \implies \text{locally compact } T$ 
  by (metis locally_compact_closedin closedin_retract)

```

```

lemma homotopic_into_retract:
   $\llbracket f' S \subseteq T; g' S \subseteq T; T \text{ retract\_of } U; \text{homotopic\_with\_canon } (\lambda x. \text{True}) S U f g \rrbracket$ 
 $\implies \text{homotopic\_with\_canon } (\lambda x. \text{True}) S T f g$ 
  apply (subst (asm) homotopic_with_def)
  apply (simp add: homotopic_with_retract_of_def retraction_def, clarify)
  apply (rule_tac x=r  $\circ$  h in exI)
  apply (rule conjI continuous_intros | erule continuous_on_subset | force simp:

```

```
image_subset_iff)+
done
```

```
lemma retract_of_locally_connected:
  assumes locally_connected T S retract_of T
  shows locally_connected S
  using assms
  by (auto simp: idempotent_imp_retraction intro!: retraction_openin_vimage_iff
    elim!: locally_connected_quotient_image retract_ofE)
```

```
lemma retract_of_locally_path_connected:
  assumes locally_path_connected T S retract_of T
  shows locally_path_connected S
  using assms
  by (auto simp: idempotent_imp_retraction intro!: retraction_openin_vimage_iff
    elim!: locally_path_connected_quotient_image retract_ofE)
```

A few simple lemmas about deformation retracts

```
lemma deformation_retract_imp_homotopy_eqv:
  fixes S :: 'a::euclidean_space set
  assumes homotopic_with_canon ( $\lambda x. \text{True}$ ) S S id r and r: retraction S T r
  shows S homotopy_eqv T
proof -
  have homotopic_with_canon ( $\lambda x. \text{True}$ ) S S (id  $\circ$  r) id
    by (simp add: assms(1) homotopic_with_symD)
  moreover have homotopic_with_canon ( $\lambda x. \text{True}$ ) T T (r  $\circ$  id) id
    using r unfolding retraction_def
    by (metis eq_id_iff homotopic_with_id2 topspace_euclidean_subtopology)
  ultimately
  show ?thesis
    unfolding homotopy_equivalent_space_def
    by (metis (no_types, lifting) continuous_map_subtopology_eu continuous_on_id'
      id_def image_id r retraction_def)
qed
```

```
lemma deformation_retract:
  fixes S :: 'a::euclidean_space set
  shows ( $\exists r. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S S \text{ id } r \wedge \text{retraction } S T r$ )
 $\longleftrightarrow$ 
 $T \text{ retract\_of } S \wedge (\exists f. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S S \text{ id } f \wedge f ' S \subseteq T)$ 
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (auto simp: retract_of_def retraction_def)
next
  assume ?rhs
  then show ?lhs
```

```

    apply (clarsimp simp add: retract_of_def retraction_def)
    apply (rule_tac x=r in exI, simp)
    apply (rule homotopic_with_trans, assumption)
    apply (rule_tac f = r ∘ f and g=r ∘ id in homotopic_with_eq)
    apply (rule_tac Y=S in homotopic_with_compose_continuous_left)
    apply (auto simp: homotopic_with_sym)
  done
qed

lemma deformation_retract_of_contractibleSing:
  fixes S :: 'a::euclidean_space set
  assumes contractible S a ∈ S
  obtains r where homotopic_with_canon (λx. True) S S id r retraction S {a} r
proof -
  have {a} retract_of S
  by (simp add: ⟨a ∈ S⟩)
  moreover have homotopic_with_canon (λx. True) S S id (λx. a)
  using assms
  by (auto simp: contractible_def homotopic_into_contractible image_subset_iff)
  moreover have (λx. a) ' S ⊆ {a}
  by (simp add: image_subsetI)
  ultimately show ?thesis
  using that deformation_retract by metis
qed

```

```

lemma continuous_on_compact_surface_projection_aux:
  fixes S :: 'a::t2_space set
  assumes compact S S ⊆ T image q T ⊆ S
    and contp: continuous_on T p
    and ∧x. x ∈ S ⇒ q x = x
    and [simp]: ∧x. x ∈ T ⇒ q(p x) = q x
    and ∧x. x ∈ T ⇒ p(q x) = p x
  shows continuous_on T q
proof -
  have *: image p T = image p S
  using assms by auto (metis imageI subset_iff)
  have contp': continuous_on S p
  by (rule continuous_on_subset [OF contp ⟨S ⊆ T⟩])
  have continuous_on (p ' T) q
  by (simp add: * assms(1) assms(2) assms(5) continuous_on_inv contp' rev_subsetD)
  then have continuous_on T (q ∘ p)
  by (rule continuous_on_compose [OF contp])
  then show ?thesis
  by (rule continuous_on_eq [of _ q ∘ p]) (simp add: o_def)
qed

```

```

lemma continuous_on_compact_surface_projection:
  fixes S :: 'a::real_normed_vector set

```

```

assumes compact S
  and S:  $S \subseteq V - \{0\}$  and cone V
  and iff:  $\bigwedge x k. x \in V - \{0\} \implies 0 < k \wedge (k *_R x) \in S \longleftrightarrow d x = k$ 
  shows continuous_on (V - {0}) ( $\lambda x. d x *_R x$ )
proof (rule continuous_on_compact_surface_projection_aux [OF ‹compact S›
S])
  show ( $\lambda x. d x *_R x$ ) ‘ (V - {0})  $\subseteq S$ 
    using iff by auto
  show continuous_on (V - {0}) ( $\lambda x. \text{inverse}(\text{norm } x) *_R x$ )
    by (intro continuous_intros) force
  show  $\bigwedge x. x \in S \implies d x *_R x = x$ 
    by (metis S zero_less_one local.iff scaleR_one subset_eq)
  show  $d (x /_R \text{norm } x) *_R (x /_R \text{norm } x) = d x *_R x$  if  $x \in V - \{0\}$  for x
    using iff [of inverse(norm x) *_R x norm x * d x, symmetric] iff that ‹cone V›
    by (simp add: field_simps cone_def zero_less_mult_iff)
  show  $d x *_R x /_R \text{norm } (d x *_R x) = x /_R \text{norm } x$  if  $x \in V - \{0\}$  for x
  proof -
    have  $0 < d x$ 
      using local.iff that by blast
    then show ?thesis
      by simp
  qed
qed

```

6.31.2 Kuhn Simplices

```

lemma bij_betw_singleton_eq:
  assumes f: bij_betw f A B and g: bij_betw g A B and a:  $a \in A$ 
  assumes eq: ( $\bigwedge x. x \in A \implies x \neq a \implies f x = g x$ )
  shows  $f a = g a$ 
proof -
  have  $f ‘ (A - \{a\}) = g ‘ (A - \{a\})$ 
    by (intro image_cong) (simp_all add: eq)
  then have  $B - \{f a\} = B - \{g a\}$ 
    using f g a by (auto simp: bij_betw_def inj_on_image_set_diff set_eq_iff)
  moreover have  $f a \in B$   $g a \in B$ 
    using f g a by (auto simp: bij_betw_def)
  ultimately show ?thesis
    by auto
qed

```

```

lemmas swap_apply1 = swap_apply(1)
lemmas swap_apply2 = swap_apply(2)

```

```

lemma pointwise_minimal_pointwise_maximal:
  fixes s :: (nat  $\Rightarrow$  nat) set
  assumes finite s
  and  $s \neq \{\}$ 
  and  $\forall x \in s. \forall y \in s. x \leq y \vee y \leq x$ 

```

```

shows  $\exists a \in s. \forall x \in s. a \leq x$ 
and  $\exists a \in s. \forall x \in s. x \leq a$ 
using assms
proof (induct s rule: finite_ne_induct)
  case (insert b s)
  assume *:  $\forall x \in \text{insert } b \ s. \forall y \in \text{insert } b \ s. x \leq y \vee y \leq x$ 
  then obtain u l where  $l \in s \ \forall b \in s. l \leq b \ u \in s \ \forall b \in s. b \leq u$ 
  using insert by auto
  with * show  $\exists a \in \text{insert } b \ s. \forall x \in \text{insert } b \ s. a \leq x \ \exists a \in \text{insert } b \ s. \forall x \in \text{insert } b \ s.$ 
 $x \leq a$ 
  using *[rule_format, of b u] *[rule_format, of b l] by (metis insert_iff order.trans)+
qed auto

```

```

lemma kuhn_labelling_lemma:
fixes P Q :: 'a::euclidean_space  $\Rightarrow$  bool
assumes  $\forall x. P \ x \longrightarrow P \ (f \ x)$ 
and  $\forall x. P \ x \longrightarrow (\forall i \in \text{Basis}. Q \ i \longrightarrow 0 \leq x \cdot i \wedge x \cdot i \leq 1)$ 
shows  $\exists l. (\forall x. \forall i \in \text{Basis}. l \ x \ i \leq (1::nat)) \wedge$ 
 $(\forall x. \forall i \in \text{Basis}. P \ x \wedge Q \ i \wedge (x \cdot i = 0) \longrightarrow (l \ x \ i = 0)) \wedge$ 
 $(\forall x. \forall i \in \text{Basis}. P \ x \wedge Q \ i \wedge (x \cdot i = 1) \longrightarrow (l \ x \ i = 1)) \wedge$ 
 $(\forall x. \forall i \in \text{Basis}. P \ x \wedge Q \ i \wedge (l \ x \ i = 0) \longrightarrow x \cdot i \leq f \ x \cdot i) \wedge$ 
 $(\forall x. \forall i \in \text{Basis}. P \ x \wedge Q \ i \wedge (l \ x \ i = 1) \longrightarrow f \ x \cdot i \leq x \cdot i)$ 
proof –
  { fix x i
    let ?R =  $\lambda y. (P \ x \wedge Q \ i \wedge x \cdot i = 0 \longrightarrow y = (0::nat)) \wedge$ 
 $(P \ x \wedge Q \ i \wedge x \cdot i = 1 \longrightarrow y = 1) \wedge$ 
 $(P \ x \wedge Q \ i \wedge y = 0 \longrightarrow x \cdot i \leq f \ x \cdot i) \wedge$ 
 $(P \ x \wedge Q \ i \wedge y = 1 \longrightarrow f \ x \cdot i \leq x \cdot i)$ 
    { assume  $P \ x \ Q \ i \ i \in \text{Basis}$  with assms have  $0 \leq f \ x \cdot i \wedge f \ x \cdot i \leq 1$  by
      auto }
    then have  $i \in \text{Basis} \Longrightarrow ?R \ 0 \vee ?R \ 1$  by auto }
  then show ?thesis
    unfolding all_conj_distrib[symmetric] Ball_def
    by (subst choice_iff[symmetric])+ blast
qed

```

The key "counting" observation, somewhat abstracted

```

lemma kuhn_counting_lemma:
fixes bnd compo compo' face S F
defines  $nF \ s == \text{card } \{f \in F. \text{face } f \ s \wedge \text{compo}' \ f\}$ 
assumes [simp, intro]: finite F — faces and [simp, intro]: finite S — simplices
and  $\bigwedge f. f \in F \Longrightarrow \text{bnd } f \Longrightarrow \text{card } \{s \in S. \text{face } f \ s\} = 1$ 
and  $\bigwedge f. f \in F \Longrightarrow \neg \text{bnd } f \Longrightarrow \text{card } \{s \in S. \text{face } f \ s\} = 2$ 
and  $\bigwedge s. s \in S \Longrightarrow \text{compo } s \Longrightarrow nF \ s = 1$ 
and  $\bigwedge s. s \in S \Longrightarrow \neg \text{compo } s \Longrightarrow nF \ s = 0 \vee nF \ s = 2$ 
and odd ( $\text{card } \{f \in F. \text{compo}' \ f \wedge \text{bnd } f\}$ )
shows odd ( $\text{card } \{s \in S. \text{compo } s\}$ )

```

proof –
have $(\sum s \mid s \in S \wedge \neg \text{compo } s. \text{ nF } s) + (\sum s \mid s \in S \wedge \text{compo } s. \text{ nF } s) =$
 $(\sum_{s \in S}. \text{ nF } s)$
by $(\text{subst } \text{sum.union\_disjoint}[\text{symmetric}]) (\text{auto intro!}: \text{sum.cong})$
also have $\dots = (\sum_{s \in S}. \text{card } \{f \in \{f \in F. \text{compo}' f \wedge \text{bnd } f\}. \text{face } f s\}) +$
 $(\sum_{s \in S}. \text{card } \{f \in \{f \in F. \text{compo}' f \wedge \neg \text{bnd } f\}. \text{face } f s\})$
unfolding $\text{sum.distrib}[\text{symmetric}]$
by $(\text{subst } \text{card\_Un\_disjoint}[\text{symmetric}])$
 $(\text{auto simp: nF\_def intro!}: \text{sum.cong } \text{arg\_cong}[\text{where } f = \text{card}])$
also have $\dots = 1 * \text{card } \{f \in F. \text{compo}' f \wedge \text{bnd } f\} + 2 * \text{card } \{f \in F. \text{compo}' f$
 $\wedge \neg \text{bnd } f\}$
using $\text{assms}(4,5)$ **by** $(\text{fastforce intro!}: \text{arg\_cong2}[\text{where } f = (+)] \text{sum\_multicount})$
finally have $\text{odd } ((\sum s \mid s \in S \wedge \neg \text{compo } s. \text{ nF } s) + \text{card } \{s \in S. \text{compo } s\})$
using $\text{assms}(6,8)$ **by** simp
moreover have $(\sum s \mid s \in S \wedge \neg \text{compo } s. \text{ nF } s) =$
 $(\sum s \mid s \in S \wedge \neg \text{compo } s \wedge \text{ nF } s = 0. \text{ nF } s) + (\sum s \mid s \in S \wedge \neg \text{compo } s \wedge$
 $\text{ nF } s = 2. \text{ nF } s)$
using $\text{assms}(7)$ **by** $(\text{subst } \text{sum.union\_disjoint}[\text{symmetric}]) (\text{fastforce intro!}: \text{sum.cong}) +$
ultimately show $?thesis$
by auto
qed

The odd/even result for faces of complete vertices, generalized

lemma $\text{kuhn\_complete\_lemma}$:

assumes $[\text{simp}]$: finite simplices
and face : $\bigwedge f s. \text{face } f s \longleftrightarrow (\exists a \in s. f = s - \{a\})$
and $\text{card\_s}[\text{simp}]$: $\bigwedge s. s \in \text{simplices} \implies \text{card } s = n + 2$
and rl_bd : $\bigwedge s. s \in \text{simplices} \implies \text{rl } 's \subseteq \{.. \text{Suc } n\}$
and bnd : $\bigwedge f s. s \in \text{simplices} \implies \text{face } f s \implies \text{bnd } f \implies \text{card } \{s \in \text{simplices}. \text{face}$
 $f s\} = 1$
and nbnd : $\bigwedge f s. s \in \text{simplices} \implies \text{face } f s \implies \neg \text{bnd } f \implies \text{card } \{s \in \text{simplices}. \text{face } f s\} = 2$
and odd_card : $\text{odd } (\text{card } \{f. (\exists s \in \text{simplices}. \text{face } f s) \wedge \text{rl } 'f = \{..n\} \wedge \text{bnd } f\})$
shows $\text{odd } (\text{card } \{s \in \text{simplices}. (\text{rl } 's = \{.. \text{Suc } n\})\})$

proof $(\text{rule } \text{kuhn\_counting\_lemma})$

have $\text{finite\_s}[\text{simp}]$: $\bigwedge s. s \in \text{simplices} \implies \text{finite } s$
by $(\text{metis add\_is\_0 zero\_neq\_numeral card.infinite assms}(3))$

let $?F = \{f. \exists s \in \text{simplices}. \text{face } f s\}$

have F_eq : $?F = (\bigcup s \in \text{simplices}. \bigcup a \in s. \{s - \{a\}\})$

by (auto simp: face)

show $\text{finite } ?F$

using $\langle \text{finite simplices} \rangle$ **unfolding** F_eq **by** auto

show $\text{card } \{s \in \text{simplices}. \text{face } f s\} = 1$ **if** $f \in ?F$ **bnd** f **for** f

using bnd that **by** auto

```

show card {s ∈ simplices. face f s} = 2 if f ∈ ?F ¬ bnd f for f
using nbnd that by auto

show odd (card {f ∈ {f. ∃ s ∈ simplices. face f s}. rl ' f = {..n} ∧ bnd f})
using odd_card by simp

fix s assume s[simp]: s ∈ simplices
let ?S = {f ∈ {f. ∃ s ∈ simplices. face f s}. face f s ∧ rl ' f = {..n}}
have ?S = (λa. s - {a}) ' {a ∈ s. rl ' (s - {a}) = {..n}}
using s by (fastforce simp: face)
then have card_S: card ?S = card {a ∈ s. rl ' (s - {a}) = {..n}}
by (auto intro!: card_image inj_onI)

{ assume rl: rl ' s = {..Suc n}
  then have inj_rl: inj_on rl s
    by (intro eq_card_imp_inj_on) auto
  moreover obtain a where rl a = Suc n a ∈ s
    by (metis atMost_iff image_iff le_Suc_eq rl)
  ultimately have n: {..n} = rl ' (s - {a})
    by (auto simp: inj_on_image_set_diff rl)
  have {a ∈ s. rl ' (s - {a}) = {..n}} = {a}
    using inj_rl ⟨a ∈ s⟩ by (auto simp: n inj_on_image_eq_iff[OF inj_rl])
  then show card ?S = 1
    unfolding card_S by simp }

{ assume rl: rl ' s ≠ {..Suc n}
  show card ?S = 0 ∨ card ?S = 2
  proof cases
    assume *: {..n} ⊆ rl ' s
    with rl_rl_bd[OF s] have rl_s: rl ' s = {..n}
      by (auto simp: atMost_Suc subset_insert_iff split: if_split_asm)
    then have ¬ inj_on rl s
      by (intro pigeonhole) simp
    then obtain a b where ab: a ∈ s b ∈ s rl a = rl b a ≠ b
      by (auto simp: inj_on_def)
    then have eq: rl ' (s - {a}) = rl ' s
      by auto
    with ab have inj: inj_on rl (s - {a})
      by (intro eq_card_imp_inj_on) (auto simp: rl_s card_Diff_singleton_if)

    { fix x assume x ∈ s x ∉ {a, b}
      then have rl ' s - {rl x} = rl ' ((s - {a}) - {x})
        by (auto simp: eq inj_on_image_set_diff[OF inj])
      also have ... = rl ' (s - {x})
        using ab ⟨x ∉ {a, b}⟩ by auto
      also assume ... = rl ' s
      finally have False
        using ⟨x ∈ s⟩ by auto }
    moreover

```

```

{ fix x assume x ∈ {a, b} with ab have x ∈ s ∧ rl ' (s - {x}) = rl ' s
  by (simp add: set_eq_iff image_iff Bex_def) metis }
ultimately have {a ∈ s. rl ' (s - {a}) = {..n}} = {a, b}
  unfolding rl_s[symmetric] by fastforce
with ⟨a ≠ b⟩ show card ?S = 0 ∨ card ?S = 2
  unfolding card_S by simp
next
assume ¬ {..n} ⊆ rl ' s
then have ∧x. rl ' (s - {x}) ≠ {..n}
  by auto
then show card ?S = 0 ∨ card ?S = 2
  unfolding card_S by simp
qed }
qed fact

locale kuhn_simplex =
  fixes p n and base upd and s :: (nat ⇒ nat) set
  assumes base: base ∈ {..< n} → {..< p}
  assumes base_out: ∧i. n ≤ i ⇒ base i = p
  assumes upd: bij_betw upd {..< n} {..< n}
  assumes s_pre: s = (λi j. if j ∈ upd' {..< i} then Suc (base j) else base j) ' {..
n}
begin

definition enum i j = (if j ∈ upd' {..< i} then Suc (base j) else base j)

lemma s_eq: s = enum ' {.. n}
  unfolding s_pre enum_def[abs_def] ..

lemma upd_space: i < n ⇒ upd i < n
  using upd by (auto dest!: bij_betwE)

lemma s_space: s ⊆ {..< n} → {.. p}
proof -
{ fix i assume i ≤ n then have enum i ∈ {..< n} → {.. p}
  proof (induct i)
    case 0 then show ?case
      using base by (auto simp: Pi_iff less_imp_le enum_def)
  next
    case (Suc i) with base show ?case
      by (auto simp: Pi_iff Suc_le_eq less_imp_le enum_def intro: upd_space)
  qed }
then show ?thesis
  by (auto simp: s_eq)
qed

lemma inj_upd: inj_on upd {..< n}
  using upd by (simp add: bij_betw_def)

```



```

lemma inj_enum: inj_on enum {.. $n$ }
proof -
  { fix  $x\ y :: nat$  assume  $x \neq y\ x \leq n\ y \leq n$ 
    with upd have upd ' $\{.. $x$ \} \neq \{.. $y$ \}$ 
    by (subst inj_on_image_eq_iff[where  $C = \{.. $n$ \}$ ]) (auto simp: bij_betw_def)
    then have enum  $x \neq$  enum  $y$ 
    by (auto simp: enum_def fun_eq_iff) }
  then show ?thesis
  by (auto simp: inj_on_def)
qed

lemma enum_0: enum 0 = base
  by (simp add: enum_def[abs_def])

lemma base_in_s: base  $\in$  s
  unfolding s_eq by (subst enum_0[symmetric]) auto

lemma enum_in:  $i \leq n \implies$  enum  $i \in$  s
  unfolding s_eq by auto

lemma one_step:
  assumes  $a: a \in s\ j < n$ 
  assumes *:  $\bigwedge a'. a' \in s \implies a' \neq a \implies a' j = p'$ 
  shows  $a j \neq p'$ 
proof
  assume  $a j = p'$ 
  with * a have  $\bigwedge a'. a' \in s \implies a' j = p'$ 
  by auto
  then have  $\bigwedge i. i \leq n \implies$  enum  $i j = p'$ 
  unfolding s_eq by auto
  from this[of 0] this[of  $n$ ] have  $j \notin$  upd ' $\{.. $n$ \}$ 
  by (auto simp: enum_def fun_eq_iff split: if_split_asm)
  with upd ' $j < n$ ' show False
  by (auto simp: bij_betw_def)
qed

lemma upd_inj:  $i < n \implies j < n \implies$  upd  $i =$  upd  $j \longleftrightarrow i = j$ 
  using upd by (auto simp: bij_betw_def inj_on_eq_iff)

lemma upd_surj: upd ' $\{.. $n$ \} = \{.. $n$ \}$ 
  using upd by (auto simp: bij_betw_def)

lemma in_upd_image:  $A \subseteq \{.. $n$ \} \implies i < n \implies$  upd  $i \in$  upd ' $A \longleftrightarrow i \in A$ 
  using inj_on_image_mem_iff[of upd ' $\{.. $n$ \}$ ] upd
  by (auto simp: bij_betw_def)

lemma enum_inj:  $i \leq n \implies j \leq n \implies$  enum  $i =$  enum  $j \longleftrightarrow i = j$ 
  using inj_enum by (auto simp: inj_on_eq_iff)

```

lemma *in_enum_image*: $A \subseteq \{.. n\} \implies i \leq n \implies \text{enum } i \in \text{enum } A \iff i \in A$

using *inj_on_image_mem_iff*[*OF inj_enum*] **by** *auto*

lemma *enum_mono*: $i \leq n \implies j \leq n \implies \text{enum } i \leq \text{enum } j \iff i \leq j$
by (*auto simp: enum_def le_fun_def in_upd_image Ball_def[symmetric]*)

lemma *enum_strict_mono*: $i \leq n \implies j \leq n \implies \text{enum } i < \text{enum } j \iff i < j$
using *enum_mono*[*of i j*] *enum_inj*[*of i j*] **by** (*auto simp: le_less*)

lemma *chain*: $a \in s \implies b \in s \implies a \leq b \vee b \leq a$
by (*auto simp: s_eq enum_mono*)

lemma *less*: $a \in s \implies b \in s \implies a < b \iff a < b$
using *chain*[*of a b*] **by** (*auto simp: less_fun_def le_fun_def not_le[symmetric]*)

lemma *enum_0_bot*: $a \in s \implies a = \text{enum } 0 \iff (\forall a' \in s. a \leq a')$
unfolding *s_eq* **by** (*auto simp: enum_mono Ball_def*)

lemma *enum_n_top*: $a \in s \implies a = \text{enum } n \iff (\forall a' \in s. a' \leq a)$
unfolding *s_eq* **by** (*auto simp: enum_mono Ball_def*)

lemma *enum_Suc*: $i < n \implies \text{enum } (\text{Suc } i) = (\text{enum } i)(\text{upd } i := \text{Suc } (\text{enum } i (\text{upd } i)))$
by (*auto simp: fun_eq_iff enum_def upd_inj*)

lemma *enum_eq_p*: $i \leq n \implies n \leq j \implies \text{enum } i j = p$
by (*induct i*) (*auto simp: enum_Suc enum_0 base_out upd_space not_less[symmetric]*)

lemma *out_eq_p*: $a \in s \implies n \leq j \implies a j = p$
unfolding *s_eq* **by** (*auto simp: enum_eq_p*)

lemma *s_le_p*: $a \in s \implies a j \leq p$
using *out_eq_p*[*of a j*] *s_space* **by** (*cases j < n*) *auto*

lemma *le_Suc_base*: $a \in s \implies a j \leq \text{Suc } (\text{base } j)$
unfolding *s_eq* **by** (*auto simp: enum_def*)

lemma *base_le*: $a \in s \implies \text{base } j \leq a j$
unfolding *s_eq* **by** (*auto simp: enum_def*)

lemma *enum_le_p*: $i \leq n \implies j < n \implies \text{enum } i j \leq p$
using *enum_in*[*of i*] *s_space* **by** *auto*

lemma *enum_less*: $a \in s \implies i < n \implies \text{enum } i < a \iff \text{enum } (\text{Suc } i) \leq a$
unfolding *s_eq* **by** (*auto simp: enum_strict_mono enum_mono*)

lemma *ksimplex_0*:
 $n = 0 \implies s = \{(\lambda x. p)\}$

using s_eq enum_def base_out by auto

lemma replace_0:

assumes $j < n$ $a \in s$ and $p: \forall x \in s - \{a\}. x j = 0$ and $x \in s$
shows $x \leq a$

proof cases

assume $x \neq a$

have $a j \neq 0$

using assms by (intro one_step[where $a=a$]) auto

with less[OF $\langle x \in s \rangle \langle a \in s \rangle$, of j] p[rule_format, of x] $\langle x \in s \rangle \langle x \neq a \rangle$

show ?thesis

by auto

qed simp

lemma replace_1:

assumes $j < n$ $a \in s$ and $p: \forall x \in s - \{a\}. x j = p$ and $x \in s$
shows $a \leq x$

proof cases

assume $x \neq a$

have $a j \neq p$

using assms by (intro one_step[where $a=a$]) auto

with enum_le_p[of j] $\langle j < n \rangle \langle a \in s \rangle$

have $a j < p$

by (auto simp: less_le s_eq)

with less[OF $\langle a \in s \rangle \langle x \in s \rangle$, of j] p[rule_format, of x] $\langle x \in s \rangle \langle x \neq a \rangle$

show ?thesis

by auto

qed simp

end

locale kuhn_simplex_pair = $s: \text{kuhn\_simplex } p \ n \ b\_s \ u\_s \ s + t: \text{kuhn\_simplex}$
 $p \ n \ b\_t \ u\_t \ t$

for $p \ n \ b\_s \ u\_s \ s \ b\_t \ u\_t \ t$

begin

lemma enum_eq:

assumes $l: i \leq l \ l \leq j$ and $j + d \leq n$

assumes eq: $s.enum \ ' \{i .. j\} = t.enum \ ' \{i + d .. j + d\}$

shows $s.enum \ l = t.enum \ (l + d)$

using l proof (induct l rule: dec_induct)

case base

then have $s: s.enum \ i \in t.enum \ ' \{i + d .. j + d\}$ and $t: t.enum \ (i + d) \in$
 $s.enum \ ' \{i .. j\}$

using eq by auto

from t $\langle i \leq j \rangle \langle j + d \leq n \rangle$ have $s.enum \ i \leq t.enum \ (i + d)$

by (auto simp: s.enum_mono)

moreover from s $\langle i \leq j \rangle \langle j + d \leq n \rangle$ have $t.enum \ (i + d) \leq s.enum \ i$

by (auto simp: t.enum_mono)

```

ultimately show ?case
  by auto
next
case (step l)
moreover from step.premis ⟨j + d ≤ n⟩ have
  s.enum l < s.enum (Suc l)
  t.enum (l + d) < t.enum (Suc l + d)
  by (simp_all add: s.enum_strict_mono t.enum_strict_mono)
moreover have
  s.enum (Suc l) ∈ t.enum ‘ {i + d .. j + d}
  t.enum (Suc l + d) ∈ s.enum ‘ {i .. j}
  using step ⟨j + d ≤ n⟩ eq by (auto simp: s.enum_inj t.enum_inj)
ultimately have s.enum (Suc l) = t.enum (Suc (l + d))
  using ⟨j + d ≤ n⟩
  by (intro antisym s.enum_less[THEN iffD1] t.enum_less[THEN iffD1])
    (auto intro!: s.enum_in t.enum_in)
then show ?case by simp
qed

lemma ksimpler_eq_bot:
  assumes a: a ∈ s ∧ a' ∈ s ⇒ a ≤ a'
  assumes b: b ∈ t ∧ b' ∈ t ⇒ b ≤ b'
  assumes eq: s - {a} = t - {b}
  shows s = t
proof cases
  assume n = 0 with s.ksimpler_0 t.ksimpler_0 show ?thesis by simp
next
  assume n ≠ 0
  have s.enum 0 = (s.enum (Suc 0)) (u_s 0 := s.enum (Suc 0) (u_s 0) - 1)
    t.enum 0 = (t.enum (Suc 0)) (u_t 0 := t.enum (Suc 0) (u_t 0) - 1)
    using ⟨n ≠ 0⟩ by (simp_all add: s.enum_Suc t.enum_Suc)
  moreover have e0: a = s.enum 0 b = t.enum 0
    using a b by (simp_all add: s.enum_0_bot t.enum_0_bot)
  moreover
  { fix j assume 0 < j j ≤ n
    moreover have s - {a} = s.enum ‘ {Suc 0 .. n} t - {b} = t.enum ‘ {Suc 0
    .. n}
    unfolding s.s_eq t.s_eq e0 by (auto simp: s.enum_inj t.enum_inj)
    ultimately have s.enum j = t.enum j
      using enum_eq[of 1 j n 0] eq by auto }
  note enum_eq = this
  then have s.enum (Suc 0) = t.enum (Suc 0)
    using ⟨n ≠ 0⟩ by auto
  moreover
  { fix j assume Suc j < n
    with enum_eq[of Suc j] enum_eq[of Suc (Suc j)]
    have u_s (Suc j) = u_t (Suc j)
      using s.enum_Suc[of Suc j] t.enum_Suc[of Suc j]
      by (auto simp: fun_eq_iff split: if_split_asm) }

```

```

then have  $\bigwedge j. 0 < j \implies j < n \implies u\_s\ j = u\_t\ j$ 
  by (auto simp: gr0_conv_Suc)
with  $\langle n \neq 0 \rangle$  have  $u\_t\ 0 = u\_s\ 0$ 
  by (intro bij_betw_singleton_eq[OF t.upd s.upd, of 0]) auto
ultimately have  $a = b$ 
  by simp
with assms show  $s = t$ 
  by auto
qed

```

lemma *ksimplex_eq_top*:

```

assumes  $a: a \in s \bigwedge a'. a' \in s \implies a' \leq a$ 
assumes  $b: b \in t \bigwedge b'. b' \in t \implies b' \leq b$ 
assumes  $eq: s - \{a\} = t - \{b\}$ 
shows  $s = t$ 
proof (cases n)
  assume  $n = 0$  with  $s.ksimplex\_0\ t.ksimplex\_0$  show ?thesis by simp
next
  case (Suc n')
  have  $s.enum\ n = (s.enum\ n')\ (u\_s\ n' := Suc\ (s.enum\ n'\ (u\_s\ n')))$ 
     $t.enum\ n = (t.enum\ n')\ (u\_t\ n' := Suc\ (t.enum\ n'\ (u\_t\ n')))$ 
  using Suc by (simp_all add: s.enum_Suc t.enum_Suc)
  moreover have  $en: a = s.enum\ n\ b = t.enum\ n$ 
    using  $a\ b$  by (simp_all add: s.enum_n_top t.enum_n_top)
  moreover
  { fix j assume  $j < n$ 
    moreover have  $s - \{a\} = s.enum\ ' \{0 .. n'\}\ t - \{b\} = t.enum\ ' \{0 .. n'\}$ 
      unfolding s.s_eq t.s_eq en by (auto simp: s.enum_inj t.enum_inj Suc)
    ultimately have  $s.enum\ j = t.enum\ j$ 
      using enum_eq[of 0 j n' 0] eq Suc by auto }
  note enum_eq = this
  then have  $s.enum\ n' = t.enum\ n'$ 
    using Suc by auto
  moreover
  { fix j assume  $j < n'$ 
    with enum_eq[of j] enum_eq[of Suc j]
    have  $u\_s\ j = u\_t\ j$ 
      using s.enum_Suc[of j] t.enum_Suc[of j]
      by (auto simp: Suc fun_eq_iff split: if_split_asm) }
  then have  $\bigwedge j. j < n' \implies u\_s\ j = u\_t\ j$ 
    by (auto simp: gr0_conv_Suc)
  then have  $u\_t\ n' = u\_s\ n'$ 
    by (intro bij_betw_singleton_eq[OF t.upd s.upd, of n']) (auto simp: Suc)
  ultimately have  $a = b$ 
    by simp
  with assms show  $s = t$ 
    by auto
qed

```

end

inductive *ksimplex* **for** $p\ n :: \text{nat}$ **where**

ksimplex: $\text{kuhn\_simplex } p\ n\ \text{base upd } s \implies \text{ksimplex } p\ n\ s$

lemma *finite_ksimplexes*: $\text{finite } \{s. \text{ksimplex } p\ n\ s\}$

proof (*rule finite_subset*)

{ **fix** $a\ s$ **assume** $\text{ksimplex } p\ n\ s\ a \in s$

then obtain $b\ u$ **where** $\text{kuhn\_simplex } p\ n\ b\ u\ s$ **by** (*auto elim: ksimplex.cases*)

then interpret $\text{kuhn\_simplex } p\ n\ b\ u\ s$.

from $s\_space\ \langle a \in s \rangle\ \text{out\_eq\_p}[OF\ \langle a \in s \rangle]$

have $a \in (\lambda f\ x. \text{if } n \leq x \text{ then } p \text{ else } f\ x)\ '(\{..< n\} \rightarrow_E \{.. p\})$

by (*auto simp: image_iff subset_eq Pi_iff split: if_split_asm*

intro!: $\text{bexI}[of\_restrict\ a\ \{..< n\}]$) }

then show $\{s. \text{ksimplex } p\ n\ s\} \subseteq \text{Pow } ((\lambda f\ x. \text{if } n \leq x \text{ then } p \text{ else } f\ x)\ '(\{..< n\} \rightarrow_E \{.. p\}))$

by *auto*

qed (*simp add: finite_PiE*)

lemma *ksimplex_card*:

assumes $\text{ksimplex } p\ n\ s$ **shows** $\text{card } s = \text{Suc } n$

using *assms* **proof** *cases*

case ($\text{ksimplex } u\ b$)

then interpret $\text{kuhn\_simplex } p\ n\ u\ b\ s$.

show *?thesis*

by (*simp add: card_image s_eq inj_enum*)

qed

lemma *simplex_top_face*:

assumes $0 < p\ \forall x \in s'. x\ n = p$

shows $\text{ksimplex } p\ n\ s' \longleftrightarrow (\exists s\ a. \text{ksimplex } p\ (\text{Suc } n)\ s \wedge a \in s \wedge s' = s - \{a\})$

using *assms*

proof *safe*

fix $s\ a$ **assume** $\text{ksimplex } p\ (\text{Suc } n)\ s$ **and** $a: a \in s$ **and** $na: \forall x \in s - \{a\}. x\ n = p$

then show $\text{ksimplex } p\ n\ (s - \{a\})$

proof *cases*

case ($\text{ksimplex } \text{base upd}$)

then interpret $\text{kuhn\_simplex } p\ \text{Suc } n\ \text{base upd } s$.

have $a\ n < p$

using *one_step[of a n p] na <a∈s> s_space* **by** (*auto simp: less_le*)

then have $a = \text{enum } 0$

using $\langle a \in s \rangle\ na$ **by** (*subst enum_0_bot*) (*auto simp: le_less intro!: less[of a _ n]*)

then have $s\_eq: s - \{a\} = \text{enum } ' \text{Suc } ' \{.. n\}$

using s_eq **by** (*simp add: atMost_Suc_eq_insert_0 insert_ident in_enum_image subset_eq*)

then have $\text{enum } 1 \in s - \{a\}$

```

    by auto
  then have upd 0 = n
    using ⟨a n < p⟩ ⟨a = enum 0⟩ na[rule_format, of enum 1]
    by (auto simp: fun_eq_iff enum_Suc split: if_split_asm)
  then have bij_betw upd (Suc ‘{.. $n$ }) {.. $n$ }
    using upd
    by (subst notIn_Un_bij_betw3[where b=0])
      (auto simp: lessThan_Suc[symmetric] lessThan_Suc_eq_insert_0)
  then have bij_betw (upd ∘ Suc) {.. $n$ } {.. $n$ }
    by (rule bij_betw_trans[rotated]) (auto simp: bij_betw_def)

  have a n = p - 1
    using enum_Suc[of 0] na[rule_format, OF ⟨enum 1 ∈ s - {a}⟩] ⟨a = enum
0⟩ by (auto simp: ⟨upd 0 = n⟩)

  show ?thesis
  proof (rule ksimplex.intros, standard)
    show bij_betw (upd ∘ Suc) {.. $n$ } {.. $n$ } by fact
    show base(n := p) ∈ {.. $n$ } → {.. $p$ } ∧ i. n ≤ i ⇒ (base(n := p)) i = p
      using base base_out by (auto simp: Pi_iff)

    have ∧i. Suc ‘{.. $i$ } = {.. $Suc i$ } - {0}
      by (auto simp: image_iff Ball_def) arith
    then have upd_Suc: ∧i. i ≤ n ⇒ (upd ∘ Suc) ‘{.. $i$ } = upd ‘{.. $Suc i$ }
      - {n}
      using ⟨upd 0 = n⟩ upd_inj by (auto simp add: image_iff less_Suc_eq_0_disj)
    have n_in_upd: ∧i. n ∈ upd ‘{.. $Suc i$ }
      using ⟨upd 0 = n⟩ by auto

    define f' where f' i j =
      (if j ∈ (upd ∘ Suc) ‘{.. $i$ } then Suc ((base(n := p)) j) else (base(n := p)) j)
    for i j
    { fix x i
      assume i [arith]: i ≤ n
      with upd_Suc have (upd ∘ Suc) ‘{.. $i$ } = upd ‘{.. $Suc i$ } - {n} .
      with ⟨a n < p⟩ ⟨a = enum 0⟩ ⟨upd 0 = n⟩ ⟨a n = p - 1⟩
      have enum (Suc i) x = f' i x
        by (auto simp add: f'_def enum_def) }
    then show s - {a} = f' ‘{.. $n$ }
      unfolding s_eq_image_comp by (intro image_cong) auto
    qed
  qed
next
  assume ksimplex p n s' and *: ∀ x ∈ s'. x n = p
  then show ∃ s a. ksimplex p (Suc n) s ∧ a ∈ s ∧ s' = s - {a}
  proof cases
    case (ksimplex base upd)
    then interpret kuhn_simplex p n base upd s' .
    define b where b = base (n := p - 1)

```

```

define u where u i = (case i of 0 ⇒ n | Suc i ⇒ upd i) for i

have ksimplex p (Suc n) (s' ∪ {b})
proof (rule ksimplex.intros, standard)
  show b ∈ {.. $Suc\ n$ } → {.. $p$ }
  using base ⟨0 < p⟩ unfolding lessThan_Suc b_def by (auto simp: PiE_iff)
  show  $\bigwedge i. Suc\ n \leq i \implies b\ i = p$ 
  using base_out by (auto simp: b_def)

  have bij_betw u (Suc ' {.. $n$ } ∪ {0}) ({.. $n$ } ∪ {u 0})
  using upd
  by (intro notIn_Un_bij_betw) (auto simp: u_def bij_betw_def image_comp
comp_def inj_on_def)
  then show bij_betw u {.. $Suc\ n$ } {.. $Suc\ n$ }
  by (simp add: u_def lessThan_Suc[symmetric] lessThan_Suc_eq_insert_0)

define f' where f' i j = (if j ∈ u' {.. $i$ } then Suc (b j) else b j) for i j

have u_eq:  $\bigwedge i. i \leq n \implies u\ ' \{.. $Suc\ i$ \} = upd\ ' \{.. $i$ \} \cup \{n\}$ 
  by (auto simp: u_def image_iff upd_inj Ball_def split: nat.split) arith

{ fix x have  $x \leq n \implies n \notin upd\ ' \{.. $x$ \}$ 
  using upd_space by (simp add: image_iff neq_iff) }
note n_not_upd = this

have *: f' ' {.. $Suc\ n$ } = f' ' (Suc ' {.. $n$ } ∪ {0})
  unfolding atMost_Suc_eq_insert_0 by simp
also have ... = (f' ∘ Suc) ' {.. $n$ } ∪ {b}
  by (auto simp: f'_def)
also have (f' ∘ Suc) ' {.. $n$ } = s'
  using ⟨0 < p⟩ base_out[of n]
  unfolding s_eq_enum_def[abs_def] f'_def[abs_def] upd_space
  by (intro image_cong) (simp_all add: u_eq b_def fun_eq_iff n_not_upd)
finally show s' ∪ {b} = f' ' {.. $Suc\ n$ } ..
qed
moreover have b ∉ s'
  using * ⟨0 < p⟩ by (auto simp: b_def)
ultimately show ?thesis by auto
qed
qed

lemma ksimplex_replace_0:
  assumes s: ksimplex p n s and a: a ∈ s
  assumes j: j < n and p:  $\forall x \in s - \{a\}. x\ j = 0$ 
  shows card {s'. ksimplex p n s' ∧ ( $\exists b \in s'. s' - \{b\} = s - \{a\}$ )} = 1
  using s
proof cases
  case (ksimplex b_s u_s)

```



```

{ fix t b assume ksimplex p n t
  then obtain b_t u_t where kuhn_simplex p n b_t u_t t
    by (auto elim: ksimplex.cases)
  interpret kuhn_simplex_pair p n b_s u_s s b_t u_t t
    by intro_locales fact+

  assume b:  $b \in t - \{b\} = s - \{a\}$ 
  with a j p s.replace_0[of _ a] t.replace_0[of _ b] have  $s = t$ 
    by (intro ksimplex_eq_top[of a b]) auto }
then have  $\{s'. \text{ksimplex } p \ n \ s' \wedge (\exists b \in s'. s' - \{b\} = s - \{a\})\} = \{s\}$ 
  using  $s \langle a \in s \rangle$  by auto
then show ?thesis
  by simp
qed

```

```

lemma ksimplex_replace_1:
  assumes s: ksimplex p n s and a:  $a \in s$ 
  assumes j:  $j < n$  and p:  $\forall x \in s - \{a\}. x \ j = p$ 
  shows  $\text{card } \{s'. \text{ksimplex } p \ n \ s' \wedge (\exists b \in s'. s' - \{b\} = s - \{a\})\} = 1$ 
  using s
proof cases
  case (ksimplex b_s u_s)

```

```

{ fix t b assume ksimplex p n t
  then obtain b_t u_t where kuhn_simplex p n b_t u_t t
    by (auto elim: ksimplex.cases)
  interpret kuhn_simplex_pair p n b_s u_s s b_t u_t t
    by intro_locales fact+

  assume b:  $b \in t - \{b\} = s - \{a\}$ 
  with a j p s.replace_1[of _ a] t.replace_1[of _ b] have  $s = t$ 
    by (intro ksimplex_eq_bot[of a b]) auto }
then have  $\{s'. \text{ksimplex } p \ n \ s' \wedge (\exists b \in s'. s' - \{b\} = s - \{a\})\} = \{s\}$ 
  using  $s \langle a \in s \rangle$  by auto
then show ?thesis
  by simp
qed

```

```

lemma ksimplex_replace_2:
  assumes s: ksimplex p n s and a  $\in s$  and  $n \neq 0$ 
  and lb:  $\forall j < n. \exists x \in s - \{a\}. x \ j \neq 0$ 
  and ub:  $\forall j < n. \exists x \in s - \{a\}. x \ j \neq p$ 
  shows  $\text{card } \{s'. \text{ksimplex } p \ n \ s' \wedge (\exists b \in s'. s' - \{b\} = s - \{a\})\} = 2$ 
  using s
proof cases
  case (ksimplex base upd)
  then interpret kuhn_simplex p n base upd s .

```

```

from  $\langle a \in s \rangle$  obtain i where  $i \leq n$   $a = \text{enum } i$ 

```

```

unfolding s_eq by auto

from ⟨i ≤ n⟩ have i = 0 ∨ i = n ∨ (0 < i ∧ i < n)
  by linarith
then have ∃!s'. s' ≠ s ∧ ksimplex p n s' ∧ (∃ b∈s'. s - {a} = s' - {b})
proof (elim disjE conjE)
  assume i = 0
  define rot where [abs_def]: rot i = (if i + 1 = n then 0 else i + 1) for i
  let ?upd = upd ∘ rot

  have rot: bij_betw rot {..

```

```

      (auto simp add: upd_inj)
qed
then have enum ' Suc ' {.. $n$ } =  $f'$  ' {.. $n$ }
  by (force simp: enum_inj)
also have Suc ' {.. $n$ } = {.. $n$ } - {0}
  by (auto simp: image_iff Ball_def) arith
also have {.. $n$ } = {.. $n$ } - { $n$ }
  by auto
finally have eq:  $s - \{a\} = f' ' \{.. $n$ \} - \{f' n\}$ 
  unfolding s_eq < $a = \text{enum } i$ > < $i = 0$ >
  by (simp add: inj_on_image_set_diff[OF inj_enum] inj_on_image_set_diff[OF
inj_f'])

have enum 0 <  $f' 0$ 
  using < $n \neq 0$ > by (simp add: enum_strict_mono  $f'_{eq\_enum}$ )
also have ... <  $f' n$ 
  using < $n \neq 0$ > b.enum_strict_mono[of 0  $n$ ] unfolding b_enum by simp
finally have  $a \neq f' n$ 
  using < $a = \text{enum } i$ > < $i = 0$ > by auto

{ fix  $t\ c$  assume  $\text{ksimplex } p\ n\ t\ c \in t$  and eq_sma:  $s - \{a\} = t - \{c\}$ 
  obtain  $b\ u$  where  $\text{kuhn\_simplex } p\ n\ b\ u\ t$ 
    using < $\text{ksimplex } p\ n\ t$ > by (auto elim:  $\text{ksimplex.cases}$ )
  then interpret  $t$ :  $\text{kuhn\_simplex } p\ n\ b\ u\ t$  .

  { fix  $x$  assume  $x \in s\ x \neq a$ 
    then have  $x (\text{upd } 0) = \text{enum } (\text{Suc } 0) (\text{upd } 0)$ 
      by (auto simp: < $a = \text{enum } i$ > < $i = 0$ > s_eq_enum_def enum_inj) }
  then have eq_upd0:  $\forall x \in t - \{c\}. x (\text{upd } 0) = \text{enum } (\text{Suc } 0) (\text{upd } 0)$ 
    unfolding eq_sma[symmetric] by auto
  then have  $c (\text{upd } 0) \neq \text{enum } (\text{Suc } 0) (\text{upd } 0)$ 
    using < $n \neq 0$ > by (intro t.one_step[OF < $c \in t$ >]) (auto simp: upd_space)
  then have  $c (\text{upd } 0) < \text{enum } (\text{Suc } 0) (\text{upd } 0) \vee c (\text{upd } 0) > \text{enum } (\text{Suc } 0) (\text{upd } 0)$ 
    by auto
  then have  $t = s \vee t = f' ' \{.. $n$ \}$ 
    proof (elim disjE conjE)
      assume *:  $c (\text{upd } 0) < \text{enum } (\text{Suc } 0) (\text{upd } 0)$ 
      interpret st:  $\text{kuhn\_simplex\_pair } p\ n\ \text{base } \text{upd } s\ b\ u\ t$  ..
      { fix  $x$  assume  $x \in t$  with * < $c \in t$ > eq_upd0[rule_format, of  $x$ ] have  $c \leq x$ 
        by (auto simp: le_less intro!: t.less[of _ _ upd 0]) }
      note top = this
      have  $s = t$ 
        using < $a = \text{enum } i$ > < $i = 0$ > < $c \in t$ >
        by (intro st.ksimplex_eq_bot[OF _ _ _ eq_sma])
        (auto simp: s_eq_enum_mono t.s_eq t.enum_mono top)
      then show ?thesis by simp
    next
      assume *:  $c (\text{upd } 0) > \text{enum } (\text{Suc } 0) (\text{upd } 0)$ 

```

```

interpret st: kuhn_simplex_pair p n enum (Suc 0) upd ∘ rot f' ' {.. n} b
u t ..
  have eq: f' ' {..n} - {f' n} = t - {c}
    using eq_sma eq by simp
  { fix x assume x ∈ t with * ⟨c ∈ t⟩ eq_upd0[rule_format, of x] have x ≤ c
    by (auto simp: le_less intro!: t.less[of _ _ upd 0]) }
  note top = this
  have f' ' {..n} = t
    using ⟨a = enum i⟩ ⟨i = 0⟩ ⟨c ∈ t⟩
    by (intro st.ksimplex_eq_top[OF _ _ _ eq])
    (auto simp: b.s_eq b.enum_mono t.s_eq t.enum_mono b.enum[symmetric]
top)
  then show ?thesis by simp
qed }
with ks_f' eq ⟨a ≠ f' n⟩ ⟨n ≠ 0⟩ show ?thesis
  apply (intro ex1I[of _ f' ' {.. n}])
  apply auto []
  applymetis
done
next
  assume i = n
  from ⟨n ≠ 0⟩ obtain n' where n': n = Suc n'
    by (cases n) auto

define rot where rot i = (case i of 0 ⇒ n' | Suc i ⇒ i) for i
let ?upd = upd ∘ rot

have rot: bij_betw rot {..<n} {..<n}
  by (auto simp: bij_betw_def inj_on_def image_iff Bex_def rot_def n' split:
nat.splits)
arith
from rot upd have bij_betw ?upd {..<n} {..<n}
  by (rule bij_betw_trans)

define b where b = base (upd n' := base (upd n') - 1)
define f' where [abs_def]: f' i j = (if j ∈ ?upd' {..<i} then Suc (b j) else b
j) for i j

interpret b: kuhn_simplex p n b upd ∘ rot f' ' {.. n}
proof
  { fix i assume n ≤ i then show b i = p
    using base_out[of i] upd_space[of n'] by (auto simp: b_def n') }
  show b ∈ {..<n} → {..<p}
    using base ⟨n ≠ 0⟩ upd_space[of n']
    by (auto simp: b_def PiE_def Pi_iff Ball_def upd_space extensional_def
n')

  show bij_betw ?upd {..<n} {..<n} by fact
qed (simp add: f'_def)

```

```

have f': b.enum = f' unfolding f'_def b.enum_def[abs_def] ..
have ks_f': ksimplex p n (b.enum ' {.. n})
  unfolding f' by rule unfold_locales

have 0 < n
  using ⟨n ≠ 0⟩ by auto

{ from ⟨a = enum i⟩ ⟨n ≠ 0⟩ ⟨i = n⟩ lb upd_space[of n]
  obtain i' where i' ≤ n enum i' ≠ enum n 0 < enum i' (upd n')
    unfolding s_eq by (auto simp: enum_inj n')
    moreover have enum i' (upd n') = base (upd n')
      unfolding enum_def using ⟨i' ≤ n⟩ ⟨enum i' ≠ enum n⟩ by (auto simp:
n' upd_inj enum_inj)
    ultimately have 0 < base (upd n')
      by auto }
then have benum1: b.enum (Suc 0) = base
  unfolding b.enum_Suc[OF ⟨0 < n⟩] b.enum_0 by (auto simp: b_def rot_def)

have [simp]: ⋀j. Suc j < n ⟹ rot ' {..< Suc j} = {n'} ∪ {..< j}
  by (auto simp: rot_def image_iff Ball_def split: nat.splits)
have rot_simps: ⋀j. rot (Suc j) = j rot 0 = n'
  by (simp_all add: rot_def)

{ fix j assume j: Suc j ≤ n then have b.enum (Suc j) = enum j
  by (induct j) (auto simp: benum1 enum_0 b.enum_Suc enum_Suc rot_simps)
}

note b_enum_eq_enum = this
then have enum ' {..< n} = b.enum ' Suc ' {..< n}
  by (auto simp: image_comp intro!: image_cong)
also have Suc ' {..< n} = {.. n} - {0}
  by (auto simp: image_iff Ball_def) arith
also have {..< n} = {.. n} - {n}
  by auto
finally have eq: s - {a} = b.enum ' {.. n} - {b.enum 0}
  unfolding s_eq ⟨a = enum i⟩ ⟨i = n⟩
  using inj_on_image_set_diff[OF inj_enum Diff_subset, of {n}]
    inj_on_image_set_diff[OF b.inj_enum Diff_subset, of {0}]
  by (simp add: comp_def)

have b.enum 0 ≤ b.enum n
  by (simp add: b.enum_mono)
also have b.enum n < enum n
  using ⟨n ≠ 0⟩ by (simp add: enum_strict_mono b_enum_eq_enum n')
finally have a ≠ b.enum 0
  using ⟨a = enum i⟩ ⟨i = n⟩ by auto

{ fix t c assume ksimplex p n t c ∈ t and eq_sma: s - {a} = t - {c}
  obtain b' u where kuhn_simplex p n b' u t
    using ⟨ksimplex p n t⟩ by (auto elim: ksimplex.cases)

```

```

then interpret t: kuhn_simplex p n b' u t .

{ fix x assume x ∈ s x ≠ a
  then have x (upd n') = enum n' (upd n')
    by (auto simp: ⟨a = enum i⟩ n' ⟨i = n⟩ s_eq enum_def enum_inj
in_upd_image) }
then have eq_upd0: ∀ x ∈ t - {c}. x (upd n') = enum n' (upd n')
  unfolding eq_sma[symmetric] by auto
then have c (upd n') ≠ enum n' (upd n')
using ⟨n ≠ 0⟩ by (intro t.one_step[OF ⟨c ∈ t⟩]) (auto simp: n' upd_space[unfolded
n'])
then have c (upd n') < enum n' (upd n') ∨ c (upd n') > enum n' (upd n')
  by auto
then have t = s ∨ t = b.enum ' {..n}
proof (elim disjE conjE)
  assume *: c (upd n') > enum n' (upd n')
  interpret st: kuhn_simplex_pair p n base upd s b' u t ..
  { fix x assume x ∈ t with * ⟨c ∈ t⟩ eq_upd0[rule_format, of x] have x ≤ c
    by (auto simp: le_less intro!: t.less[of _ _ upd n']) }
  note top = this
  have s = t
    using ⟨a = enum i⟩ ⟨i = n⟩ ⟨c ∈ t⟩
    by (intro st.ksimplex_eq_top[OF _ _ _ eq_sma])
    (auto simp: s_eq enum_mono t.s_eq t.enum_mono top)
  then show ?thesis by simp
next
  assume *: c (upd n') < enum n' (upd n')
  interpret st: kuhn_simplex_pair p n b upd ∘ rot f' ' {.. n} b' u t ..
  have eq: f' ' {..n} - {b.enum 0} = t - {c}
    using eq_sma eq f' by simp
  { fix x assume x ∈ t with * ⟨c ∈ t⟩ eq_upd0[rule_format, of x] have c ≤ x
    by (auto simp: le_less intro!: t.less[of _ _ upd n']) }
  note bot = this
  have f' ' {..n} = t
    using ⟨a = enum i⟩ ⟨i = n⟩ ⟨c ∈ t⟩
    by (intro st.ksimplex_eq_bot[OF _ _ _ eq])
    (auto simp: b.s_eq b.enum_mono t.s_eq t.enum_mono bot)
  with f' show ?thesis by simp
qed }
with ks_f' eq ⟨a ≠ b.enum 0⟩ ⟨n ≠ 0⟩ show ?thesis
apply (intro ex1I[of _ b.enum ' {.. n}])
apply auto []
apply metis
done
next
  assume i: 0 < i i < n
  define i' where i' = i - 1
  with i have Suc i' < n
    by simp

```

```

with i have Suc_i': Suc i' = i
  by (simp add: i'_def)

let ?upd = Fun.swap i' i upd
from i upd have bij_betw ?upd {.. $n$ } {.. $n$ }
  by (subst bij_betw_swap_iff) (auto simp: i'_def)

define f' where [abs_def]: f' i j = (if j ∈ ?upd {.. $i$ } then Suc (base j) else
base j)
  for i j
interpret b: kuhn_simplex p n base ?upd f' {.. $n$ }
proof
  show base ∈ {.. $n$ } → {.. $p$ } by (rule base)
  { fix i assume  $n \leq i$  then show base i = p by (rule base_out) }
  show bij_betw ?upd {.. $n$ } {.. $n$ } by fact
qed (simp add: f'_def)
have f': b.enum = f'_def b.enum_def [abs_def] ..
have ks_f': ksimplex p n (b.enum {.. $n$ })
  unfolding f' by rule unfold_locales

have {i} ⊆ {.. $n$ }
  using i by auto
{ fix j assume  $j \leq n$ 
  moreover have  $j < i \vee i = j \vee i < j$  by arith
  moreover note i
  ultimately have enum j = b.enum j  $\longleftrightarrow j \neq i$ 
    apply (simp only: fun_eq_iff enum_def b.enum_def flip: image_comp)
    apply (cases ⟨i = j⟩)
    apply simp
    apply (metis Suc_i' ⟨i ≤ n⟩ imageI in_upd_image lessI lessThan_iff
lessThan_subset_iff less_irrefl_nat transpose_apply_second transpose_commute)
    apply (subst transpose_image_eq)
    apply (auto simp add: i'_def)
  done
}
note enum_eq_benum = this
then have enum {.. $n$ } - {i} = b.enum {.. $n$ } - {i}
  by (intro image_cong) auto
then have eq: s - {a} = b.enum {.. $n$ } - {b.enum i}
  unfolding s_eq ⟨a = enum i⟩
  using inj_on_image_set_diff[OF inj_enum Diff_subset ⟨{i} ⊆ {.. $n$ }⟩]
    inj_on_image_set_diff[OF b.inj_enum Diff_subset ⟨{i} ⊆ {.. $n$ }⟩]
  by (simp add: comp_def)

have a ≠ b.enum i
  using ⟨a = enum i⟩ enum_eq_benum i by auto

{ fix t c assume ksimplex p n t c ∈ t and eq_sma: s - {a} = t - {c}
  obtain b' u where kuhn_simplex p n b' u t

```

```

    using ⟨ksimplex p n t⟩ by (auto elim: ksimplex.cases)
  then interpret t: kuhn_simplex p n b' u t .
  have enum i' ∈ s - {a} enum (i + 1) ∈ s - {a}
    using ⟨a = enum i⟩ i enum_in by (auto simp: enum_inj i'_def)
  then obtain l k where
    l: t.enum l = enum i' l ≤ n t.enum l ≠ c and
    k: t.enum k = enum (i + 1) k ≤ n t.enum k ≠ c
    unfolding eq_sma by (auto simp: t.s_eq)
  with i have t.enum l < t.enum k
    by (simp add: enum_strict_mono i'_def)
  with ⟨l ≤ n⟩ ⟨k ≤ n⟩ have l < k
    by (simp add: t.enum_strict_mono)
  { assume Suc l = k
    have enum (Suc (Suc i')) = t.enum (Suc l)
      using i by (simp add: k ⟨Suc l = k⟩ i'_def)
    then have False
      using ⟨l < k⟩ ⟨k ≤ n⟩ ⟨Suc i' < n⟩
      by (auto simp: t.enum_Suc enum_Suc l upd_inj fun_eq_iff split:
if_split_asm)
    (metis Suc_lessD n_not_Suc_n upd_inj) }
  with ⟨l < k⟩ have Suc l < k
    by arith
  have c_eq: c = t.enum (Suc l)
  proof (rule ccontr)
    assume c ≠ t.enum (Suc l)
    then have t.enum (Suc l) ∈ s - {a}
      using ⟨l < k⟩ ⟨k ≤ n⟩ by (simp add: t.s_eq eq_sma)
    then obtain j where t.enum (Suc l) = enum j j ≤ n enum j ≠ enum i
      unfolding s_eq ⟨a = enum i⟩ by auto
    with i have t.enum (Suc l) ≤ t.enum l ∨ t.enum k ≤ t.enum (Suc l)
      by (auto simp: i'_def enum_mono enum_inj l k)
    with ⟨Suc l < k⟩ ⟨k ≤ n⟩ show False
      by (simp add: t.enum_mono)
  qed

  { have t.enum (Suc (Suc l)) ∈ s - {a}
    unfolding eq_sma c_eq t.s_eq using ⟨Suc l < k⟩ ⟨k ≤ n⟩ by (auto simp:
t.enum_inj)
    then obtain j where eq: t.enum (Suc (Suc l)) = enum j and j ≤ n j ≠ i
      by (auto simp: s_eq ⟨a = enum i⟩)
    moreover have enum i' < t.enum (Suc (Suc l))
      unfolding l(1)[symmetric] using ⟨Suc l < k⟩ ⟨k ≤ n⟩ by (auto simp:
t.enum_strict_mono)
    ultimately have i' < j
      using i by (simp add: enum_strict_mono i'_def)
    with ⟨j ≠ i⟩ ⟨j ≤ n⟩ have t.enum k ≤ t.enum (Suc (Suc l))
      unfolding i'_def by (simp add: enum_mono k eq)
    then have k ≤ Suc (Suc l)
      using ⟨k ≤ n⟩ ⟨Suc l < k⟩ by (simp add: t.enum_mono) }

```



```

with ‹Suc l < k› have Suc (Suc l) = k by simp
then have enum (Suc (Suc i')) = t.enum (Suc (Suc l))
  using i by (simp add: k i'_def)
also have ... = (enum i') (u l := Suc (enum i' (u l)), u (Suc l) := Suc
(enum i' (u (Suc l))))
  using ‹Suc l < k› ‹k ≤ n› by (simp add: t.enum_Suc l t.upd_inj)
finally have (u l = upd i' ∧ u (Suc l) = upd (Suc i')) ∨
  (u l = upd (Suc i') ∧ u (Suc l) = upd i')
  using ‹Suc i' < n› by (auto simp: enum_Suc fun_eq_iff split: if_split_asm)

then have t = s ∨ t = b.enum ' {..n}
proof (elim disjE conjE)
  assume u: u l = upd i'
  have c = t.enum (Suc l) unfolding c_eq ..
  also have t.enum (Suc l) = enum (Suc i')
    using u ‹l < k› ‹k ≤ n› ‹Suc i' < n› by (simp add: enum_Suc t.enum_Suc
l)
  also have ... = a
    using ‹a = enum i› i by (simp add: i'_def)
  finally show ?thesis
    using eq_sma ‹a ∈ s› ‹c ∈ t› by auto
next
  assume u: u l = upd (Suc i')
  define B where B = b.enum ' {..n}
  have b.enum i' = enum i'
    using enum_eq_benum[of i'] i by (auto simp: i'_def gr0_conv_Suc)
  have c = t.enum (Suc l) unfolding c_eq ..
  also have t.enum (Suc l) = b.enum (Suc i')
    using u ‹l < k› ‹k ≤ n› ‹Suc i' < n›
    by (simp_all add: enum_Suc t.enum_Suc l b.enum_Suc ‹b.enum i' =
enum i'›)
    (simp add: Suc_i')
  also have ... = b.enum i
    using i by (simp add: i'_def)
  finally have c = b.enum i .
  then have t - {c} = B - {c} c ∈ B
    unfolding eq_sma[symmetric] eq B_def using i by auto
  with ‹c ∈ t› have t = B
    by auto
  then show ?thesis
    by (simp add: B_def)
qed }
with ks_f' eq ‹a ≠ b.enum i› ‹n ≠ 0› ‹i ≤ n› show ?thesis
  apply (intro ex1I[of _ b.enum ' {.. n}])
  apply auto []
  apply metis
done
qed
then show ?thesis

```

```

    using s < a ∈ s > by (simp add: card_2_iff' Ex1_def) metis
qed

```

Hence another step towards concreteness.

lemma *kuhn_simplex_lemma*:

```

    assumes ∀ s. ksimplex p (Suc n) s ⟶ rl ' s ⊆ {.. Suc n}
    and odd (card {f. ∃ s a. ksimplex p (Suc n) s ∧ a ∈ s ∧ (f = s - {a}) ∧
        rl ' f = {..n} ∧ ((∃ j ≤ n. ∀ x ∈ f. x j = 0) ∨ (∃ j ≤ n. ∀ x ∈ f. x j = p))})
    shows odd (card {s. ksimplex p (Suc n) s ∧ rl ' s = {..Suc n}})
proof (rule kuhn_complete_lemma[OF finite_ksimplexes refl, unfolded mem_Collect_eq,
    where bnd = λ f. (∃ j ∈ {..n}. ∀ x ∈ f. x j = 0) ∨ (∃ j ∈ {..n}. ∀ x ∈ f. x j = p)],
    safe del: notI)

    have *: ∧ x y. x = y ⟹ odd (card x) ⟹ odd (card y)
    by auto
    show odd (card {f. (∃ s ∈ {s. ksimplex p (Suc n) s}. ∃ a ∈ s. f = s - {a}) ∧
        rl ' f = {..n} ∧ ((∃ j ∈ {..n}. ∀ x ∈ f. x j = 0) ∨ (∃ j ∈ {..n}. ∀ x ∈ f. x j = p))})
    apply (rule *[OF_ assms(2)])
    apply (auto simp: atLeast0AtMost)
    done

```

next

```

fix s assume s: ksimplex p (Suc n) s
then show card s = n + 2
  by (simp add: ksimplex_card)

fix a assume a: a ∈ s then show rl a ≤ Suc n
  using assms(1) s by (auto simp: subset_eq)

let ?S = {t. ksimplex p (Suc n) t ∧ (∃ b ∈ t. s - {a} = t - {b})}
{ fix j assume j: j ≤ n ∀ x ∈ s - {a}. x j = 0
  with s a show card ?S = 1
    using ksimplex_replace_0[of p n + 1 s a]
    by (subst eq_commute) simp }

{ fix j assume j: j ≤ n ∀ x ∈ s - {a}. x j = p
  with s a show card ?S = 1
    using ksimplex_replace_1[of p n + 1 s a]
    by (subst eq_commute) simp }

{ assume card ?S ≠ 2 ∧ (∃ j ∈ {..n}. ∀ x ∈ s - {a}. x j = p)
  with s a show ∃ j ∈ {..n}. ∀ x ∈ s - {a}. x j = 0
    using ksimplex_replace_2[of p n + 1 s a]
    by (subst (asm) eq_commute) auto }

```

qed

Reduced labelling

definition *reduced* :: $\text{nat} \Rightarrow (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat}$ **where** *reduced* $n\ x = (\text{LEAST } k. k = n \vee x\ k \neq 0)$

lemma *reduced_labelling*:

shows *reduced* $n\ x \leq n$

and $\forall i < \text{reduced } n\ x. x\ i = 0$

and *reduced* $n\ x = n \vee x\ (\text{reduced } n\ x) \neq 0$

proof –

show *reduced* $n\ x \leq n$

unfolding *reduced_def* **by** (rule *LeastI2_wellorder*[**where** $a=n$]) *auto*

show $\forall i < \text{reduced } n\ x. x\ i = 0$

unfolding *reduced_def* **by** (rule *LeastI2_wellorder*[**where** $a=n$]) *fastforce*+

show *reduced* $n\ x = n \vee x\ (\text{reduced } n\ x) \neq 0$

unfolding *reduced_def* **by** (rule *LeastI2_wellorder*[**where** $a=n$]) *fastforce*+

qed

lemma *reduced_labelling_unique*:

$r \leq n \implies \forall i < r. x\ i = 0 \implies r = n \vee x\ r \neq 0 \implies \text{reduced } n\ x = r$

unfolding *reduced_def* **by** (rule *LeastI2_wellorder*[**where** $a=n$]) (metis *le_less not_le*)+

lemma *reduced_labelling_zero*: $j < n \implies x\ j = 0 \implies \text{reduced } n\ x \neq j$

using *reduced_labelling*[*of* $n\ x$] **by** *auto*

lemma *reduce_labelling_zero*[*simp*]: *reduced* $0\ x = 0$

by (rule *reduced_labelling_unique*) *auto*

lemma *reduced_labelling_nonzero*: $j < n \implies x\ j \neq 0 \implies \text{reduced } n\ x \leq j$

using *reduced_labelling*[*of* $n\ x$] **by** (elim *allE*[**where** $x=j$]) *auto*

lemma *reduced_labelling_Suc*: *reduced* $(\text{Suc } n)\ x \neq \text{Suc } n \implies \text{reduced } (\text{Suc } n)\ x = \text{reduced } n\ x$

using *reduced_labelling*[*of* $\text{Suc } n\ x$]

by (intro *reduced_labelling_unique*[*symmetric*]) *auto*

lemma *complete_face_top*:

assumes $\forall x \in f. \forall j \leq n. x\ j = 0 \longrightarrow \text{lab } x\ j = 0$

and $\forall x \in f. \forall j \leq n. x\ j = p \longrightarrow \text{lab } x\ j = 1$

and *eq*: $(\text{reduced } (\text{Suc } n) \circ \text{lab})\ 'f = \{..n\}$

shows $((\exists j \leq n. \forall x \in f. x\ j = 0) \vee (\exists j \leq n. \forall x \in f. x\ j = p)) \longleftrightarrow (\forall x \in f. x\ n = p)$

proof (*safe del: disjCI*)

fix $x\ j$ **assume** $j: j \leq n \vee x \in f. x\ j = 0$

{ **fix** x **assume** $x \in f$ **with** *assms* j **have** *reduced* $(\text{Suc } n)\ (\text{lab } x) \neq j$

by (intro *reduced_labelling_zero*) *auto* }

moreover **have** $j \in (\text{reduced } (\text{Suc } n) \circ \text{lab})\ 'f$

using *j eq* **by** *auto*

ultimately **show** $x\ n = p$

by *force*

```

next
  fix x j assume j: j ≤ n ∀ x ∈ f. x j = p and x: x ∈ f
  have j = n
  proof (rule ccontr)
    assume ¬ ?thesis
    { fix x assume x ∈ f
      with assms j have reduced (Suc n) (lab x) ≤ j
      by (intro reduced_labelling_nonzero) auto
      then have reduced (Suc n) (lab x) ≠ n
      using ⟨j ≠ n⟩ ⟨j ≤ n⟩ by simp }
    moreover
    have n ∈ (reduced (Suc n) ∘ lab) ' f
    using eq by auto
    ultimately show False
    by force
  qed
  moreover have j ∈ (reduced (Suc n) ∘ lab) ' f
  using j eq by auto
  ultimately show x n = p
  using j x by auto
qed auto

```

Hence we get just about the nice induction.

lemma *kuhn_induction*:

```

assumes 0 < p
  and lab_0: ∀ x. ∀ j ≤ n. (∀ j. x j ≤ p) ∧ x j = 0 ⟶ lab x j = 0
  and lab_1: ∀ x. ∀ j ≤ n. (∀ j. x j ≤ p) ∧ x j = p ⟶ lab x j = 1
  and odd: odd (card {s. ksimplex p n s ∧ (reduced n ∘ lab) ' s = {..n}})
shows odd (card {s. ksimplex p (Suc n) s ∧ (reduced (Suc n) ∘ lab) ' s = {..Suc
n}})
proof -
  let ?rl = reduced (Suc n) ∘ lab and ?ext = λ f v. ∃ j ≤ n. ∀ x ∈ f. x j = v
  let ?ext = λ s. (∃ j ≤ n. ∀ x ∈ s. x j = 0) ∨ (∃ j ≤ n. ∀ x ∈ s. x j = p)
  have ∀ s. ksimplex p (Suc n) s ⟶ ?rl ' s ⊆ {..Suc n}
    by (simp add: reduced_labelling_subset_eq)
  moreover
  have {s. ksimplex p n s ∧ (reduced n ∘ lab) ' s = {..n}} =
    {f. ∃ s a. ksimplex p (Suc n) s ∧ a ∈ s ∧ f = s - {a} ∧ ?rl ' f = {..n} ∧
?ext f}
  proof (intro set_eqI, safe del: disjCI equalityI disjE)
    fix s assume s: ksimplex p n s and rl: (reduced n ∘ lab) ' s = {..n}
    from s obtain u b where kuhn_simplex p n u b s by (auto elim: ksimplex.cases)
    then interpret kuhn_simplex p n u b s .
    have all_eq_p: ∀ x ∈ s. x n = p
      by (auto simp: out_eq_p)
    moreover
    { fix x assume x ∈ s
      with lab_1 [rule_format, of n x] all_eq_p s le_p [of x]
      have ?rl x ≤ n

```

```

    by (auto intro!: reduced_labelling_nonzero)
  then have ?rl x = reduced n (lab x)
    by (auto intro!: reduced_labelling_Suc) }
then have ?rl ' s = {..n}
  using rl by (simp cong: image_cong)
moreover
obtain t a where ksimplex p (Suc n) t a ∈ t s = t - {a}
  using s unfolding simplex_top_face[OF ‹0 < p› all_eq_p] by auto
ultimately
show ∃ t a. ksimplex p (Suc n) t ∧ a ∈ t ∧ s = t - {a} ∧ ?rl ' s = {..n} ∧
?ext s
  by auto
next
fix x s a assume s: ksimplex p (Suc n) s and rl: ?rl ' (s - {a}) = {.. n}
  and a: a ∈ s and ?ext (s - {a})
  from s obtain u b where kuhn_simplex p (Suc n) u b s by (auto elim:
ksimplex.cases)
  then interpret kuhn_simplex p Suc n u b s .
  have all_eq_p: ∀ x ∈ s. x (Suc n) = p
    by (auto simp: out_eq_p)

{ fix x assume x ∈ s - {a}
  then have ?rl x ∈ ?rl ' (s - {a})
    by auto
  then have ?rl x ≤ n
    unfolding rl by auto
  then have ?rl x = reduced n (lab x)
    by (auto intro!: reduced_labelling_Suc) }
then show rl': (reduced n lab) ' (s - {a}) = {..n}
  unfolding rl[symmetric] by (intro image_cong) auto

from ‹?ext (s - {a})›
have all_eq_p: ∀ x ∈ s - {a}. x n = p
proof (elim disjE exE conjE)
  fix j assume j ≤ n ∀ x ∈ s - {a}. x j = 0
  with lab_0[rule_format, of j] all_eq_p s_le_p
  have ∧x. x ∈ s - {a} ⇒ reduced (Suc n) (lab x) ≠ j
    by (intro reduced_labelling_zero) auto
  moreover have j ∈ ?rl ' (s - {a})
    using ‹j ≤ n› unfolding rl by auto
  ultimately show ?thesis
    by force
next
fix j assume j ≤ n and eq_p: ∀ x ∈ s - {a}. x j = p
show ?thesis
proof cases
  assume j = n with eq_p show ?thesis by simp
next
  assume j ≠ n

```

```

{ fix x assume x: x ∈ s - {a}
  have reduced n (lab x) ≤ j
  proof (rule reduced_labelling_nonzero)
    show lab x j ≠ 0
    using lab_1[rule_format, of j x] x_s_le_p[of x] eq_p ⟨j ≤ n⟩ by auto
    show j < n
    using ⟨j ≤ n⟩ ⟨j ≠ n⟩ by simp
  qed
  then have reduced n (lab x) ≠ n
  using ⟨j ≤ n⟩ ⟨j ≠ n⟩ by simp }
moreover have n ∈ (reduced n o lab) ‘ (s - {a})
  unfolding rl' by auto
ultimately show ?thesis
  by force
qed
qed
show ksimplex p n (s - {a})
  unfolding simplex_top_face[OF ⟨0 < p⟩ all_eq_p] using s a by auto
qed
ultimately show ?thesis
  using assms by (intro kuhn_simplex_lemma) auto
qed

```

And so we get the final combinatorial result.

```

lemma ksimplex_0: ksimplex p 0 s ⟷ s = {(λx. p)}
proof
  assume ksimplex p 0 s then show s = {(λx. p)}
  by (blast dest: kuhn_simplex.ksimplex_0 elim: ksimplex.cases)
next
  assume s: s = {(λx. p)}
  show ksimplex p 0 s
  proof (intro ksimplex, unfold locales)
    show (λ_. p) ∈ {..<0::nat} → {..

```

```

lemma kuhn_combinatorial:
  assumes 0 < p
  and ∀ x j. (∀ j. x j ≤ p) ∧ j < n ∧ x j = 0 ⟶ lab x j = 0
  and ∀ x j. (∀ j. x j ≤ p) ∧ j < n ∧ x j = p ⟶ lab x j = 1
  shows odd (card {s. ksimplex p n s ∧ (reduced n o lab) ‘ s = {..n}})
  (is odd (card (?M n)))
  using assms
proof (induct n)
  case 0 then show ?case
  by (simp add: ksimplex_0 cong: conj_cong)
next

```

```

case (Suc n)
then have odd (card (?M n))
  by force
with Suc show ?case
  using kuhn_induction[of p n] by (auto simp: comp_def)
qed

lemma kuhn_lemma:
  fixes n p :: nat
  assumes 0 < p
    and  $\forall x. (\forall i < n. x\ i \leq p) \longrightarrow (\forall i < n. \text{label } x\ i = (0::nat) \vee \text{label } x\ i = 1)$ 
    and  $\forall x. (\forall i < n. x\ i \leq p) \longrightarrow (\forall i < n. x\ i = 0 \longrightarrow \text{label } x\ i = 0)$ 
    and  $\forall x. (\forall i < n. x\ i \leq p) \longrightarrow (\forall i < n. x\ i = p \longrightarrow \text{label } x\ i = 1)$ 
  obtains q where  $\forall i < n. q\ i < p$ 
    and  $\forall i < n. \exists r\ s. (\forall j < n. q\ j \leq r\ j \wedge r\ j \leq q\ j + 1) \wedge (\forall j < n. q\ j \leq s\ j \wedge s\ j \leq q\ j + 1) \wedge \text{label } r\ i \neq \text{label } s\ i$ 
proof -
  let ?rl = reduced n o label
  let ?A = {s. ksimplex p n s  $\wedge$  ?rl ' s = {..n}}
  have odd (card ?A)
    using assms by (intro kuhn_combinatorial[of p n label]) auto
  then have ?A  $\neq$  {}
    by (rule odd_card_imp_not_empty)
  then obtain s b u where kuhn_simplex p n b u s and rl: ?rl ' s = {..n}
    by (auto elim: ksimplex.cases)
  interpret kuhn_simplex p n b u s by fact

  show ?thesis
proof (intro that[of b] allI impI)
  fix i
  assume i < n
  then show b i < p
    using base by auto
next
  fix i
  assume i < n
  then have i  $\in$  {.. n} Suc i  $\in$  {.. n}
    by auto
  then obtain u v where u: u  $\in$  s Suc i = ?rl u and v: v  $\in$  s i = ?rl v
    unfolding rl[symmetric] by blast

  have label u i  $\neq$  label v i
    using reduced_labelling [of n label u] reduced_labelling [of n label v]
      u(2)[symmetric] v(2)[symmetric] <i < n>
    by auto
  moreover
  have b j  $\leq$  u j u j  $\leq$  b j + 1 b j  $\leq$  v j v j  $\leq$  b j + 1 if j < n for j
    using that base_le[OF <u $\in$ s>] le_Suc_base[OF <u $\in$ s>] base_le[OF <v $\in$ s>]
      le_Suc_base[OF <v $\in$ s>]

```

```

    by auto
    ultimately show  $\exists r s. (\forall j < n. b\ j \leq r\ j \wedge r\ j \leq b\ j + 1) \wedge$ 
       $(\forall j < n. b\ j \leq s\ j \wedge s\ j \leq b\ j + 1) \wedge \text{label } r\ i \neq \text{label } s\ i$ 
    by blast
  qed
qed

```

Main result for the unit cube

lemma *kuhn_labelling_lemma'*:

```

  assumes  $(\forall x :: \text{nat} \Rightarrow \text{real}. P\ x \longrightarrow P\ (f\ x))$ 
    and  $\forall x. P\ x \longrightarrow (\forall i :: \text{nat}. Q\ i \longrightarrow 0 \leq x\ i \wedge x\ i \leq 1)$ 
  shows  $\exists l. (\forall x\ i. l\ x\ i \leq (1 :: \text{nat})) \wedge$ 
     $(\forall x\ i. P\ x \wedge Q\ i \wedge x\ i = 0 \longrightarrow l\ x\ i = 0) \wedge$ 
     $(\forall x\ i. P\ x \wedge Q\ i \wedge x\ i = 1 \longrightarrow l\ x\ i = 1) \wedge$ 
     $(\forall x\ i. P\ x \wedge Q\ i \wedge l\ x\ i = 0 \longrightarrow x\ i \leq f\ x\ i) \wedge$ 
     $(\forall x\ i. P\ x \wedge Q\ i \wedge l\ x\ i = 1 \longrightarrow f\ x\ i \leq x\ i)$ 
proof -
  have and_forall_thm:  $\bigwedge P\ Q. (\forall x. P\ x) \wedge (\forall x. Q\ x) \longleftrightarrow (\forall x. P\ x \wedge Q\ x)$ 
    by auto
  have *:  $\forall x\ y :: \text{real}. 0 \leq x \wedge x \leq 1 \wedge 0 \leq y \wedge y \leq 1 \longrightarrow x \neq 1 \wedge x \leq y \vee x \neq$ 
     $0 \wedge y \leq x$ 
    by auto
  show ?thesis
    unfolding and_forall_thm
    apply (subst choice_iff[symmetric])+
    apply rule
    apply rule
  proof -
    fix  $x\ x'$ 
    let ?R =  $\lambda y :: \text{nat}. (P\ x \wedge Q\ x' \wedge x\ x' = 0 \longrightarrow y = 0) \wedge$ 
       $(P\ x \wedge Q\ x' \wedge x\ x' = 1 \longrightarrow y = 1) \wedge$ 
       $(P\ x \wedge Q\ x' \wedge y = 0 \longrightarrow x\ x' \leq (f\ x)\ x') \wedge$ 
       $(P\ x \wedge Q\ x' \wedge y = 1 \longrightarrow (f\ x)\ x' \leq x\ x')$ 
    have  $0 \leq f\ x\ x' \wedge f\ x\ x' \leq 1$  if  $P\ x\ Q\ x'$ 
      using assms(2)[rule_format, of  $f\ x\ x'$ ] that
      apply (drule_tac assms(1)[rule_format])
      apply auto
    done
    then have  $?R\ 0 \vee ?R\ 1$ 
      by auto
    then show  $\exists y \leq 1. ?R\ y$ 
      by auto
  qed
qed

```


6.31.3 Brouwer's fixed point theorem

We start proving Brouwer's fixed point theorem for the unit cube = $\text{cbox } 0 \text{ One}$.

```

lemma brouwer_cube:
  fixes  $f :: 'a :: \text{euclidean\_space} \Rightarrow 'a$ 
  assumes  $\text{continuous\_on } (\text{cbox } 0 \text{ One}) f$ 
  and  $f \text{ ' cbox } 0 \text{ One} \subseteq \text{cbox } 0 \text{ One}$ 
  shows  $\exists x \in \text{cbox } 0 \text{ One. } f x = x$ 
proof (rule ccontr)
  define  $n$  where  $n = \text{DIM}('a)$ 
  have  $n: 1 \leq n \ 0 < n \ n \neq 0$ 
  unfolding  $n\_def$  by (auto simp: Suc_le_eq)
  assume  $\neg ?thesis$ 
  then have  $*$ :  $\neg (\exists x \in \text{cbox } 0 \text{ One. } f x - x = 0)$ 
  by auto
  obtain  $d$  where
     $d: d > 0 \ \bigwedge x. x \in \text{cbox } 0 \text{ One} \implies d \leq \text{norm } (f x - x)$ 
  apply (rule brouwer_compactness_lemma[OF compact_cbox _ *])
  apply (rule continuous_intros assms)+
  apply blast
  done
  have  $*$ :  $\forall x. x \in \text{cbox } 0 \text{ One} \longrightarrow f x \in \text{cbox } 0 \text{ One}$ 
   $\forall x. x \in (\text{cbox } 0 \text{ One} :: 'a \text{ set}) \longrightarrow (\forall i \in \text{Basis. } \text{True} \longrightarrow 0 \leq x \cdot i \wedge x \cdot i \leq 1)$ 
  using  $\text{assms}(2)[\text{unfolded image_subset_iff Ball_def}]$ 
  unfolding  $\text{cbox\_def}$ 
  by auto
  obtain  $\text{label} :: 'a \Rightarrow 'a \Rightarrow \text{nat}$  where  $\text{label} [\text{rule\_format}]$ :
     $\forall x. \forall i \in \text{Basis. } \text{label } x \ i \leq 1$ 
     $\forall x. \forall i \in \text{Basis. } x \in \text{cbox } 0 \text{ One} \wedge x \cdot i = 0 \longrightarrow \text{label } x \ i = 0$ 
     $\forall x. \forall i \in \text{Basis. } x \in \text{cbox } 0 \text{ One} \wedge x \cdot i = 1 \longrightarrow \text{label } x \ i = 1$ 
     $\forall x. \forall i \in \text{Basis. } x \in \text{cbox } 0 \text{ One} \wedge \text{label } x \ i = 0 \longrightarrow x \cdot i \leq f x \cdot i$ 
     $\forall x. \forall i \in \text{Basis. } x \in \text{cbox } 0 \text{ One} \wedge \text{label } x \ i = 1 \longrightarrow f x \cdot i \leq x \cdot i$ 
  using  $\text{kuhn\_labelling\_lemma}[OF *]$  by auto
  note  $\text{label} = \text{this} [\text{rule\_format}]$ 
  have  $\text{lem1}$ :  $\forall x \in \text{cbox } 0 \text{ One. } \forall y \in \text{cbox } 0 \text{ One. } \forall i \in \text{Basis. } \text{label } x \ i \neq \text{label } y \ i \longrightarrow$ 
     $|f x \cdot i - x \cdot i| \leq \text{norm } (f y - f x) + \text{norm } (y - x)$ 
  proof safe
    fix  $x \ y :: 'a$ 
    assume  $x: x \in \text{cbox } 0 \text{ One}$  and  $y: y \in \text{cbox } 0 \text{ One}$ 
    fix  $i$ 
    assume  $i: \text{label } x \ i \neq \text{label } y \ i \ i \in \text{Basis}$ 
    have  $*$ :  $\bigwedge x \ y \ f x \ f y :: \text{real. } x \leq f x \wedge f y \leq y \vee f x \leq x \wedge y \leq f y \implies$ 
       $|f x - x| \leq |f y - f x| + |y - x|$  by auto
    have  $|f x - x| \cdot i \leq |f y - f x| \cdot i + |(y - x) \cdot i|$ 
    proof (cases  $\text{label } x \ i = 0$ )
      case  $\text{True}$ 
      then have  $\text{fxy}$ :  $\neg f y \cdot i \leq y \cdot i \implies f x \cdot i \leq x \cdot i$ 
      by (metis  $\text{True } i \ \text{label}(1) \ \text{label}(5) \ \text{le\_antisym less\_one not\_le\_imp\_less } y$ )

```

```

show ?thesis
unfolding inner_simps
by (rule *) (auto simp: True i label x y fxy)
next
case False
then show ?thesis
  using label [OF ‹i ∈ Basis›] i(1) x y
  apply (auto simp: inner_diff_left le_Suc_eq)
  by (metis *)
qed
also have ... ≤ norm (f y - f x) + norm (y - x)
  by (simp add: add_mono i(2) norm_bound_Basis_le)
finally show |f x · i - x · i| ≤ norm (f y - f x) + norm (y - x)
  unfolding inner_simps .
qed
have ∃ e > 0. ∀ x ∈ cbox 0 One. ∀ y ∈ cbox 0 One. ∀ z ∈ cbox 0 One. ∀ i ∈ Basis.
  norm (x - z) < e ⟶ norm (y - z) < e ⟶ label x i ≠ label y i ⟶
  |(f(z) - z) · i| < d / (real n)
proof -
  have d': d / real n / 8 > 0
  using d(1) by (simp add: n_def)
  have *: uniformly_continuous_on (cbox 0 One) f
  by (rule compact_uniformly_continuous[OF assms(1) compact_cbox])
  obtain e where e:
    e > 0
    ∧ x x'. x ∈ cbox 0 One ⟹
      x' ∈ cbox 0 One ⟹
        norm (x' - x) < e ⟹
          norm (f x' - f x) < d / real n / 8
  using *[unfolded uniformly_continuous_on_def, rule_format, OF d']
  unfolding dist_norm
  by blast
show ?thesis
proof (intro exI conjI ballI impI)
  show 0 < min (e / 2) (d / real n / 8)
  using d' e by auto
  fix x y z i
  assume as:
    x ∈ cbox 0 One y ∈ cbox 0 One z ∈ cbox 0 One
    norm (x - z) < min (e / 2) (d / real n / 8)
    norm (y - z) < min (e / 2) (d / real n / 8)
    label x i ≠ label y i
  assume i: i ∈ Basis
  have *: ∧ z fz x fx n1 n2 n3 n4 d4 d :: real. |fx - x| ≤ n1 + n2 ⟹
    |fx - fz| ≤ n3 ⟹ |x - z| ≤ n4 ⟹
    n1 < d4 ⟹ n2 < 2 * d4 ⟹ n3 < d4 ⟹ n4 < d4 ⟹
    (8 * d4 = d) ⟹ |fz - z| < d
  by auto
  show |(f z - z) · i| < d / real n

```

```

    unfolding inner_simps
  proof (rule *)
    show  $|f x \cdot i - x \cdot i| \leq \text{norm } (f y - f x) + \text{norm } (y - x)$ 
      using as(1) as(2) as(6) i lem1 by blast
    show  $\text{norm } (f x - f z) < d / \text{real } n / 8$ 
      using d' e as by auto
    show  $|f x \cdot i - f z \cdot i| \leq \text{norm } (f x - f z) |x \cdot i - z \cdot i| \leq \text{norm } (x - z)$ 
      unfolding inner_diff_left[symmetric]
      by (rule Basis_le_norm[OF i])+
    have tria:  $\text{norm } (y - x) \leq \text{norm } (y - z) + \text{norm } (x - z)$ 
      using dist_triangle[of y x z, unfolded dist_norm]
      unfolding norm_minus_commute
      by auto
    also have  $\dots < e / 2 + e / 2$ 
      using as(4) as(5) by auto
    finally show  $\text{norm } (f y - f x) < d / \text{real } n / 8$ 
      using as(1) as(2) e(2) by auto
    have  $\text{norm } (y - z) + \text{norm } (x - z) < d / \text{real } n / 8 + d / \text{real } n / 8$ 
      using as(4) as(5) by auto
    with tria show  $\text{norm } (y - x) < 2 * (d / \text{real } n / 8)$ 
      by auto
  qed (use as in auto)
qed
then
obtain e where e:
  e > 0
   $\bigwedge x y z i. x \in \text{cbox } 0 \text{ One} \implies$ 
   $y \in \text{cbox } 0 \text{ One} \implies$ 
   $z \in \text{cbox } 0 \text{ One} \implies$ 
   $i \in \text{Basis} \implies$ 
   $\text{norm } (x - z) < e \wedge \text{norm } (y - z) < e \wedge \text{label } x \ i \neq \text{label } y \ i \implies$ 
   $|(f z - z) \cdot i| < d / \text{real } n$ 
  by blast
obtain p :: nat where p:  $1 + \text{real } n / e \leq \text{real } p$ 
  using real_arch_simple ..
have  $1 + \text{real } n / e > 0$ 
  using e(1) n by (simp add: add_pos_pos)
then have p > 0
  using p by auto

obtain b :: nat  $\Rightarrow$  'a where b:  $\text{bij\_betw } b \ \{..< n\} \ \text{Basis}$ 
  by atomize_elim (auto simp: n_def intro!: finite_same_card_bij)
define b' where  $b' = \text{inv\_into } \{..< n\} \ b$ 
then have b':  $\text{bij\_betw } b' \ \text{Basis } \{..< n\}$ 
  using bij_betw_inv_into[OF b] by auto
then have b'_Basis:  $\bigwedge i. i \in \text{Basis} \implies b' \ i \in \{..< n\}$ 
  unfolding bij_betw_def by (auto simp: set_eq_iff)
have bb'[simp]:  $\bigwedge i. i \in \text{Basis} \implies b \ (b' \ i) = i$ 

```

```

    unfolding b'_def
    using b
    by (auto simp: f_inv_into_f bij_betw_def)
  have b'[simp]:  $\bigwedge i. i < n \implies b' (b i) = i$ 
    unfolding b'_def
    using b
    by (auto simp: inv_into_f_eq bij_betw_def)
  have *:  $\bigwedge x :: nat. x = 0 \vee x = 1 \longleftrightarrow x \leq 1$ 
    by auto
  have b'':  $\bigwedge j. j < n \implies b j \in Basis$ 
    using b unfolding bij_betw_def by auto
  have q1:  $0 < p \forall x. (\forall i < n. x i \leq p) \longrightarrow$ 
     $(\forall i < n. (label (\sum i \in Basis. (real (x (b' i)) / real p) *_{\mathbb{R}} i) \circ b) i = 0 \vee$ 
     $(label (\sum i \in Basis. (real (x (b' i)) / real p) *_{\mathbb{R}} i) \circ b) i = 1)$ 
    unfolding *
    using  $\langle p > 0 \rangle \langle n > 0 \rangle$ 
    using label(1)[OF b'']
    by auto
  { fix x :: nat  $\Rightarrow$  nat and i assume  $\forall i < n. x i \leq p \wedge i < n \wedge x i = p \vee x i = 0$ 
    then have  $(\sum i \in Basis. (real (x (b' i)) / real p) *_{\mathbb{R}} i) \in (cbox 0 One :: 'a set)$ 
      using b'_Basis
      by (auto simp: cbox_def bij_betw_def zero_le_divide_iff divide_le_eq_1) }
  note cube = this
  have q2:  $\forall x. (\forall i < n. x i \leq p) \longrightarrow (\forall i < n. x i = 0 \longrightarrow$ 
     $(label (\sum i \in Basis. (real (x (b' i)) / real p) *_{\mathbb{R}} i) \circ b) i = 0)$ 
    unfolding o_def using cube  $\langle p > 0 \rangle$  by (intro allI impI label(2)) (auto simp:
  b'')
  have q3:  $\forall x. (\forall i < n. x i \leq p) \longrightarrow (\forall i < n. x i = p \longrightarrow$ 
     $(label (\sum i \in Basis. (real (x (b' i)) / real p) *_{\mathbb{R}} i) \circ b) i = 1)$ 
    using cube  $\langle p > 0 \rangle$  unfolding o_def by (intro allI impI label(3)) (auto simp:
  b'')
  obtain q where q:
     $\forall i < n. q i < p$ 
     $\forall i < n.$ 
     $\exists r s. (\forall j < n. q j \leq r j \wedge r j \leq q j + 1) \wedge$ 
     $(\forall j < n. q j \leq s j \wedge s j \leq q j + 1) \wedge$ 
     $(label (\sum i \in Basis. (real (r (b' i)) / real p) *_{\mathbb{R}} i) \circ b) i \neq$ 
     $(label (\sum i \in Basis. (real (s (b' i)) / real p) *_{\mathbb{R}} i) \circ b) i$ 
    by (rule kuhn_lemma[OF q1 q2 q3])
  define z :: 'a where  $z = (\sum i \in Basis. (real (q (b' i)) / real p) *_{\mathbb{R}} i)$ 
  have  $\exists i \in Basis. d / real n \leq |(f z - z) \cdot i|$ 
  proof (rule ccontr)
    have  $\forall i \in Basis. q (b' i) \in \{0..p\}$ 
      using q(1) b'
      by (auto intro: less_imp_le simp: bij_betw_def)
    then have  $z \in cbox 0 One$ 
      unfolding z_def cbox_def
      using b'_Basis
      by (auto simp: bij_betw_def zero_le_divide_iff divide_le_eq_1)

```

```

then have d_fz_z:  $d \leq \text{norm } (f \, z - z)$ 
  by (rule d)
assume  $\neg ?thesis$ 
then have as:  $\forall i \in \text{Basis}. |f \, z \cdot i - z \cdot i| < d / \text{real } n$ 
  using  $\langle n > 0 \rangle$ 
  by (auto simp: not_le inner_diff)
have  $\text{norm } (f \, z - z) \leq (\sum i \in \text{Basis}. |f \, z \cdot i - z \cdot i|)$ 
  unfolding inner_diff_left[symmetric]
  by (rule norm_le_l1)
also have  $\dots < (\sum (i::'a) \in \text{Basis}. d / \text{real } n)$ 
  by (meson as finite_Basis nonempty_Basis sum_strict_mono)
also have  $\dots = d$ 
  using DIM_positive[where 'a='a] by (auto simp: n_def)
finally show False
  using d_fz_z by auto
qed
then obtain i where  $i: i \in \text{Basis} \, d / \text{real } n \leq |(f \, z - z) \cdot i| \dots$ 
have *:  $b' \, i < n$ 
  using i and b'[unfolded bij_betw_def]
  by auto
obtain r s where rs:
 $\bigwedge j. j < n \implies q \, j \leq r \, j \wedge r \, j \leq q \, j + 1$ 
 $\bigwedge j. j < n \implies q \, j \leq s \, j \wedge s \, j \leq q \, j + 1$ 
 $(\text{label } (\sum i \in \text{Basis}. (\text{real } (r \, (b' \, i)) / \text{real } p) *_{\mathbb{R}} i) \circ b) \, (b' \, i) \neq$ 
 $(\text{label } (\sum i \in \text{Basis}. (\text{real } (s \, (b' \, i)) / \text{real } p) *_{\mathbb{R}} i) \circ b) \, (b' \, i)$ 
  using q(2)[rule_format, OF *] by blast
have b'_im:  $\bigwedge i. i \in \text{Basis} \implies b' \, i < n$ 
  using b' unfolding bij_betw_def by auto
define r' :: 'a where  $r' = (\sum i \in \text{Basis}. (\text{real } (r \, (b' \, i)) / \text{real } p) *_{\mathbb{R}} i)$ 
have  $\bigwedge i. i \in \text{Basis} \implies r \, (b' \, i) \leq p$ 
  apply (rule order_trans)
  apply (rule rs(1)[OF b'_im, THEN conjunct2])
  using q(1)[rule_format, OF b'_im]
  apply (auto simp: Suc_le_eq)
  done
then have r'  $\in \text{cbox } 0 \, \text{One}$ 
  unfolding r'_def cbox_def
  using b'_Basis
  by (auto simp: bij_betw_def zero_le_divide_iff divide_le_eq_1)
define s' :: 'a where  $s' = (\sum i \in \text{Basis}. (\text{real } (s \, (b' \, i)) / \text{real } p) *_{\mathbb{R}} i)$ 
have  $\bigwedge i. i \in \text{Basis} \implies s \, (b' \, i) \leq p$ 
  using b'_im q(1) rs(2) by fastforce
then have s'  $\in \text{cbox } 0 \, \text{One}$ 
  unfolding s'_def cbox_def
  using b'_Basis by (auto simp: bij_betw_def zero_le_divide_iff divide_le_eq_1)
have z  $\in \text{cbox } 0 \, \text{One}$ 
  unfolding z_def cbox_def
  using b'_Basis q(1)[rule_format, OF b'_im]  $\langle p > 0 \rangle$ 
  by (auto simp: bij_betw_def zero_le_divide_iff divide_le_eq_1 less_imp_le)

```

```

{
  have (∑ i∈Basis. |real (r (b' i)) - real (q (b' i))|) ≤ (∑ (i::'a)∈Basis. 1)
    by (rule sum_mono) (use rs(1)[OF b'_im] in force)
  also have ... < e * real p
    using p ⟨e > 0⟩ ⟨p > 0⟩
    by (auto simp: field_simps n_def)
  finally have (∑ i∈Basis. |real (r (b' i)) - real (q (b' i))|) < e * real p .
}
moreover
{
  have (∑ i∈Basis. |real (s (b' i)) - real (q (b' i))|) ≤ (∑ (i::'a)∈Basis. 1)
    by (rule sum_mono) (use rs(2)[OF b'_im] in force)
  also have ... < e * real p
    using p ⟨e > 0⟩ ⟨p > 0⟩
    by (auto simp: field_simps n_def)
  finally have (∑ i∈Basis. |real (s (b' i)) - real (q (b' i))|) < e * real p .
}
ultimately
have norm (r' - z) < e and norm (s' - z) < e
  unfolding r'_def s'_def z_def
  using ⟨p > 0⟩
  apply (rule_tac[!] le_less_trans[OF norm_le_l1])
  apply (auto simp: field_simps sum_divide_distrib[symmetric] inner_diff_left)
  done
then have |(f z - z) • i| < d / real n
  using rs(3) i
  unfolding r'_def[symmetric] s'_def[symmetric] o_def bb'
  by (intro e(2)[OF ⟨r'∈cbox 0 One⟩ ⟨s'∈cbox 0 One⟩ ⟨z∈cbox 0 One⟩]) auto
then show False
  using i by auto
qed

```

Next step is to prove it for nonempty interiors.

lemma *brouwer_weak*:

fixes $f :: 'a::euclidean\_space \Rightarrow 'a$

assumes *compact S*

and *convex S*

and *interior S ≠ {}*

and *continuous_on S f*

and $f \text{ ` } S \subseteq S$

obtains x **where** $x \in S$ **and** $f x = x$

proof –

let $?U = \text{cbox } 0 \text{ One} :: 'a \text{ set}$

have $\sum \text{Basis} /_R 2 \in \text{interior } ?U$

proof (rule interiorI)

let $?I = (\bigcap i \in \text{Basis}. \{x::'a. 0 < x \cdot i\} \cap \{x. x \cdot i < 1\})$

show *open ?I*

by (intro open_INT finite_Basis ballI open_Int, auto intro: open_Collect_less simp: continuous_on_inner)

```

  show  $\sum Basis /_R 2 \in ?I$ 
  by simp
  show  $?I \subseteq cbox\ 0\ One$ 
  unfolding cbox_def by force
qed
then have *: interior  $?U \neq \{\}$  by fast
have *:  $?U$  homeomorphic  $S$ 
  using homeomorphic_convex_compact[OF convex_box(1) compact_cbox *
assms(2,1,3)] .
have  $\forall f. continuous\_on\ ?U\ f \wedge f' : ?U \subseteq ?U \longrightarrow$ 
  ( $\exists x \in ?U. f\ x = x$ )
  using brouwer_cube by auto
then show ?thesis
  unfolding homeomorphic_fixpoint_property[OF *]
  using assms
  by (auto intro: that)
qed

```

Then the particular case for closed balls.

```

lemma brouwer_ball:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'a$ 
  assumes  $e > 0$ 
    and continuous_on (cball  $a\ e$ )  $f$ 
    and  $f' : cball\ a\ e \subseteq cball\ a\ e$ 
  obtains  $x$  where  $x \in cball\ a\ e$  and  $f\ x = x$ 
  using brouwer_weak[OF compact_cball convex_cball, of  $a\ e\ f$ ]
  unfolding interior_cball_ball_eq_empty
  using assms by auto

```

And finally we prove Brouwer's fixed point theorem in its general version.

```

theorem brouwer:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'a$ 
  assumes  $S: compact\ S\ convex\ S\ S \neq \{\}$ 
    and conf: continuous_on  $S\ f$ 
    and fim:  $f' : S \subseteq S$ 
  obtains  $x$  where  $x \in S$  and  $f\ x = x$ 
proof -
  have  $\exists e > 0. S \subseteq cball\ 0\ e$ 
    using compact_imp_bounded[OF compact_S] unfolding bounded_pos
    by auto
  then obtain  $e$  where  $e: e > 0\ S \subseteq cball\ 0\ e$ 
    by blast
  have  $\exists x \in cball\ 0\ e. (f \circ closest\_point\ S)\ x = x$ 
  proof (rule_tac brouwer_ball[OF e(1)])
    show continuous_on (cball  $0\ e$ ) ( $f \circ closest\_point\ S$ )
      apply (rule continuous_on_compose)
      using S compact_eq_bounded_closed continuous_on_closest_point apply
blast
    by (meson S conf closest_point_in_set compact_imp_closed continuous_on_subset

```

```

image_subsetI)
  show (f ∘ closest_point S) ' cball 0 e ⊆ cball 0 e
  by clarsimp (metis S fim closest_point_exists(1) compact_eq_bounded_closed
e(2) image_subset_iff mem_cball_0 subsetCE)
qed (use assms in auto)
then obtain x where x: x ∈ cball 0 e (f ∘ closest_point S) x = x ..
have x ∈ S
  by (metis closest_point_in_set comp_apply compact_imp_closed fim im-
age_eqI S(1) S(3) subset_iff x(2))
then have *: closest_point S x = x
  by (rule closest_point_self)
show thesis
proof
  show closest_point S x ∈ S
  by (simp add: * ⟨x ∈ S⟩)
  show f (closest_point S x) = closest_point S x
  using * x(2) by auto
qed
qed

```

6.31.4 Applications

So we get the no-retraction theorem.

```

corollary no_retraction_cball:
  fixes a :: 'a::euclidean_space
  assumes e > 0
  shows ¬ (frontier (cball a e) retract_of (cball a e))
proof
  assume *: frontier (cball a e) retract_of (cball a e)
  have **: ⋀x a. a - (2 *R a - xa) = - (a - xa)
    using scaleR_left_distrib[of 1 1 a] by auto
  obtain x where x: x ∈ {x. norm (a - x) = e} 2 *R a - x = x
  proof (rule retract_fixpoint_property[OF *, of λx. scaleR 2 a - x])
    show continuous_on (frontier (cball a e)) ((-) (2 *R a))
    by (intro continuous_intros)
    show (-) (2 *R a) ' frontier (cball a e) ⊆ frontier (cball a e)
    by clarsimp (metis ** dist_norm norm_minus_cancel)
  qed (auto simp: dist_norm intro: brouwer_ball[OF assms])
  then have scaleR 2 a = scaleR 1 x + scaleR 1 x
    by (auto simp: algebra_simps)
  then have a = x
    unfolding scaleR_left_distrib[symmetric]
    by auto
  then show False
    using x assms by auto
qed

```

```

corollary contractible_sphere:
  fixes a :: 'a::euclidean_space

```



```

  shows contractible(sphere a r)  $\longleftrightarrow$   $r \leq 0$ 
proof (cases 0 < r)
  case True
  then show ?thesis
    unfolding contractible_def nullhomotopic_from_sphere_extension
    using no_retraction_cball [OF True, of a]
    by (auto simp: retract_of_def retraction_def)
next
  case False
  then show ?thesis
    unfolding contractible_def nullhomotopic_from_sphere_extension
    using less_eq_real_def by auto
qed

corollary connected_sphere_eq:
  fixes a :: 'a :: euclidean_space
  shows connected(sphere a r)  $\longleftrightarrow$   $2 \leq \text{DIM}('a) \vee r \leq 0$ 
  (is ?lhs = ?rhs)
proof (cases r 0::real rule: linorder_cases)
  case less
  then show ?thesis by auto
next
  case equal
  then show ?thesis by auto
next
  case greater
  show ?thesis
  proof
    assume L: ?lhs
    have False if 1: DIM('a) = 1
    proof -
      obtain x y where xy: sphere a r = {x,y} x  $\neq$  y
      using sphere_1D_doubleton [OF 1 greater]
      by (metis dist_self greater insertI1 less_add_same_cancel1 mem_sphere
mult_2 not_le zero_le_dist)
      then have finite (sphere a r)
      by auto
      with L <r > 0> xy show False
      using connected_finite_iff_sing by auto
    qed
    with greater show ?rhs
    by (metis DIM_ge_Suc0 One_nat_def Suc_1 le_antisym not_less_eq_eq)
  next
    assume ?rhs
    then show ?lhs
    using connected_sphere_greater by auto
  qed
qed

```

```

corollary path_connected_sphere_eq:
  fixes  $a :: 'a :: euclidean\_space$ 
  shows  $path\_connected(sphere\ a\ r) \longleftrightarrow 2 \leq DIM('a) \vee r \leq 0$ 
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    using connected_sphere_eq_path_connected_imp_connected by blast
next
  assume  $R: ?rhs$ 
  then show ?lhs
    by (auto simp: contractible_imp_path_connected contractible_sphere path_connected_sphere)
qed

```

```

proposition frontier_subset_retraction:
  fixes  $S :: 'a::euclidean\_space\ set$ 
  assumes bounded  $S$  and fros:  $frontier\ S \subseteq T$ 
    and contf: continuous_on (closure  $S$ )  $f$ 
    and fim:  $f\ 'S \subseteq T$ 
    and fid:  $\bigwedge x. x \in T \implies f\ x = x$ 
  shows  $S \subseteq T$ 
proof (rule ccontr)
  assume  $\neg S \subseteq T$ 
  then obtain  $a$  where  $a \in S\ a \notin T$  by blast
  define  $g$  where  $g \equiv \lambda z. \text{if } z \in \text{closure } S \text{ then } f\ z \text{ else } z$ 
  have continuous_on (closure  $S \cup \text{closure}(-S)$ )  $g$ 
    unfolding g_def
    apply (rule continuous_on_cases)
    using fros fid frontier_closures by (auto simp: contf)
  moreover have  $\text{closure } S \cup \text{closure}(-S) = UNIV$ 
    using closure_Un by fastforce
  ultimately have contg: continuous_on  $UNIV\ g$  by metis
  obtain  $B$  where  $0 < B$  and  $B: \text{closure } S \subseteq \text{ball } a\ B$ 
    using  $\langle \text{bounded } S \rangle$  bounded_subset_ballD by blast
  have notga:  $g\ x \neq a$  for  $x$ 
    unfolding g_def using fros fim  $\langle a \notin T \rangle$ 
    apply (auto simp: frontier_def)
    using fid interior_subset apply fastforce
    by (simp add:  $\langle a \in S \rangle$  closure_def)
  define  $h$  where  $h \equiv (\lambda y. a + (B / \text{norm}(y - a)) *_{\mathbb{R}} (y - a)) \circ g$ 
  have  $\neg (\text{frontier } (\text{cball } a\ B) \text{ retract\_of } (\text{cball } a\ B))$ 
    by (metis no_retraction_cball  $\langle 0 < B \rangle$ )
  then have  $\bigwedge k. \neg \text{retraction } (\text{cball } a\ B) (\text{frontier } (\text{cball } a\ B))\ k$ 
    by (simp add: retract_of_def)
  moreover have retraction (cball  $a\ B$ ) (frontier (cball  $a\ B$ ))  $h$ 
    unfolding retraction_def
proof (intro conjI ballI)
  show  $\text{frontier } (\text{cball } a\ B) \subseteq \text{cball } a\ B$ 
    by force

```

```

show continuous_on (cball a B) h
  unfolding h_def
  by (intro continuous_intros) (use contg continuous_on_subset notga in auto)
show h ' cball a B  $\subseteq$  frontier (cball a B)
  using '0 < B' by (auto simp: h_def notga dist_norm)
show  $\bigwedge x. x \in \text{frontier (cball a B)} \implies h\ x = x$ 
  apply (auto simp: h_def algebra_simps)
  apply (simp add: vector_add_divide_simps notga)
  by (metis (no_types, opaque_lifting) B add.commute dist_commute dist_norm
g_def mem_ball not_less_iff_gr_or_eq subset_eq)
qed
ultimately show False by simp
qed

```

Punctured affine hulls, etc

lemma *rel_frontier_deformation_retract_of_punctured_convex*:

```

fixes S :: 'a::euclidean_space set
assumes convex S convex T bounded S
  and arelS: a  $\in$  rel_interior S
  and relS: rel_frontier S  $\subseteq$  T
  and affS: T  $\subseteq$  affine hull S
obtains r where homotopic_with_canon ( $\lambda x. \text{True}$ ) (T - {a}) (T - {a}) id r
  retraction (T - {a}) (rel_frontier S) r

```

proof –

```

have  $\exists d. 0 < d \wedge (a + d *_R l) \in \text{rel\_frontier } S \wedge$ 
  ( $\forall e. 0 \leq e \wedge e < d \implies (a + e *_R l) \in \text{rel\_interior } S$ )
  if  $(a + l) \in \text{affine hull } S \wedge l \neq 0$  for l
  apply (rule ray_to_rel_frontier [OF 'bounded S' arelS])
  apply (rule that)+
  by metis
then obtain dd
  where dd1:  $\bigwedge l. [(a + l) \in \text{affine hull } S; l \neq 0] \implies 0 < dd\ l \wedge (a + dd\ l *_R$ 
l)  $\in$  rel_frontier S
  and dd2:  $\bigwedge l\ e. [(a + l) \in \text{affine hull } S; e < dd\ l; 0 \leq e; l \neq 0] \implies$ 
   $(a + e *_R l) \in \text{rel\_interior } S$ 
  by metis+
have aaffS: a  $\in$  affine hull S
  by (meson arelS subsetD hull_inc rel_interior_subset)
have (( $\lambda z. z - a$ ) ' (affine hull S - {a})) = (( $\lambda z. z - a$ ) ' (affine hull S)) - {0}
  by auto
moreover have continuous_on ((( $\lambda z. z - a$ ) ' (affine hull S)) - {0}) ( $\lambda x. dd$ 
x *R x)

```

```

proof (rule continuous_on_compact_surface_projection)
show compact (rel_frontier (( $\lambda z. z - a$ ) ' S))
  by (simp add: 'bounded S' bounded_translation_minus compact_rel_frontier_bounded)
have reeq: rel_frontier (( $\lambda z. z - a$ ) ' S) = ( $\lambda z. z - a$ ) ' rel_frontier S
  using rel_frontier_translation [of -a] add.commute by simp

```

```

also have ...  $\subseteq (\lambda z. z - a) \text{ ' } (\text{affine hull } S) - \{0\}$ 
  using rel_frontier_affine_hull arelS rel_frontier_def by fastforce
finally show rel_frontier  $((\lambda z. z - a) \text{ ' } S) \subseteq (\lambda z. z - a) \text{ ' } (\text{affine hull } S) - \{0\}$  .
show cone  $((\lambda z. z - a) \text{ ' } (\text{affine hull } S))$ 
  by (rule subspace_imp_cone)
  (use aaffS in <simp add: subspace_affine image_comp o_def affine_translation_aux [of a]>)
show  $(0 < k \wedge k *_{\mathbb{R}} x \in \text{rel\_frontier } ((\lambda z. z - a) \text{ ' } S)) \longleftrightarrow (dd \ x = k)$ 
  if  $x \in (\lambda z. z - a) \text{ ' } (\text{affine hull } S) - \{0\}$  for  $k \ x$ 
proof
  show  $dd \ x = k \implies 0 < k \wedge k *_{\mathbb{R}} x \in \text{rel\_frontier } ((\lambda z. z - a) \text{ ' } S)$ 
  using dd1 [of x] that image_iff by (fastforce simp add: releq)
next
  assume k:  $0 < k \wedge k *_{\mathbb{R}} x \in \text{rel\_frontier } ((\lambda z. z - a) \text{ ' } S)$ 
  have False if  $dd \ x < k$ 
  proof -
    have  $k \neq 0 \wedge a + k *_{\mathbb{R}} x \in \text{closure } S$ 
    using k closure_translation [of -a]
    by (auto simp: rel_frontier_def cong: image_cong_simp)
    then have segsub:  $\text{open\_segment } a \ (a + k *_{\mathbb{R}} x) \subseteq \text{rel\_interior } S$ 
    by (metis rel_interior_closure_convex_segment [OF <convex S> arelS])
    have  $x \neq 0$  and xaffS:  $a + x \in \text{affine hull } S$ 
    using x by auto
    then have  $0 < dd \ x$  and inS:  $a + dd \ x *_{\mathbb{R}} x \in \text{rel\_frontier } S$ 
    using dd1 by auto
    moreover have  $a + dd \ x *_{\mathbb{R}} x \in \text{open\_segment } a \ (a + k *_{\mathbb{R}} x)$ 
    using k <x ≠ 0> <0 < dd x>
    apply (simp add: in_segment)
    apply (rule_tac x = dd x / k in exI)
    apply (simp add: field_simps that)
    apply (simp add: vector_add_divide_simps algebra_simps)
    done
    ultimately show ?thesis
    using segsub by (auto simp: rel_frontier_def)
  qed
  moreover have False if  $k < dd \ x$ 
  using x k that rel_frontier_def
  by (fastforce simp: algebra_simps releq dest!: dd2)
  ultimately show  $dd \ x = k$ 
  by fastforce
qed
qed
ultimately have *: continuous_on  $((\lambda z. z - a) \text{ ' } (\text{affine hull } S - \{a\})) (\lambda x. dd \ x *_{\mathbb{R}} x)$ 
  by auto
have continuous_on  $(\text{affine hull } S - \{a\}) ((\lambda x. a + dd \ x *_{\mathbb{R}} x) \circ (\lambda z. z - a))$ 
  by (intro * continuous_intros continuous_on_compose)
with affS have contdd: continuous_on  $(T - \{a\}) ((\lambda x. a + dd \ x *_{\mathbb{R}} x) \circ (\lambda z.$ 

```

```

z - a))
  by (blast intro: continuous_on_subset)
show ?thesis
proof
  show homotopic_with_canon ( $\lambda x. \text{True}$ ) ( $T - \{a\}$ ) ( $T - \{a\}$ ) id ( $\lambda x. a + dd$ 
( $x - a$ )  $*_R$  ( $x - a$ ))
  proof (rule homotopic_with_linear)
    show continuous_on ( $T - \{a\}$ ) id
      by (intro continuous_intros continuous_on_compose)
    show continuous_on ( $T - \{a\}$ ) ( $\lambda x. a + dd$  ( $x - a$ )  $*_R$  ( $x - a$ ))
      using contdd by (simp add: o_def)
    show closed_segment (id x) ( $a + dd$  ( $x - a$ )  $*_R$  ( $x - a$ ))  $\subseteq T - \{a\}$ 
      if  $x \in T - \{a\}$  for x
  proof (clarsimp simp: in_segment, intro conjI)
    fix u::real assume u:  $0 \leq u \leq 1$ 
    have  $a + dd$  ( $x - a$ )  $*_R$  ( $x - a$ )  $\in T$ 
    by (metis DiffD1 DiffD2 add commute add.right_neutral affS dd1 diff_add_cancel
relS singletonI subsetCE that)
    then show  $(1 - u) *_R x + u *_R (a + dd$  ( $x - a$ )  $*_R$  ( $x - a$ ))  $\in T$ 
      using convexD [OF <convex T>] that u by simp
    have iff:  $(1 - u) *_R x + u *_R (a + d *_R (x - a)) = a \longleftrightarrow$ 
       $(1 - u + u * d) *_R (x - a) = 0$  for d
      by (auto simp: algebra_simps)
    have  $x \in T \wedge x \neq a$  using that by auto
    then have axa:  $a + (x - a) \in \text{affine hull } S$ 
      by (metis (no_types) add commute affS diff_add_cancel rev_subsetD)
    then have  $\neg dd$  ( $x - a$ )  $\leq 0 \wedge a + dd$  ( $x - a$ )  $*_R$  ( $x - a$ )  $\in \text{rel\_frontier } S$ 
      using <x  $\neq$  a> dd1 by fastforce
    with <x  $\neq$  a> show  $(1 - u) *_R x + u *_R (a + dd$  ( $x - a$ )  $*_R$  ( $x - a$ ))  $\neq a$ 
      apply (auto simp: iff)
      using less_eq_real_def mult_le_0_iff not_less u by fastforce
  qed
qed
show retraction ( $T - \{a\}$ ) ( $\text{rel\_frontier } S$ ) ( $\lambda x. a + dd$  ( $x - a$ )  $*_R$  ( $x - a$ ))
proof (simp add: retraction_def, intro conjI ballI)
  show  $\text{rel\_frontier } S \subseteq T - \{a\}$ 
    using arelS relS rel_frontier_def by fastforce
  show continuous_on ( $T - \{a\}$ ) ( $\lambda x. a + dd$  ( $x - a$ )  $*_R$  ( $x - a$ ))
    using contdd by (simp add: o_def)
  show  $(\lambda x. a + dd$  ( $x - a$ )  $*_R$  ( $x - a$ )) ' ( $T - \{a\}$ )  $\subseteq \text{rel\_frontier } S$ 
    apply (auto simp: rel_frontier_def)
  apply (metis Diff_subset add commute affS dd1 diff_add_cancel eq_iff_diff_eq_0
rel_frontier_def subset_iff)
  by (metis DiffE add commute affS dd1 diff_add_cancel eq_iff_diff_eq_0
rel_frontier_def rev_subsetD)
  show  $a + dd$  ( $x - a$ )  $*_R$  ( $x - a$ ) = x if x:  $x \in \text{rel\_frontier } S$  for x
  proof -
    have  $x \neq a$ 
      using that arelS by (auto simp: rel_frontier_def)

```

```

have False if dd (x - a) < 1
proof -
  have x ∈ closure S
  using x by (auto simp: rel_frontier_def)
  then have segsub: open_segment a x ⊆ rel_interior S
  by (metis rel_interior_closure_convex_segment [OF ⟨convex S⟩ arelS])
  have xaffS: x ∈ affine hull S
  using affS relS x by auto
  then have 0 < dd (x - a) and inS: a + dd (x - a) *R (x - a) ∈
rel_frontier S
  using dd1 by (auto simp: ⟨x ≠ a⟩)
  moreover have a + dd (x - a) *R (x - a) ∈ open_segment a x
  using ⟨x ≠ a⟩ ⟨0 < dd (x - a)⟩
  apply (simp add: in_segment)
  apply (rule_tac x = dd (x - a) in exI)
  apply (simp add: algebra_simps that)
  done
  ultimately show ?thesis
  using segsub by (auto simp: rel_frontier_def)
qed
moreover have False if 1 < dd (x - a)
  using x that dd2 [of x - a 1] ⟨x ≠ a⟩ closure_affine_hull
  by (auto simp: rel_frontier_def)
ultimately have dd (x - a) = 1 — similar to another proof above
  by fastforce
with that show ?thesis
  by (simp add: rel_frontier_def)
qed
qed
qed
qed

corollary rel_frontier_retract_of_punctured_affine_hull:
  fixes S :: 'a::euclidean_space set
  assumes bounded S convex S a ∈ rel_interior S
  shows rel_frontier S retract_of (affine hull S - {a})
apply (rule rel_frontier_deformation_retract_of_punctured_convex [of S affine
hull S a])
apply (auto simp: affine_imp_convex rel_frontier_affine_hull retract_of_def assms)
done

corollary rel_boundary_retract_of_punctured_affine_hull:
  fixes S :: 'a::euclidean_space set
  assumes compact S convex S a ∈ rel_interior S
  shows (S - rel_interior S) retract_of (affine hull S - {a})
by (metis assms closure_closed compact_eq_bounded_closed rel_frontier_def
rel_frontier_retract_of_punctured_affine_hull)

lemma homotopy_eqv_rel_frontier_punctured_convex:

```

```

fixes  $S :: 'a::euclidean\_space\ set$ 
assumes  $convex\ S\ bounded\ S\ a \in rel\_interior\ S\ convex\ T\ rel\_frontier\ S \subseteq T\ T$ 
 $\subseteq affine\ hull\ S$ 
shows  $(rel\_frontier\ S)\ homotopy\_eqv\ (T - \{a\})$ 
apply  $(rule\ rel\_frontier\_deformation\_retract\_of\_punctured\_convex\ [of\ S\ T])$ 
using  $assms$ 
apply  $auto$ 
using  $deformation\_retract\_imp\_homotopy\_eqv\ homotopy\_equivalent\_space\_sym$ 
by  $blast$ 

```

```

lemma  $homotopy\_eqv\_rel\_frontier\_punctured\_affine\_hull$ :
fixes  $S :: 'a::euclidean\_space\ set$ 
assumes  $convex\ S\ bounded\ S\ a \in rel\_interior\ S$ 
shows  $(rel\_frontier\ S)\ homotopy\_eqv\ (affine\ hull\ S - \{a\})$ 
apply  $(rule\ homotopy\_eqv\_rel\_frontier\_punctured\_convex)$ 
using  $assms\ rel\_frontier\_affine\_hull\ by\ force+$ 

```

```

lemma  $path\_connected\_sphere\_gen$ :
assumes  $convex\ S\ bounded\ S\ aff\_dim\ S \neq 1$ 
shows  $path\_connected(rel\_frontier\ S)$ 
proof  $(cases\ rel\_interior\ S = \{\})$ 
case  $True$ 
then show  $?thesis$ 
by  $(simp\ add: \langle convex\ S \rangle\ convex\_imp\_path\_connected\ rel\_frontier\_def)$ 
next
case  $False$ 
then show  $?thesis$ 
by  $(metis\ aff\_dim\_affine\_hull\ affine\_affine\_hull\ affine\_imp\_convex\ all\_not\_in\_conv$ 
 $assms\ path\_connected\_punctured\_convex\ rel\_frontier\_retract\_of\_punctured\_affine\_hull$ 
 $retract\_of\_path\_connected)$ 
qed

```

```

lemma  $connected\_sphere\_gen$ :
assumes  $convex\ S\ bounded\ S\ aff\_dim\ S \neq 1$ 
shows  $connected(rel\_frontier\ S)$ 
by  $(simp\ add: assms\ path\_connected\_imp\_connected\ path\_connected\_sphere\_gen)$ 

```

Borsuk-style characterization of separation

```

lemma  $continuous\_on\_Borsuk\_map$ :
 $a \notin s \implies continuous\_on\ s\ (\lambda x. inverse(norm\ (x - a)) *_R (x - a))$ 
by  $(rule\ continuous\_intros\ | force)+$ 

```

```

lemma  $Borsuk\_map\_into\_sphere$ :
 $(\lambda x. inverse(norm\ (x - a)) *_R (x - a))\ 's \subseteq sphere\ 0\ 1 \longleftrightarrow (a \notin s)$ 
by  $auto\ (metis\ eq\_iff\_diff\_eq\ 0\ left\_inverse\ norm\_eq\_zero)$ 

```

```

lemma  $Borsuk\_maps\_homotopic\_in\_path\_component$ :
assumes  $path\_component\ (-\ s)\ a\ b$ 

```

```

shows homotopic_with_canon ( $\lambda x. \text{True}$ )  $s$  ( $\text{sphere } 0 \ 1$ )
  ( $\lambda x. \text{inverse}(\text{norm}(x - a)) *_R (x - a)$ )
  ( $\lambda x. \text{inverse}(\text{norm}(x - b)) *_R (x - b)$ )
proof -
  obtain  $g$  where  $\text{path } g \text{ path\_image } g \subseteq -s$   $\text{pathstart } g = a$   $\text{pathfinish } g = b$ 
  using  $\text{assms}$  by ( $\text{auto simp: path\_component\_def}$ )
  then show ?thesis
    apply ( $\text{simp add: path\_def path\_image\_def pathstart\_def pathfinish\_def homotopic\_with\_def}$ )
    apply ( $\text{rule\_tac } x = \lambda z. \text{inverse}(\text{norm}(\text{snd } z - (g \circ \text{fst})z)) *_R (\text{snd } z - (g \circ \text{fst})z)$  in  $\text{exI}$ )
    apply ( $\text{intro conjI continuous\_intros}$ )
    apply ( $\text{rule continuous\_intros | erule continuous\_on\_subset | fastforce simp: divide\_simps sphere\_def}$ ) +
    done
qed

lemma non_extensible_Borsuk_map:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  assumes  $\text{compact } s$  and  $\text{cin: } c \in \text{components}(-s)$  and  $\text{boc: bounded } c$  and  $a \in c$ 
  shows  $\neg (\exists g. \text{continuous\_on } (s \cup c) \ g \wedge$ 
     $g \text{ ` } (s \cup c) \subseteq \text{sphere } 0 \ 1 \wedge$ 
     $(\forall x \in s. g \ x = \text{inverse}(\text{norm}(x - a)) *_R (x - a)))$ 
proof -
  have  $\text{closed } s$  using  $\text{assms}$  by ( $\text{simp add: compact\_imp\_closed}$ )
  have  $c \subseteq -s$ 
  using  $\text{assms}$  by ( $\text{simp add: in\_components\_subset}$ )
  with  $\langle a \in c \rangle$  have  $a \notin s$  by blast
  then have  $\text{ceq: } c = \text{connected\_component\_set } (-s) \ a$ 
  by ( $\text{metis } \langle a \in c \rangle \text{ cin components\_iff connected\_component\_eq}$ )
  then have  $\text{bounded } (s \cup \text{connected\_component\_set } (-s) \ a)$ 
  using  $\langle \text{compact } s \rangle \text{ boc compact\_imp\_bounded}$  by auto
  with  $\text{bounded\_subset\_ballD}$  obtain  $r$  where  $0 < r$  and  $r: (s \cup \text{connected\_component\_set } (-s) \ a) \subseteq \text{ball } a \ r$ 
  by blast
  { fix  $g$ 
    assume  $\text{continuous\_on } (s \cup c) \ g$ 
     $g \text{ ` } (s \cup c) \subseteq \text{sphere } 0 \ 1$ 
    and  $[\text{simp}]: \bigwedge x. x \in s \implies g \ x = (x - a) /_R \text{norm } (x - a)$ 
    then have  $[\text{simp}]: \bigwedge x. x \in s \cup c \implies \text{norm } (g \ x) = 1$ 
    by force
    have  $\text{cb\_eq: } \text{cball } a \ r = (s \cup \text{connected\_component\_set } (-s) \ a) \cup$ 
       $(\text{cball } a \ r - \text{connected\_component\_set } (-s) \ a)$ 
    using  $\text{ball\_subset\_cball [of } a \ r]$   $r$  by auto
    have  $\text{cont1: continuous\_on } (s \cup \text{connected\_component\_set } (-s) \ a)$ 
       $(\lambda x. a + r *_R g \ x)$ 
    apply ( $\text{rule continuous\_intros}$ ) +
    using  $\langle \text{continuous\_on } (s \cup c) \ g \rangle \text{ ceq}$  by blast
  }

```



```

have cont2: continuous_on (cball a r - connected_component_set (- s) a)
  ((λx. a + r *R ((x - a) /R norm (x - a))))
by (rule continuous_intros | force simp: ⟨a ∉ s⟩)+
have 1: continuous_on (cball a r)
  ((λx. if connected_component (- s) a x
    then a + r *R g x
    else a + r *R ((x - a) /R norm (x - a))))
apply (subst cb_eq)
apply (rule continuous_on_cases [OF _ _ cont1 cont2])
using ceq cin
apply (auto intro: closed_Un_complement_component
  simp: ⟨closed s⟩ open_Cmpl open_connected_component)
done
have 2: (λx. a + r *R g x) ' (cball a r ∩ connected_component_set (- s) a)
  ⊆ sphere a r
using ⟨0 < r⟩ by (force simp: dist_norm ceq)
have retraction (cball a r) (sphere a r)
  ((λx. if x ∈ connected_component_set (- s) a
    then a + r *R g x
    else a + r *R ((x - a) /R norm (x - a))))
using ⟨0 < r⟩
apply (simp add: retraction_def dist_norm 1 2, safe)
apply (force simp: dist_norm abs_if mult_less_0_iff divide_simps ⟨a ∉ s⟩)
using r
by (auto simp: dist_norm norm_minus_commute)
then have False
using no_retraction_cball
  [OF ⟨0 < r⟩, of a, unfolded retract_of_def, simplified, rule_format,
  of λx. if x ∈ connected_component_set (- s) a
    then a + r *R g x
    else a + r *R inverse(norm(x - a)) *R (x - a)]
by blast
}
then show ?thesis
by blast
qed

```

Proving surjectivity via Brouwer fixpoint theorem

```

lemma brouwer_surjective:
  fixes f :: 'n::euclidean_space ⇒ 'n
  assumes compact T
  and convex T
  and T ≠ {}
  and continuous_on T f
  and ∧ x y. [x ∈ S; y ∈ T] ⇒ x + (y - f y) ∈ T
  and x ∈ S
  shows ∃ y ∈ T. f y = x
proof -

```

```

have *:  $\bigwedge x y. f y = x \longleftrightarrow x + (y - f y) = y$ 
  by (auto simp add: algebra_simps)
show ?thesis
  unfolding *
  apply (rule brouwer[OF assms(1-3), of  $\lambda y. x + (y - f y)$ ])
  apply (intro continuous_intros)
  using assms
  apply auto
  done
qed

```

```

lemma brouwer_surjective_cball:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'n
  assumes continuous_on (cball a e) f
    and e > 0
    and x  $\in$  S
    and  $\bigwedge x y. [x \in S; y \in \text{cball } a \text{ } e] \implies x + (y - f y) \in \text{cball } a \text{ } e$ 
  shows  $\exists y \in \text{cball } a \text{ } e. f y = x$ 
  apply (rule brouwer_surjective)
  apply (rule compact_cball_convex_cball)+
  unfolding cball_eq_empty
  using assms
  apply auto
  done

```

Inverse function theorem

See Sussmann: "Multidifferential calculus", Theorem 2.1.1

```

lemma sussmann_open_mapping:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::euclidean_space
  assumes open S
    and contf: continuous_on S f
    and x  $\in$  S
    and derf: (f has_derivative f') (at x)
    and bounded_linear g' f'  $\circ$  g' = id
    and T  $\subseteq$  S
    and x: x  $\in$  interior T
  shows f x  $\in$  interior (f ` T)
proof -
  interpret f': bounded_linear f'
    using assms unfolding has_derivative_def by auto
  interpret g': bounded_linear g'
    using assms by auto
  obtain B where B: 0 < B  $\forall x. \text{norm } (g' x) \leq \text{norm } x * B$ 
    using bounded_linear.pos_bounded[OF assms(5)] by blast
  hence *: 1 / (2 * B) > 0 by auto
  obtain e0 where e0:
    0 < e0
     $\forall y. \text{norm } (y - x) < e0 \longrightarrow \text{norm } (f y - f x - f' (y - x)) \leq 1 / (2 * B) *$ 

```

```

norm (y - x)
  using derf unfolding has_derivative_at_alt
  using * by blast
obtain e1 where e1: 0 < e1 cball x e1 ⊆ T
  using mem_interior_cball x by blast
have *: 0 < e0 / B 0 < e1 / B using e0 e1 B by auto
obtain e where e: 0 < e e < e0 / B e < e1 / B
  using field_lbound_gt_zero[OF *] by blast
have lem: ∃ y ∈ cball (f x) e. f (x + g' (y - f x)) = z if z ∈ cball (f x) (e / 2) for
z
proof (rule brouwer_surjective_cball)
  have z: z ∈ S if as: y ∈ cball (f x) e z = x + (g' y - g' (f x)) for y z
  proof-
    have dist x z = norm (g' (f x) - g' y)
      unfolding as(2) and dist_norm by auto
    also have ... ≤ norm (f x - y) * B
      by (metis B(2) g'.diff)
    also have ... ≤ e * B
      by (metis B(1) dist_norm mem_cball mult_le_cancel_iff1 that(1))
    also have ... ≤ e1
      using B(1) e(3) pos_less_divide_eq by fastforce
    finally have z ∈ cball x e1
      by force
    then show z ∈ S
      using e1 assms(7) by auto
  qed
show continuous_on (cball (f x) e) (λy. f (x + g' (y - f x)))
  unfolding g'.diff
proof (rule continuous_on_compose2 [OF _ _ order_refl, of _ _ f])
  show continuous_on ((λy. x + (g' y - g' (f x))) ' cball (f x) e) f
    by (rule continuous_on_subset[OF contf]) (use z in blast)
  show continuous_on (cball (f x) e) (λy. x + (g' y - g' (f x)))
    by (intro continuous_intros linear_continuous_on[OF ‹bounded_linear g'›])
  qed
next
fix y z
assume y: y ∈ cball (f x) (e / 2) and z: z ∈ cball (f x) e
have norm (g' (z - f x)) ≤ norm (z - f x) * B
  using B by auto
also have ... ≤ e * B
  by (metis B(1) z dist_norm mem_cball norm_minus_commute mult_le_cancel_iff1)
also have ... < e0
  using B(1) e(2) pos_less_divide_eq by blast
finally have *: norm (x + g' (z - f x) - x) < e0
  by auto
have **: f x + f' (x + g' (z - f x) - x) = z
  using assms(6)[unfolded o_def id_def, THEN cong]
  by auto
have norm (f x - (y + (z - f (x + g' (z - f x))))) ≤

```

```

      norm (f (x + g' (z - f x)) - z) + norm (f x - y)
    using norm_triangle_ineq[of f (x + g' (z - f x)) - z f x - y]
    by (auto simp add: algebra_simps)
  also have ... ≤ 1 / (B * 2) * norm (g' (z - f x)) + norm (f x - y)
    using e0(2)[rule_format, OF *]
    by (simp only: algebra_simps **) auto
  also have ... ≤ 1 / (B * 2) * norm (g' (z - f x)) + e/2
    using y by (auto simp: dist_norm)
  also have ... ≤ 1 / (B * 2) * B * norm (z - f x) + e/2
    using * B by (auto simp add: field_simps)
  also have ... ≤ 1 / 2 * norm (z - f x) + e/2
    by auto
  also have ... ≤ e/2 + e/2
    using B(1) ⟨norm (z - f x) * B ≤ e * B⟩ by auto
  finally show y + (z - f (x + g' (z - f x))) ∈ cball (f x) e
    by (auto simp: dist_norm)
qed (use e that in auto)
show ?thesis
  unfolding mem_interior
proof (intro exI conjI subsetI)
  fix y
  assume y ∈ ball (f x) (e / 2)
  then have *: y ∈ cball (f x) (e / 2)
    by auto
  obtain z where z: z ∈ cball (f x) e ∧ f (x + g' (z - f x)) = y
    using lem * by blast
  then have norm (g' (z - f x)) ≤ norm (z - f x) * B
    using B
    by (auto simp add: field_simps)
  also have ... ≤ e * B
    by (metis B(1) dist_norm mem_cball norm_minus_commute mult_le_cancel_iff1
z(1))
  also have ... ≤ e1
    using e B unfolding less_divide_eq by auto
  finally have x + g'(z - f x) ∈ T
    by (metis add_diff_cancel diff_diff_add dist_norm e1(2) mem_cball norm_minus_commute
subset_eq)
  then show y ∈ f ' T
    using z by auto
qed (use e in auto)
qed

```

Hence the following eccentric variant of the inverse function theorem. This has no continuity assumptions, but we do need the inverse function. We could put $f' \circ g = I$ but this happens to fit with the minimal linear algebra theory I've set up so far.

lemma *has_derivative_inverse_strong*:
fixes $f :: 'n::\text{euclidean\_space} \Rightarrow 'n$
assumes *open S*

```

    and  $x \in S$ 
    and contf: continuous_on  $S$   $f$ 
    and gf:  $\bigwedge x. x \in S \implies g(f\ x) = x$ 
    and derf: (f has_derivative  $f'$ ) (at  $x$ )
    and id:  $f' \circ g' = \text{id}$ 
  shows (g has_derivative  $g'$ ) (at ( $f\ x$ ))
proof -
  have linf: bounded_linear  $f'$ 
    using derf unfolding has_derivative_def by auto
  then have ling: bounded_linear  $g'$ 
    unfolding linear_conv_bounded_linear[symmetric]
    using id right_inverse_linear by blast
  moreover have  $g' \circ f' = \text{id}$ 
    using id linf ling
    unfolding linear_conv_bounded_linear[symmetric]
    using linear_inverse_left
    by auto
  moreover have *:  $\bigwedge T. \llbracket T \subseteq S; x \in \text{interior } T \rrbracket \implies f\ x \in \text{interior } (f' \text{ ` } T)$ 
    apply (rule sussmann_open_mapping)
    apply (rule assms ling)+
    apply auto
    done
  have continuous (at ( $f\ x$ ))  $g$ 
    unfolding continuous_at Lim_at
  proof (rule, rule)
    fix  $e :: \text{real}$ 
    assume  $e > 0$ 
    then have  $f\ x \in \text{interior } (f' \text{ ` } (\text{ball } x\ e \cap S))$ 
      using *[rule_format, of  $\text{ball } x\ e \cap S$ ]  $\langle x \in S \rangle$ 
      by (auto simp add: interior_open[OF open_ball] interior_open[OF assms(1)])
    then obtain  $d$  where  $d: 0 < d$   $\text{ball } (f\ x)\ d \subseteq f' \text{ ` } (\text{ball } x\ e \cap S)$ 
      unfolding mem_interior by blast
    show  $\exists d > 0. \forall y. 0 < \text{dist } y\ (f\ x) \wedge \text{dist } y\ (f\ x) < d \longrightarrow \text{dist } (g\ y)\ (g\ (f\ x)) < e$ 
  proof (intro exI allI impI conjI)
    fix  $y$ 
    assume  $0 < \text{dist } y\ (f\ x) \wedge \text{dist } y\ (f\ x) < d$ 
    then have  $g\ y \in g' \text{ ` } f' \text{ ` } (\text{ball } x\ e \cap S)$ 
      by (metis  $d(2)$  dist_commute mem_ball rev_image_eqI subset_iff)
    then show  $\text{dist } (g\ y)\ (g\ (f\ x)) < e$ 
      using gf[OF  $\langle x \in S \rangle$ ]
      by (simp add: assms(4) dist_commute image_iff)
    qed (use  $d$  in auto)
  qed
  moreover have  $f\ x \in \text{interior } (f' \text{ ` } S)$ 
    apply (rule sussmann_open_mapping)
    apply (rule assms ling)+
    using interior_open[OF assms(1)] and  $\langle x \in S \rangle$ 
    apply auto

```

```

done
moreover have  $f(g\ y) = y$  if  $y \in \text{interior}(f^{-1} S)$  for  $y$ 
  by (metis gf_imageE interiorE_subsetD that)
ultimately show ?thesis using assms
  by (metis has_derivative_inverse_basic_x open_interior)
qed

```

A rewrite based on the other domain.

```

lemma has_derivative_inverse_strong_x:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'a$ 
  assumes open  $S$ 
  and  $g\ y \in S$ 
  and continuous_on  $S\ f$ 
  and  $\bigwedge x. x \in S \implies g(f\ x) = x$ 
  and  $(f\ \text{has\_derivative}\ f')\ (\text{at}\ (g\ y))$ 
  and  $f' \circ g' = \text{id}$ 
  and  $f(g\ y) = y$ 
  shows  $(g\ \text{has\_derivative}\ g')\ (\text{at}\ y)$ 
  using has_derivative_inverse_strong[OF assms(1–6)]
  unfolding assms(7)
  by simp

```

On a region.

```

theorem has_derivative_inverse_on:
  fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'n$ 
  assumes open  $S$ 
  and derf:  $\bigwedge x. x \in S \implies (f\ \text{has\_derivative}\ f'(x))\ (\text{at}\ x)$ 
  and  $\bigwedge x. x \in S \implies g(f\ x) = x$ 
  and  $f' x \circ g' x = \text{id}$ 
  and  $x \in S$ 
  shows  $(g\ \text{has\_derivative}\ g'(x))\ (\text{at}\ (f\ x))$ 
proof (rule has_derivative_inverse_strong[where  $g'=g' x$  and  $f=f$ ])
  show continuous_on  $S\ f$ 
  unfolding continuous_on_eq_continuous_at[OF ‹open  $S$ ›]
  using derf has_derivative_continuous by blast
qed (use assms in auto)

```

end

6.32 Fashoda Meet Theorem

```

theory Fashoda_Theorem
imports Brouwer_Fixpoint Path_Connected Cartesian_Euclidean_Space
begin

```

6.32.1 Bijections between intervals

```

definition interval_bij ::  $'a \times 'a \Rightarrow 'a \times 'a \Rightarrow 'a \Rightarrow 'a::\text{euclidean\_space}$ 

```

where $interval\_bij =$
 $(\lambda(a, b) (u, v) x. (\sum_{i \in Basis. (u \cdot i + (x \cdot i - a \cdot i) / (b \cdot i - a \cdot i) * (v \cdot i - u \cdot i))$
 $*_R i))$

lemma $interval\_bij\_affine$:

$interval\_bij (a, b) (u, v) = (\lambda x. (\sum_{i \in Basis. ((v \cdot i - u \cdot i) / (b \cdot i - a \cdot i) * (x \cdot i)) *_R$
 $i) +$
 $(\sum_{i \in Basis. (u \cdot i - (v \cdot i - u \cdot i) / (b \cdot i - a \cdot i) * (a \cdot i)) *_R i))$
by (auto simp add: $interval\_bij\_def$ sum.distrib [symmetric] scaleR_add_left
[symmetric]
fun_eq_iff intro!: sum.cong)
(simp add: algebra_simps diff_divide_distrib [symmetric])

lemma $continuous\_interval\_bij$:

fixes $a b :: 'a::euclidean\_space$
shows $continuous (at x) (interval\_bij (a, b) (u, v))$
by (auto simp add: divide_inverse $interval\_bij\_def$ intro!: continuous_sum con-
tinuous_intros)

lemma $continuous\_on\_interval\_bij$: $continuous\_on s (interval\_bij (a, b) (u, v))$

apply (rule continuous_at_imp_continuous_on)

apply (rule, rule continuous_interval_bij)

done

lemma $in\_interval\_interval\_bij$:

fixes $a b u v x :: 'a::euclidean\_space$
assumes $x \in cbox a b$
and $cbox u v \neq \{\}$
shows $interval\_bij (a, b) (u, v) x \in cbox u v$
apply (simp only: $interval\_bij\_def$ split_conv mem_box inner_sum_left_Basis
cong: ball_cong)
apply safe
proof -
fix $i :: 'a$
assume $i: i \in Basis$
have $cbox a b \neq \{\}$
using assms **by** auto
with i **have** $*: a \cdot i \leq b \cdot i$ $u \cdot i \leq v \cdot i$
using assms(2) **by** (auto simp add: box_eq_empty)
have $x: a \cdot i \leq x \cdot i$ $x \cdot i \leq b \cdot i$
using assms(1)[unfolded mem_box] **using** i **by** auto
have $0 \leq (x \cdot i - a \cdot i) / (b \cdot i - a \cdot i) * (v \cdot i - u \cdot i)$
using x **by** auto
then show $u \cdot i \leq u \cdot i + (x \cdot i - a \cdot i) / (b \cdot i - a \cdot i) * (v \cdot i - u \cdot i)$
using $*$ **by** auto
have $((x \cdot i - a \cdot i) / (b \cdot i - a \cdot i)) * (v \cdot i - u \cdot i) \leq 1 * (v \cdot i - u \cdot i)$
apply (rule mult_right_mono)
unfolding divide_le_eq_1
using x

2700

```

    apply auto
  done
  then show  $u \cdot i + (x \cdot i - a \cdot i) / (b \cdot i - a \cdot i) * (v \cdot i - u \cdot i) \leq v \cdot i$ 
    using * by auto
qed

```

```

lemma interval_bij_bij:
   $\forall (i::'a::euclidean\_space) \in \text{Basis}. a \cdot i < b \cdot i \wedge u \cdot i < v \cdot i \implies$ 
    interval_bij (a, b) (u, v) (interval_bij (u, v) (a, b) x) = x
  by (auto simp: interval_bij_def euclidean_eq_iff[where 'a='a])

```

```

lemma interval_bij_bij_cart: fixes  $x::\text{real}^n$  assumes  $\forall i. a \cdot i < b \cdot i \wedge u \cdot i < v \cdot i$ 
  shows interval_bij (a,b) (u,v) (interval_bij (u,v) (a,b) x) = x
  using assms by (intro interval_bij_bij) (auto simp: Basis_vec_def inner_axis)

```

6.32.2 Fashoda meet theorem

```

lemma infnorm_2:
  fixes  $x::\text{real}^2$ 
  shows infnorm x = max |x$1| |x$2|
  unfolding infnorm_cart UNIV_2 by (rule cSup_eq) auto

```

```

lemma infnorm_eq_1_2:
  fixes  $x::\text{real}^2$ 
  shows infnorm x = 1  $\longleftrightarrow$ 
    |x$1|  $\leq 1 \wedge$  |x$2|  $\leq 1 \wedge$  (x$1 = -1  $\vee$  x$1 = 1  $\vee$  x$2 = -1  $\vee$  x$2 = 1)
  unfolding infnorm_2 by auto

```

```

lemma infnorm_eq_1_imp:
  fixes  $x::\text{real}^2$ 
  assumes infnorm x = 1
  shows |x$1|  $\leq 1$  and |x$2|  $\leq 1$ 
  using assms unfolding infnorm_eq_1_2 by auto

```

```

proposition fashoda_unit:
  fixes  $f g::\text{real} \Rightarrow \text{real}^2$ 
  assumes  $f' \{-1 .. 1\} \subseteq \text{cbox } (-1) 1$ 
    and  $g' \{-1 .. 1\} \subseteq \text{cbox } (-1) 1$ 
    and continuous_on  $\{-1 .. 1\} f$ 
    and continuous_on  $\{-1 .. 1\} g$ 
    and  $f (-1) \$1 = -1$ 
    and  $f 1 \$1 = 1$   $g (-1) \$2 = -1$ 
    and  $g 1 \$2 = 1$ 
  shows  $\exists s \in \{-1 .. 1\}. \exists t \in \{-1 .. 1\}. f s = g t$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  note as = this[unfolded bex_simps, rule_format]
  define sqprojection

```



```

  where [abs_def]: sqprojection z = (inverse (infnorm z)) *R z for z :: real^2
define negatex :: real^2 ⇒ real^2
  where negatex x = (vector [-(x$1), x$2]) for x
have lem1: ∀ z::real^2. infnorm (negatex z) = infnorm z
  unfolding negatex_def infnorm_2 vector_2 by auto
have lem2: ∀ z. z ≠ 0 ⟶ infnorm (sqprojection z) = 1
  unfolding sqprojection_def infnorm_mul[unfolded scalar_mult_eq_scaleR]
  by (simp add: real_abs_infnorm infnorm_eq_0)
let ?F = λw::real^2. (f ∘ (λx. x$1)) w - (g ∘ (λx. x$2)) w
have *: ∧i. (λx::real^2. x $ i) ' cbox (- 1) 1 = {-1..1}
proof
  show (λx::real^2. x $ i) ' cbox (- 1) 1 ⊆ {-1..1} for i
    by (auto simp: mem_box_cart)
  show {-1..1} ⊆ (λx::real^2. x $ i) ' cbox (- 1) 1 for i
    by (clarsimp simp: image_iff mem_box_cart Bex_def) (metis (no_types,
opaque_lifting) vec_component)
qed
{
  fix x
  assume x ∈ (λw. (f ∘ (λx. x $ 1)) w - (g ∘ (λx. x $ 2)) w) ' (cbox (- 1)
(1::real^2))
  then obtain w :: real^2 where w:
    w ∈ cbox (- 1) 1
    x = (f ∘ (λx. x $ 1)) w - (g ∘ (λx. x $ 2)) w
  unfolding image_iff ..
  then have x ≠ 0
    using as[of w$1 w$2]
    unfolding mem_box_cart atLeastAtMost_iff
    by auto
} note x0 = this
have 1: box (- 1) (1::real^2) ≠ {}
  unfolding interval_eq_empty_cart by auto
have negatex (x + y) $ i = (negatex x + negatex y) $ i ∧ negatex (c *R x) $ i
= (c *R negatex x) $ i
  for i x y c
  using exhaust_2 [of i] by (auto simp: negatex_def)
then have bounded_linear negatex
  by (simp add: bounded_linearI' vec_eq_iff)
then have 2: continuous_on (cbox (- 1) 1) (negatex ∘ sqprojection ∘ ?F)
  apply (intro continuous_intros continuous_on_component)
  unfolding * sqprojection_def
  apply (intro assms continuous_intros)+
  apply (simp_all add: infnorm_eq_0 x0 linear_continuous_on)
done
have 3: (negatex ∘ sqprojection ∘ ?F) ' cbox (-1) 1 ⊆ cbox (-1) 1
  unfolding subset_eq
proof (rule, goal_cases)
  case (1 x)
  then obtain y :: real^2 where y:

```

```

      y ∈ cbox (− 1) 1
      x = (negatex ∘ sqprojection ∘ (λw. (f ∘ (λx. x $ 1)) w − (g ∘ (λx. x $ 2))
w)) y
    unfolding image_iff ..
  have ?F y ≠ 0
  by (rule x0) (use y in auto)
  then have *: infnorm (sqprojection (?F y)) = 1
  unfolding y o_def
  by (rule lem2[rule_format])
  have inf1: infnorm x = 1
  unfolding *[symmetric] y o_def
  by (rule lem1[rule_format])
  show x ∈ cbox (−1) 1
  unfolding mem_box_cart interval_box_cart infnorm_2
  proof
    fix i
    show (− 1) $ i ≤ x $ i ∧ x $ i ≤ 1 $ i
    using exhaust_2 [of i] inf1 by (auto simp: infnorm_2)
  qed
  qed
  obtain x :: real^2 where x:
    x ∈ cbox (− 1) 1
    (negatex ∘ sqprojection ∘ (λw. (f ∘ (λx. x $ 1)) w − (g ∘ (λx. x $ 2)) w)) x
= x
  apply (rule brouwer_weak[of cbox (− 1) (1::real^2) negatex ∘ sqprojection ∘
?F])
  apply (rule compact_cbox convex_box)+
  unfolding interior_cbox
  apply (rule 1 2 3)+
  apply blast
  done
  have ?F x ≠ 0
  by (rule x0) (use x in auto)
  then have *: infnorm (sqprojection (?F x)) = 1
  unfolding o_def
  by (rule lem2[rule_format])
  have nx: infnorm x = 1
  apply (subst x(2)[symmetric])
  unfolding *[symmetric] o_def
  apply (rule lem1[rule_format])
  done
  have iff: 0 < sqprojection x$i ⟷ 0 < x$i sqprojection x$i < 0 ⟷ x$i < 0
  if x ≠ 0 for x i
  proof −
    have inverse (infnorm x) > 0
    by (simp add: infnorm_pos_lt that)
    then show (0 < sqprojection x $ i) = (0 < x $ i)
    and (sqprojection x $ i < 0) = (x $ i < 0)
    unfolding sqprojection_def vector_component_simps vector_scaleR_component

```

```

real_scaleR_def
  unfolding zero_less_mult_iff mult_less_0_iff
  by (auto simp add: field_simps)
qed
have x1:  $x \ \$ \ 1 \in \{-1..1::real\}$   $x \ \$ \ 2 \in \{-1..1::real\}$ 
  using x(1) unfolding mem_box_cart by auto
then have nz:  $f(x \ \$ \ 1) - g(x \ \$ \ 2) \neq 0$ 
  using as by auto
consider  $x \ \$ \ 1 = -1 \mid x \ \$ \ 1 = 1 \mid x \ \$ \ 2 = -1 \mid x \ \$ \ 2 = 1$ 
  using nx unfolding infnorm_eq_1_2 by auto
then show False
proof cases
case 1
  then have *:  $f(x \ \$ \ 1) \ \$ \ 1 = -1$ 
    using assms(5) by auto
  have sqprojection  $(f(x \ \$ \ 1) - g(x \ \$ \ 2)) \ \$ \ 1 > 0$ 
    using x(2)[unfolded o_def vec_eq_iff, THEN spec[where x=1]]
    by (auto simp: negatex_def 1)
  moreover
  from x1 have  $g(x \ \$ \ 2) \in \text{cbox}(-1) \ 1$ 
    using assms(2) by blast
  ultimately show False
    unfolding iff[OF nz] vector_component_simps * mem_box_cart
    using not_le by auto
next
case 2
  then have *:  $f(x \ \$ \ 1) \ \$ \ 1 = 1$ 
    using assms(6) by auto
  have sqprojection  $(f(x \ \$ \ 1) - g(x \ \$ \ 2)) \ \$ \ 1 < 0$ 
    using x(2)[unfolded o_def vec_eq_iff, THEN spec[where x=1]] 2
    by (auto simp: negatex_def)
  moreover have  $g(x \ \$ \ 2) \in \text{cbox}(-1) \ 1$ 
    using assms(2) x1 by blast
  ultimately show False
    unfolding iff[OF nz] vector_component_simps * mem_box_cart
    using not_le by auto
next
case 3
  then have *:  $g(x \ \$ \ 2) \ \$ \ 2 = -1$ 
    using assms(7) by auto
  have sqprojection  $(f(x \ \$ \ 1) - g(x \ \$ \ 2)) \ \$ \ 2 < 0$ 
    using x(2)[unfolded o_def vec_eq_iff, THEN spec[where x=2]] 3 by (auto
simp: negatex_def)
  moreover
  from x1 have  $f(x \ \$ \ 1) \in \text{cbox}(-1) \ 1$ 
    using assms(1) by blast
  ultimately show False
    unfolding iff[OF nz] vector_component_simps * mem_box_cart
    by (erule_tac x=2 in allE) auto

```

```

next
case 4
then have *:  $g(x\ \$\ 2)\ \$\ 2 = 1$ 
using assms(8) by auto
have sqprojection  $(f(x\$1) - g(x\$2))\ \$\ 2 > 0$ 
using  $x(2)[unfolded\ o\_def\ vec\_eq\_iff, THEN\ spec[where\ x=2]]\ 4$  by (auto
simp: negatex_def)
moreover
from x1 have  $f(x\ \$\ 1) \in cbox\ (-1)\ 1$ 
using assms(1) by blast
ultimately show False
unfolding iff[OF nz] vector_component_simps * mem_box_cart
by (erule_tac  $x=2$  in allE) auto
qed
qed

```

proposition fashoda_unit_path:

```

fixes f g :: real  $\Rightarrow$  real2
assumes path f
and path g
and path_image  $f \subseteq cbox\ (-1)\ 1$ 
and path_image  $g \subseteq cbox\ (-1)\ 1$ 
and (pathstart f)$1 = -1
and (pathfinish f)$1 = 1
and (pathstart g)$2 = -1
and (pathfinish g)$2 = 1
obtains z where  $z \in path\_image\ f$  and  $z \in path\_image\ g$ 
proof -
note assms=assms[unfolded path_def pathstart_def pathfinish_def path_image_def]
define iscale where [abs_def]:  $iscale\ z = inverse\ 2 *_{\mathbb{R}} (z + 1)$  for  $z :: real$ 
have isc:  $iscale\ ' \{-1..1\} \subseteq \{0..1\}$ 
unfolding iscale_def by auto
have  $\exists s \in \{-1..1\}. \exists t \in \{-1..1\}. (f \circ iscale)\ s = (g \circ iscale)\ t$ 
proof (rule fashoda_unit)
show  $(f \circ iscale)\ ' \{-1..1\} \subseteq cbox\ (-1)\ 1$   $(g \circ iscale)\ ' \{-1..1\} \subseteq cbox\ (-1)\ 1$ 
using isc and assms(3-4) by (auto simp add: image_comp [symmetric])
have *: continuous_on  $\{-1..1\}$  iscale
unfolding iscale_def by (rule continuous_intros)+
show continuous_on  $\{-1..1\}$   $(f \circ iscale)$  continuous_on  $\{-1..1\}$   $(g \circ iscale)$ 
apply -
apply (rule_tac [!] continuous_on_compose[OF *])
apply (rule_tac [!] continuous_on_subset[OF _ isc])
apply (rule assms)+
done
have *:  $(1 / 2) *_{\mathbb{R}} (1 + (1::real^1)) = 1$ 
unfolding vec_eq_iff by auto
show  $(f \circ iscale)\ (-1)\ \$\ 1 = -1$ 
and  $(f \circ iscale)\ 1\ \$\ 1 = 1$ 

```

```

    and (g ∘ iscale) (- 1) $ 2 = -1
    and (g ∘ iscale) 1 $ 2 = 1
    unfolding o_def iscale_def
    using assms
    by (auto simp add: *)
qed
then obtain s t where st:
  s ∈ {- 1..1}
  t ∈ {- 1..1}
  (f ∘ iscale) s = (g ∘ iscale) t
  by auto
show thesis
  apply (rule_tac z = f (iscale s) in that)
  using st
  unfolding o_def path_image_def image_iff
  apply -
  apply (rule_tac x=iscale s in bexI)
  prefer 3
  apply (rule_tac x=iscale t in bexI)
  using isc[unfolded subset_eq, rule_format]
  apply auto
  done
qed

theorem fashoda:
  fixes b :: real^2
  assumes path f
    and path g
    and path_image f ⊆ cbox a b
    and path_image g ⊆ cbox a b
    and (pathstart f)$1 = a$1
    and (pathfinish f)$1 = b$1
    and (pathstart g)$2 = a$2
    and (pathfinish g)$2 = b$2
  obtains z where z ∈ path_image f and z ∈ path_image g
proof -
  fix P Q S
  presume P ∨ Q ∨ S
  P ⇒ thesis and Q ⇒ thesis and S ⇒ thesis
  then show thesis
    by auto
next
  have cbox a b ≠ {}
    using assms(3) using path_image_nonempty[of f] by auto
  then have a ≤ b
    unfolding interval_eq_empty_cart less_eq_vec_def by (auto simp add: not_less)
  then show a$1 = b$1 ∨ a$2 = b$2 ∨ (a$1 < b$1 ∧ a$2 < b$2)
    unfolding less_eq_vec_def forall_2 by auto
next
  assume as: a$1 = b$1

```

```

have  $\exists z \in \text{path\_image } g. z \$ 2 = (\text{pathstart } f) \$ 2$ 
  apply (rule connected_ivt_component_cart)
  apply (rule connected_path_image assms)+
  apply (rule pathstart_in_path_image)
  apply (rule pathfinish_in_path_image)
  unfolding assms using assms(3)[unfolded path_image_def subset_eq, rule_format, of
f 0]
  unfolding pathstart_def
  apply (auto simp add: less_eq_vec_def mem_box_cart)
  done
then obtain  $z :: \text{real}^2$  where  $z: z \in \text{path\_image } g \ z \$ 2 = \text{pathstart } f \$ 2 ..$ 
have  $z \in \text{cbox } a \ b$ 
  using z(1) assms(4)
  unfolding path_image_def
  by blast
then have  $z = f \ 0$ 
  unfolding vec_eq_iff forall_2
  unfolding z(2) pathstart_def
  using assms(3)[unfolded path_image_def subset_eq mem_box_cart, rule_format, of
f 0 1]
  unfolding mem_box_cart
  apply (erule_tac  $x=1$  in allE)
  using as
  apply auto
  done
then show thesis
  apply -
  apply (rule that[OF _ z(1)])
  unfolding path_image_def
  apply auto
  done
next
assume as:  $a \$ 2 = b \$ 2$ 
have  $\exists z \in \text{path\_image } f. z \$ 1 = (\text{pathstart } g) \$ 1$ 
  apply (rule connected_ivt_component_cart)
  apply (rule connected_path_image assms)+
  apply (rule pathstart_in_path_image)
  apply (rule pathfinish_in_path_image)
  unfolding assms
  using assms(4)[unfolded path_image_def subset_eq, rule_format, of g 0]
  unfolding pathstart_def
  apply (auto simp add: less_eq_vec_def mem_box_cart)
  done
then obtain  $z$  where  $z: z \in \text{path\_image } f \ z \$ 1 = \text{pathstart } g \$ 1 ..$ 
have  $z \in \text{cbox } a \ b$ 
  using z(1) assms(3)
  unfolding path_image_def
  by blast
then have  $z = g \ 0$ 

```

```

    unfolding vec_eq_iff forall_2
    unfolding z(2) pathstart_def
    using assms(4)[unfolded path_image_def subset_eq mem_box_cart, rule_format, of
g 0 2]
    unfolding mem_box_cart
    apply (erule_tac x=2 in allE)
    using as
    apply auto
    done
then show thesis
  apply -
  apply (rule that[OF z(1)])
  unfolding path_image_def
  apply auto
  done
next
  assume as: a $ 1 < b $ 1 ∧ a $ 2 < b $ 2
  have int_nem: cbox (-1) (1::real^2) ≠ {}
    unfolding interval_eq_empty_cart by auto
  obtain z :: real^2 where z:
    z ∈ (interval_bij (a, b) (- 1, 1) ∘ f) ‘ {0..1}
    z ∈ (interval_bij (a, b) (- 1, 1) ∘ g) ‘ {0..1}
  apply (rule fashoda_unit_path[of interval_bij (a,b) (- 1,1) ∘ f interval_bij
(a,b) (- 1,1) ∘ g])
  unfolding path_def path_image_def pathstart_def pathfinish_def
  apply (rule_tac[1-2] continuous_on_compose)
  apply (rule assms[unfolded path_def] continuous_on_interval_bij)+
  unfolding subset_eq
  apply (rule_tac[1-2] ballI)
proof -
  fix x
  assume x ∈ (interval_bij (a, b) (- 1, 1) ∘ f) ‘ {0..1}
  then obtain y where y:
    y ∈ {0..1}
    x = (interval_bij (a, b) (- 1, 1) ∘ f) y
  unfolding image_iff ..
  show x ∈ cbox (- 1) 1
    unfolding y_o_def
    apply (rule in_interval_interval_bij)
    using y(1)
    using assms(3)[unfolded path_image_def subset_eq] int_nem
    apply auto
    done
next
  fix x
  assume x ∈ (interval_bij (a, b) (- 1, 1) ∘ g) ‘ {0..1}
  then obtain y where y:
    y ∈ {0..1}
    x = (interval_bij (a, b) (- 1, 1) ∘ g) y

```

```

    unfolding image_iff ..
  show  $x \in \text{cbox } (-1) 1$ 
    unfolding y_o_def
    apply (rule in_interval_interval_bij)
    using y(1)
    using assms(4)[unfolded path_image_def subset_eq] int_nem
    apply auto
    done
  next
  show (interval_bij (a, b) (-1, 1)  $\circ$  f) 0 1 = -1
    and (interval_bij (a, b) (-1, 1)  $\circ$  f) 1 1 = 1
    and (interval_bij (a, b) (-1, 1)  $\circ$  g) 0 2 = -1
    and (interval_bij (a, b) (-1, 1)  $\circ$  g) 1 2 = 1
    using assms as
    by (simp_all add: cart_eq_inner_axis pathstart_def pathfinish_def interval_bij_def)
      (simp_all add: inner_axis)
  qed
  from z(1) obtain zf where zf:
    zf  $\in$  {0..1}
    z = (interval_bij (a, b) (-1, 1)  $\circ$  f) zf
    unfolding image_iff ..
  from z(2) obtain zg where zg:
    zg  $\in$  {0..1}
    z = (interval_bij (a, b) (-1, 1)  $\circ$  g) zg
    unfolding image_iff ..
  have *:  $\forall i. (-1) \$ i < (1::\text{real}^2) \$ i \wedge a \$ i < b \$ i$ 
    unfolding forall_2
    using as
    by auto
  show thesis
  proof (rule_tac z=interval_bij (-1,1) (a,b) z in that)
    show interval_bij (-1, 1) (a, b) z  $\in$  path_image f
      using zf by (simp add: interval_bij_bij_cart[OF *] path_image_def)
    show interval_bij (-1, 1) (a, b) z  $\in$  path_image g
      using zg by (simp add: interval_bij_bij_cart[OF *] path_image_def)
  qed
qed

```

6.32.3 Some slightly ad hoc lemmas I use below

lemma segment_vertical:

```

  fixes a :: real^2
  assumes a$1 = b$1
  shows  $x \in \text{closed\_segment } a b \iff$ 
     $x$1 = a$1 \wedge x$1 = b$1 \wedge (a$2 \leq x$2 \wedge x$2 \leq b$2 \vee b$2 \leq x$2 \wedge x$2 \leq a$2)$ 
    (is _ = ?R)
  proof -

```



```

let ?L =  $\exists u. (x \$ 1 = (1 - u) * a \$ 1 + u * b \$ 1 \wedge x \$ 2 = (1 - u) * a \$ 2 + u * b \$ 2) \wedge 0 \leq u \wedge u \leq 1$ 
{
  presume ?L  $\implies$  ?R and ?R  $\implies$  ?L
  then show ?thesis
    unfolding closed_segment_def mem_Collect_eq
    unfolding vec_eq_iff forall_2 scalar_mult_eq_scaleR[symmetric] vector_component_simps
    by blast
}
{
  assume ?L
  then obtain u where u:
    x $ 1 = (1 - u) * a $ 1 + u * b $ 1
    x $ 2 = (1 - u) * a $ 2 + u * b $ 2
    0  $\leq$  u
    u  $\leq$  1
  by blast
  { fix b a
    assume b + u * a > a + u * b
    then have (1 - u) * b > (1 - u) * a
      by (auto simp add: field_simps)
    then have b  $\geq$  a
      apply (drule_tac mult_left_less_imp_less)
      using u
      apply auto
      done
    then have u * a  $\leq$  u * b
      apply -
      apply (rule mult_left_mono[OF _ u(3)])
      using u(3-4)
      apply (auto simp add: field_simps)
      done
    } note * = this
  {
    fix a b
    assume u * b > u * a
    then have (1 - u) * a  $\leq$  (1 - u) * b
      apply -
      apply (rule mult_left_mono)
      apply (drule mult_left_less_imp_less)
      using u
      apply auto
      done
    then have a + u * b  $\leq$  b + u * a
      by (auto simp add: field_simps)
    } note ** = this
  }
  then show ?R
    unfolding u assms
    using u

```

```

    by (auto simp add: field_simps not_le intro: * **)
  }
  {
    assume ?R
    then show ?L
    proof (cases x$2 = b$2)
      case True
      then show ?L
        apply (rule_tac x=(x$2 - a$2) / (b$2 - a$2) in exI)
        unfolding assms True using ‹?R› apply (auto simp add: field_simps)
        done
    next
      case False
      then show ?L
        apply (rule_tac x=1 - (x$2 - b$2) / (a$2 - b$2) in exI)
        unfolding assms using ‹?R› apply (auto simp add: field_simps)
        done
    qed
  }
qed

lemma segment_horizontal:
  fixes a :: real^2
  assumes a$2 = b$2
  shows x ∈ closed_segment a b ⟷
    x$2 = a$2 ∧ x$2 = b$2 ∧ (a$1 ≤ x$1 ∧ x$1 ≤ b$1 ∨ b$1 ≤ x$1 ∧ x$1 ≤
a$1)
  (is _ = ?R)
proof -
  let ?L = ∃ u. (x $ 1 = (1 - u) * a $ 1 + u * b $ 1 ∧ x $ 2 = (1 - u) * a $ 2
+ u * b $ 2) ∧ 0 ≤ u ∧ u ≤ 1
  {
    presume ?L ⟹ ?R and ?R ⟹ ?L
    then show ?thesis
      unfolding closed_segment_def mem_Collect_eq
      unfolding vec_eq_iff forall_2 scalar_mult_eq_scaleR[symmetric] vector_component_simps
      by blast
  }
  {
    assume ?L
    then obtain u where u:
      x $ 1 = (1 - u) * a $ 1 + u * b $ 1
      x $ 2 = (1 - u) * a $ 2 + u * b $ 2
      0 ≤ u
      u ≤ 1
    by blast
    {
      fix b a
      assume b + u * a > a + u * b
    }
  }

```

```

    then have  $(1 - u) * b > (1 - u) * a$ 
      by (auto simp add: field_simps)
    then have  $b \geq a$ 
      apply (drule_tac mult_left_less_imp_less)
      using u
      apply auto
      done
    then have  $u * a \leq u * b$ 
      apply -
      apply (rule mult_left_mono[OF u(3)])
      using u(3-4)
      apply (auto simp add: field_simps)
      done
  } note * = this
{
  fix a b
  assume  $u * b > u * a$ 
  then have  $(1 - u) * a \leq (1 - u) * b$ 
    apply -
    apply (rule mult_left_mono)
    apply (drule mult_left_less_imp_less)
    using u
    apply auto
    done
  then have  $a + u * b \leq b + u * a$ 
    by (auto simp add: field_simps)
} note ** = this
then show ?R
  unfolding u assms
  using u
  by (auto simp add: field_simps not_le intro: * **)
}
{
  assume ?R
  then show ?L
  proof (cases  $x\$1 = b\$1$ )
  case True
  then show ?L
    apply (rule_tac  $x=(x\$1 - a\$1) / (b\$1 - a\$1)$  in exI)
    unfolding assms True
    using ‹?R›
    apply (auto simp add: field_simps)
    done
  next
  case False
  then show ?L
    apply (rule_tac  $x=1 - (x\$1 - b\$1) / (a\$1 - b\$1)$  in exI)
    unfolding assms
    using ‹?R›

```

```

      apply (auto simp add: field_simps)
    done
  qed
}
qed

```

6.32.4 Useful Fashoda corollary pointed out to me by Tom Hales

corollary *fashoda_interlace*:

```

  fixes a :: real^2
  assumes path f
    and path g
    and paf: path_image f ⊆ cbox a b
    and pag: path_image g ⊆ cbox a b
    and (pathstart f)$2 = a$2
    and (pathfinish f)$2 = a$2
    and (pathstart g)$2 = a$2
    and (pathfinish g)$2 = a$2
    and (pathstart f)$1 < (pathstart g)$1
    and (pathstart g)$1 < (pathfinish f)$1
    and (pathfinish f)$1 < (pathfinish g)$1
  obtains z where z ∈ path_image f and z ∈ path_image g
proof -
  have cbox a b ≠ {}
    using path_image_nonempty[of f] using assms(3) by auto
  note ab=this[unfolded interval_eq_empty_cart not_ex forall_2 not_less]
  have pathstart f ∈ cbox a b
    and pathfinish f ∈ cbox a b
    and pathstart g ∈ cbox a b
    and pathfinish g ∈ cbox a b
    using pathstart_in_path_image pathfinish_in_path_image
    using assms(3-4)
    by auto
  note startfin = this[unfolded mem_box_cart forall_2]
  let ?P1 = linepath (vector[a$1 - 2, a$2 - 2]) (vector[(pathstart f)$1, a$2 - 2])
  +++
  linepath(vector[(pathstart f)$1, a$2 - 2])(pathstart f) +++ f +++
  linepath(pathfinish f)(vector[(pathfinish f)$1, a$2 - 2]) +++
  linepath(vector[(pathfinish f)$1, a$2 - 2])(vector[b$1 + 2, a$2 - 2])
  let ?P2 = linepath(vector[(pathstart g)$1, (pathstart g)$2 - 3])(pathstart g)
  +++ g +++
  linepath(pathfinish g)(vector[(pathfinish g)$1, a$2 - 1]) +++
  linepath(vector[(pathfinish g)$1, a$2 - 1])(vector[b$1 + 1, a$2 - 1]) +++
  linepath(vector[b$1 + 1, a$2 - 1])(vector[b$1 + 1, b$2 + 3])
  let ?a = vector[a$1 - 2, a$2 - 3]
  let ?b = vector[b$1 + 2, b$2 + 3]
  have P1P2: path_image ?P1 = path_image (linepath (vector[a$1 - 2, a$2 - 2])
  (vector[(pathstart f)$1, a$2 - 2])) ∪

```

```

    path_image (linepath(vector[(pathstart f)$1, a$2 - 2])(pathstart f))  $\cup$  path_image
f  $\cup$ 
    path_image (linepath(pathfinish f)(vector[(pathfinish f)$1, a$2 - 2]))  $\cup$ 
    path_image (linepath(vector[(pathfinish f)$1, a$2 - 2])(vector[b$1 + 2, a$2
- 2]))
    path_image ?P2 = path_image(linepath(vector[(pathstart g)$1, (pathstart g)$2
- 3])(pathstart g))  $\cup$  path_image g  $\cup$ 
    path_image(linepath(pathfinish g)(vector[(pathfinish g)$1, a$2 - 1]))  $\cup$ 
    path_image(linepath(vector[(pathfinish g)$1, a$2 - 1])(vector[b$1 + 1, a$2 -
1]))  $\cup$ 
    path_image(linepath(vector[b$1 + 1, a$2 - 1])(vector[b$1 + 1, b$2 + 3]))
using assms(1-2)
  by(auto simp add: path_image_join)
  have abab: cbox a b  $\subseteq$  cbox ?a ?b
    unfolding interval_cbox_cart[symmetric]
    by (auto simp add: less_eq_vec_def forall_2)
  obtain z where
    z  $\in$  path_image
      (linepath (vector [a $ 1 - 2, a $ 2 - 2]) (vector [pathstart f $ 1, a $ 2
- 2]) +++
        linepath (vector [pathstart f $ 1, a $ 2 - 2]) (pathstart f) +++
        f +++
        linepath (pathfinish f) (vector [pathfinish f $ 1, a $ 2 - 2]) +++
        linepath (vector [pathfinish f $ 1, a $ 2 - 2]) (vector [b $ 1 + 2, a $ 2
- 2]))
    z  $\in$  path_image
      (linepath (vector [pathstart g $ 1, pathstart g $ 2 - 3]) (pathstart g) +++
        g +++
        linepath (pathfinish g) (vector [pathfinish g $ 1, a $ 2 - 1]) +++
        linepath (vector [pathfinish g $ 1, a $ 2 - 1]) (vector [b $ 1 + 1, a $ 2
- 1]) +++
        linepath (vector [b $ 1 + 1, a $ 2 - 1]) (vector [b $ 1 + 1, b $ 2 + 3]))
    apply (rule fashoda[of ?P1 ?P2 ?a ?b])
  unfolding pathstart_join pathfinish_join pathstart_linepath pathfinish_linepath
vector_2
  proof -
    show path ?P1 and path ?P2
      using assms by auto
    show path_image ?P1  $\subseteq$  cbox ?a ?b path_image ?P2  $\subseteq$  cbox ?a ?b
      unfolding P1P2 path_image_linepath using startfin paf pag
      by (auto simp: mem_box_cart segment_horizontal segment_vertical forall_2)
    show a $ 1 - 2 = a $ 1 - 2
      and b $ 1 + 2 = b $ 1 + 2
      and pathstart g $ 2 - 3 = a $ 2 - 3
      and b $ 2 + 3 = b $ 2 + 3
      by (auto simp add: assms)
  qed
  note z=this[unfolded P1P2 path_image_linepath]
  show thesis

```

```

proof (rule that[of z])
  have (z ∈ closed_segment (vector [a $ 1 - 2, a $ 2 - 2]) (vector [pathstart f
$ 1, a $ 2 - 2])) ∨
    z ∈ closed_segment (vector [pathstart f $ 1, a $ 2 - 2]) (pathstart f)) ∨
    z ∈ closed_segment (pathfinish f) (vector [pathfinish f $ 1, a $ 2 - 2]) ∨
    z ∈ closed_segment (vector [pathfinish f $ 1, a $ 2 - 2]) (vector [b $ 1 + 2,
a $ 2 - 2])) ⇒
    (((z ∈ closed_segment (vector [pathstart g $ 1, pathstart g $ 2 - 3]) (pathstart
g)) ∨
      z ∈ closed_segment (pathfinish g) (vector [pathfinish g $ 1, a $ 2 - 1])) ∨
      z ∈ closed_segment (vector [pathfinish g $ 1, a $ 2 - 1]) (vector [b $ 1 + 1,
a $ 2 - 1])) ∨
      z ∈ closed_segment (vector [b $ 1 + 1, a $ 2 - 1]) (vector [b $ 1 + 1, b $
2 + 3])) ⇒ False
proof (simp only: segment_vertical segment_horizontal vector_2, goal_cases)
case prems: 1
have pathfinish f ∈ cbox a b
  using assms(3) pathfinish_in_path_image[of f] by auto
then have 1 + b $ 1 ≤ pathfinish f $ 1 ⇒ False
  unfolding mem_box_cart forall_2 by auto
then have z$1 ≠ pathfinish f$1
  using prems(2)
  using assms ab
  by (auto simp add: field_simps)
moreover have pathstart f ∈ cbox a b
  using assms(3) pathstart_in_path_image[of f]
  by auto
then have 1 + b $ 1 ≤ pathstart f $ 1 ⇒ False
  unfolding mem_box_cart forall_2
  by auto
then have z$1 ≠ pathstart f$1
  using prems(2) using assms ab
  by (auto simp add: field_simps)
ultimately have *: z$2 = a$2 - 2
  using prems(1) by auto
have z$1 ≠ pathfinish g$1
  using prems(2) assms ab
  by (auto simp add: field_simps *)
moreover have pathstart g ∈ cbox a b
  using assms(4) pathstart_in_path_image[of g]
  by auto
note this[unfolded mem_box_cart forall_2]
then have z$1 ≠ pathstart g$1
  using prems(1) assms ab
  by (auto simp add: field_simps *)
ultimately have a $ 2 - 1 ≤ z $ 2 ∧ z $ 2 ≤ b $ 2 + 3 ∨ b $ 2 + 3 ≤ z
$ 2 ∧ z $ 2 ≤ a $ 2 - 1
  using prems(2) unfolding * assms by (auto simp add: field_simps)
then show False

```

```

      unfolding * using ab by auto
    qed
  then have  $z \in \text{path\_image } f \vee z \in \text{path\_image } g$ 
    using z unfolding Un_iff by blast
  then have  $z': z \in \text{cbox } a \ b$ 
    using assms(3-4) by auto
  have  $a \$ 2 = z \$ 2 \implies (z \$ 1 = \text{pathstart } f \$ 1 \vee z \$ 1 = \text{pathfinish } f \$ 1)$ 
 $\implies$ 
     $z = \text{pathstart } f \vee z = \text{pathfinish } f$ 
    unfolding vec_eq_iff forall_2 assms
    by auto
  with z' show  $z \in \text{path\_image } f$ 
    using z(1)
    unfolding Un_iff mem_box_cart forall_2
    by (simp only: segment_vertical segment_horizontal vector_2) (auto simp:
    assms)
  have  $a \$ 2 = z \$ 2 \implies (z \$ 1 = \text{pathstart } g \$ 1 \vee z \$ 1 = \text{pathfinish } g \$ 1)$ 
 $\implies$ 
     $z = \text{pathstart } g \vee z = \text{pathfinish } g$ 
    unfolding vec_eq_iff forall_2 assms
    by auto
  with z' show  $z \in \text{path\_image } g$ 
    using z(2)
    unfolding Un_iff mem_box_cart forall_2
    by (simp only: segment_vertical segment_horizontal vector_2) (auto simp:
    assms)
  qed
qed
end

```

6.33 Vector Cross Products in 3 Dimensions

```

theory Cross3
  imports Determinants Cartesian_Euclidean_Space
begin

context includes no_Set_Product_syntax
begin — locally disable syntax for set product, to avoid warnings

definition cross3 ::  $[\text{real}^3, \text{real}^3] \Rightarrow \text{real}^3$  (infixr  $\times$  80)
  where  $a \times b \equiv$ 
    vector  $[a\$2 * b\$3 - a\$3 * b\$2,$ 
       $a\$3 * b\$1 - a\$1 * b\$3,$ 
       $a\$1 * b\$2 - a\$2 * b\$1]$ 

end

bundle cross3_syntax begin

```

```

notation cross3 (infixr × 80)
no_notation Product_Type.Times (infixr × 80)
end

```

```

bundle no_cross3_syntax begin
no_notation cross3 (infixr × 80)
notation Product_Type.Times (infixr × 80)
end

```

```

unbundle cross3_syntax

```

6.33.1 Basic lemmas

lemmas *cross3_simps* = *cross3_def inner_vec_def sum_3 det_3 vec_eq_iff vector_def algebra_simps*

lemma *dot_cross_self*: $x \cdot (x \times y) = 0$ $x \cdot (y \times x) = 0$ $(x \times y) \cdot y = 0$ $(y \times x) \cdot y = 0$
by (*simp_all add: orthogonal_def cross3_simps*)

lemma *orthogonal_cross*: *orthogonal* ($x \times y$) *x* *orthogonal* ($x \times y$) *y*
 $\text{orthogonal } y (x \times y)$ *orthogonal* ($x \times y$) *x*
by (*simp_all add: orthogonal_def dot_cross_self*)

lemma *cross_zero_left* [*simp*]: $0 \times x = 0$ **and** *cross_zero_right* [*simp*]: $x \times 0 = 0$ **for** $x::\text{real}^3$
by (*simp_all add: cross3_simps*)

lemma *cross_skew*: $(x \times y) = -(y \times x)$ **for** $x::\text{real}^3$
by (*simp add: cross3_simps*)

lemma *cross_refl* [*simp*]: $x \times x = 0$ **for** $x::\text{real}^3$
by (*simp add: cross3_simps*)

lemma *cross_add_left*: $(x + y) \times z = (x \times z) + (y \times z)$ **for** $x::\text{real}^3$
by (*simp add: cross3_simps*)

lemma *cross_add_right*: $x \times (y + z) = (x \times y) + (x \times z)$ **for** $x::\text{real}^3$
by (*simp add: cross3_simps*)

lemma *cross_mult_left*: $(c *_R x) \times y = c *_R (x \times y)$ **for** $x::\text{real}^3$
by (*simp add: cross3_simps*)

lemma *cross_mult_right*: $x \times (c *_R y) = c *_R (x \times y)$ **for** $x::\text{real}^3$
by (*simp add: cross3_simps*)

lemma *cross_minus_left* [*simp*]: $(-x) \times y = -(x \times y)$ **for** $x::\text{real}^3$
by (*simp add: cross3_simps*)

lemma *cross_minus_right* [simp]: $x \times -y = -(x \times y)$ **for** $x::\text{real}^3$
by (simp add: cross3_simps)

lemma *left_diff_distrib*: $(x - y) \times z = x \times z - y \times z$ **for** $x::\text{real}^3$
by (simp add: cross3_simps)

lemma *right_diff_distrib*: $x \times (y - z) = x \times y - x \times z$ **for** $x::\text{real}^3$
by (simp add: cross3_simps)

hide_fact (open) *left_diff_distrib right_diff_distrib*

proposition *Jacobi*: $x \times (y \times z) + y \times (z \times x) + z \times (x \times y) = 0$ **for** $x::\text{real}^3$
by (simp add: cross3_simps)

proposition *Lagrange*: $x \times (y \times z) = (x \cdot z) *_R y - (x \cdot y) *_R z$
by (simp add: cross3_simps) (metis (full_types) exhaust_3)

proposition *cross_triple*: $(x \times y) \cdot z = (y \times z) \cdot x$
by (simp add: cross3_def inner_vec_def sum_3 vec_eq_iff algebra_simps)

lemma *cross_components*:
 $(x \times y)_1 = x_2 * y_3 - y_2 * x_3$ $(x \times y)_2 = x_3 * y_1 - y_3 * x_1$ $(x \times y)_3 = x_1 * y_2 - y_1 * x_2$
by (simp_all add: cross3_def inner_vec_def sum_3 vec_eq_iff algebra_simps)

lemma *cross_basis*: $(\text{axis } 1 \ 1) \times (\text{axis } 2 \ 1) = \text{axis } 3 \ 1$ $(\text{axis } 2 \ 1) \times (\text{axis } 1 \ 1) = -(\text{axis } 3 \ 1)$
 $(\text{axis } 2 \ 1) \times (\text{axis } 3 \ 1) = \text{axis } 1 \ 1$ $(\text{axis } 3 \ 1) \times (\text{axis } 2 \ 1) = -(\text{axis } 1 \ 1)$
 $(\text{axis } 3 \ 1) \times (\text{axis } 1 \ 1) = \text{axis } 2 \ 1$ $(\text{axis } 1 \ 1) \times (\text{axis } 3 \ 1) = -(\text{axis } 2 \ 1)$
using *exhaust_3*
by (force simp add: axis_def cross3_simps)+

lemma *cross_basis_nonzero*:
 $u \neq 0 \implies u \times \text{axis } 1 \ 1 \neq 0 \vee u \times \text{axis } 2 \ 1 \neq 0 \vee u \times \text{axis } 3 \ 1 \neq 0$
by (clarsimp simp add: axis_def cross3_simps) (metis exhaust_3)

lemma *cross_dot_cancel*:
fixes $x::\text{real}^3$
assumes *deq*: $x \cdot y = x \cdot z$ **and** *veq*: $x \times y = x \times z$ **and** $x: x \neq 0$
shows $y = z$
proof –
have $x \cdot x \neq 0$
by (simp add: x)
then have $y - z = 0$
using *veq*
by (metis (no_types, lifting) Cross3.right_diff_distrib Lagrange deq eq_iff_diff_eq_0 inner_diff_right scale_eq_0_iff)

2718

```

    then show ?thesis
      using eq_iff_diff_eq_0 by blast
qed

```

```

lemma norm_cross_dot: (norm (x × y))2 + (x · y)2 = (norm x * norm y)2
  unfolding power2_norm_eq_inner power_mult_distrib
  by (simp add: cross3_simps power2_eq_square)

```

```

lemma dot_cross_det: x · (y × z) = det(vector[x,y,z])
  by (simp add: cross3_simps)

```

```

lemma cross_cross_det: (w × x) × (y × z) = det(vector[w,x,z]) *R y - det(vector[w,x,y])
  *R z
  using exhaust_3 by (force simp add: cross3_simps)

```

```

proposition dot_cross: (w × x) · (y × z) = (w · y) * (x · z) - (w · z) * (x · y)
  by (force simp add: cross3_simps)

```

```

proposition norm_cross: (norm (x × y))2 = (norm x)2 * (norm y)2 - (x · y)2
  unfolding power2_norm_eq_inner power_mult_distrib
  by (simp add: cross3_simps power2_eq_square)

```

```

lemma cross_eq_0: x × y = 0 ⟷ collinear{0,x,y}
proof -
  have x × y = 0 ⟷ norm (x × y) = 0
    by simp
  also have ... ⟷ (norm x * norm y)2 = (x · y)2
    using norm_cross [of x y] by (auto simp: power_mult_distrib)
  also have ... ⟷ |x · y| = norm x * norm y
    using power2_eq_iff
    by (metis (mono_tags, opaque_lifting) abs_minus abs_norm_cancel abs_power2
  norm_mult power_abs real_norm_def)
  also have ... ⟷ collinear {0, x, y}
    by (rule norm_cauchy_schwarz_equal)
  finally show ?thesis .
qed

```

```

lemma cross_eq_self: x × y = x ⟷ x = 0 x × y = y ⟷ y = 0
  apply (metis cross_zero_left dot_cross_self(1) inner_eq_zero_iff)
  by (metis cross_zero_right dot_cross_self(2) inner_eq_zero_iff)

```

```

lemma norm_and_cross_eq_0:
  x · y = 0 ∧ x × y = 0 ⟷ x = 0 ∨ y = 0 (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (metis cross_dot_cancel cross_zero_right inner_zero_right)
qed auto

```

```

lemma bilinear_cross: bilinear( $\times$ )
  apply (auto simp add: bilinear_def linear_def)
  apply unfold_locales
  apply (simp add: cross_add_right)
  apply (simp add: cross_mult_right)
  apply (simp add: cross_add_left)
  apply (simp add: cross_mult_left)
  done

```

6.33.2 Preservation by rotation, or other orthogonal transformation up to sign

```

lemma cross_matrix_mult: transpose A *v ((A *v x)  $\times$  (A *v y)) = det A *R (x
 $\times$  y)
  apply (simp add: vec_eq_iff )
  apply (simp add: vector_matrix_mult_def matrix_vector_mult_def forall_3
cross3_simps)
  done

```

```

lemma cross_orthogonal_matrix:
  assumes orthogonal_matrix A
  shows (A *v x)  $\times$  (A *v y) = det A *R (A *v (x  $\times$  y))
proof -
  have mat 1 = transpose (A ** transpose A)
  by (metis (no_types) assms orthogonal_matrix_def transpose_mat)
  then show ?thesis
  by (metis (no_types) vector_matrix_mul_rid vector_transpose_matrix cross_matrix_mult
matrix_vector_mul_assoc matrix_vector_mult_scaleR)
qed

```

```

lemma cross_rotation_matrix: rotation_matrix A  $\implies$  (A *v x)  $\times$  (A *v y) =
A *v (x  $\times$  y)
  by (simp add: rotation_matrix_def cross_orthogonal_matrix)

```

```

lemma cross_rotoinversion_matrix: rotoinversion_matrix A  $\implies$  (A *v x)  $\times$  (A
*v y) = - A *v (x  $\times$  y)
  by (simp add: rotoinversion_matrix_def cross_orthogonal_matrix scaleR_matrix_vector_assoc)

```

```

lemma cross_orthogonal_transformation:
  assumes orthogonal_transformation f
  shows (f x)  $\times$  (f y) = det(matrix f) *R f(x  $\times$  y)
proof -
  have orth: orthogonal_matrix (matrix f)
  using assms orthogonal_transformation_matrix by blast
  have matrix f *v z = f z for z
  using assms orthogonal_transformation_matrix by force
  with cross_orthogonal_matrix [OF orth] show ?thesis
  by simp
qed

```

```

lemma cross_linear_image:
   $\llbracket \text{linear } f; \bigwedge x. \text{norm}(f\ x) = \text{norm } x; \det(\text{matrix } f) = 1 \rrbracket$ 
     $\implies (f\ x) \times (f\ y) = f(x \times y)$ 
  by (simp add: cross_orthogonal_transformation orthogonal_transformation)

```

6.33.3 Continuity

```

lemma continuous_cross:  $\llbracket \text{continuous } F\ f; \text{continuous } F\ g \rrbracket \implies \text{continuous } F$ 
   $(\lambda x. (f\ x) \times (g\ x))$ 
  apply (subst continuous_componentwise)
  apply (clarsimp simp add: cross3_simps)
  apply (intro continuous_intros; simp)
done

```

```

lemma continuous_on_cross:
  fixes  $f :: 'a::t2\_space \Rightarrow \text{real}^3$ 
  shows  $\llbracket \text{continuous\_on } S\ f; \text{continuous\_on } S\ g \rrbracket \implies \text{continuous\_on } S\ (\lambda x. (f\ x)$ 
   $\times (g\ x))$ 
  by (simp add: continuous_on_eq_continuous_within continuous_cross)

```

```

unbundle no_cross3_syntax

```

```

end

```

6.34 Bounded Continuous Functions

```

theory Bounded_Continuous_Function
  imports
    Topology_Euclidean_Space
    Uniform_Limit
begin

```

6.34.1 Definition

```

definition bcontfun =  $\{f. \text{continuous\_on } UNIV\ f \wedge \text{bounded } (\text{range } f)\}$ 

```

```

typedef (overloaded) ( $'a, 'b$ ) bcontfun (( $\_ \Rightarrow_C \_$ ) [22, 21] 21) =
  bcontfun::('a::topological_space \Rightarrow 'b::metric_space) set
morphisms apply_bcontfun Bcontfun
by (auto intro: continuous_intros simp: bounded_def bcontfun_def)

```

```

declare [coercion apply_bcontfun :: ('a::topological_space \Rightarrow_C 'b::metric_space) \Rightarrow
 $'a \Rightarrow 'b$ ]]

```

```

setup_lifting type_definition_bcontfun

```

```

lemma continuous_on_apply_bcontfun[intro, simp]:  $\text{continuous\_on } T\ (\text{apply\_bcontfun } x)$ 

```

```

and bounded_apply_bcontfun[intro, simp]: bounded (range (apply_bcontfun x))
using apply_bcontfun[of x]
by (auto simp: bcontfun_def intro: continuous_on_subset)

```

```

lemma bcontfun_eqI: ( $\bigwedge x. \text{apply\_bcontfun } f \ x = \text{apply\_bcontfun } g \ x$ )  $\implies f = g$ 
by transfer auto

```

```

lemma bcontfunE:
  assumes  $f \in \text{bcontfun}$ 
  obtains  $g$  where  $f = \text{apply\_bcontfun } g$ 
by (blast intro: apply_bcontfun_cases assms )

```

```

lemma const_bcontfun: ( $\lambda x. b$ )  $\in \text{bcontfun}$ 
by (auto simp: bcontfun_def image_def)

```

```

lift_definition const_bcontfun::'b::metric_space  $\Rightarrow$  ('a::topological_space  $\Rightarrow_C$  'b)
is  $\lambda c \_. c$ 
by (rule const_bcontfun)

```

```

instantiation bcontfun :: (topological_space, metric_space) metric_space
begin

```

```

lift_definition dist_bcontfun :: 'a  $\Rightarrow_C$  'b  $\Rightarrow$  'a  $\Rightarrow_C$  'b  $\Rightarrow$  real
is  $\lambda f \ g. (\text{SUP } x. \text{dist } (f \ x) \ (g \ x))$  .

```

```

definition uniformity_bcontfun :: ('a  $\Rightarrow_C$  'b  $\times$  'a  $\Rightarrow_C$  'b) filter
where uniformity_bcontfun = (INF  $e \in \{0 <.. \}$ . principal  $\{(x, y). \text{dist } x \ y < e\}$ )

```

```

definition open_bcontfun :: ('a  $\Rightarrow_C$  'b) set  $\Rightarrow$  bool
where open_bcontfun  $S = (\forall x \in S. \forall_F (x', y) \text{ in uniformity. } x' = x \longrightarrow y \in S)$ 

```

```

lemma bounded_dist_le_SUP_dist:
  bounded (range  $f$ )  $\implies$  bounded (range  $g$ )  $\implies \text{dist } (f \ x) \ (g \ x) \leq (\text{SUP } x. \text{dist } (f \ x) \ (g \ x))$ 
by (auto intro!: cSUP_upper bounded_imp_bdd_above bounded_dist_comp)

```

```

lemma dist_bounded:
  fixes  $f \ g :: 'a \Rightarrow_C 'b$ 
  shows  $\text{dist } (f \ x) \ (g \ x) \leq \text{dist } f \ g$ 
by transfer (auto intro!: bounded_dist_le_SUP_dist simp: bcontfun_def)

```

```

lemma dist_bound:
  fixes  $f \ g :: 'a \Rightarrow_C 'b$ 
  assumes  $\bigwedge x. \text{dist } (f \ x) \ (g \ x) \leq b$ 
  shows  $\text{dist } f \ g \leq b$ 
using assms
by transfer (auto intro!: cSUP_least)

```

```

lemma dist_fun_lt_imp_dist_val_lt:
  fixes f g :: 'a  $\Rightarrow_C$  'b
  assumes dist f g < e
  shows dist (f x) (g x) < e
  using dist_bounded assms by (rule le_less_trans)

instance
proof
  fix f g h :: 'a  $\Rightarrow_C$  'b
  show dist f g = 0  $\longleftrightarrow$  f = g
  proof
    have  $\bigwedge x. \text{dist } (f x) (g x) \leq \text{dist } f g$ 
    by (rule dist_bounded)
    also assume dist f g = 0
    finally show f = g
    by (auto simp: apply_bcontfun_inject[symmetric])
  qed (auto simp: dist_bcontfun_def intro!: cSup_eq)
  show dist f g  $\leq$  dist f h + dist g h
  proof (rule dist_bound)
    fix x
    have dist (f x) (g x)  $\leq$  dist (f x) (h x) + dist (g x) (h x)
    by (rule dist_triangle2)
    also have dist (f x) (h x)  $\leq$  dist f h
    by (rule dist_bounded)
    also have dist (g x) (h x)  $\leq$  dist g h
    by (rule dist_bounded)
    finally show dist (f x) (g x)  $\leq$  dist f h + dist g h
    by simp
  qed
qed (rule open_bcontfun_def uniformity_bcontfun_def)+

end

lift_definition PiC :: 'a :: topological_space set  $\Rightarrow$  ('a  $\Rightarrow$  'b set)  $\Rightarrow$  ('a  $\Rightarrow_C$  'b :: metric_space)
set
  is  $\lambda I X. \text{Pi } I X \cap \text{bcontfun}$ 
  by auto

lemma mem_PiC_iff:  $x \in \text{PiC } I X \longleftrightarrow \text{apply\_bcontfun } x \in \text{Pi } I X$ 
  by transfer simp

lemmas mem_PiCD = mem_PiC_iff[THEN iffD1]
  and mem_PiCI = mem_PiC_iff[THEN iffD2]

lemma tendsto_bcontfun_uniform_limit:
  fixes f :: 'i  $\Rightarrow$  'a :: topological_space  $\Rightarrow_C$  'b :: metric_space
  assumes (f  $\longrightarrow$  l) F
  shows uniform_limit UNIV f l F
proof (rule uniform_limitI)

```

```

fix e::real assume e > 0
from tendstoD[OF assms this] have  $\forall_F x \text{ in } F. \text{dist } (f x) l < e$  .
then show  $\forall_F n \text{ in } F. \forall x \in UNIV. \text{dist } ((f n) x) (l x) < e$ 
  by eventually_elim (auto simp: dist_fun_lt_imp_dist_val_lt)
qed

```

```

lemma uniform_limit_tendsto_bcontfun:
  fixes f::'i  $\Rightarrow$  'a::topological_space  $\Rightarrow_C$  'b::metric_space
  and l::'a::topological_space  $\Rightarrow_C$  'b::metric_space
  assumes uniform_limit UNIV f l F
  shows (f  $\longrightarrow$  l) F
proof (rule tendstoI)
  fix e::real assume e > 0
  then have e / 2 > 0 by simp
  from uniform_limitD[OF assms this]
  have  $\forall_F i \text{ in } F. \forall x. \text{dist } (f i x) (l x) < e / 2$  by simp
  then have  $\forall_F x \text{ in } F. \text{dist } (f x) l \leq e / 2$ 
    by eventually_elim (blast intro: dist_bound_less_imp_le)
  then show  $\forall_F x \text{ in } F. \text{dist } (f x) l < e$ 
    by eventually_elim (use <0 < e> in auto)
qed

```

```

lemma uniform_limit_bcontfunE:
  fixes f::'i  $\Rightarrow$  'a::topological_space  $\Rightarrow_C$  'b::metric_space
  and l::'a::topological_space  $\Rightarrow$  'b::metric_space
  assumes uniform_limit UNIV f l F F  $\neq$  bot
  obtains l':'a::topological_space  $\Rightarrow_C$  'b::metric_space
  where l = l' (f  $\longrightarrow$  l') F
by (metis (mono_tags, lifting) always_eventually_apply_bcontfun apply_bcontfun_cases
  assms
  bcontfun_def mem_Collect_eq uniform_limit_bounded uniform_limit_tendsto_bcontfun
  uniform_limit_theorem)

```

```

lemma closed_PiC:
  fixes I :: 'a::metric_space set
  and X :: 'a  $\Rightarrow$  'b::complete_space set
  assumes  $\bigwedge i. i \in I \implies \text{closed } (X i)$ 
  shows closed (PiC I X)
  unfolding closed_sequential_limits
proof safe
  fix f l
  assume seq:  $\forall n. f n \in \text{PiC } I X$  and lim: f  $\longrightarrow$  l
  show l  $\in$  PiC I X
proof (safe intro!: mem_PiCI)
  fix x assume x  $\in$  I
  then have closed (X x)
    using assms by simp
  moreover have eventually ( $\lambda i. f i x \in X x$ ) sequentially
    using seq <x  $\in$  I>

```

```

      by (auto intro!: eventuallyI dest!: mem_PiCD simp: Pi_iff)
    moreover note sequentially_bot
    moreover have  $(\lambda n. (f\ n)\ x) \longrightarrow l\ x$ 
      using tendsto_bcontfun_uniform_limit[OF lim]
      by (rule tendsto_uniform_limitI) simp
    ultimately show  $l\ x \in X\ x$ 
      by (rule Lim_in_closed_set)
  qed
qed

```

6.34.2 Complete Space

instance *bcontfun* :: (*metric_space*, *complete_space*) *complete_space*

proof

```

  fix  $f :: \text{nat} \Rightarrow ('a, 'b)\ \text{bcontfun}$ 
  assume Cauchy  $f$  — Cauchy equals uniform convergence
  then obtain  $g$  where uniform_limit UNIV  $f\ g$  sequentially
    using uniformly_convergent_eq_cauchy[of  $\lambda\_.$  True  $f$ ]
  unfolding Cauchy_def uniform_limit_sequentially_iff
  by (metis dist_fun_lt_imp_dist_val_lt)

```

```

  from uniform_limit_bcontfunE[OF this sequentially_bot]
  obtain  $l'$  where  $g = \text{apply\_bcontfun } l' (f \longrightarrow l')$  by metis
  then show convergent  $f$ 
    by (intro convergentI)

```

qed

6.34.3 Supremum norm for a normed vector space

instantiation *bcontfun* :: (*topological_space*, *real_normed_vector*) *real_vector*
begin

lemma *uminus_cont*: $f \in \text{bcontfun} \implies (\lambda x. -\ f\ x) \in \text{bcontfun}$ **for** $f :: 'a \Rightarrow 'b$
 by (auto simp: *bcontfun_def* intro!: *continuous_intros*)

lemma *plus_cont*: $f \in \text{bcontfun} \implies g \in \text{bcontfun} \implies (\lambda x. f\ x + g\ x) \in \text{bcontfun}$
for $f\ g :: 'a \Rightarrow 'b$
 by (auto simp: *bcontfun_def* intro!: *continuous_intros* *bounded_plus_comp*)

lemma *minus_cont*: $f \in \text{bcontfun} \implies g \in \text{bcontfun} \implies (\lambda x. f\ x - g\ x) \in \text{bcontfun}$
for $f\ g :: 'a \Rightarrow 'b$
 by (auto simp: *bcontfun_def* intro!: *continuous_intros* *bounded_minus_comp*)

lemma *scaleR_cont*: $f \in \text{bcontfun} \implies (\lambda x. a\ *_R\ f\ x) \in \text{bcontfun}$ **for** $f :: 'a \Rightarrow 'b$
 by (auto simp: *bcontfun_def* intro!: *continuous_intros* *bounded_scaleR_comp*)

lemma *bcontfun_normI*: *continuous_on* UNIV $f \implies (\bigwedge x. \text{norm } (f\ x) \leq b) \implies f \in \text{bcontfun}$
 by (auto simp: *bcontfun_def* intro: *boundedI*)

lift_definition *uminus_bcontfun*::('a $\Rightarrow_C$ 'b) $\Rightarrow$ 'a $\Rightarrow_C$ 'b **is** $\lambda f x. - f x$
by (rule *uminus_cont*)

lift_definition *plus_bcontfun*::('a $\Rightarrow_C$ 'b) $\Rightarrow$ ('a $\Rightarrow_C$ 'b) $\Rightarrow$ 'a $\Rightarrow_C$ 'b **is** $\lambda f g x. f x + g x$
by (rule *plus_cont*)

lift_definition *minus_bcontfun*::('a $\Rightarrow_C$ 'b) $\Rightarrow$ ('a $\Rightarrow_C$ 'b) $\Rightarrow$ 'a $\Rightarrow_C$ 'b **is** $\lambda f g x. f x - g x$
by (rule *minus_cont*)

lift_definition *zero_bcontfun*::'a $\Rightarrow_C$ 'b **is** $\lambda \_. 0$
by (rule *const_bcontfun*)

lemma *const_bcontfun_0_eq_0[simp]*: *const_bcontfun 0 = 0*
by *transfer simp*

lift_definition *scaleR_bcontfun*::real $\Rightarrow$ ('a $\Rightarrow_C$ 'b) $\Rightarrow$ 'a $\Rightarrow_C$ 'b **is** $\lambda r g x. r *_{\mathbb{R}} g x$
by (rule *scaleR_cont*)

lemmas [*simp*] =
const_bcontfun.rep_eq
uminus_bcontfun.rep_eq
plus_bcontfun.rep_eq
minus_bcontfun.rep_eq
zero_bcontfun.rep_eq
scaleR_bcontfun.rep_eq

instance
by *standard (auto intro!: bcontfun_eqI simp: algebra_simps)*

end

lemma *bounded_norm_le_SUP_norm*:
 $\text{bounded } (\text{range } f) \implies \text{norm } (f x) \leq (\text{SUP } x. \text{norm } (f x))$
by (*auto intro!: cSUP_upper bounded_imp_bdd_above simp: bounded_norm_comp*)

instantiation *bcontfun* :: (topological_space, real_normed_vector) real_normed_vector
begin

definition *norm_bcontfun* :: ('a, 'b) bcontfun $\Rightarrow$ real
where *norm_bcontfun f = dist f 0*

definition *sgn* (*f*::('a,'b) bcontfun) = *f /_R norm f*

instance
proof

```

fix a :: real
fix f g :: ('a, 'b) bcontfun
show dist f g = norm (f - g)
  unfolding norm_bcontfun_def
  by transfer (simp add: dist_norm)
show norm (f + g) ≤ norm f + norm g
  unfolding norm_bcontfun_def
  by transfer
    (auto intro!: cSUP_least norm_triangle_le add_mono bounded_norm_le_SUP_norm
      simp: dist_norm bcontfun_def)
show norm (a *R f) = |a| * norm f
  unfolding norm_bcontfun_def
  apply transfer
  by (rule trans[OF continuous_at_Sup_mono[symmetric]])
    (auto intro!: monoI mult_left_mono continuous_intros bounded_imp_bdd_above
      simp: bounded_norm_comp bcontfun_def image_comp)
qed (auto simp: norm_bcontfun_def sgn_bcontfun_def)

end

```

```

lemma norm_bounded:
  fixes f :: ('a::topological_space, 'b::real_normed_vector) bcontfun
  shows norm (apply_bcontfun f x) ≤ norm f
  using dist_bounded[of f x 0]
  by (simp add: dist_norm)

```

```

lemma norm_bound:
  fixes f :: ('a::topological_space, 'b::real_normed_vector) bcontfun
  assumes  $\bigwedge x. \text{norm} (\text{apply\_bcontfun } f \ x) \leq b$ 
  shows norm f ≤ b
  using dist_bound[of f 0 b] assms
  by (simp add: dist_norm)

```

6.34.4 (bounded) continuous extenstion

```

lemma continuous_on_cbox_bcontfunE:
  fixes f :: 'a::euclidean_space ⇒ 'b::metric_space
  assumes continuous_on (cbox a b) f
  obtains g :: 'a ⇒C 'b where
     $\bigwedge x. x \in \text{cbox } a \ b \implies g \ x = f \ x$ 
     $\bigwedge x. g \ x = f \ (\text{clamp } a \ b \ x)$ 
proof –
  define g where g ≡ ext_cont f a b
  have g ∈ bcontfun
  using assms
  by (auto intro!: continuous_on_ext_cont simp: g_def bcontfun_def)
    (auto simp: g_def ext_cont_def
      intro!: clamp_bounded_compact_imp_bounded[OF compact_continuous_image]
      assms)

```

```

then obtain  $h$  where  $h$ :  $g = \text{apply\_bcontfun } h$  by (rule bcontfunE)
then have  $h\ x = f\ x$  if  $x \in \text{cbox } a\ b$  for  $x$ 
  by (auto simp:  $h[\text{symmetric}]$   $g\_def$  that)
moreover
have  $h\ x = f\ (\text{clamp } a\ b\ x)$  for  $x$ 
  by (auto simp:  $h[\text{symmetric}]$   $g\_def$  ext_cont_def)
ultimately show ?thesis ..
qed

```

```

lifting_update bcontfun.lifting
lifting_forget bcontfun.lifting

```

```

end

```

6.35 Lindelöf spaces

```

theory Lindelof_Spaces
imports T1_Spaces
begin

```

```

definition Lindelof_space where
  Lindelof_space  $X \equiv$ 
     $\forall \mathcal{U}. (\forall U \in \mathcal{U}. \text{openin } X\ U) \wedge \bigcup \mathcal{U} = \text{topspace } X$ 
     $\longrightarrow (\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \bigcup \mathcal{V} = \text{topspace } X)$ 

```

```

lemma Lindelof_spaceD:
   $\llbracket \text{Lindelof\_space } X; \bigwedge U. U \in \mathcal{U} \implies \text{openin } X\ U; \bigcup \mathcal{U} = \text{topspace } X \rrbracket$ 
   $\implies \exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \bigcup \mathcal{V} = \text{topspace } X$ 
  by (auto simp: Lindelof_space_def)

```

```

lemma Lindelof_space_alt:
  Lindelof_space  $X \longleftrightarrow$ 
     $(\forall \mathcal{U}. (\forall U \in \mathcal{U}. \text{openin } X\ U) \wedge \text{topspace } X \subseteq \bigcup \mathcal{U}$ 
       $\longrightarrow (\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \text{topspace } X \subseteq \bigcup \mathcal{V}))$ 
  unfolding Lindelof_space_def
  using openin_subset by fastforce

```

```

lemma compact_imp_Lindelof_space:
  compact_space  $X \implies \text{Lindelof\_space } X$ 
  unfolding Lindelof_space_def compact_space
  by (meson uncountable_infinite)

```

```

lemma Lindelof_space_topspace_empty:
   $\text{topspace } X = \{\} \implies \text{Lindelof\_space } X$ 
  using compact_imp_Lindelof_space compact_space_topspace_empty by blast

```

```

lemma Lindelof_space_Union:
  assumes  $\mathcal{U}$ : countable  $\mathcal{U}$  and  $\text{lin}$ :  $\bigwedge U. U \in \mathcal{U} \implies \text{Lindelof\_space } (\text{subtopology } X\ U)$ 

```

```

shows Lindelof_space (subtopology X (⋃U))
proof -
  have  $\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{F} \wedge \bigcup \mathcal{U} \cap \bigcup \mathcal{V} = \text{topspace } X \cap \bigcup \mathcal{U}$ 
    if  $\mathcal{F}: \mathcal{F} \subseteq \text{Collect } (\text{openin } X)$  and  $UF: \bigcup \mathcal{U} \cap \bigcup \mathcal{F} = \text{topspace } X \cap \bigcup \mathcal{U}$ 
    for  $\mathcal{F}$ 
  proof -
    have  $\bigwedge U. \llbracket U \in \mathcal{U}; U \cap \bigcup \mathcal{F} = \text{topspace } X \cap U \rrbracket$ 
       $\implies \exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{F} \wedge U \cap \bigcup \mathcal{V} = \text{topspace } X \cap U$ 
    using lin  $\mathcal{F}$ 
    unfolding Lindelof_space_def openin_subtopology_alt Ball_def subset_iff
    [symmetric]
    by (simp add: all_subset_image imp_conjL ex_countable_subset_image)
  then obtain  $g$  where  $g: \bigwedge U. \llbracket U \in \mathcal{U}; U \cap \bigcup \mathcal{F} = \text{topspace } X \cap U \rrbracket$ 
     $\implies \text{countable } (g \text{ ` } U) \wedge (g \text{ ` } U) \subseteq \mathcal{F} \wedge U \cap \bigcup (g \text{ ` } U) =$ 
 $\text{topspace } X \cap U$ 
    by metis
  show ?thesis
  proof (intro exI conjI)
    show countable  $(\bigcup (g \text{ ` } \mathcal{U}))$ 
      using Int_commute  $UF \ g$  by (fastforce intro: countable_UN [OF  $\mathcal{U}$ ])
    show  $\bigcup (g \text{ ` } \mathcal{U}) \subseteq \mathcal{F}$ 
      using  $g \ UF$  by blast
    show  $\bigcup \mathcal{U} \cap \bigcup (\bigcup (g \text{ ` } \mathcal{U})) = \text{topspace } X \cap \bigcup \mathcal{U}$ 
  proof
    show  $\bigcup \mathcal{U} \cap \bigcup (\bigcup (g \text{ ` } \mathcal{U})) \subseteq \text{topspace } X \cap \bigcup \mathcal{U}$ 
      using  $g \ UF$  by blast
    show  $\text{topspace } X \cap \bigcup \mathcal{U} \subseteq \bigcup \mathcal{U} \cap \bigcup (\bigcup (g \text{ ` } \mathcal{U}))$ 
  proof clarsimp
    show  $\exists y \in \mathcal{U}. \exists W \in g \ y. x \in W$ 
      if  $x \in \text{topspace } X \ x \in V \ V \in \mathcal{U}$  for  $x \ V$ 
    proof -
      have  $V \cap \bigcup \mathcal{F} = \text{topspace } X \cap V$ 
        using  $UF \ \langle V \in \mathcal{U} \rangle$  by blast
      with that  $g \ [OF \ \langle V \in \mathcal{U} \rangle]$  show ?thesis by blast
    qed
  qed
qed
qed
qed
qed
then show ?thesis
  unfolding Lindelof_space_def openin_subtopology_alt Ball_def subset_iff
  [symmetric]
  by (simp add: all_subset_image imp_conjL ex_countable_subset_image)
qed

lemma countable_imp_Lindelof_space:
  assumes countable(topspace X)
  shows Lindelof_space X
proof -

```

```

have Lindelof_space (subtopology X ( $\bigcup x \in \text{topspace } X. \{x\}$ ))
proof (rule Lindelof_space_Union)
  show countable (( $\lambda x. \{x\}$ ) '  $\text{topspace } X$ )
    using assms by blast
  show Lindelof_space (subtopology X U)
    if  $U \in (\lambda x. \{x\})$  '  $\text{topspace } X$  for U
  proof -
    have compactin X U
      using that by force
    then show ?thesis
      by (meson compact_imp_Lindelof_space compact_space_subtopology)
    qed
  qed
then show ?thesis
  by simp
qed
lemma Lindelof_space_subtopology:
  Lindelof_space(subtopology X S)  $\longleftrightarrow$ 
  ( $\forall \mathcal{U}. (\forall U \in \mathcal{U}. \text{openin } X U) \wedge \text{topspace } X \cap S \subseteq \bigcup \mathcal{U}$ 
 $\longrightarrow (\exists V. \text{countable } V \wedge V \subseteq \mathcal{U} \wedge \text{topspace } X \cap S \subseteq \bigcup V)$ )
proof -
  have *: ( $S \cap \bigcup \mathcal{U} = \text{topspace } X \cap S$ ) = ( $\text{topspace } X \cap S \subseteq \bigcup \mathcal{U}$ )
  if  $\bigwedge x. x \in \mathcal{U} \implies \text{openin } X x$  for  $\mathcal{U}$ 
  by (blast dest: openin_subset [OF that])
  moreover have ( $V \subseteq \mathcal{U} \wedge S \cap \bigcup V = \text{topspace } X \cap S$ ) = ( $V \subseteq \mathcal{U} \wedge \text{topspace } X$ 
 $\cap S \subseteq \bigcup V$ )
  if  $\forall x. x \in \mathcal{U} \longrightarrow \text{openin } X x$   $\text{topspace } X \cap S \subseteq \bigcup \mathcal{U}$  countable  $V$  for  $\mathcal{U} V$ 
  using that * by blast
  ultimately show ?thesis
    unfolding Lindelof_space_def openin_subtopology_alt Ball_def
    apply (simp add: all_subset_image imp_conjL ex_countable_subset_image
flip: subset_iff)
    apply (intro all_cong1 imp_cong ex_cong, auto)
    done
  qed
lemma Lindelof_space_subtopology_subset:
   $S \subseteq \text{topspace } X$ 
 $\implies (\text{Lindelof\_space}(\text{subtopology } X S) \longleftrightarrow$ 
  ( $\forall \mathcal{U}. (\forall U \in \mathcal{U}. \text{openin } X U) \wedge S \subseteq \bigcup \mathcal{U}$ 
 $\longrightarrow (\exists V. \text{countable } V \wedge V \subseteq \mathcal{U} \wedge S \subseteq \bigcup V)))$ 
  by (metis Lindelof_space_subtopology_topspace_subtopology_topspace_subtopology_subset)
lemma Lindelof_space_closedin_subtopology:
  assumes X: Lindelof_space X and clo: closedin X S
  shows Lindelof_space (subtopology X S)
proof -
  have  $S \subseteq \text{topspace } X$ 
  by (simp add: clo closedin_subset)

```

```

then show ?thesis
proof (clarsimp simp add: Lindelof_space_subtopology_subset)
  show  $\exists V. \text{countable } V \wedge V \subseteq \mathcal{F} \wedge S \subseteq \bigcup V$ 
  if  $\forall U \in \mathcal{F}. \text{openin } X \ U \text{ and } S \subseteq \bigcup \mathcal{F}$  for  $\mathcal{F}$ 
proof -
  have  $\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \text{insert } (\text{topspace } X - S) \ \mathcal{F} \wedge \bigcup \mathcal{V} = \text{topspace } X$ 
  proof (rule Lindelof_spaceD [OF X, of insert (topspace X - S)  $\mathcal{F}$ ])
    show openin X U
    if  $U \in \text{insert } (\text{topspace } X - S) \ \mathcal{F}$  for U
    using that  $\langle \forall U \in \mathcal{F}. \text{openin } X \ U \rangle$  clo by blast
  show  $\bigcup (\text{insert } (\text{topspace } X - S) \ \mathcal{F}) = \text{topspace } X$ 
  apply auto
  apply (meson in_mono openin_closedin_eq that(1))
  using UnionE  $\langle S \subseteq \bigcup \mathcal{F} \rangle$  by auto
qed
then obtain  $\mathcal{V}$  where countable  $\mathcal{V}$   $\mathcal{V} \subseteq \text{insert } (\text{topspace } X - S) \ \mathcal{F} \bigcup \mathcal{V} =$ 
topspace X
  by metis
with  $\langle S \subseteq \text{topspace } X \rangle$ 
show ?thesis
  by (rule_tac x=( $\mathcal{V} - \{\text{topspace } X - S\}$ ) in exI) auto
qed
qed
qed

```

lemma *Lindelof_space_continuous_map_image:*

```

  assumes X: Lindelof_space X and f: continuous_map X Y f and fim: f ‘
(topspace X) = topspace Y
  shows Lindelof_space Y
proof -
  have  $\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \bigcup \mathcal{V} = \text{topspace } Y$ 
  if  $\mathcal{U}: \bigwedge U. U \in \mathcal{U} \implies \text{openin } Y \ U$  and  $UU: \bigcup \mathcal{U} = \text{topspace } Y$  for  $\mathcal{U}$ 
proof -
  define  $\mathcal{V}$  where  $\mathcal{V} \equiv (\lambda U. \{x \in \text{topspace } X. f \ x \in U\}) \ \mathcal{U}$ 
  have  $\bigwedge V. V \in \mathcal{V} \implies \text{openin } X \ V$ 
  unfolding  $\mathcal{V\_def}$  using  $\mathcal{U}$  continuous_map f by fastforce
  moreover have  $\bigcup \mathcal{V} = \text{topspace } X$ 
  unfolding  $\mathcal{V\_def}$  using UU fim by fastforce
  ultimately have  $\exists \mathcal{W}. \text{countable } \mathcal{W} \wedge \mathcal{W} \subseteq \mathcal{V} \wedge \bigcup \mathcal{W} = \text{topspace } X$ 
  using X by (simp add: Lindelof_space_def)
  then obtain  $\mathcal{C}$  where countable  $\mathcal{C}$   $\mathcal{C} \subseteq \mathcal{U}$  and  $\mathcal{C}: (\bigcup U \in \mathcal{C}. \{x \in \text{topspace } X. f$ 
 $x \in U\}) = \text{topspace } X$ 
  by (metis (no_types, lifting)  $\mathcal{V\_def}$  countable_subset_image)
  moreover have  $\bigcup \mathcal{C} = \text{topspace } Y$ 
proof
  show  $\bigcup \mathcal{C} \subseteq \text{topspace } Y$ 
  using UU  $\mathcal{C} \ \langle \mathcal{C} \subseteq \mathcal{U} \rangle$  by fastforce
  have  $y \in \bigcup \mathcal{C}$  if  $y \in \text{topspace } Y$  for y
proof -

```

```

    obtain  $x$  where  $x \in \text{topspace } X$   $y = f\ x$ 
    using that  $\text{fin}$  by  $(\text{metis } \langle y \in \text{topspace } Y \rangle \text{ image } E)$ 
    with  $C$  show  $?thesis$  by auto
  qed
  then show  $\text{topspace } Y \subseteq \bigcup C$  by blast
  qed
  ultimately show  $?thesis$ 
  by blast
  qed
  then show  $?thesis$ 
  unfolding  $\text{Lindelof\_space\_def}$ 
  by auto
  qed

lemma  $\text{Lindelof\_space\_quotient\_map\_image}$ :
   $\llbracket \text{quotient\_map } X\ Y\ q; \text{Lindelof\_space } X \rrbracket \implies \text{Lindelof\_space } Y$ 
  by  $(\text{meson } \text{Lindelof\_space\_continuous\_map\_image } \text{quotient\_imp\_continuous\_map}$ 
 $\text{quotient\_imp\_surjective\_map})$ 

lemma  $\text{Lindelof\_space\_retraction\_map\_image}$ :
   $\llbracket \text{retraction\_map } X\ Y\ r; \text{Lindelof\_space } X \rrbracket \implies \text{Lindelof\_space } Y$ 
  using  $\text{Abstract\_Topology.retraction\_imp\_quotient\_map } \text{Lindelof\_space\_quotient\_map\_image}$ 
  by blast

lemma  $\text{locally\_finite\_cover\_of\_Lindelof\_space}$ :
  assumes  $X: \text{Lindelof\_space } X$  and  $UU: \text{topspace } X \subseteq \bigcup \mathcal{U}$  and  $\text{fin: locally\_finite\_in}$ 
 $X\ \mathcal{U}$ 
  shows  $\text{countable } \mathcal{U}$ 
  proof -
    have  $UU\_eq: \bigcup \mathcal{U} = \text{topspace } X$ 
    by  $(\text{meson } UU\ \text{fin } \text{locally\_finite\_in\_def } \text{subset\_antisym})$ 
    obtain  $T$  where  $T: \bigwedge x. x \in \text{topspace } X \implies \text{openin } X\ (T\ x) \wedge x \in T\ x \wedge \text{finite}$ 
 $\{U \in \mathcal{U}. U \cap T\ x \neq \{\}\}$ 
    using  $\text{fin}$  unfolding  $\text{locally\_finite\_in\_def}$  by  $\text{metis}$ 
    then obtain  $I$  where  $\text{countable } I$   $I \subseteq \text{topspace } X$  and  $I: \text{topspace } X \subseteq \bigcup (T\ `$ 
 $I)$ 
    using  $X$  unfolding  $\text{Lindelof\_space\_alt}$ 
    by  $(\text{drule\_tac } x=\text{image } T\ (\text{topspace } X)\ \text{in spec})\ (\text{auto simp: ex\_countable\_subset\_image})$ 
    show  $?thesis$ 
    proof (rule  $\text{countable\_subset}$ )
      have  $\bigwedge i. i \in I \implies \text{countable } \{U \in \mathcal{U}. U \cap T\ i \neq \{\}\}$ 
      using  $T$ 
      by  $(\text{meson } \langle I \subseteq \text{topspace } X \rangle\ \text{in\_mono } \text{uncountable\_infinite})$ 
      then show  $\text{countable } (\text{insert } \{\} (\bigcup_{i \in I}. \{U \in \mathcal{U}. U \cap T\ i \neq \{\}\}))$ 
      by  $(\text{simp add: } \langle \text{countable } I \rangle)$ 
    qed (use  $UU\_eq\ I$  in auto)
  qed

```

```

lemma Lindelof_space_proper_map_preimage:
  assumes f: proper_map X Y f and Y: Lindelof_space Y
  shows Lindelof_space X
proof (clarsimp simp: Lindelof_space_alt)
  show  $\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \text{topspace } X \subseteq \bigcup \mathcal{V}$ 
    if U:  $\forall U \in \mathcal{U}. \text{openin } X \ U$  and sub_UU:  $\text{topspace } X \subseteq \bigcup \mathcal{U}$  for U
  proof -
    have  $\exists \mathcal{V}. \text{finite } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \{x \in \text{topspace } X. f \ x = y\} \subseteq \bigcup \mathcal{V}$  if y  $\in \text{topspace } Y$  for y
    proof (rule compactinD)
      show compactin X  $\{x \in \text{topspace } X. f \ x = y\}$ 
        using f proper_map_def that by fastforce
      qed (use sub_UU U in auto)
      then obtain V where V:  $\bigwedge y. y \in \text{topspace } Y \implies \text{finite } (\mathcal{V} \ y) \wedge \mathcal{V} \ y \subseteq \mathcal{U} \wedge \{x \in \text{topspace } X. f \ x = y\} \subseteq \bigcup (\mathcal{V} \ y)$ 
        by meson
      define W where W  $\equiv (\lambda y. \text{topspace } Y - \text{image } f \ (\text{topspace } X - \bigcup (\mathcal{V} \ y))) \text{ 'topspace } Y$ 
      have  $\forall U \in \mathcal{W}. \text{openin } Y \ U$ 
        using f U V unfolding W_def proper_map_def closed_map_def
        by (simp add: closedin_diff openin_Union openin_diff subset_iff)
      moreover have  $\text{topspace } Y \subseteq \bigcup \mathcal{W}$ 
        using V unfolding W_def by clarsimp fastforce
      ultimately have  $\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{W} \wedge \text{topspace } Y \subseteq \bigcup \mathcal{V}$ 
        using Y by (simp add: Lindelof_space_alt)
      then obtain I where countable I  $I \subseteq \text{topspace } Y$ 
        and I:  $\text{topspace } Y \subseteq (\bigcup i \in I. \text{topspace } Y - f \text{ 'topspace } X - \bigcup (\mathcal{V} \ i))$ 
        unfolding W_def ex_countable_subset_image by metis
      show ?thesis
      proof (intro exI conjI)
        have  $\bigwedge i. i \in I \implies \text{countable } (\mathcal{V} \ i)$ 
          by (meson V  $\langle I \subseteq \text{topspace } Y \rangle$  in_mono uncountable_infinite)
        with  $\langle \text{countable } I \rangle$  show  $\text{countable } (\bigcup (\mathcal{V} \text{ ' } I))$ 
          by auto
        show  $\bigcup (\mathcal{V} \text{ ' } I) \subseteq \mathcal{U}$ 
          using V  $\langle I \subseteq \text{topspace } Y \rangle$  by fastforce
        show  $\text{topspace } X \subseteq \bigcup (\bigcup (\mathcal{V} \text{ ' } I))$ 
          proof
            show  $x \in \bigcup (\bigcup (\mathcal{V} \text{ ' } I))$  if x  $\in \text{topspace } X$  for x
            proof -
              have f x  $\in \text{topspace } Y$ 
                by (meson f image_subset_iff proper_map_imp_subset_topspace that)
              then show ?thesis
                using that I by auto
            qed
          qed
        qed
      qed
    qed
  qed

```



```

lemma Lindelof_space_perfect_map_image:
   $\llbracket \text{Lindelof\_space } X; \text{perfect\_map } X \ Y \ f \rrbracket \implies \text{Lindelof\_space } Y$ 
using Lindelof_space_quotient_map_image_perfect_imp_quotient_map by blast

lemma Lindelof_space_perfect_map_image_eq:
   $\text{perfect\_map } X \ Y \ f \implies \text{Lindelof\_space } X \longleftrightarrow \text{Lindelof\_space } Y$ 
using Lindelof_space_perfect_map_image_Lindelof_space_proper_map_preimage_perfect_map_def by blast

end

```

6.36 Infinite Products

```

theory Infinite_Products
  imports Topology_Euclidean_Space Complex_Transcendental
begin

```

6.36.1 Preliminaries

```

lemma sum_le_prod:
  fixes  $f :: 'a \Rightarrow 'b :: \text{linordered\_semidom}$ 
  assumes  $\bigwedge x. x \in A \implies f \ x \geq 0$ 
  shows  $\text{sum } f \ A \leq (\prod x \in A. 1 + f \ x)$ 
  using assms
proof (induction A rule: infinite_finite_induct)
  case (insert x A)
  from insert.hyps have  $\text{sum } f \ A + f \ x * (\prod x \in A. 1) \leq (\prod x \in A. 1 + f \ x) + f \ x$ 
   $* (\prod x \in A. 1 + f \ x)$ 
  by (intro add_mono insert_mult_left_mono prod_mono) (auto intro: insert.prem)
  with insert.hyps show ?case by (simp add: algebra_simps)
qed simp_all

lemma prod_le_exp_sum:
  fixes  $f :: 'a \Rightarrow \text{real}$ 
  assumes  $\bigwedge x. x \in A \implies f \ x \geq 0$ 
  shows  $\text{prod } (\lambda x. 1 + f \ x) \ A \leq \text{exp } (\text{sum } f \ A)$ 
  using assms
proof (induction A rule: infinite_finite_induct)
  case (insert x A)
  have  $(1 + f \ x) * (\prod x \in A. 1 + f \ x) \leq \text{exp } (f \ x) * \text{exp } (\text{sum } f \ A)$ 
  using insert.prem by (intro mult_mono insert_prod_nonneg exp_ge_add_one_self)
  auto
  with insert.hyps show ?case by (simp add: algebra_simps exp_add)
qed simp_all

lemma lim_ln_1_plus_x_over_x_at_0:  $(\lambda x :: \text{real}. \ln (1 + x) / x) - 0 \rightarrow 1$ 
proof (rule lhopital)
  show  $(\lambda x :: \text{real}. \ln (1 + x)) - 0 \rightarrow 0$ 

```

```

  by (rule tendsto_eq_intros refl | simp)+
have eventually ( $\lambda x::\text{real}. x \in \{-1/2 <..< 1/2\}$ ) (nhds 0)
  by (rule eventually_nhds_in_open) auto
hence *: eventually ( $\lambda x::\text{real}. x \in \{-1/2 <..< 1/2\}$ ) (at 0)
  by (rule filter_leD [rotated]) (simp_all add: at_within_def)
show eventually ( $\lambda x::\text{real}. ((\lambda x. \ln (1 + x)) \text{ has\_field\_derivative inverse } (1 + x))$ ) (at x)) (at 0)
  using * by eventually_elim (auto intro!: derivative_eq_intros simp: field_simps)
show eventually ( $\lambda x::\text{real}. ((\lambda x. x) \text{ has\_field\_derivative } 1)$ ) (at x)) (at 0)
  using * by eventually_elim (auto intro!: derivative_eq_intros simp: field_simps)
show  $\forall_F x$  in at 0.  $x \neq 0$  by (auto simp: at_within_def eventually_inf_principal)
show ( $\lambda x::\text{real}. \text{inverse } (1 + x) / 1$ )  $-0 \rightarrow 1$ 
  by (rule tendsto_eq_intros refl | simp)+
qed auto

```

6.36.2 Definitions and basic properties

definition `raw_has_prod` :: $[\text{nat} \Rightarrow 'a::\{\text{t2\_space}, \text{comm\_semiring\_1}\}, \text{nat}, 'a] \Rightarrow \text{bool}$
where `raw_has_prod` f M $p \equiv (\lambda n. \prod_{i \leq n}. f (i+M)) \longrightarrow p \wedge p \neq 0$

The nonzero and zero cases, as in *Complex Analysis* by Joseph Bak and Donald J. Newman, page 241

definition

`has_prod` :: $(\text{nat} \Rightarrow 'a::\{\text{t2\_space}, \text{comm\_semiring\_1}\}) \Rightarrow 'a \Rightarrow \text{bool}$ (**infix** `has'_prod` 80)
where `f has_prod p` $\equiv \text{raw\_has\_prod } f \ 0 \ p \vee (\exists i \ q. p = 0 \wedge f \ i = 0 \wedge \text{raw\_has\_prod } f \ (\text{Suc } i) \ q)$

definition `convergent_prod` :: $(\text{nat} \Rightarrow 'a::\{\text{t2\_space}, \text{comm\_semiring\_1}\}) \Rightarrow \text{bool}$
where

`convergent_prod` $f \equiv \exists M \ p. \text{raw\_has\_prod } f \ M \ p$

definition `prodinf` :: $(\text{nat} \Rightarrow 'a::\{\text{t2\_space}, \text{comm\_semiring\_1}\}) \Rightarrow 'a$
 (**binder** $\prod$ 10)
where `prodinf` $f = (\text{THE } p. f \text{ has\_prod } p)$

lemmas `prod_defs` = `raw_has_prod_def has_prod_def convergent_prod_def prodinf_def`

lemma `has_prod_subst[trans]`: $f = g \Longrightarrow g \text{ has\_prod } z \Longrightarrow f \text{ has\_prod } z$
by `simp`

lemma `has_prod_cong`: $(\bigwedge n. f \ n = g \ n) \Longrightarrow f \text{ has\_prod } c \longleftrightarrow g \text{ has\_prod } c$
by `presburger`

lemma `raw_has_prod_nonzero` [`simp`]: $\neg \text{raw\_has\_prod } f \ M \ 0$
by (`simp` `add: raw_has_prod_def`)

```

lemma raw_has_prod_eq_0:
  fixes f :: nat  $\Rightarrow$  'a::{semidom,t2_space}
  assumes p: raw_has_prod f m p and i: f i = 0 i  $\geq$  m
  shows p = 0
proof -
  have eq0: ( $\prod_{k \leq n} f (k+m)$ ) = 0 if i - m  $\leq$  n for n
  proof -
    have  $\exists k \leq n. f (k+m) = 0$ 
    using i that by auto
    then show ?thesis
    by auto
  qed
  have ( $\lambda n. \prod_{i \leq n} f (i+m)$ )  $\longrightarrow$  0
  by (rule LIMSEQ_offset [where k = i-m]) (simp add: eq0)
  with p show ?thesis
  unfolding raw_has_prod_def
  using LIMSEQ_unique by blast
qed

lemma raw_has_prod_Suc:
  raw_has_prod f (Suc M) a  $\longleftrightarrow$  raw_has_prod ( $\lambda n. f (Suc n)$ ) M a
  unfolding raw_has_prod_def by auto

lemma has_prod_0_iff: f has_prod 0  $\longleftrightarrow$  ( $\exists i. f i = 0 \wedge (\exists p. \text{raw\_has\_prod } f (Suc i) p)$ )
  by (simp add: has_prod_def)

lemma has_prod_unique2:
  fixes f :: nat  $\Rightarrow$  'a::{semidom,t2_space}
  assumes f has_prod a f has_prod b shows a = b
  using assms
  by (auto simp: has_prod_def raw_has_prod_eq_0) (meson raw_has_prod_def
  sequentially_bot tendsto_unique)

lemma has_prod_unique:
  fixes f :: nat  $\Rightarrow$  'a :: {semidom,t2_space}
  shows f has_prod s  $\implies$  s = prodinf f
  by (simp add: has_prod_unique2 prodinf_def the_equality)

lemma convergent_prod_altdef:
  fixes f :: nat  $\Rightarrow$  'a :: {t2_space,comm_semiring_1}
  shows convergent_prod f  $\longleftrightarrow$  ( $\exists M L. (\forall n \geq M. f n \neq 0) \wedge (\lambda n. \prod_{i \leq n} f (i+M))$ 
 $\longrightarrow L \wedge L \neq 0$ )
proof
  assume convergent_prod f
  then obtain M L where *: ( $\lambda n. \prod_{i \leq n} f (i+M)$ )  $\longrightarrow L$  L  $\neq$  0
  by (auto simp: prod_defs)
  have f i  $\neq$  0 if i  $\geq$  M for i
  proof

```

```

    assume  $f\ i = 0$ 
    have **: eventually  $(\lambda n. (\prod i \leq n. f\ (i+M)) = 0)$  sequentially
      using eventually_ge_at_top[of  $i - M$ ]
    proof eventually_elim
      case (elim  $n$ )
      with  $\langle f\ i = 0 \rangle$  and  $\langle i \geq M \rangle$  show ?case
        by (auto intro!: bexI[of _  $i - M$ ] prod_zero)
    qed
    have  $(\lambda n. (\prod i \leq n. f\ (i+M))) \longrightarrow 0$ 
      unfolding filterlim_iff
      by (auto dest!: eventually_nhds_x_imp_x intro!: eventually_mono[OF **])
    from tendsto_unique[OF _ this *(1)] and *(2)
      show False by simp
    qed
    with * show  $(\exists M\ L. (\forall n \geq M. f\ n \neq 0) \wedge (\lambda n. \prod i \leq n. f\ (i+M)) \longrightarrow L \wedge L \neq 0)$ 
      by blast
    qed (auto simp: prod_defs)

```

6.36.3 Absolutely convergent products

definition $abs\_convergent\_prod :: (nat \Rightarrow _) \Rightarrow bool$ **where**
 $abs\_convergent\_prod\ f \longleftrightarrow convergent\_prod\ (\lambda i. 1 + norm\ (f\ i - 1))$

lemma $abs\_convergent\_prodI$:
assumes $convergent\ (\lambda n. \prod i \leq n. 1 + norm\ (f\ i - 1))$
shows $abs\_convergent\_prod\ f$
proof –
from $assms$ **obtain** L **where** $L: (\lambda n. \prod i \leq n. 1 + norm\ (f\ i - 1)) \longrightarrow L$
by (auto simp: convergent_def)
have $L \geq 1$
proof (rule tendsto_le)
show eventually $(\lambda n. (\prod i \leq n. 1 + norm\ (f\ i - 1)) \geq 1)$ sequentially
proof (intro always_eventually_allI)
fix n
have $(\prod i \leq n. 1 + norm\ (f\ i - 1)) \geq (\prod i \leq n. 1)$
by (intro prod_mono) auto
thus $(\prod i \leq n. 1 + norm\ (f\ i - 1)) \geq 1$ **by** simp
qed
qed (use L in simp_all)
hence $L \neq 0$ **by** auto
with L **show** ?thesis **unfolding** $abs\_convergent\_prod\_def$ $prod\_defs$
by (intro exI[of _ 0::nat] exI[of _ L]) auto
qed

lemma
fixes $f :: nat \Rightarrow 'a :: \{topological\_semigroup\_mult, t2\_space, idom\}$
assumes $convergent\_prod\ f$
shows $convergent\_prod\_imp\_convergent: \quad convergent\ (\lambda n. \prod i \leq n. f\ i)$

```

    and convergent_prod_to_zero_iff [simp]:  $(\lambda n. \prod_{i \leq n}. f\ i) \longrightarrow 0 \iff$ 
 $(\exists i. f\ i = 0)$ 
  proof -
    from assms obtain M L
      where M:  $\bigwedge n. n \geq M \implies f\ n \neq 0$  and  $(\lambda n. \prod_{i \leq n}. f\ (i + M)) \longrightarrow L$ 
    and L  $\neq 0$ 
    by (auto simp: convergent_prod_altdef)
    note this(2)
    also have  $(\lambda n. \prod_{i \leq n}. f\ (i + M)) = (\lambda n. \prod_{i=M..M+n}. f\ i)$ 
      by (intro ext prod.reindex_bij_witness[of _  $\lambda n. n - M$   $\lambda n. n + M$ ]) auto
    finally have  $(\lambda n. (\prod_{i < M}. f\ i) * (\prod_{i=M..M+n}. f\ i)) \longrightarrow (\prod_{i < M}. f\ i) * L$ 
      by (intro tendsto_mult tendsto_const)
    also have  $(\lambda n. (\prod_{i < M}. f\ i) * (\prod_{i=M..M+n}. f\ i)) = (\lambda n. (\prod_{i \in \{..<M\} \cup \{M..M+n\}}. f\ i))$ 
      by (subst prod.union_disjoint) auto
    also have  $(\lambda n. \{..<M\} \cup \{M..M+n\}) = (\lambda n. \{..n+M\})$  by auto
    finally have  $\lim: (\lambda n. \text{prod } f\ \{..n\}) \longrightarrow \text{prod } f\ \{..<M\} * L$ 
      by (rule LIMSEQ_offset)
    thus convergent  $(\lambda n. \prod_{i \leq n}. f\ i)$ 
      by (auto simp: convergent_def)

  show  $(\lambda n. \prod_{i \leq n}. f\ i) \longrightarrow 0 \iff (\exists i. f\ i = 0)$ 
  proof
    assume  $\exists i. f\ i = 0$ 
    then obtain i where  $f\ i = 0$  by auto
    moreover with M have  $i < M$  by (cases  $i < M$ ) auto
    ultimately have  $(\prod_{i < M}. f\ i) = 0$  by auto
    with  $\lim$  show  $(\lambda n. \prod_{i \leq n}. f\ i) \longrightarrow 0$  by simp
  next
    assume  $(\lambda n. \prod_{i \leq n}. f\ i) \longrightarrow 0$ 
    from tendsto_unique[OF _ this  $\lim$ ] and  $\langle L \neq 0 \rangle$ 
    show  $\exists i. f\ i = 0$  by auto
  qed
qed

lemma convergent_prod_iff_nz_lim:
  fixes f :: nat  $\Rightarrow$  'a :: {topological_semigroup_mult, t2_space, idom}
  assumes  $\bigwedge i. f\ i \neq 0$ 
  shows  $\text{convergent\_prod } f \iff (\exists L. (\lambda n. \prod_{i \leq n}. f\ i) \longrightarrow L \wedge L \neq 0)$ 
    (is ?lhs  $\iff$  ?rhs)
  proof
    assume ?lhs then show ?rhs
      using assms convergentD convergent_prod_imp_convergent convergent_prod_to_zero_iff
    by blast
  next
    assume ?rhs then show ?lhs
      unfolding prod_defs
      by (rule_tac  $x=0$  in exI) auto
  qed

```

```

lemma convergent_prod_iff_convergent:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{topological\_semigroup\_mult}, \text{t2\_space}, \text{idom}\}$ 
  assumes  $\bigwedge i. f\ i \neq 0$ 
  shows  $\text{convergent\_prod } f \longleftrightarrow \text{convergent } (\lambda n. \prod_{i \leq n}. f\ i) \wedge \text{lim } (\lambda n. \prod_{i \leq n}. f\ i) \neq 0$ 
  by (force simp: convergent_prod_iff_nz_lim assms convergent_def limI)

lemma bounded_imp_convergent_prod:
  fixes  $a :: \text{nat} \Rightarrow \text{real}$ 
  assumes  $1: \bigwedge n. a\ n \geq 1$  and bounded:  $\bigwedge n. (\prod_{i \leq n}. a\ i) \leq B$ 
  shows convergent_prod  $a$ 
proof -
  have bdd_above ( $\text{range}(\lambda n. \prod_{i \leq n}. a\ i)$ )
    by (meson bdd_aboveI2 bounded)
  moreover have incseq ( $\lambda n. \prod_{i \leq n}. a\ i$ )
    unfolding mono_def by (metis 1 prod_mono2 atMost_subset_iff dual_order.trans finite_atMost zero_le_one)
  ultimately obtain  $p$  where  $p: (\lambda n. \prod_{i \leq n}. a\ i) \longrightarrow p$ 
    using LIMSEQ_incseq_SUP by blast
  then have  $p \neq 0$ 
    by (metis 1 not_one_le_zero prod_ge_1 LIMSEQ_le_const)
  with 1  $p$  show ?thesis
    by (metis convergent_prod_iff_nz_lim not_one_le_zero)
qed

```

```

lemma abs_convergent_prod_altdef:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{one}, \text{real\_normed\_vector}\}$ 
  shows  $\text{abs\_convergent\_prod } f \longleftrightarrow \text{convergent } (\lambda n. \prod_{i \leq n}. 1 + \text{norm } (f\ i - 1))$ 
proof
  assume abs_convergent_prod  $f$ 
  thus convergent ( $\lambda n. \prod_{i \leq n}. 1 + \text{norm } (f\ i - 1)$ )
    by (auto simp: abs_convergent_prod_def intro!: convergent_prod_imp_convergent)
qed (auto intro: abs_convergent_prodI)

```

```

lemma Weierstrass_prod_ineq:
  fixes  $f :: 'a \Rightarrow \text{real}$ 
  assumes  $\bigwedge x. x \in A \implies f\ x \in \{0..1\}$ 
  shows  $1 - \text{sum } f\ A \leq (\prod_{x \in A}. 1 - f\ x)$ 
  using assms
proof (induction  $A$  rule: infinite_finite_induct)
  case (insert  $x\ A$ )
  from insert.hyps and insert.prems
    have  $1 - \text{sum } f\ A + f\ x * (\prod_{x \in A}. 1 - f\ x) \leq (\prod_{x \in A}. 1 - f\ x) + f\ x * (\prod_{x \in A}. 1)$ 
    by (intro insert.IH add_mono mult_left_mono prod_mono) auto
  with insert.hyps show ?case by (simp add: algebra_simps)

```

qed simp_all

lemma norm_prod_minus1_le_prod_minus1:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{real\_normed\_div\_algebra}, \text{comm\_ring\_1}\}$
shows $\text{norm} (\text{prod} (\lambda n. 1 + f n) A - 1) \leq \text{prod} (\lambda n. 1 + \text{norm} (f n)) A - 1$
proof (induction A rule: infinite_finite_induct)
case (insert x A)
from insert.hyps **have**
 $\text{norm} ((\prod n \in \text{insert } x A. 1 + f n) - 1) =$
 $\text{norm} ((\prod n \in A. 1 + f n) - 1 + f x * (\prod n \in A. 1 + f n))$
by (simp add: algebra_simps)
also have $\dots \leq \text{norm} ((\prod n \in A. 1 + f n) - 1) + \text{norm} (f x * (\prod n \in A. 1 + f n))$
by (rule norm_triangle_ineq)
also have $\text{norm} (f x * (\prod n \in A. 1 + f n)) = \text{norm} (f x) * (\prod x \in A. \text{norm} (1 + f x))$
by (simp add: prod_norm norm_mult)
also have $(\prod x \in A. \text{norm} (1 + f x)) \leq (\prod x \in A. \text{norm} (1::'a) + \text{norm} (f x))$
by (intro prod_mono norm_triangle_ineq ballI conjI) auto
also have $\text{norm} (1::'a) = 1$ **by** simp
also note insert.IH
also have $(\prod n \in A. 1 + \text{norm} (f n)) - 1 + \text{norm} (f x) * (\prod x \in A. 1 + \text{norm} (f x)) =$
 $(\prod n \in \text{insert } x A. 1 + \text{norm} (f n)) - 1$
using insert.hyps **by** (simp add: algebra_simps)
finally show ?case **by** - (simp_all add: mult_left_mono)
qed simp_all

lemma convergent_prod_imp_ev_nonzero:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{t2\_space}, \text{comm\_semiring\_1}\}$
assumes convergent_prod f
shows eventually $(\lambda n. f n \neq 0)$ sequentially
using assms **by** (auto simp: eventually_at_top_linorder convergent_prod_altdef)

lemma convergent_prod_imp_LIMSEQ:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{real\_normed\_field}\}$
assumes convergent_prod f
shows $f \longrightarrow 1$
proof -
from assms **obtain** M L **where** $L: (\lambda n. \prod i \leq n. f (i+M)) \longrightarrow L \wedge n. n \geq M \implies f n \neq 0 \wedge L \neq 0$
by (auto simp: convergent_prod_altdef)
hence $L': (\lambda n. \prod i \leq \text{Suc } n. f (i+M)) \longrightarrow L$ **by** (subst filterlim_sequentially_Suc)
have $(\lambda n. (\prod i \leq \text{Suc } n. f (i+M)) / (\prod i \leq n. f (i+M))) \longrightarrow L / L$
using L L' **by** (intro tendsto_divide) simp_all
also from L **have** $L / L = 1$ **by** simp
also have $(\lambda n. (\prod i \leq \text{Suc } n. f (i+M)) / (\prod i \leq n. f (i+M))) = (\lambda n. f (n + \text{Suc } M))$
using assms L **by** (auto simp: fun_eq_iff atMost_Suc)

finally show ?thesis by (rule LIMSEQ_offset)
qed

lemma abs_convergent_prod_imp_summable:
fixes f :: nat $\Rightarrow$ 'a :: real_normed_div_algebra
assumes abs_convergent_prod f
shows summable ($\lambda i. \text{norm } (f\ i - 1)$)
proof -
from assms have convergent ($\lambda n. \prod_{i \leq n}. 1 + \text{norm } (f\ i - 1)$)
unfolding abs_convergent_prod_def by (rule convergent_prod_imp_convergent)
then obtain L where L: ($\lambda n. \prod_{i \leq n}. 1 + \text{norm } (f\ i - 1)$) $\longrightarrow$ L
unfolding convergent_def by blast
have convergent ($\lambda n. \sum_{i \leq n}. \text{norm } (f\ i - 1)$)
proof (rule Bseq_monoseq_convergent)
have eventually ($\lambda n. (\prod_{i \leq n}. 1 + \text{norm } (f\ i - 1)) < L + 1$) sequentially
using L(1) by (rule order_tendstoD) simp_all
hence $\forall_F x$ in sequentially. $\text{norm } (\sum_{i \leq x}. \text{norm } (f\ i - 1)) \leq L + 1$
proof eventually_elim
case (elim n)
have $\text{norm } (\sum_{i \leq n}. \text{norm } (f\ i - 1)) = (\sum_{i \leq n}. \text{norm } (f\ i - 1))$
unfolding real_norm_def by (intro abs_of_nonneg sum_nonneg) simp_all
also have $\dots \leq (\prod_{i \leq n}. 1 + \text{norm } (f\ i - 1))$ by (rule sum_le_prod) auto
also have $\dots < L + 1$ by (rule elim)
finally show ?case by simp
qed
thus Bseq ($\lambda n. \sum_{i \leq n}. \text{norm } (f\ i - 1)$) by (rule BfunI)
next
show monoseq ($\lambda n. \sum_{i \leq n}. \text{norm } (f\ i - 1)$)
by (rule mono_SucI1) auto
qed
thus summable ($\lambda i. \text{norm } (f\ i - 1)$) by (simp add: summable_iff_convergent')
qed

lemma summable_imp_abs_convergent_prod:
fixes f :: nat $\Rightarrow$ 'a :: real_normed_div_algebra
assumes summable ($\lambda i. \text{norm } (f\ i - 1)$)
shows abs_convergent_prod f
proof (intro abs_convergent_prodI Bseq_monoseq_convergent)
show monoseq ($\lambda n. \prod_{i \leq n}. 1 + \text{norm } (f\ i - 1)$)
by (intro mono_SucI1)
(auto simp: atMost_Suc algebra_simps intro!: mult_nonneg_nonneg prod_nonneg)
next
show Bseq ($\lambda n. \prod_{i \leq n}. 1 + \text{norm } (f\ i - 1)$)
proof (rule Bseq_eventually_mono)
show eventually ($\lambda n. \text{norm } (\prod_{i \leq n}. 1 + \text{norm } (f\ i - 1)) \leq$
 $\text{norm } (\exp (\sum_{i \leq n}. \text{norm } (f\ i - 1)))$) sequentially
by (intro always_eventually allI) (auto simp: abs_prod exp_sum intro!:
prod_mono)
next


```

from assms have  $(\lambda n. \sum_{i \leq n}. \text{norm } (f\ i - 1)) \longrightarrow (\sum i. \text{norm } (f\ i - 1))$ 
  using sums_def le by blast
hence  $(\lambda n. \exp (\sum_{i \leq n}. \text{norm } (f\ i - 1))) \longrightarrow \exp (\sum i. \text{norm } (f\ i - 1))$ 
  by (rule tendsto_exp)
hence convergent  $(\lambda n. \exp (\sum_{i \leq n}. \text{norm } (f\ i - 1)))$ 
  by (rule convergentI)
thus Bseq  $(\lambda n. \exp (\sum_{i \leq n}. \text{norm } (f\ i - 1)))$ 
  by (rule convergent_imp_Bseq)
qed
qed

```

```

theorem abs_convergent_prod_conv_summable:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \text{real\_normed\_div\_algebra}$ 
  shows  $\text{abs\_convergent\_prod } f \longleftrightarrow \text{summable } (\lambda i. \text{norm } (f\ i - 1))$ 
  by (blast intro: abs_convergent_prod_imp_summable summable_imp_abs_convergent_prod)

```

```

lemma abs_convergent_prod_imp_LIMSEQ:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{comm\_ring\_1}, \text{real\_normed\_div\_algebra}\}$ 
  assumes abs_convergent_prod  $f$ 
  shows  $f \longrightarrow 1$ 
proof -
  from assms have summable  $(\lambda n. \text{norm } (f\ n - 1))$ 
    by (rule abs_convergent_prod_imp_summable)
  from summable_LIMSEQ_zero[OF this] have  $(\lambda n. f\ n - 1) \longrightarrow 0$ 
    by (simp add: tendsto_norm_zero_iff)
  from tendsto_add[OF this tendsto_const[of 1]] show ?thesis by simp
qed

```

```

lemma abs_convergent_prod_imp_ev_nonzero:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{comm\_ring\_1}, \text{real\_normed\_div\_algebra}\}$ 
  assumes abs_convergent_prod  $f$ 
  shows eventually  $(\lambda n. f\ n \neq 0)$  sequentially
proof -
  from assms have  $f \longrightarrow 1$ 
    by (rule abs_convergent_prod_imp_LIMSEQ)
  hence eventually  $(\lambda n. \text{dist } (f\ n) 1 < 1)$  at_top
    by (auto simp: tendsto_iff)
  thus ?thesis by eventually_elim auto
qed

```

6.36.4 Ignoring initial segments

```

lemma convergent_prod_offset:
  assumes convergent_prod  $(\lambda n. f\ (n + m))$ 
  shows convergent_prod  $f$ 
proof -
  from assms obtain  $M\ L$  where  $(\lambda n. \prod_{k \leq n}. f\ (k + (M + m))) \longrightarrow L\ L \neq 0$ 
    by (auto simp: prod_defs add.assoc)

```

```

    thus convergent_prod f
    unfolding prod_defs by blast
qed

```

```

lemma abs_convergent_prod_offset:
  assumes abs_convergent_prod (λn. f (n + m))
  shows abs_convergent_prod f
  using assms unfolding abs_convergent_prod_def by (rule convergent_prod_offset)

```

```

lemma raw_has_prod_ignore_initial_segment:
  fixes f :: nat ⇒ 'a :: real_normed_field
  assumes raw_has_prod f M p N ≥ M
  obtains q where raw_has_prod f N q
proof -
  have p: (λn. ∏ k≤n. f (k + M)) ⟶ p and p ≠ 0
    using assms by (auto simp: raw_has_prod_def)
  then have nz: ∧ n. n ≥ M ⟹ f n ≠ 0
    using assms by (auto simp: raw_has_prod_eq_0)
  define C where C = (∏ k<N-M. f (k + M))
  from nz have [simp]: C ≠ 0
    by (auto simp: C_def)

  from p have (λi. ∏ k≤i + (N-M). f (k + M)) ⟶ p
    by (rule LIMSEQ_ignore_initial_segment)
  also have (λi. ∏ k≤i + (N-M). f (k + M)) = (λn. C * (∏ k≤n. f (k + N)))
  proof (rule ext, goal_cases)
    case (1 n)
    have {..n+(N-M)} = {.. $(N-M)$ } ∪ {(N-M)..n+(N-M)} by auto
    also have (∏ k∈... f (k + M)) = C * (∏ k=(N-M)..n+(N-M). f (k + M))
    unfolding C_def by (rule prod.union_disjoint) auto
    also have (∏ k=(N-M)..n+(N-M). f (k + M)) = (∏ k≤n. f (k + (N-M) + M))
    by (intro ext prod.reindex_bij_witness[of _ λk. k + (N-M) λk. k - (N-M)])
    auto
  finally show ?case
    using ⟨N ≥ M⟩ by (simp add: add_ac)
  qed
  finally have (λn. C * (∏ k≤n. f (k + N)) / C) ⟶ p / C
    by (intro tendsto_divide tendsto_const) auto
  hence (λn. ∏ k≤n. f (k + N)) ⟶ p / C by simp
  moreover from ⟨p ≠ 0⟩ have p / C ≠ 0 by simp
  ultimately show ?thesis
    using raw_has_prod_def that by blast
qed

```

```

corollary convergent_prod_ignore_initial_segment:
  fixes f :: nat ⇒ 'a :: real_normed_field

```

```

assumes convergent_prod f
shows convergent_prod ( $\lambda n. f (n + m)$ )
using assms
unfolding convergent_prod_def
apply clarify
apply (erule_tac  $N=M+m$  in raw_has_prod_ignore_initial_segment)
apply (auto simp add: raw_has_prod_def add_ac)
done

```

```

corollary convergent_prod_ignore_nonzero_segment:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \text{real\_normed\_field}$ 
  assumes  $f: \text{convergent\_prod } f$  and  $\text{nz}: \bigwedge i. i \geq M \implies f\ i \neq 0$ 
  shows  $\exists p. \text{raw\_has\_prod } f\ M\ p$ 
  using convergent_prod_ignore_initial_segment [OF f]
  by (metis convergent_LIMSEQ_iff convergent_prod_iff convergent_le_add_same_cancel2
    nz_prod_defs(1) zero_order(1))

```

```

corollary abs_convergent_prod_ignore_initial_segment:
  assumes abs_convergent_prod f
  shows abs_convergent_prod ( $\lambda n. f (n + m)$ )
  using assms unfolding abs_convergent_prod_def
  by (rule convergent_prod_ignore_initial_segment)

```

6.36.5 More elementary properties

```

theorem abs_convergent_prod_imp_convergent_prod:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{real\_normed\_div\_algebra}, \text{complete\_space}, \text{comm\_ring } 1\}$ 
  assumes abs_convergent_prod f
  shows convergent_prod f

```

proof –

```

  from assms have eventually ( $\lambda n. f\ n \neq 0$ ) sequentially
    by (rule abs_convergent_prod_imp_ev_nonzero)
  then obtain N where  $N: f\ n \neq 0$  if  $n \geq N$  for  $n$ 
    by (auto simp: eventually_at_top_linorder)
  let  $?P = \lambda n. \prod i \leq n. f\ (i + N)$  and  $?Q = \lambda n. \prod i \leq n. 1 + \text{norm } (f\ (i + N) -$ 
     $1)$ 

```

have *Cauchy* $?P$

proof (*rule* *CauchyI'*, *goal_cases*)

case ($1\ \varepsilon$)

from *assms* **have** *abs_convergent_prod* ($\lambda n. f\ (n + N)$)

by (*rule* *abs_convergent_prod_ignore_initial_segment*)

hence *Cauchy* $?Q$

unfolding *abs_convergent_prod_def*

by (*intro* *convergent_Cauchy* *convergent_prod_imp_convergent*)

from *CauchyD* [*OF this* 1] **obtain** *M* **where** $M: \text{norm } (?Q\ m - ?Q\ n) < \varepsilon$ **if** $m \geq M\ n \geq M$ **for** $m\ n$

by *blast*

show *?case*

```

proof (rule exI[of _ M], safe, goal_cases)
  case (1 m n)
  have dist (?P m) (?P n) = norm (?P n - ?P m)
    by (simp add: dist_norm norm_minus_commute)
  also from 1 have {..n} = {..m} ∪ {m<..n} by auto
  hence norm (?P n - ?P m) = norm (?P m * (∏ k∈{m<..n}. f (k + N)) -
    ?P m)
    by (subst prod.union_disjoint [symmetric]) (auto simp: algebra_simps)
  also have ... = norm (?P m * ((∏ k∈{m<..n}. f (k + N)) - 1))
    by (simp add: algebra_simps)
  also have ... = (∏ k≤m. norm (f (k + N))) * norm ((∏ k∈{m<..n}. f (k
    + N)) - 1)
    by (simp add: norm_mult prod_norm)
  also have ... ≤ ?Q m * ((∏ k∈{m<..n}. 1 + norm (f (k + N) - 1)) - 1)
    using norm_prod_minus1_le_prod_minus1[of λk. f (k + N) - 1 {m<..n}]
      norm_triangle_ineq[of 1 f k - 1 for k]
    by (intro mult_mono prod_mono ballI conjI norm_prod_minus1_le_prod_minus1
    prod_nonneg) auto
  also have ... = ?Q m * (∏ k∈{m<..n}. 1 + norm (f (k + N) - 1)) - ?Q
    m
    by (simp add: algebra_simps)
  also have ?Q m * (∏ k∈{m<..n}. 1 + norm (f (k + N) - 1)) =
    (∏ k∈{..m}∪{m<..n}. 1 + norm (f (k + N) - 1))
    by (rule prod.union_disjoint [symmetric]) auto
  also from 1 have {..m}∪{m<..n} = {..n} by auto
  also have ?Q n - ?Q m ≤ norm (?Q n - ?Q m) by simp
  also from 1 have ... < ε by (intro M) auto
  finally show ?case .
qed
qed
hence conv: convergent ?P by (rule Cauchy_convergent)
then obtain L where L: ?P ⟶ L
  by (auto simp: convergent_def)

have L ≠ 0
proof
  assume [simp]: L = 0
  from tendsto_norm[OF L] have limit: (λn. ∏ k≤n. norm (f (k + N))) ⟶ 0
    by (simp add: prod_norm)

  from assms have (λn. f (n + N)) ⟶ 1
    by (intro abs_convergent_prod_imp_LIMSEQ abs_convergent_prod_ignore_initial_segment)
  hence eventually (λn. norm (f (n + N) - 1) < 1) sequentially
    by (auto simp: tendsto_iff dist_norm)
  then obtain M0 where M0: norm (f (n + N) - 1) < 1 if n ≥ M0 for n
    by (auto simp: eventually_at_top_linorder)

  {

```

```

fix M assume M: M ≥ M0
with M0 have M: norm (f (n + N) - 1) < 1 if n ≥ M for n using that
by simp

have (λn. ∏ k≤n. 1 - norm (f (k+M+N) - 1)) ⟶ 0
proof (rule tendsto_sandwich)
  show eventually (λn. (∏ k≤n. 1 - norm (f (k+M+N) - 1)) ≥ 0)
sequentially
  using M by (intro always_eventually prod_nonneg allI ballI) (auto intro:
less_imp_le)
  have norm (1::'a) - norm (f (i + M + N) - 1) ≤ norm (f (i + M +
N)) for i
  using norm_triangle_ineq3[of f (i + M + N) 1] by simp
  thus eventually (λn. (∏ k≤n. 1 - norm (f (k+M+N) - 1)) ≤ (∏ k≤n.
norm (f (k+M+N)))) at_top
  using M by (intro always_eventually allI prod_mono ballI conjI) (auto
intro: less_imp_le)

define C where C = (∏ k<M. norm (f (k + N)))
from N have [simp]: C ≠ 0 by (auto simp: C_def)
from L have (λn. norm (∏ k≤n+M. f (k + N))) ⟶ 0
by (intro LIMSEQ_ignore_initial_segment) (simp add: tendsto_norm_zero_iff)
also have (λn. norm (∏ k≤n+M. f (k + N))) = (λn. C * (∏ k≤n. norm
(f (k + M + N))))
proof (rule ext, goal_cases)
  case (1 n)
  have {..n+M} = {..<M} ∪ {M..n+M} by auto
  also have norm (∏ k∈... f (k + N)) = C * norm (∏ k=M..n+M. f (k
+ N))
  unfolding C_def by (subst prod.union_disjoint) (auto simp: norm_mult
prod_norm)
  also have (∏ k=M..n+M. f (k + N)) = (∏ k≤n. f (k + N + M))
  by (intro prod.reindex_bij_witness[of _ λi. i + M λi. i - M]) auto
  finally show ?case by (simp add: add_ac prod_norm)
qed
finally have (λn. C * (∏ k≤n. norm (f (k + M + N))) / C) ⟶ 0 /
C
  by (intro tendsto_divide tendsto_const) auto
thus (λn. ∏ k≤n. norm (f (k + M + N))) ⟶ 0 by simp
qed simp_all

have 1 - (∑ i. norm (f (i + M + N) - 1)) ≤ 0
proof (rule tendsto_le)
  show eventually (λn. 1 - (∑ k≤n. norm (f (k+M+N) - 1)) ≤
(∏ k≤n. 1 - norm (f (k+M+N) - 1))) at_top
  using M by (intro always_eventually allI Weierstrass_prod_ineq) (auto
intro: less_imp_le)
  show (λn. ∏ k≤n. 1 - norm (f (k+M+N) - 1)) ⟶ 0 by fact
  show (λn. 1 - (∑ k≤n. norm (f (k + M + N) - 1)))

```

```

       $\longrightarrow 1 - (\sum i. \text{norm } (f (i + M + N) - 1))$ 
    by (intro tendsto_intros summable_LIMSEQ' summable_ignore_initial_segment

      abs_convergent_prod_imp_summable assms)
  qed simp_all
  hence  $(\sum i. \text{norm } (f (i + M + N) - 1)) \geq 1$  by simp
  also have  $\dots + (\sum i < M. \text{norm } (f (i + N) - 1)) = (\sum i. \text{norm } (f (i + N) - 1))$ 
- 1))
  by (intro suminf_split_initial_segment [symmetric] summable_ignore_initial_segment
      abs_convergent_prod_imp_summable assms)
  finally have  $1 + (\sum i < M. \text{norm } (f (i + N) - 1)) \leq (\sum i. \text{norm } (f (i + N) - 1))$  by simp
} note * = this

  have  $1 + (\sum i. \text{norm } (f (i + N) - 1)) \leq (\sum i. \text{norm } (f (i + N) - 1))$ 
  proof (rule tendsto_le)
    show  $(\lambda M. 1 + (\sum i < M. \text{norm } (f (i + N) - 1))) \longrightarrow 1 + (\sum i. \text{norm } (f (i + N) - 1))$ 
    by (intro tendsto_intros summable_LIMSEQ summable_ignore_initial_segment

      abs_convergent_prod_imp_summable assms)
    show eventually  $(\lambda M. 1 + (\sum i < M. \text{norm } (f (i + N) - 1)) \leq (\sum i. \text{norm } (f (i + N) - 1)))$  at_top
    using eventually_ge_at_top[of M0] by eventually_elim (use * in auto)
  qed simp_all
  thus False by simp
qed
with L show ?thesis by (auto simp: prod_defs)
qed

lemma raw_has_prod_cases:
  fixes f :: nat  $\Rightarrow$  'a :: {idom, topological_semigroup_mult, t2_space}
  assumes raw_has_prod f M p
  obtains i where  $i < M$  f i = 0 | p where raw_has_prod f 0 p
proof -
  have  $(\lambda n. \prod i \leq n. f (i + M)) \longrightarrow p$  p  $\neq 0$ 
  using assms unfolding raw_has_prod_def by blast+
  then have  $(\lambda n. \text{prod } f \{..<M\} * (\prod i \leq n. f (i + M))) \longrightarrow \text{prod } f \{..<M\} * p$ 
  by (metis tendsto_mult_left)
  moreover have  $\text{prod } f \{..<M\} * (\prod i \leq n. f (i + M)) = \text{prod } f \{..n+M\}$  for n
  proof -
    have  $\{..n+M\} = \{..<M\} \cup \{M..n+M\}$ 
    by auto
    then have  $\text{prod } f \{..n+M\} = \text{prod } f \{..<M\} * \text{prod } f \{M..n+M\}$ 
    by simp (subst prod.union_disjoint; force)
    also have  $\dots = \text{prod } f \{..<M\} * (\prod i \leq n. f (i + M))$ 
    by (metis (mono_tags, lifting) add.left_neutral atMost_atLeast0 prod.shift_bounds_cl_nat_ivl)
    finally show ?thesis by metis

```

```

qed
ultimately have  $(\lambda n. \text{prod } f \ \{..n\}) \longrightarrow \text{prod } f \ \{..<M\} * p$ 
  by (auto intro: LIMSEQ_offset [where k=M])
then have raw_has_prod  $f \ 0 \ (\text{prod } f \ \{..<M\} * p)$  if  $\forall i < M. f \ i \neq 0$ 
  using  $\langle p \neq 0 \rangle$  assms that by (auto simp: raw_has_prod_def)
then show thesis
  using that by blast
qed

```

```

corollary convergent_prod_offset_0:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{idom}, \text{topological\_semigroup\_mult}, \text{t2\_space}\}$ 
  assumes convergent_prod  $f \ \bigwedge i. f \ i \neq 0$ 
  shows  $\exists p. \text{raw\_has\_prod } f \ 0 \ p$ 
  using assms convergent_prod_def raw_has_prod_cases by blast

```

```

lemma prodinf_eq_lim:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{idom}, \text{topological\_semigroup\_mult}, \text{t2\_space}\}$ 
  assumes convergent_prod  $f \ \bigwedge i. f \ i \neq 0$ 
  shows  $\text{prodinf } f = \lim (\lambda n. \prod_{i \leq n} f \ i)$ 
  using assms convergent_prod_offset_0 [OF assms]
  by (simp add: prod_defs lim_def) (metis (no_types) assms(1) convergent_prod_to_zero_iff)

```

```

lemma has_prod_one[simp, intro]:  $(\lambda n. 1) \text{ has\_prod } 1$ 
  unfolding prod_defs by auto

```

```

lemma convergent_prod_one[simp, intro]: convergent_prod  $(\lambda n. 1)$ 
  unfolding prod_defs by auto

```

```

lemma prodinf_cong:  $(\bigwedge n. f \ n = g \ n) \implies \text{prodinf } f = \text{prodinf } g$ 
  by presburger

```

```

lemma convergent_prod_cong:
  fixes  $f \ g :: \text{nat} \Rightarrow 'a :: \{\text{field}, \text{topological\_semigroup\_mult}, \text{t2\_space}\}$ 
  assumes ev: eventually  $(\lambda x. f \ x = g \ x)$  sequentially and  $f: \bigwedge i. f \ i \neq 0$  and  $g: \bigwedge i. g \ i \neq 0$ 
  shows convergent_prod  $f = \text{convergent\_prod } g$ 
proof -
  from assms obtain  $N$  where  $N: \forall n \geq N. f \ n = g \ n$ 
    by (auto simp: eventually_at_top_linorder)
  define  $C$  where  $C = (\prod_{k < N} f \ k / g \ k)$ 
  with  $g$  have  $C \neq 0$ 
    by (simp add: f)
  have *: eventually  $(\lambda n. \text{prod } f \ \{..n\} = C * \text{prod } g \ \{..n\})$  sequentially
    using eventually_ge_at_top[of  $N$ ]
  proof eventually_elim
    case (elim  $n$ )
    then have  $\{..n\} = \{..<N\} \cup \{N..n\}$ 
      by auto
    also have  $\text{prod } f \ \dots = \text{prod } f \ \{..<N\} * \text{prod } f \ \{N..n\}$ 

```

```

    by (intro prod.union_disjoint) auto
  also from N have prod f {N..n} = prod g {N..n}
    by (intro prod.cong) simp_all
  also have prod f {.. $N$ } * prod g {N..n} =  $C$  * (prod g {.. $N$ } * prod g
{N..n})
    unfolding C_def by (simp add: g prod_dividef)
  also have prod g {.. $N$ } * prod g {N..n} = prod g ({.. $N$ }  $\cup$  {N..n})
    by (intro prod.union_disjoint [symmetric]) auto
  also from elim have {.. $N$ }  $\cup$  {N..n} = {..n}
    by auto
  finally show prod f {..n} =  $C$  * prod g {..n} .
qed
then have cong: convergent ( $\lambda n$ . prod f {..n}) = convergent ( $\lambda n$ .  $C$  * prod g
{..n})
  by (rule convergent_cong)
show ?thesis
proof
  assume cf: convergent_prod f
  with f have  $\neg (\lambda n$ . prod f {..n})  $\longrightarrow$  0
    by simp
  then have  $\neg (\lambda n$ . prod g {..n})  $\longrightarrow$  0
    using *  $\langle C \neq 0 \rangle$  filterlim_cong by fastforce
  then show convergent_prod g
    by (metis convergent_mult_const_iff  $\langle C \neq 0 \rangle$  cong cf convergent_LIMSEQ_iff
convergent_prod_iff convergent convergent_prod_imp_convergent g)
  next
    assume cg: convergent_prod g
    have  $\exists a$ .  $C * a \neq 0 \wedge (\lambda n$ . prod g {..n})  $\longrightarrow$  a
      by (metis (no_types)  $\langle C \neq 0 \rangle$  cg convergent_prod_iff_nz_lim divide_eq_0_iff
g nonzero_mult_div_cancel_right)
    then show convergent_prod f
      using * tendsto_mult_left filterlim_cong
      by (fastforce simp add: convergent_prod_iff_nz_lim f)
  qed
qed

```

lemma has_prod_finite:

```

  fixes f :: nat  $\Rightarrow$  'a:: {semidom,t2_space}
  assumes [simp]: finite N
  and f:  $\bigwedge n$ .  $n \notin N \implies f\ n = 1$ 
  shows f has_prod ( $\prod_{n \in N}$  f n)
proof -
  have eq: prod f {..n + Suc (Max N)} = prod f N for n
  proof (rule prod.mono_neutral_right)
    show  $N \subseteq \{..n + \text{Suc} (\text{Max } N)\}$ 
      by (auto simp: le_Suc_eq trans_le_add2)
    show  $\forall i \in \{..n + \text{Suc} (\text{Max } N)\} - N$ . f i = 1
      using f by blast
  qed auto

```



```

show ?thesis
proof (cases  $\forall n \in N. f\ n \neq 0$ )
  case True
  then have  $\text{prod } f\ N \neq 0$ 
  by simp
  moreover have  $(\lambda n. \text{prod } f\ \{..n\}) \longrightarrow \text{prod } f\ N$ 
  by (rule LIMSEQ_offset[of _ Suc (Max N)]) (simp add: eq atLeast0LessThan
del: add_Suc_right)
  ultimately show ?thesis
  by (simp add: raw_has_prod_def has_prod_def)
next
  case False
  then obtain  $k$  where  $k \in N$  and  $f\ k = 0$ 
  by auto
  let ?Z =  $\{n \in N. f\ n = 0\}$ 
  have maxge:  $\text{Max } ?Z \geq n$  if  $f\ n = 0$  for  $n$ 
  using Max_ge [of ?Z]  $\langle \text{finite } N \rangle$   $\langle f\ n = 0 \rangle$ 
  by (metis (mono_tags) Collect_mem_eq f_finite_Collect_conjI mem_Collect_eq
zero_neq_one)
  let ?q =  $\text{prod } f\ \{\text{Suc } (\text{Max } ?Z).. \text{Max } N\}$ 
  have [simp]:  $?q \neq 0$ 
  using maxge Suc_n_not_le_n le_trans by force
  have eq:  $(\prod_{i \leq n + \text{Max } N} f\ (\text{Suc } (i + \text{Max } ?Z))) = ?q$  for  $n$ 
  proof -
    have  $(\prod_{i \leq n + \text{Max } N} f\ (\text{Suc } (i + \text{Max } ?Z))) = \text{prod } f\ \{\text{Suc } (\text{Max } ?Z)..n + \text{Max } N + \text{Suc } (\text{Max } ?Z)\}$ 
    proof (rule prod.reindex_cong [where  $l = \lambda i. i + \text{Suc } (\text{Max } ?Z)$ , THEN sym])
      show  $\{\text{Suc } (\text{Max } ?Z)..n + \text{Max } N + \text{Suc } (\text{Max } ?Z)\} = (\lambda i. i + \text{Suc } (\text{Max } ?Z))\ \{..n + \text{Max } N\}$ 
      using le_Suc_ex by fastforce
    qed (auto simp: inj_on_def)
    also have  $\dots = ?q$ 
    by (rule prod.mono_neutral_right)
      (use Max.coboundedI [OF  $\langle \text{finite } N \rangle$ ] f in  $\langle \text{force}+ \rangle$ )
  finally show ?thesis .
qed
have  $q: \text{raw\_has\_prod } f\ (\text{Suc } (\text{Max } ?Z))\ ?q$ 
proof (simp add: raw_has_prod_def)
  show  $(\lambda n. \prod_{i \leq n} f\ (\text{Suc } (i + \text{Max } ?Z))) \longrightarrow ?q$ 
  by (rule LIMSEQ_offset[of _ (Max N)]) (simp add: eq)
qed
show ?thesis
  unfolding has_prod_def
proof (intro disjI2 exI conjI)
  show  $\text{prod } f\ N = 0$ 
  using  $\langle f\ k = 0 \rangle$   $\langle k \in N \rangle$   $\langle \text{finite } N \rangle$  prod_zero by blast
  show  $f\ (\text{Max } ?Z) = 0$ 
  using Max_in [of ?Z]  $\langle \text{finite } N \rangle$   $\langle f\ k = 0 \rangle$   $\langle k \in N \rangle$  by auto

```

2750

```

      qed (use q in auto)
    qed
  qed

```

```

corollary has_prod_0:
  fixes f :: nat  $\Rightarrow$  'a::{semidom,t2_space}
  assumes  $\bigwedge n. f\ n = 1$ 
  shows f has_prod 1
  by (simp add: assms has_prod_cong)

```

```

lemma prodinf_zero[simp]: prodinf ( $\lambda n. 1::'a::real\_normed\_field$ ) = 1
  using has_prod_unique by force

```

```

lemma convergent_prod_finite:
  fixes f :: nat  $\Rightarrow$  'a::{idom,t2_space}
  assumes finite N  $\bigwedge n. n \notin N \implies f\ n = 1$ 
  shows convergent_prod f
proof -
  have  $\exists n\ p. \text{raw\_has\_prod}\ f\ n\ p$ 
    using assms has_prod_def has_prod_finite by blast
  then show ?thesis
    by (simp add: convergent_prod_def)
qed

```

```

lemma has_prod_If_finite_set:
  fixes f :: nat  $\Rightarrow$  'a::{idom,t2_space}
  shows finite A  $\implies (\lambda r. \text{if } r \in A \text{ then } f\ r \text{ else } 1) \text{ has\_prod } (\prod_{r \in A} f\ r)$ 
  using has_prod_finite[of A ( $\lambda r. \text{if } r \in A \text{ then } f\ r \text{ else } 1$ )]
  by simp

```

```

lemma has_prod_If_finite:
  fixes f :: nat  $\Rightarrow$  'a::{idom,t2_space}
  shows finite {r. P r}  $\implies (\lambda r. \text{if } P\ r \text{ then } f\ r \text{ else } 1) \text{ has\_prod } (\prod_{r \mid P\ r} f\ r)$ 
  using has_prod_If_finite_set[of {r. P r}] by simp

```

```

lemma convergent_prod_If_finite_set[simp, intro]:
  fixes f :: nat  $\Rightarrow$  'a::{idom,t2_space}
  shows finite A  $\implies \text{convergent\_prod } (\lambda r. \text{if } r \in A \text{ then } f\ r \text{ else } 1)$ 
  by (simp add: convergent_prod_finite)

```

```

lemma convergent_prod_If_finite[simp, intro]:
  fixes f :: nat  $\Rightarrow$  'a::{idom,t2_space}
  shows finite {r. P r}  $\implies \text{convergent\_prod } (\lambda r. \text{if } P\ r \text{ then } f\ r \text{ else } 1)$ 
  using convergent_prod_def has_prod_If_finite has_prod_def by fastforce

```

```

lemma has_prod_single:
  fixes f :: nat  $\Rightarrow$  'a::{idom,t2_space}
  shows ( $\lambda r. \text{if } r = i \text{ then } f\ r \text{ else } 1$ ) has_prod f i
  using has_prod_If_finite[of  $\lambda r. r = i$ ] by simp

```

```

context
  fixes  $f :: \text{nat} \Rightarrow 'a :: \text{real\_normed\_field}$ 
begin

lemma convergent_prod_imp_has_prod:
  assumes convergent_prod  $f$ 
  shows  $\exists p. f \text{ has\_prod } p$ 
proof -
  obtain  $M$   $p$  where  $p: \text{raw\_has\_prod } f \ M \ p$ 
    using assms convergent_prod_def by blast
  then have  $p \neq 0$ 
    using raw_has_prod_nonzero by blast
  with  $p$  have  $\text{fnz}: f \ i \neq 0 \text{ if } i \geq M \text{ for } i$ 
    using raw_has_prod_eq_0 that by blast
  define  $C$  where  $C = (\prod_{n < M}. f \ n)$ 
  show ?thesis
  proof (cases  $\forall n \leq M. f \ n \neq 0$ )
    case True
    then have  $C \neq 0$ 
      by (simp add: C_def)
    then show ?thesis
      by (meson True assms convergent_prod_offset_0 fnz has_prod_def nat_le_linear)
  next
    case False
    let  $?N = \text{GREATEST } n. f \ n = 0$ 
    have  $0: f \ ?N = 0$ 
      using fnz False
      by (metis (mono_tags, lifting) GreatestI_ex_nat nat_le_linear)
    have  $f \ i \neq 0 \text{ if } i > ?N \text{ for } i$ 
      by (metis (mono_tags, lifting) Greatest_le_nat fnz leD linear that)
    then have  $\exists p. \text{raw\_has\_prod } f \ (\text{Suc } ?N) \ p$ 
      using assms by (auto simp: intro!: convergent_prod_ignore_nonzero_segment)
    then show ?thesis
      unfolding has_prod_def using  $0$  by blast
  qed
qed

lemma convergent_prod_has_prod [intro]:
  shows convergent_prod  $f \implies f \text{ has\_prod } (\text{prodinf } f)$ 
  unfolding prodinf_def
  by (metis convergent_prod_imp_has_prod has_prod_unique theI')

lemma convergent_prod_LIMSEQ:
  shows convergent_prod  $f \implies (\lambda n. \prod_{i \leq n}. f \ i) \longrightarrow \text{prodinf } f$ 
  by (metis convergent_LIMSEQ_iff convergent_prod_has_prod convergent_prod_imp_convergent

    convergent_prod_to_zero_iff raw_has_prod_eq_0 has_prod_def prodinf_eq_lim
    zero_le)

```

```

theorem has_prod_iff:  $f \text{ has\_prod } x \longleftrightarrow \text{convergent\_prod } f \wedge \text{prodinf } f = x$ 
proof
  assume  $f \text{ has\_prod } x$ 
  then show  $\text{convergent\_prod } f \wedge \text{prodinf } f = x$ 
    apply safe
    using convergent_prod_def has_prod_def apply blast
    using has_prod_unique by blast
qed auto

```

```

lemma convergent_prod_has_prod_iff:  $\text{convergent\_prod } f \longleftrightarrow f \text{ has\_prod } \text{prodinf } f$ 
by (auto simp: has_prod_iff convergent_prod_has_prod)

```

```

lemma prodinf_finite:
  assumes  $N$ : finite  $N$ 
  and  $f$ :  $\bigwedge n. n \notin N \implies f\ n = 1$ 
  shows  $\text{prodinf } f = (\prod_{n \in N}. f\ n)$ 
  using has_prod_finite[OF assms, THEN has_prod_unique] by simp

end

```

6.36.6 Infinite products on ordered topological monoids

```

lemma LIMSEQ_prod_0:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{semidom}, \text{topological\_space}\}$ 
  assumes  $f\ i = 0$ 
  shows  $(\lambda n. \text{prod } f\ \{..n\}) \longrightarrow 0$ 
proof (subst tendsto_cong)
  show  $\forall_F n \text{ in sequentially. } \text{prod } f\ \{..n\} = 0$ 
  proof
    show  $\text{prod } f\ \{..n\} = 0$  if  $n \geq i$  for  $n$ 
    using that assms by auto
  qed
qed auto

```

```

lemma LIMSEQ_prod_nonneg:
  fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{linordered\_semidom}, \text{linorder\_topology}\}$ 
  assumes  $0$ :  $\bigwedge n. 0 \leq f\ n$  and  $a$ :  $(\lambda n. \text{prod } f\ \{..n\}) \longrightarrow a$ 
  shows  $a \geq 0$ 
  by (simp add: 0 prod_nonneg LIMSEQ_le_const [OF a])

```

```

context
  fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{linordered\_semidom}, \text{linorder\_topology}\}$ 
begin

```

```

lemma has_prod_le:
  assumes  $f$ :  $f \text{ has\_prod } a$  and  $g$ :  $g \text{ has\_prod } b$  and  $le$ :  $\bigwedge n. 0 \leq f\ n \wedge f\ n \leq g\ n$ 

```

```

  shows  $a \leq b$ 
proof (cases  $a=0 \vee b=0$ )
case True
then show ?thesis
proof
  assume [simp]:  $a=0$ 
  have  $b \geq 0$ 
  proof (rule LIMSEQ_prod_nonneg)
    show  $(\lambda n. \text{prod } g \ \{..n\}) \longrightarrow b$ 
    using  $g$  by (auto simp: has_prod_def raw_has_prod_def LIMSEQ_prod_0)
  qed (use le_order_trans in auto)
  then show ?thesis
  by auto
next
  assume [simp]:  $b=0$ 
  then obtain  $i$  where  $g \ i = 0$ 
  using  $g$  by (auto simp: prod_defs)
  then have  $f \ i = 0$ 
  using antisym le by force
  then have  $a=0$ 
  using  $f$  by (auto simp: prod_defs LIMSEQ_prod_0 LIMSEQ_unique)
  then show ?thesis
  by auto
qed
next
case False
then show ?thesis
using assms
unfolding has_prod_def raw_has_prod_def
by (force simp: LIMSEQ_prod_0 intro!: LIMSEQ_le prod_mono)
qed

lemma prodinf_le:
  assumes  $f: f \text{ has\_prod } a$  and  $g: g \text{ has\_prod } b$  and  $le: \bigwedge n. 0 \leq f \ n \wedge f \ n \leq g \ n$ 
  shows  $\text{prodinf } f \leq \text{prodinf } g$ 
  using has_prod_le [OF assms] has_prod_unique  $f \ g$  by blast

end

```

```

lemma prod_le_produf:
  fixes  $f :: \text{nat} \Rightarrow 'a::\{\text{linordered\_idom}, \text{linorder\_topology}\}$ 
  assumes  $f \text{ has\_prod } a \wedge i. 0 \leq f \ i \wedge i. i \geq n \implies 1 \leq f \ i$ 
  shows  $\text{prod } f \ \{..<n\} \leq \text{prodinf } f$ 
  by (rule has_prod_le[OF has_prod_If_finite_set]) (use assms has_prod_unique in auto)

```

```

lemma prodinf_nonneg:
  fixes  $f :: \text{nat} \Rightarrow 'a::\{\text{linordered\_idom}, \text{linorder\_topology}\}$ 

```

```

assumes  $f$  has_prod  $a \wedge i. 1 \leq f i$ 
shows  $1 \leq \text{prodinf } f$ 
using prod_le_produf[of  $f a 0$ ] assms
by (metis order_trans prod_ge_1 zero_le_one)

```

```

lemma produf_le_const:
  fixes  $f :: \text{nat} \Rightarrow \text{real}$ 
  assumes convergent_prod  $f \wedge n. \text{prod } f \{.. $n\} \leq x$ 
  shows produf  $f \leq x$ 
  by (metis lessThan_Suc_atMost assms convergent_prod LIMSEQ LIMSEQ_le_const2)$ 
```

```

lemma produf_eq_one_iff [simp]:
  fixes  $f :: \text{nat} \Rightarrow \text{real}$ 
  assumes  $f: \text{convergent\_prod } f$  and  $ge1: \wedge n. 1 \leq f n$ 
  shows produf  $f = 1 \longleftrightarrow (\forall n. f n = 1)$ 
proof
  assume produf  $f = 1$ 
  then have  $(\lambda n. \prod_{i < n} f i) \longrightarrow 1$ 
  using convergent_prod_LIMSEQ[of  $f$ ] assms by (simp add: LIMSEQ_lessThan_iff_atMost)
  then have  $\wedge i. (\prod_{n \in \{i\}} f n) \leq 1$ 
  proof (rule LIMSEQ_le_const)
    have  $1 \leq \text{prod } f n$  for  $n$ 
    by (simp add: ge1 prod_ge_1)
    have  $\text{prod } f \{.. $n\} = 1$  for  $n$ 
    by (metis  $\wedge n. 1 \leq \text{prod } f n \langle \text{produf } f = 1 \rangle$  antisym  $f$  convergent_prod_has_prod$ 
     $ge1$  order_trans prod_le_produf zero_le_one)
    then have  $(\prod_{n \in \{i\}} f n) \leq \text{prod } f \{.. $n\}$  if  $n \geq \text{Suc } i$  for  $i n$ 
    by (metis mult.left_neutral order_refl prod.cong prod.neutral_const prod.lessThan_Suc)
    then show  $\exists N. \forall n \geq N. (\prod_{n \in \{i\}} f n) \leq \text{prod } f \{.. $n\}$  for  $i$ 
    by blast
  qed
  qed
  with ge1 show  $\forall n. f n = 1$ 
  by (auto intro!: antisym)
qed (metis produf_zero fun_eq_iff)$$ 
```

```

lemma produf_pos_iff:
  fixes  $f :: \text{nat} \Rightarrow \text{real}$ 
  assumes convergent_prod  $f \wedge n. 1 \leq f n$ 
  shows  $1 < \text{prodinf } f \longleftrightarrow (\exists i. 1 < f i)$ 
  using prod_le_produf[of  $f 1$ ] produf_eq_one_iff
  by (metis convergent_prod_has_prod assms less_le produf_nonneg)

```

```

lemma less_1_produf2:
  fixes  $f :: \text{nat} \Rightarrow \text{real}$ 
  assumes convergent_prod  $f \wedge n. 1 \leq f n$  and  $1 < f i$ 
  shows  $1 < \text{prodinf } f$ 
proof -
  have  $1 < (\prod_{n < \text{Suc } i} f n)$ 
  using assms by (intro less_1_prod2[where  $i=i$ ]) auto

```

also have $\dots \leq \text{prodinf } f$
 by (intro prod_le_prodf) (use assms order_trans zero_le_one in ‹blast+›)
 finally show ?thesis .
 qed

lemma less_1_prodf:
 fixes $f :: \text{nat} \Rightarrow \text{real}$
 shows $\llbracket \text{convergent\_prod } f; \bigwedge n. 1 < f\ n \rrbracket \implies 1 < \text{prodinf } f$
 by (intro less_1_prodf2[where $i=1$]) (auto intro: less_imp_le)

lemma prodinf_nonzero:
 fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{idom}, \text{topological\_semigroup\_mult}, \text{t2\_space}\}$
 assumes $\text{convergent\_prod } f \wedge i. f\ i \neq 0$
 shows $\text{prodinf } f \neq 0$
 by (metis assms convergent_prod_offset_0 has_prod_unique raw_has_prod_def has_prod_def)

lemma less_0_prodf:
 fixes $f :: \text{nat} \Rightarrow \text{real}$
 assumes $f: \text{convergent\_prod } f$ and $0: \bigwedge i. f\ i > 0$
 shows $0 < \text{prodinf } f$
 proof –
 have $\text{prodinf } f \neq 0$
 by (metis assms less_irrefl prodinf_nonzero)
 moreover have $0 < (\prod_{n < i. f\ n})$ for i
 by (simp add: 0 prod_pos)
 then have $\text{prodinf } f \geq 0$
 using convergent_prod_LIMSEQ [OF f] LIMSEQ_prod_nonneg 0 less_le by blast
 ultimately show ?thesis
 by auto
 qed

lemma prod_less_prodf2:
 fixes $f :: \text{nat} \Rightarrow \text{real}$
 assumes $f: \text{convergent\_prod } f$ and $1: \bigwedge m. m \geq n \implies 1 \leq f\ m$ and $0: \bigwedge m. 0 < f\ m$ and $i: n \leq i \wedge 1 < f\ i$
 shows $\text{prod } f\ \{..<n\} < \text{prodinf } f$
 proof –
 have $\text{prod } f\ \{..<n\} \leq \text{prod } f\ \{..<i\}$
 by (rule prod_mono2) (use assms less_le in auto)
 then have $\text{prod } f\ \{..<n\} < f\ i * \text{prod } f\ \{..<i\}$
 using mult_less_le_imp_less[of $1\ f\ i\ \text{prod } f\ \{..<n\}\ \text{prod } f\ \{..<i\}$] assms
 by (simp add: prod_pos)
 moreover have $\text{prod } f\ \{..<\text{Suc } i\} \leq \text{prodinf } f$
 using prod_le_prodf[of $f\ \text{Suc } i$]
 by (meson 0 1 Suc_leD convergent_prod_has_prod $f\ \langle n \leq i \rangle$ le_trans less_eq_real_def)
 ultimately show ?thesis
 by (metis le_less_trans mult.commute not_le prod_lessThan_Suc)

2756

qed

```
lemma prod_less_prodnf:
  fixes f :: nat ⇒ real
  assumes f: convergent_prod f and 1:  $\bigwedge m. m \geq n \implies 1 < f\ m$  and 0:  $\bigwedge m. 0 < f\ m$ 
  shows prod f {.. $n$ } < prodnf f
  by (meson 0 1 f le_less prod_less_prodnf2)
```

```
lemma raw_has_prodnf_bounded:
  fixes f :: nat ⇒ real
  assumes pos:  $\bigwedge n. 1 \leq f\ n$ 
  and le:  $\bigwedge n. (\prod_{i < n} f\ i) \leq x$ 
  shows  $\exists p. \text{raw\_has\_prod}\ f\ 0\ p$ 
  unfolding raw_has_prodnf_def add_0_right
proof (rule exI LIMSEQ_incseq_SUP conjI)
  show bdd_above (range ( $\lambda n. \text{prod}\ f\ \{..n\}$ ))
  by (metis bdd_aboveI2 le_lessThan_Suc_atMost)
  then have ( $\text{SUP}\ i. \text{prod}\ f\ \{..i\}$ ) > 0
  by (metis UNIV_I cSUP_upper less_le_trans pos prod_pos zero_less_one)
  then show ( $\text{SUP}\ i. \text{prod}\ f\ \{..i\}$ )  $\neq 0$ 
  by auto
  show incseq ( $\lambda n. \text{prod}\ f\ \{..n\}$ )
  using pos order_trans [OF zero_le_one] by (auto simp: mono_def intro!: prod_mono2)
qed
```

```
lemma convergent_prodnf_nonneg_bounded:
  fixes f :: nat ⇒ real
  assumes  $\bigwedge n. 1 \leq f\ n \wedge (\prod_{i < n} f\ i) \leq x$ 
  shows convergent_prod f
  using convergent_prod_def raw_has_prodnf_bounded [OF assms] by blast
```

6.36.7 Infinite products on topological spaces

context

```
  fixes f g :: nat ⇒ 'a::t2_space, topological_semigroup_mult, idom}
begin
```

```
lemma raw_has_prod_mult:  $\llbracket \text{raw\_has\_prod}\ f\ M\ a; \text{raw\_has\_prod}\ g\ M\ b \rrbracket \implies$ 
 $\text{raw\_has\_prod}\ (\lambda n. f\ n * g\ n)\ M\ (a * b)$ 
  by (force simp add: prod.distrib tendsto_mult raw_has_prod_def)
```

```
lemma has_prod_mult_nz:  $\llbracket f\ \text{has\_prod}\ a; g\ \text{has\_prod}\ b; a \neq 0; b \neq 0 \rrbracket \implies (\lambda n.$ 
 $f\ n * g\ n)\ \text{has\_prod}\ (a * b)$ 
  by (simp add: raw_has_prod_mult has_prod_def)
```

end


```

context
  fixes f g :: nat  $\Rightarrow$  'a::real_normed_field
begin

lemma has_prod_mult:
  assumes f: f has_prod a and g: g has_prod b
  shows ( $\lambda n. f\ n * g\ n$ ) has_prod (a * b)
  using f [unfolded has_prod_def]
proof (elim disjE exE conjE)
  assume f0: raw_has_prod f 0 a
  show ?thesis
    using g [unfolded has_prod_def]
  proof (elim disjE exE conjE)
    assume g0: raw_has_prod g 0 b
    with f0 show ?thesis
      by (force simp add: has_prod_def prod.distrib tendsto_mult raw_has_prod_def)
  next
    fix j q
    assume b = 0 and g j = 0 and q: raw_has_prod g (Suc j) q
    obtain p where p: raw_has_prod f (Suc j) p
      using f0 raw_has_prod_ignore_initial_segment by blast
    then have Ex (raw_has_prod ( $\lambda n. f\ n * g\ n$ ) (Suc j))
      using q raw_has_prod_mult by blast
    then show ?thesis
      using  $\langle b = 0 \rangle \langle g\ j = 0 \rangle$  has_prod_0_iff by fastforce
  qed
next
  fix i p
  assume a = 0 and f i = 0 and p: raw_has_prod f (Suc i) p
  show ?thesis
    using g [unfolded has_prod_def]
  proof (elim disjE exE conjE)
    assume g0: raw_has_prod g 0 b
    obtain q where q: raw_has_prod g (Suc i) q
      using g0 raw_has_prod_ignore_initial_segment by blast
    then have Ex (raw_has_prod ( $\lambda n. f\ n * g\ n$ ) (Suc i))
      using raw_has_prod_mult p by blast
    then show ?thesis
      using  $\langle a = 0 \rangle \langle f\ i = 0 \rangle$  has_prod_0_iff by fastforce
  next
    fix j q
    assume b = 0 and g j = 0 and q: raw_has_prod g (Suc j) q
    obtain p' where p': raw_has_prod f (Suc (max i j)) p'
      by (metis raw_has_prod_ignore_initial_segment max_Suc_Suc max_def p)
    moreover
    obtain q' where q': raw_has_prod g (Suc (max i j)) q'
      by (metis raw_has_prod_ignore_initial_segment max.cobounded2 max_Suc_Suc
q)

```

2758

```

ultimately show ?thesis
  using ‹b = 0› by (simp add: has_prod_def) (metis ‹f i = 0› ‹g j = 0›
raw_has_prod_mult max_def)
qed
qed

lemma convergent_prod_mult:
  assumes f: convergent_prod f and g: convergent_prod g
  shows convergent_prod (λn. f n * g n)
  unfolding convergent_prod_def
proof -
  obtain M p N q where p: raw_has_prod f M p and q: raw_has_prod g N q
  using convergent_prod_def f g by blast+
  then obtain p' q' where p': raw_has_prod f (max M N) p' and q': raw_has_prod
g (max M N) q'
  by (meson raw_has_prod_ignore_initial_segment max.cobounded1 max.cobounded2)
  then show ∃ M p. raw_has_prod (λn. f n * g n) M p
  using raw_has_prod_mult by blast
qed

```

```

lemma prodinf_mult: convergent_prod f ⟹ convergent_prod g ⟹ prodinf f *
prodinf g = (∏ n. f n * g n)
  by (intro has_prod_unique has_prod_mult convergent_prod_has_prod)

end

```

```

context
  fixes f :: 'i ⇒ nat ⇒ 'a::real_normed_field
  and I :: 'i set
begin

```

```

lemma has_prod_prod: (∧i. i ∈ I ⟹ (f i) has_prod (x i)) ⟹ (λn. ∏ i∈I. f i
n) has_prod (∏ i∈I. x i)
  by (induct I rule: infinite_finite_induct) (auto intro!: has_prod_mult)

```

```

lemma prodinf_prod: (∧i. i ∈ I ⟹ convergent_prod (f i)) ⟹ (∏ n. ∏ i∈I. f i
n) = (∏ i∈I. ∏ n. f i n)
  using has_prod_unique[OF has_prod_prod, OF convergent_prod_has_prod] by
simp

```

```

lemma convergent_prod_prod: (∧i. i ∈ I ⟹ convergent_prod (f i)) ⟹ conver-
gent_prod (λn. ∏ i∈I. f i n)
  using convergent_prod_has_prod_iff has_prod_prod prodinf_prod by force

```

end

6.36.8 Infinite summability on real normed fields

```

context

```

fixes $f :: nat \Rightarrow 'a::real\_normed\_field$
begin

lemma *raw_has_prod_Suc_iff*: $raw\_has\_prod\ f\ M\ (a * f\ M) \longleftrightarrow raw\_has\_prod\ (\lambda n. f\ (Suc\ n))\ M\ a \wedge f\ M \neq 0$

proof $-$

have $raw\_has\_prod\ f\ M\ (a * f\ M) \longleftrightarrow (\lambda i. \prod_{j \leq Suc\ i} f\ (j + M)) \longrightarrow a * f\ M \wedge a * f\ M \neq 0$

by (*subst filterlim_sequentially_Suc*) (*simp add: raw_has_prod_def*)

also have $\dots \longleftrightarrow (\lambda i. (\prod_{j \leq i} f\ (Suc\ j + M)) * f\ M) \longrightarrow a * f\ M \wedge a * f\ M \neq 0$

by (*simp add: ac_simps atMost_Suc_eq_insert_0 image_Suc_atMost prod.atLeast1_atMost_eq lessThan_Suc_atMost*)

del: prod.cl_ivl_Suc)

also have $\dots \longleftrightarrow raw\_has\_prod\ (\lambda n. f\ (Suc\ n))\ M\ a \wedge f\ M \neq 0$

proof safe

assume *tends*: $(\lambda i. (\prod_{j \leq i} f\ (Suc\ j + M)) * f\ M) \longrightarrow a * f\ M$ **and** $0: a * f\ M \neq 0$

with *tendsto_divide*[*OF tends tendsto_const, of f M*]

show $raw\_has\_prod\ (\lambda n. f\ (Suc\ n))\ M\ a$

by (*simp add: raw_has_prod_def*)

qed (*auto intro: tendsto_mult_right simp: raw_has_prod_def*)

finally show *?thesis* .

qed

lemma *has_prod_Suc_iff*:

assumes $f\ 0 \neq 0$ **shows** $(\lambda n. f\ (Suc\ n))\ has\_prod\ a \longleftrightarrow f\ has\_prod\ (a * f\ 0)$

proof (*cases a = 0*)

case True

then show *?thesis*

proof (*simp add: has_prod_def, safe*)

fix $i\ x$

assume $f\ (Suc\ i) = 0$ **and** $raw\_has\_prod\ (\lambda n. f\ (Suc\ n))\ (Suc\ i)\ x$

then obtain y **where** $raw\_has\_prod\ f\ (Suc\ (Suc\ i))\ y$

by (*metis (no_types) raw_has_prod_eq_0 Suc_n_not_le_n raw_has_prod_Suc_iff raw_has_prod_ignore_initial_segment raw_has_prod_nonzero linear*)

then show $\exists i. f\ i = 0 \wedge \exists x\ (raw\_has\_prod\ f\ (Suc\ i))$

using $\langle f\ (Suc\ i) = 0 \rangle$ **by** *blast*

next

fix $i\ x$

assume $f\ i = 0$ **and** $x: raw\_has\_prod\ f\ (Suc\ i)\ x$

then obtain j **where** $j: i = Suc\ j$

by (*metis assms not0_implies_Suc*)

moreover have $\exists y. raw\_has\_prod\ (\lambda n. f\ (Suc\ n))\ i\ y$

using x **by** (*auto simp: raw_has_prod_def*)

then show $\exists i. f\ (Suc\ i) = 0 \wedge \exists x\ (raw\_has\_prod\ (\lambda n. f\ (Suc\ n))\ (Suc\ i))$

using $\langle f\ i = 0 \rangle\ j$ **by** *blast*

qed

next

2760

```

    case False
  then show ?thesis
    by (auto simp: has_prod_def raw_has_prod_Suc_iff assms)
qed

lemma convergent_prod_Suc_iff [simp]:
  shows convergent_prod ( $\lambda n. f (Suc n)$ ) = convergent_prod f
proof
  assume convergent_prod f
  then obtain M L where M_nz:  $\forall n \geq M. f n \neq 0$  and
    M_L: ( $\lambda n. \prod_{i \leq n}. f (i + M)$ )  $\longrightarrow L$  and  $L \neq 0$ 
    unfolding convergent_prod_altdef by auto
  have ( $\lambda n. \prod_{i \leq n}. f (Suc (i + M))$ )  $\longrightarrow L / f M$ 
  proof -
    have ( $\lambda n. \prod_{i \in \{0..Suc\ n\}}. f (i + M)$ )  $\longrightarrow L$ 
      using M_L
    apply (subst (asm) filterlim_sequentially_Suc[symmetric])
    using atLeast0AtMost by auto
  then have ( $\lambda n. f M * (\prod_{i \in \{0..n\}}. f (Suc (i + M)))$ )  $\longrightarrow L$ 
    apply (subst (asm) prod.atLeast0_atMost_Suc_shift)
    by simp
  then have ( $\lambda n. (\prod_{i \in \{0..n\}}. f (Suc (i + M)))$ )  $\longrightarrow L / f M$ 
    apply (drule_tac tendsto_divide)
    using M_nz[rule_format, of M, simplified] by auto
  then show ?thesis unfolding atLeast0AtMost .
qed

then show convergent_prod ( $\lambda n. f (Suc n)$ ) unfolding convergent_prod_altdef
  apply (rule_tac exI[where x=M])
  apply (rule_tac exI[where x=L/f M])
  using M_nz <math>L \neq 0</math> by auto
next
  assume convergent_prod ( $\lambda n. f (Suc n)$ )
  then obtain M where  $\exists L. (\forall n \geq M. f (Suc n) \neq 0) \wedge (\lambda n. \prod_{i \leq n}. f (Suc (i + M))) \longrightarrow L \wedge L \neq 0$ 
    unfolding convergent_prod_altdef by auto
  then show convergent_prod f unfolding convergent_prod_altdef
    apply (rule_tac exI[where x=Suc M])
    using Suc_le_D by auto
qed

lemma raw_has_prod_inverse:
  assumes raw_has_prod f M a shows raw_has_prod ( $\lambda n. inverse (f n)$ ) M
    (inverse a)
  using assms unfolding raw_has_prod_def by (auto dest: tendsto_inverse simp:
    prod_inversef [symmetric])

lemma has_prod_inverse:
  assumes f_has_prod a shows ( $\lambda n. inverse (f n)$ ) has_prod (inverse a)
  using assms raw_has_prod_inverse unfolding has_prod_def by auto

```

```

lemma convergent_prod_inverse:
  assumes convergent_prod f
  shows convergent_prod  $(\lambda n. \text{inverse } (f\ n))$ 
  using assms unfolding convergent_prod_def by (blast intro: raw_has_prod_inverse
elim: )

end

context
  fixes f :: nat  $\Rightarrow$  'a::real_normed_field'
begin

lemma raw_has_prod_Suc_iff': raw_has_prod f M a  $\longleftrightarrow$  raw_has_prod  $(\lambda n. f$ 
(Suc n)) M  $(a / f\ M) \wedge f\ M \neq 0$ 
  by (metis raw_has_prod_eq_0 add.commute add.left_neutral raw_has_prod_Suc_iff
raw_has_prod_nonzero le_add1 nonzero_mult_div_cancel_right times_divide_eq_left)

lemma has_prod_divide: f has_prod a  $\implies$  g has_prod b  $\implies$   $(\lambda n. f\ n / g\ n)$ 
has_prod  $(a / b)$ 
  unfolding divide_inverse by (intro has_prod_inverse has_prod_mult)

lemma convergent_prod_divide:
  assumes f: convergent_prod f and g: convergent_prod g
  shows convergent_prod  $(\lambda n. f\ n / g\ n)$ 
  using f g has_prod_divide has_prod_iff by blast

lemma prodnf_divide: convergent_prod f  $\implies$  convergent_prod g  $\implies$  prodnf f /
prodnf g =  $(\prod n. f\ n / g\ n)$ 
  by (intro has_prod_unique has_prod_divide convergent_prod_has_prod)

lemma prodnf_inverse: convergent_prod f  $\implies$   $(\prod n. \text{inverse } (f\ n)) = \text{inverse}$ 
 $(\prod n. f\ n)$ 
  by (intro has_prod_unique [symmetric] has_prod_inverse convergent_prod_has_prod)

lemma has_prod_Suc_imp:
  assumes  $(\lambda n. f\ (\text{Suc } n))$  has_prod a
  shows f has_prod  $(a * f\ 0)$ 
proof –
  have f has_prod  $(a * f\ 0)$  when raw_has_prod  $(\lambda n. f\ (\text{Suc } n))$  0 a
  apply (cases f 0=0)
  using that unfolding has_prod_def raw_has_prod_Suc
  by (auto simp add: raw_has_prod_Suc_iff)
moreover have f has_prod  $(a * f\ 0)$  when
 $(\exists i\ q. a = 0 \wedge f\ (\text{Suc } i) = 0 \wedge \text{raw\_has\_prod } (\lambda n. f\ (\text{Suc } n))\ (\text{Suc } i)\ q)$ 
proof –
  from that
  obtain i q where  $a = 0 \wedge f\ (\text{Suc } i) = 0$  raw_has_prod  $(\lambda n. f\ (\text{Suc } n))$   $(\text{Suc } i)\ q$ 
  by auto

```

```

    then show ?thesis unfolding has_prod_def
    by (auto intro!: exI[where x=Suc i] simp: raw_has_prod_Suc)
qed
ultimately show f has_prod (a * f 0) using assms unfolding has_prod_def
by auto
qed

```

```

lemma has_prod_iff_shift:
  assumes  $\bigwedge i. i < n \implies f i \neq 0$ 
  shows  $(\lambda i. f (i + n)) \text{ has\_prod } a \longleftrightarrow f \text{ has\_prod } (a * (\prod_{i < n}. f i))$ 
  using assms
proof (induct n arbitrary: a)
  case 0
  then show ?case by simp
next
  case (Suc n)
  then have  $(\lambda i. f (Suc i + n)) \text{ has\_prod } a \longleftrightarrow (\lambda i. f (i + n)) \text{ has\_prod } (a * f n)$ 
  by (subst has_prod_Suc_iff) auto
  with Suc show ?case
  by (simp add: ac_simps)
qed

```

```

corollary has_prod_iff_shift':
  assumes  $\bigwedge i. i < n \implies f i \neq 0$ 
  shows  $(\lambda i. f (i + n)) \text{ has\_prod } (a / (\prod_{i < n}. f i)) \longleftrightarrow f \text{ has\_prod } a$ 
  by (simp add: assms has_prod_iff_shift)

```

```

lemma has_prod_one_iff_shift:
  assumes  $\bigwedge i. i < n \implies f i = 1$ 
  shows  $(\lambda i. f (i + n)) \text{ has\_prod } a \longleftrightarrow (\lambda i. f i) \text{ has\_prod } a$ 
  by (simp add: assms has_prod_iff_shift)

```

```

lemma convergent_prod_iff_shift [simp]:
  shows  $\text{convergent\_prod } (\lambda i. f (i + n)) \longleftrightarrow \text{convergent\_prod } f$ 
  apply safe
  using convergent_prod_offset apply blast
  using convergent_prod_ignore_initial_segment convergent_prod_def by blast

```

```

lemma has_prod_split_initial_segment:
  assumes  $f \text{ has\_prod } a \wedge \bigwedge i. i < n \implies f i \neq 0$ 
  shows  $(\lambda i. f (i + n)) \text{ has\_prod } (a / (\prod_{i < n}. f i))$ 
  using assms has_prod_iff_shift' by blast

```

```

lemma prodinf_divide_initial_segment:
  assumes  $\text{convergent\_prod } f \wedge \bigwedge i. i < n \implies f i \neq 0$ 
  shows  $(\prod i. f (i + n)) = (\prod i. f i) / (\prod_{i < n}. f i)$ 
  by (rule has_prod_unique[symmetric]) (auto simp: assms has_prod_iff_shift)

```

```

lemma prodimf_split_initial_segment:
  assumes convergent_prod  $f \wedge i. i < n \implies f\ i \neq 0$ 
  shows  $\text{prodimf } f = (\prod i. f\ (i + n)) * (\prod i < n. f\ i)$ 
  by (auto simp add: assms prodinf_divide_initial_segment)

lemma prodimf_split_head:
  assumes convergent_prod  $f\ f\ 0 \neq 0$ 
  shows  $(\prod n. f\ (\text{Suc } n)) = \text{prodimf } f / f\ 0$ 
  using prodimf_split_initial_segment[of 1] assms by simp

end

context
  fixes  $f :: \text{nat} \Rightarrow 'a :: \text{real\_normed\_field}$ 
begin

lemma convergent_prod_inverse_iff [simp]: convergent_prod  $(\lambda n. \text{inverse } (f\ n))$ 
 $\longleftrightarrow$  convergent_prod  $f$ 
  by (auto dest: convergent_prod_inverse)

lemma convergent_prod_const_iff [simp]:
  fixes  $c :: 'a :: \{\text{real\_normed\_field}\}$ 
  shows convergent_prod  $(\lambda \_. c) \longleftrightarrow c = 1$ 
proof
  assume convergent_prod  $(\lambda \_. c)$ 
  then show  $c = 1$ 
    using convergent_prod_imp_LIMSEQ LIMSEQ_unique by blast
next
  assume  $c = 1$ 
  then show convergent_prod  $(\lambda \_. c)$ 
    by auto
qed

lemma has_prod_power:  $f$  has_prod  $a \implies (\lambda i. f\ i \wedge n)$  has_prod  $(a \wedge n)$ 
  by (induction  $n$ ) (auto simp: has_prod_mult)

lemma convergent_prod_power: convergent_prod  $f \implies \text{convergent\_prod } (\lambda i. f\ i \wedge n)$ 
  by (induction  $n$ ) (auto simp: convergent_prod_mult)

lemma prodimf_power: convergent_prod  $f \implies \text{prodimf } (\lambda i. f\ i \wedge n) = \text{prodimf } f \wedge n$ 
  by (metis has_prod_unique convergent_prod_imp_has_prod has_prod_power)

end

```

6.36.9 Exponentials and logarithms

context

2764

```
fixes f :: nat => 'a::{real_normed_field,banach}
begin
```

```
lemma sums_imp_has_prod_exp:
  assumes f sums s
  shows raw_has_prod ( $\lambda i. \exp (f i)$ ) 0 ( $\exp s$ )
  using assms continuous_on_exp [of UNIV  $\lambda x::'a. x$ ]
  using continuous_on_tendsto_compose [of UNIV  $\exp (\lambda n. \text{sum } f \{..n\}) s$ ]
  by (simp add: prod_defs sums_def le_exp_sum)
```

```
lemma convergent_prod_exp:
  assumes summable f
  shows convergent_prod ( $\lambda i. \exp (f i)$ )
  using sums_imp_has_prod_exp assms unfolding summable_def convergent_prod_def
  by blast
```

```
lemma prodinf_exp:
  assumes summable f
  shows prodinf ( $\lambda i. \exp (f i)$ ) =  $\exp (\text{suminf } f)$ 
proof -
  have f sums suminf f
    using assms by blast
  then have ( $\lambda i. \exp (f i)$ ) has_prod  $\exp (\text{suminf } f)$ 
    by (simp add: has_prod_def sums_imp_has_prod_exp)
  then show ?thesis
    by (rule has_prod_unique [symmetric])
qed
```

end

```
theorem convergent_prod_iff_summable_real:
  fixes a :: nat => real
  assumes  $\bigwedge n. a\ n > 0$ 
  shows convergent_prod ( $\lambda k. 1 + a\ k$ )  $\longleftrightarrow$  summable a (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain p where raw_has_prod ( $\lambda k. 1 + a\ k$ ) 0 p
    by (metis assms add_less_same_cancel2 convergent_prod_offset_0 not_one_less_zero)
  then have to_p: ( $\lambda n. \prod_{k \leq n}. 1 + a\ k$ )  $\longrightarrow$  p
    by (auto simp: raw_has_prod_def)
  moreover have le: ( $\sum_{k \leq n}. a\ k$ )  $\leq$  ( $\prod_{k \leq n}. 1 + a\ k$ ) for n
    by (rule sum_le_prod) (use assms less_le in force)
  have ( $\prod_{k \leq n}. 1 + a\ k$ )  $\leq$  p for n
  proof (rule incseq_le [OF _ to_p])
    show incseq ( $\lambda n. \prod_{k \leq n}. 1 + a\ k$ )
      using assms by (auto simp: mono_def order.strict_implies_order intro!:
prod_mono2)
    qed
    with le have ( $\sum_{k \leq n}. a\ k$ )  $\leq$  p for n
```



```

    by (metis order_trans)
  with assms bounded_imp_summable show ?rhs
    by (metis not_less order.asym)
next
  assume R: ?rhs
  have  $(\prod_{k \leq n}. 1 + a\ k) \leq \exp(\text{suminf } a)$  for  $n$ 
  proof -
    have  $(\prod_{k \leq n}. 1 + a\ k) \leq \exp(\sum_{k \leq n}. a\ k)$  for  $n$ 
      by (rule prod_le_exp_sum) (use assms less_le in force)
    moreover have  $\exp(\sum_{k \leq n}. a\ k) \leq \exp(\text{suminf } a)$  for  $n$ 
      unfolding exp_le_cancel_iff
      by (meson sum_le_suminf R assms finite_atMost less_eq_real_def)
    ultimately show ?thesis
      by (meson order_trans)
  qed
  then obtain  $L$  where  $L: (\lambda n. \prod_{k \leq n}. 1 + a\ k) \longrightarrow L$ 
    by (metis assms bounded_imp_convergent_prod convergent_prod_iff_nz_lim
      le_add_same_cancel1 le_add_same_cancel2 less_le not_le zero_le_one)
  moreover have  $L \neq 0$ 
  proof
    assume  $L = 0$ 
    with  $L$  have  $(\lambda n. \prod_{k \leq n}. 1 + a\ k) \longrightarrow 0$ 
      by simp
    moreover have  $(\prod_{k \leq n}. 1 + a\ k) > 1$  for  $n$ 
      by (simp add: assms less_1_prod)
    ultimately show False
      by (meson Lim_bounded2 not_one_le_zero less_imp_le)
  qed
  ultimately show ?lhs
    using assms convergent_prod_iff_nz_lim
    by (metis add_less_same_cancel1 less_le not_le zero_less_one)
qed

lemma exp_suminf_producing_real:
  fixes  $f :: \text{nat} \Rightarrow \text{real}$ 
  assumes  $ge0: \bigwedge n. f\ n \geq 0$  and  $ac: \text{abs\_convergent\_prod } (\lambda n. \exp(f\ n))$ 
  shows  $\text{producing } (\lambda i. \exp(f\ i)) = \exp(\text{suminf } f)$ 
  proof -
    have summable  $f$ 
      using  $ac$  unfolding abs_convergent_prod_conv_summable
    proof (elim summable_comparison_test')
      fix  $n$ 
      have  $|f\ n| = f\ n$ 
        by (simp add: ge0)
      also have  $\dots \leq \exp(f\ n) - 1$ 
        by (metis diff_diff_add exp_ge_add_one_self ge_iff_diff_ge_0)
      finally show  $\text{norm } (f\ n) \leq \text{norm } (\exp(f\ n) - 1)$ 
        by simp
    qed
  qed

```

2766

```

    then show ?thesis
      by (simp add: prodinf_exp)
qed

```

```

lemma has_prod_imp_sums_ln_real:
  fixes f :: nat  $\Rightarrow$  real
  assumes raw_has_prod f 0 p and 0:  $\bigwedge x. f\ x > 0$ 
  shows  $(\lambda i. \ln (f\ i))\ \text{sums}\ (\ln\ p)$ 
proof -
  have p > 0
    using assms unfolding prod_defs by (metis LIMSEQ_prod_nonneg less_eq_real_def)
  then show ?thesis
    using assms continuous_on_ln [of {0<..}  $\lambda x. x$ ]
    using continuous_on_tendsto_compose [of {0<..}  $\ln (\lambda n. \text{prod } f\ \{..n\})\ p$ ]
    by (auto simp: prod_defs sums_def le ln_prod order_tendstoD)
qed

```

```

lemma summable_ln_real:
  fixes f :: nat  $\Rightarrow$  real
  assumes f: convergent_prod f and 0:  $\bigwedge x. f\ x > 0$ 
  shows summable  $(\lambda i. \ln (f\ i))$ 
proof -
  obtain M p where raw_has_prod f M p
    using f convergent_prod_def by blast
  then consider i where i < M  $f\ i = 0 \mid p$  where raw_has_prod f 0 p
    using raw_has_prod_cases by blast
  then show ?thesis
  proof cases
    case 1
    with 0 show ?thesis
      by (metis less_irrefl)
  next
    case 2
    then show ?thesis
      using 0 has_prod_imp_sums_ln_real summable_def by blast
  qed
qed

```

```

lemma suminf_ln_real:
  fixes f :: nat  $\Rightarrow$  real
  assumes f: convergent_prod f and 0:  $\bigwedge x. f\ x > 0$ 
  shows  $\text{suminf } (\lambda i. \ln (f\ i)) = \ln (\text{prodinf } f)$ 
proof -
  have f has_prod prodinf f
    by (simp add: f has_prod_iff)
  then have raw_has_prod f 0 (prodinf f)
    by (metis 0 has_prod_def less_irrefl)
  then have  $(\lambda i. \ln (f\ i))\ \text{sums}\ \ln (\text{prodinf } f)$ 
    using 0 has_prod_imp_sums_ln_real by blast

```

```

then show ?thesis
  by (rule sums_unique [symmetric])
qed

```

```

lemma prodinf_exp_real:
  fixes f :: nat ⇒ real
  assumes f: convergent_prod f and 0:  $\bigwedge x. f\ x > 0$ 
  shows prodinf f = exp (suminf (λi. ln (f i)))
  by (simp add: 0 f less_0_prodinf suminf_ln_real)

```

```

theorem Ln_prodinf_complex:
  fixes z :: nat ⇒ complex
  assumes z:  $\bigwedge j. z\ j \neq 0$  and ξ:  $\xi \neq 0$ 
  shows ((λn.  $\prod_{j \leq n}. z\ j$ ) ⟶ ξ) ⟷ (∃ k. (λn. ( $\sum_{j \leq n}. Ln\ (z\ j)$ )) ⟶ Ln
  ξ + of_int k * (of_real(2*pi) * i)) (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  have pnz: ( $\prod_{j \leq n}. z\ j$ ) ≠ 0 for n
    using z by auto
  define Θ where Θ ≡ Arg ξ + 2*pi
  then have Θ > pi
    using Arg_def mpi_less_Im_Ln by fastforce
  have ξ_eq: ξ = cmod ξ * exp (i * Θ)
    using Arg_def Arg_eq ξ unfolding Θ_def by (simp add: algebra_simps
  exp_add)
  define ϑ where ϑ ≡ λn. THE t. is_Arg ( $\prod_{j \leq n}. z\ j$ ) t ∧ t ∈ {Θ - pi <.. $\Theta + pi$ }
  have uniq: ∃! s. is_Arg ( $\prod_{j \leq n}. z\ j$ ) s ∧ s ∈ {Θ - pi <.. $\Theta + pi$ } for n
    using Argument_exists_unique [OF pnz] by metis
  have ϑ: is_Arg ( $\prod_{j \leq n}. z\ j$ ) (ϑ n) and ϑ_interval: ϑ n ∈ {Θ - pi <.. $\Theta + pi$ } for
  n
    unfolding ϑ_def
    using theI' [OF uniq] by metis+
  have ϑ_pos:  $\bigwedge j. \vartheta\ j > 0$ 
    using ϑ_interval ⟨Θ > pi⟩ by simp (meson diff_gt_0_iff_gt less_trans)
  have ( $\prod_{j \leq n}. z\ j$ ) = cmod ( $\prod_{j \leq n}. z\ j$ ) * exp (i * ϑ n) for n
    using ϑ by (auto simp: is_Arg_def)
  then have eq: (λn.  $\prod_{j \leq n}. z\ j$ ) = (λn. cmod ( $\prod_{j \leq n}. z\ j$ ) * exp (i * ϑ n))
    by simp
  then have (λn. (cmod ( $\prod_{j \leq n}. z\ j$ ) * exp (i * (ϑ n)))) ⟶ ξ
    using L by force
  then obtain k where k: (λj. ϑ j - of_int (k j) * (2 * pi)) ⟶ Θ
    using L by (subst (asm) ξ_eq) (auto simp add: eq z ξ polar_convergence)
  moreover have  $\forall_F n$  in sequentially. k n = 0
  proof -
    have *: k j = 0 if dist (ϑ j - real_of_int k j * 2) V < 1 ϑ j ∈ {V - 1 <.. $V + 1$ }
    for k j ϑ j V
      using that by (auto simp: dist_norm)
    have  $\forall_F j$  in sequentially. dist (ϑ j - of_int (k j) * (2 * pi)) Θ < pi

```

```

    using tendstoD [OF k] pi_gt_zero by blast
  then show ?thesis
  proof (rule eventually_mono)
    fix j
    assume d: dist (∅ j - real_of_int (k j) * (2 * pi)) Θ < pi
    show k j = 0
    by (rule * [of ∅ j/pi _ Θ/pi])
      (use ∅_interval [of j] d in ⟨simp_all add: divide_simps dist_norm⟩)
  qed
qed
ultimately have ∅toΘ: ∅ ⟶ Θ
  apply (simp only: tendsto_def)
  apply (erule all_forward imp_forward asm_rl)+
  apply (drule (1) eventually_conj)
  apply (auto elim: eventually_mono)
  done
then have to0: (λn. |∅ (Suc n) - ∅ n|) ⟶ 0
  by (metis (full_types) diff_self filterlim_sequentially_Suc tendsto_diff tend-
sto_rabs_zero)
  have ∃ k. Im (∑ j ≤ n. Ln (z j)) - of_int k * (2*pi) = ∅ n for n
  proof (rule is_Arg_exp_diff_2pi)
    show is_Arg (exp (∑ j ≤ n. Ln (z j))) (∅ n)
    using pnz ∅ by (simp add: is_Arg_def exp_sum prod_norm)
  qed
  then have ∃ k. (∑ j ≤ n. Im (Ln (z j))) = ∅ n + of_int k * (2*pi) for n
  by (simp add: algebra_simps)
  then obtain k where k: ∧ n. (∑ j ≤ n. Im (Ln (z j))) = ∅ n + of_int (k n) *
(2*pi)
  by metis
  obtain K where ∀_F n in sequentially. k n = K
  proof -
    have k_le: (2*pi) * |k (Suc n) - k n| ≤ |∅ (Suc n) - ∅ n| + |Im (Ln (z (Suc
n))))| for n
    proof -
      have (∑ j ≤ Suc n. Im (Ln (z j))) - (∑ j ≤ n. Im (Ln (z j))) = Im (Ln (z
(Suc n)))
      by simp
      then show ?thesis
      using k [of Suc n] k [of n] by (auto simp: abs_if algebra_simps)
    qed
  qed
  have z ⟶ 1
  using L ξ convergent_prod_iff_nz_lim z by (blast intro: convergent_prod_imp_LIMSEQ)
  with z have (λn. Ln (z n)) ⟶ Ln 1
  using isCont_tendsto_compose [OF continuous_at_Ln] nonpos_Reals_one_I
by blast
  then have (λn. Ln (z n)) ⟶ 0
  by simp
  then have (λn. |Im (Ln (z (Suc n)))|) ⟶ 0
  by (metis LIMSEQ_unique ⟨z ⟶ 1⟩ continuous_at_Ln filterlim_sequentially_Suc

```

```

isCont tendsto_compose nonpos_Reals_one_I tendsto_Im tendsto_rabs_zero_iff
zero_complex.simps(2))
  then have  $\forall_F n$  in sequentially.  $|Im (Ln (z (Suc n)))| < 1$ 
    by (simp add: order_tendsto_iff)
  moreover have  $\forall_F n$  in sequentially.  $|\vartheta (Suc n) - \vartheta n| < 1$ 
    using to0 by (simp add: order_tendsto_iff)
  ultimately have  $\forall_F n$  in sequentially.  $(2*pi) * |k (Suc n) - k n| < 1 + 1$ 
  proof (rule eventually_elim2)
    fix n
    assume  $|Im (Ln (z (Suc n)))| < 1$  and  $|\vartheta (Suc n) - \vartheta n| < 1$ 
    with k_le [of n] show  $2 * pi * real\_of\_int |k (Suc n) - k n| < 1 + 1$ 
      by linarith
  qed
  then have  $\forall_F n$  in sequentially.  $real\_of\_int |k (Suc n) - k n| < 1$ 
  proof (rule eventually_mono)
    fix n :: nat
    assume  $2 * pi * |k (Suc n) - k n| < 1 + 1$ 
    then have  $|k (Suc n) - k n| < 2 / (2*pi)$ 
      by (simp add: field_simps)
    also have ... < 1
      using pi_ge_two by auto
    finally show  $real\_of\_int |k (Suc n) - k n| < 1$  .
  qed
  then obtain N where  $N: \bigwedge n. n \geq N \implies |k (Suc n) - k n| = 0$ 
    using eventually_sequentially_less_irrefl of_int_abs by fastforce
  have  $k (N+i) = k N$  for i
  proof (induction i)
    case (Suc i)
    with N [of N+i] show ?case
      by auto
  qed simp
  then have  $\bigwedge n. n \geq N \implies k n = k N$ 
    using le_Suc_ex by auto
  then show ?thesis
    by (force simp add: eventually_sequentially intro: that)
  qed
  with  $\vartheta$ to $\Theta$  have  $(\lambda n. (\sum_{j \leq n}. Im (Ln (z j)))) \longrightarrow \Theta + of\_int K * (2*pi)$ 
    by (simp add: k tendsto_add tendsto_mult tendsto_eventually)
  moreover have  $(\lambda n. (\sum_{k \leq n}. Re (Ln (z k)))) \longrightarrow Re (Ln \xi)$ 
    using assms continuous_imp_tendsto [OF isCont_ln tendsto_norm [OF L]]
    by (simp add: o_def flip: prod_norm ln_prod)
  ultimately show ?rhs
    by (rule_tac x=K+1 in exI) (auto simp: tendsto_complex_iff  $\Theta\_def$  Arg_def
    assms algebra_simps)
next
  assume ?rhs
  then obtain r where  $r: (\lambda n. (\sum_{k \leq n}. Ln (z k))) \longrightarrow Ln \xi + of\_int r * (of\_real(2*pi) * i)$  ..
  have  $(\lambda n. exp (\sum_{k \leq n}. Ln (z k))) \longrightarrow \xi$ 

```

```

    using assms continuous_imp_tendsto [OF isCont_exp r] exp_integer_2pi [of
r]
    by (simp add: o_def exp_add algebra_simps)
  moreover have  $\exp(\sum_{k \leq n} Ln(z k)) = (\prod_{k \leq n} z k)$  for  $n$ 
    by (simp add: exp_sum add_eq_0_iff assms)
  ultimately show ?lhs
    by auto
qed

```

Prop 17.2 of Bak and Newman, Complex Analysis, p.242

```

proposition convergent_prod_iff_summable_complex:
  fixes  $z :: nat \Rightarrow complex$ 
  assumes  $\bigwedge k. z k \neq 0$ 
  shows  $\text{convergent\_prod } (\lambda k. z k) \longleftrightarrow \text{summable } (\lambda k. Ln(z k))$  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain  $p$  where  $p: (\lambda n. \prod_{k \leq n} z k) \longrightarrow p$  and  $p \neq 0$ 
    using convergent_prod_LIMSEQ prodinf_nonzero add_eq_0_iff assms by
fastforce
  then show ?rhs
    using Ln_prodinf_complex assms
    by (auto simp: prodinf_nonzero summable_def sums_def le)
next
  assume  $R: ?rhs$ 
  have  $(\prod_{k \leq n} z k) = \exp(\sum_{k \leq n} Ln(z k))$  for  $n$ 
    by (simp add: exp_sum add_eq_0_iff assms)
  then have  $(\lambda n. \prod_{k \leq n} z k) \longrightarrow \exp(\text{suminf } (\lambda k. Ln(z k)))$ 
    using continuous_imp_tendsto [OF isCont_exp summable_LIMSEQ' [OF R]]
  by (simp add: o_def)
  then show ?lhs
    by (subst convergent_prod_iff_convergent) (auto simp: convergent_def tend-
sto_Lim assms add_eq_0_iff)
qed

```

Prop 17.3 of Bak and Newman, Complex Analysis

```

proposition summable_imp_convergent_prod_complex:
  fixes  $z :: nat \Rightarrow complex$ 
  assumes  $z: \text{summable } (\lambda k. \text{norm } (z k))$  and  $\text{non0}: \bigwedge k. z k \neq -1$ 
  shows  $\text{convergent\_prod } (\lambda k. 1 + z k)$ 
proof -
  note if_cong [cong] power_Suc [simp del]
  obtain  $N$  where  $N: \bigwedge k. k \geq N \implies \text{norm } (z k) < 1/2$ 
    using summable_LIMSEQ_zero [OF z]
  by (metis diff_zero dist_norm half_gt_zero_iff less_numeral_extra(1) lim_sequentially
tendsto_norm_zero_iff)
  have  $\text{norm } (Ln(1 + z k)) \leq 2 * \text{norm } (z k)$  if  $k \geq N$  for  $k$ 
proof (cases  $z k = 0$ )
  case False
  let ?f =  $\lambda i. \text{cmod } ((-1) ^ i * z k ^ i / \text{of\_nat } (\text{Suc } i))$ 

```

```

have normf: norm (?f n) ≤ (1 / 2) ^ n for n
proof -
  have norm (?f n) = cmod (z k) ^ n / cmod (1 + of_nat n)
    by (auto simp: norm_divide norm_mult norm_power)
  also have ... ≤ cmod (z k) ^ n
    by (auto simp: field_split_simps mult_le_cancel_left1 in_Reals_norm)
  also have ... ≤ (1 / 2) ^ n
    using N [OF that] by (simp add: power_mono)
  finally show norm (?f n) ≤ (1 / 2) ^ n .
qed
have summablef: summable ?f
  by (intro normf summable_comparison_test' [OF summable_geometric [of
1/2]]) auto
  have (λn. (- 1) ^ Suc n / of_nat n * z k ^ n) sums Ln (1 + z k)
    using Ln_series [of z k] N that by fastforce
  then have *: (λi. z k * (((- 1) ^ i * z k ^ i) / (Suc i))) sums Ln (1 + z k)
    using sums_split_initial_segment [where n=1] by (force simp: power_Suc
mult_ac)
  then have norm (Ln (1 + z k)) = norm (suminf (λi. z k * (((- 1) ^ i * z k ^ i) / (Suc i))))
    using sums_unique by force
  also have ... = norm (z k * suminf (λi. ((- 1) ^ i * z k ^ i) / (Suc i)))
    apply (subst suminf_mult)
    using * False
    by (auto simp: sums_summable intro: summable_mult_D [of z k])
  also have ... = norm (z k) * norm (suminf (λi. ((- 1) ^ i * z k ^ i) / (Suc
i)))
    by (simp add: norm_mult)
  also have ... ≤ norm (z k) * suminf (λi. norm (((- 1) ^ i * z k ^ i) / (Suc
i)))
    by (intro mult_left_mono summable_norm summablef) auto
  also have ... ≤ norm (z k) * suminf (λi. (1/2) ^ i)
    by (intro mult_left_mono suminf_le) (use summable_geometric [of 1/2]
summablef normf in auto)
  also have ... ≤ norm (z k) * 2
    using suminf_geometric [of 1/2::real] by simp
  finally show ?thesis
    by (simp add: mult_ac)
qed simp
then have summable (λk. Ln (1 + z k))
  by (metis summable_comparison_test summable_mult z)
with non0 show ?thesis
  by (simp add: add_eq_0_iff convergent_prod_iff summable_complex)
qed

lemma summable_Ln_complex:
  fixes z :: nat ⇒ complex
  assumes convergent_prod z ∧ k. z k ≠ 0
  shows summable (λk. Ln (z k))

```

using *convergent_prod_def* *assms* *convergent_prod_iff_summable_complex* **by** *blast*

6.36.10 Embeddings from the reals into some complete real normed field

lemma *tendsto_eq_of_real_lim*:
assumes $(\lambda n. \text{of\_real } (f\ n) :: 'a :: \{\text{complete\_space}, \text{real\_normed\_field}\}) \longrightarrow q$
shows $q = \text{of\_real } (\lim f)$
proof –
have *convergent* $(\lambda n. \text{of\_real } (f\ n) :: 'a)$
using *assms* *convergent_def* **by** *blast*
then have *convergent* *f*
unfolding *convergent_def*
by (*simp add: convergent_eq_Cauchy Cauchy_def*)
then show *?thesis*
by (*metis LIMSEQ_unique assms convergentD sequentially_bot tendsto_Lim tendsto_of_real*)
qed

lemma *tendsto_eq_of_real*:
assumes $(\lambda n. \text{of\_real } (f\ n) :: 'a :: \{\text{complete\_space}, \text{real\_normed\_field}\}) \longrightarrow q$
obtains *r* **where** $q = \text{of\_real } r$
using *tendsto_eq_of_real_lim* *assms* **by** *blast*

lemma *has_prod_of_real_iff [simp]*:
 $(\lambda n. \text{of\_real } (f\ n) :: 'a :: \{\text{complete\_space}, \text{real\_normed\_field}\}) \text{ has\_prod of\_real } c \longleftrightarrow f \text{ has\_prod } c$
(is ?lhs = ?rhs)
proof
assume *?lhs*
then show *?rhs*
apply (*auto simp: prod_defs LIMSEQ_prod_0 tendsto_of_real_iff simp flip: of_real_prod*)
using *tendsto_eq_of_real*
by (*metis of_real_0 tendsto_of_real_iff*)
next
assume *?rhs*
with *tendsto_of_real_iff* **show** *?lhs*
by (*fastforce simp: prod_defs simp flip: of_real_prod*)
qed
end

6.37 Infinite sums

In this theory, we introduce the definition of infinite sums, i.e., sums ranging over an infinite, potentially uncountable index set with no particular

ordering. (This is different from series. Those are sums indexed by natural numbers, and the order of the index set matters.)

Our definition is quite standard: $s := \sum_{x \in A} f(x)$ is the limit of finite sums $s_F := \sum_{x \in F} f(x)$ for increasing F . That is, s is the limit of the net s_F where F are finite subsets of A ordered by inclusion. We believe that this is the standard definition for such sums. See, e.g., Definition 4.11 in [2]. This definition is quite general: it is well-defined whenever f takes values in some commutative monoid endowed with a Hausdorff topology. (Examples are reals, complex numbers, normed vector spaces, and more.)

```
theory Infinite_Sum
  imports
    Elementary_Topology
    HOL-Library.Extended_Nonnegative_Real
    HOL-Library.Complex_Order
begin
```

6.37.1 Definition and syntax

definition *has_sum* :: $\langle 'a \Rightarrow 'b :: \{comm\_monoid\_add, topological\_space\} \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow bool \rangle$ **where**
 $\langle has\_sum\ f\ A\ x \longleftrightarrow (sum\ f \longrightarrow x)\ (finite\_subsets\_at\_top\ A) \rangle$

definition *summable_on* :: $\langle 'a \Rightarrow 'b :: \{comm\_monoid\_add, topological\_space\} \Rightarrow 'a\ set \Rightarrow bool \rangle$ (**infixr** *summable'_on* 46) **where**
 $f\ summable\_on\ A \longleftrightarrow (\exists x. has\_sum\ f\ A\ x)$

definition *infsum* :: $\langle 'a \Rightarrow 'b :: \{comm\_monoid\_add, t2\_space\} \Rightarrow 'a\ set \Rightarrow 'b \rangle$ **where**
 $infsum\ f\ A = (if\ f\ summable\_on\ A\ then\ Lim\ (finite\_subsets\_at\_top\ A)\ (sum\ f)\ else\ 0)$

abbreviation *abs_summable_on* :: $\langle 'a \Rightarrow 'b :: real\_normed\_vector \rangle \Rightarrow 'a\ set \Rightarrow bool$ (**infixr** *abs'_summable'_on* 46) **where**
 $f\ abs\_summable\_on\ A \equiv (\lambda x. norm\ (f\ x))\ summable\_on\ A$

syntax (ASCII)

$\_infsum :: pttrn \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow 'b :: topological\_comm\_monoid\_add\ ((3INFSUM\ \_ / \_) [0, 51, 10] 10)$

syntax

$\_infsum :: pttrn \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow 'b :: topological\_comm\_monoid\_add\ ((2\sum_{\infty} (\_ / \_) / \_) [0, 51, 10] 10)$

translations — Beware of argument permutation!

$\sum_{i \in A} b \Rightarrow CONST\ infsum\ (\lambda i. b)\ A$

syntax (ASCII)

$\_univinfsum :: pttrn \Rightarrow 'a \Rightarrow 'a\ ((3INFSUM\ \_ / \_) [0, 10] 10)$

syntax

$\_univinfsum :: pttrn \Rightarrow 'a \Rightarrow 'a\ ((2\sum_{\infty} \_ / \_) [0, 10] 10)$

translations

$$\sum_{\infty} x. t \Rightarrow \text{CONST infsum } (\lambda x. t) (\text{CONST UNIV})$$
syntax (ASCII)
$$\_qinfsum :: \text{pttrn} \Rightarrow \text{bool} \Rightarrow 'a \Rightarrow 'a \ ((3\text{INFSUM } \_ \mid / \_./ \_) [0, 0, 10] 10)$$
syntax

$$\_qinfsum :: \text{pttrn} \Rightarrow \text{bool} \Rightarrow 'a \Rightarrow 'a \ ((2\sum_{\infty} \_ \mid (\_) ./ \_) [0, 0, 10] 10)$$
translations

$$\sum_{\infty} x | P. t \Rightarrow \text{CONST infsum } (\lambda x. t) \{x. P\}$$
print_translation <

let

$$\text{fun sum\_tr'} [\text{Abs } (x, Tx, t), \text{Const } (@\{\text{const\_syntax Collect}\}, \_) \$ \text{Abs } (y, Ty, P)] =$$

$$\text{if } x <> y \text{ then raise Match}$$

else

let

$$\text{val } x' = \text{Syntax\_Trans.mark\_bound\_body } (x, Tx);$$

$$\text{val } t' = \text{subst\_bound } (x', t);$$

$$\text{val } P' = \text{subst\_bound } (x', P);$$

in

$$\text{Syntax.const } @\{\text{syntax\_const\_qinfsum}\} \$ \text{Syntax\_Trans.mark\_bound\_abs}$$

$$(x, Tx) \$ P' \$ t'$$

end

$$\mid \text{sum\_tr'} \_ = \text{raise Match};$$

$$\text{in } [(@\{\text{const\_syntax infsum}\}, K \text{ sum\_tr'})] \text{ end}$$

>

6.37.2 General properties**lemma** *infsumI*:
$$\text{fixes } f g :: \langle 'a \Rightarrow 'b :: \{\text{comm\_monoid\_add}, t2\_space\} \rangle$$

$$\text{assumes } \langle \text{has\_sum } f A x \rangle$$

$$\text{shows } \langle \text{infsum } f A = x \rangle$$

$$\text{by } (\text{metis } \text{assms } \text{finite\_subsets\_at\_top\_neq\_bot } \text{infsum\_def } \text{summable\_on\_def } \text{has\_sum\_def } \text{tendsto\_Lim})$$
lemma *infsum_eqI*:
$$\text{fixes } f g :: \langle 'a \Rightarrow 'b :: \{\text{comm\_monoid\_add}, t2\_space\} \rangle$$

$$\text{assumes } \langle x = y \rangle$$

$$\text{assumes } \langle \text{has\_sum } f A x \rangle$$

$$\text{assumes } \langle \text{has\_sum } g B y \rangle$$

$$\text{shows } \langle \text{infsum } f A = \text{infsum } g B \rangle$$

$$\text{by } (\text{metis } \text{assms}(1) \text{ assms}(2) \text{ assms}(3) \text{ finite\_subsets\_at\_top\_neq\_bot } \text{infsum\_def } \text{summable\_on\_def } \text{has\_sum\_def } \text{tendsto\_Lim})$$
lemma *infsum_eqI'*:
$$\text{fixes } f g :: \langle 'a \Rightarrow 'b :: \{\text{comm\_monoid\_add}, t2\_space\} \rangle$$

$$\text{assumes } \langle \bigwedge x. \text{has\_sum } f A x \longleftrightarrow \text{has\_sum } g B x \rangle$$

shows $\langle \text{infsum } f \ A = \text{infsum } g \ B \rangle$
by (*metis* *assms* *infsum_def* *infsum_eqI* *summable_on_def*)

lemma *infsum_not_exists*:
fixes $f :: \langle 'a \Rightarrow 'b :: \{\text{comm\_monoid\_add}, \text{t2\_space}\} \rangle$
assumes $\langle \neg f \text{ summable\_on } A \rangle$
shows $\langle \text{infsum } f \ A = 0 \rangle$
by (*simp* *add: assms infsum_def*)

lemma *summable_iff_has_sum_infsum*: $f \text{ summable\_on } A \longleftrightarrow \text{has\_sum } f \ A$
(*infsum* $f \ A$)
using *infsumI* *summable_on_def* **by** *blast*

lemma *has_sum_infsum[simp]*:
assumes $\langle f \text{ summable\_on } S \rangle$
shows $\langle \text{has\_sum } f \ S \ (\text{infsum } f \ S) \rangle$
using *assms* **by** (*auto* *simp: summable_on_def infsum_def has_sum_def tendsto_Lim*)

lemma *has_sum_cong_neutral*:
fixes $f \ g :: \langle 'a \Rightarrow 'b :: \{\text{comm\_monoid\_add}, \text{topological\_space}\} \rangle$
assumes $\langle \bigwedge x. x \in T - S \implies g \ x = 0 \rangle$
assumes $\langle \bigwedge x. x \in S - T \implies f \ x = 0 \rangle$
assumes $\langle \bigwedge x. x \in S \cap T \implies f \ x = g \ x \rangle$
shows $\text{has\_sum } f \ S \ x \longleftrightarrow \text{has\_sum } g \ T \ x$
proof –
have $\langle \text{eventually } P \ (\text{filtermap } (\text{sum } f) \ (\text{finite\_subsets\_at\_top } S))$
 $= \text{eventually } P \ (\text{filtermap } (\text{sum } g) \ (\text{finite\_subsets\_at\_top } T)) \rangle$ **for** P
proof
assume $\langle \text{eventually } P \ (\text{filtermap } (\text{sum } f) \ (\text{finite\_subsets\_at\_top } S)) \rangle$
then obtain $F0$ **where** $\langle \text{finite } F0 \rangle$ **and** $\langle F0 \subseteq S \rangle$ **and** $F0\_P: \langle \bigwedge F. \text{finite } F$
 $\implies F \subseteq S \implies F \supseteq F0 \implies P \ (\text{sum } f \ F) \rangle$
by (*metis* (*no_types*, *lifting*) *eventually_filtermap* *eventually_finite_subsets_at_top*)
define $F0'$ **where** $\langle F0' = F0 \cap T \rangle$
have [*simp*]: $\langle \text{finite } F0' \rangle$ $\langle F0' \subseteq T \rangle$
by (*simp_all* *add: F0'_def* $\langle \text{finite } F0 \rangle$)
have $\langle P \ (\text{sum } g \ F) \rangle$ **if** $\langle \text{finite } F \rangle$ $\langle F \subseteq T \rangle$ $\langle F \supseteq F0' \rangle$ **for** F
proof –
have $\langle P \ (\text{sum } f \ ((F \cap S) \cup (F0 \cap S))) \rangle$
apply (*rule* $F0\_P$)
using $\langle F0 \subseteq S \rangle$ $\langle \text{finite } F0 \rangle$ **that** **by** *auto*
also have $\langle \text{sum } f \ ((F \cap S) \cup (F0 \cap S)) = \text{sum } g \ F \rangle$
apply (*rule* *sum.mono_neutral_cong*)
using *that* $\langle \text{finite } F0 \rangle$ $F0'_\text{def}$ *assms* **by** *auto*
finally show *?thesis* .
qed
with $\langle F0' \subseteq T \rangle$ $\langle \text{finite } F0' \rangle$ **show** $\langle \text{eventually } P \ (\text{filtermap } (\text{sum } g) \ (\text{finite\_subsets\_at\_top } T)) \rangle$
by (*metis* (*no_types*, *lifting*) *eventually_filtermap* *eventually_finite_subsets_at_top*)

```

next
  assume ⟨eventually P (filtermap (sum g) (finite_subsets_at_top T))⟩
  then obtain F0 where ⟨finite F0⟩ and ⟨F0 ⊆ T⟩ and F0_P: ⟨∧F. finite F
    ⇒ F ⊆ T ⇒ F ⊇ F0 ⇒ P (sum g F)⟩
  by (metis (no_types, lifting) eventually_filtermap eventually_finite_subsets_at_top)
  define F0' where ⟨F0' = F0 ∩ S⟩
  have [simp]: ⟨finite F0'⟩ ⟨F0' ⊆ S⟩
  by (simp_all add: F0'_def ⟨finite F0⟩)
  have ⟨P (sum f F)⟩ if ⟨finite F⟩ ⟨F ⊆ S⟩ ⟨F ⊇ F0'⟩ for F
  proof -
    have ⟨P (sum g ((F∩T) ∪ (F0∩T)))⟩
    apply (rule F0_P)
    using ⟨F0 ⊆ T⟩ ⟨finite F0⟩ that by auto
    also have ⟨sum g ((F∩T) ∪ (F0∩T)) = sum f F⟩
    apply (rule sum.mono_neutral_cong)
    using that ⟨finite F0⟩ F0'_def assms by auto
    finally show ?thesis .
  qed
  with ⟨F0' ⊆ S⟩ ⟨finite F0'⟩ show ⟨eventually P (filtermap (sum f) (finite_subsets_at_top
    S))⟩
  by (metis (no_types, lifting) eventually_filtermap eventually_finite_subsets_at_top)
  qed

  then have tendsto_x: (sum f ⟶ x) (finite_subsets_at_top S) ⟷ (sum g
    ⟶ x) (finite_subsets_at_top T) for x
  by (simp add: le_filter_def filterlim_def)

  then show ?thesis
  by (simp add: has_sum_def)
  qed

lemma summable_on_cong_neutral:
  fixes f g :: 'a ⇒ 'b::{comm_monoid_add, topological_space}
  assumes ⟨∧x. x∈T-S ⇒ g x = 0⟩
  assumes ⟨∧x. x∈S-T ⇒ f x = 0⟩
  assumes ⟨∧x. x∈S∩T ⇒ f x = g x⟩
  shows f summable_on S ⟷ g summable_on T
  using has_sum_cong_neutral[of T S g f, OF assms]
  by (simp add: summable_on_def)

lemma infsum_cong_neutral:
  fixes f g :: 'a ⇒ 'b::{comm_monoid_add, t2_space}
  assumes ⟨∧x. x∈T-S ⇒ g x = 0⟩
  assumes ⟨∧x. x∈S-T ⇒ f x = 0⟩
  assumes ⟨∧x. x∈S∩T ⇒ f x = g x⟩
  shows ⟨infsum f S = infsum g T⟩
  apply (rule infsum_eqI')
  using assms by (rule has_sum_cong_neutral)

```

```

lemma has_sum_cong:
  assumes  $\bigwedge x. x \in A \implies f\ x = g\ x$ 
  shows  $\text{has\_sum}\ f\ A\ x \longleftrightarrow \text{has\_sum}\ g\ A\ x$ 
  using assms by (intro has_sum_cong_neutral) auto

lemma summable_on_cong:
  assumes  $\bigwedge x. x \in A \implies f\ x = g\ x$ 
  shows  $f\ \text{summable\_on}\ A \longleftrightarrow g\ \text{summable\_on}\ A$ 
  by (metis assms summable_on_def has_sum_cong)

lemma infsum_cong:
  assumes  $\bigwedge x. x \in A \implies f\ x = g\ x$ 
  shows  $\text{infsum}\ f\ A = \text{infsum}\ g\ A$ 
  using assms infsum_eqI' has_sum_cong by blast

lemma summable_on_cofin_subset:
  fixes  $f :: 'a \Rightarrow 'b::\text{topological\_ab\_group\_add}$ 
  assumes  $f\ \text{summable\_on}\ A$  and  $[simp]: \text{finite}\ F$ 
  shows  $f\ \text{summable\_on}\ (A - F)$ 
proof -
  from assms(1) obtain  $x$  where  $\text{lim\_f}: (\text{sum}\ f \longrightarrow x)\ (\text{finite\_subsets\_at\_top}\ A)$ 
  unfolding summable_on_def has_sum_def by auto
  define  $F'$  where  $F' = F \cap A$ 
  with assms have  $\text{finite}\ F'$  and  $A - F = A - F'$ 
  by auto
  have  $\text{filtermap}\ ((\cup)F')\ (\text{finite\_subsets\_at\_top}\ (A - F))$ 
     $\leq \text{finite\_subsets\_at\_top}\ A$ 
  proof (rule filter_leI)
    fix  $P$  assume eventually  $P\ (\text{finite\_subsets\_at\_top}\ A)$ 
    then obtain  $X$  where  $[simp]: \text{finite}\ X$  and  $XA: X \subseteq A$ 
      and  $P: \forall Y. \text{finite}\ Y \wedge X \subseteq Y \wedge Y \subseteq A \longrightarrow P\ Y$ 
    unfolding eventually_finite_subsets_at_top by auto
    define  $X'$  where  $X' = X - F$ 
    hence  $[simp]: \text{finite}\ X'$  and  $[simp]: X' \subseteq A - F$ 
    using  $XA$  by auto
    hence  $\text{finite}\ Y \wedge X' \subseteq Y \wedge Y \subseteq A - F \longrightarrow P\ (F' \cup Y)$  for  $Y$ 
    using  $P\ XA$  unfolding  $X'_\text{def}$  using  $F'_\text{def}$   $\langle \text{finite}\ F' \rangle$  by blast
    thus eventually  $P\ (\text{filtermap}\ ((\cup)F')\ (\text{finite\_subsets\_at\_top}\ (A - F)))$ 
    unfolding eventually_filtermap_eventually_finite_subsets_at_top
    by (rule_tac  $x = X'$  in exI, simp)
  qed
  with  $\text{lim\_f}$  have  $(\text{sum}\ f \longrightarrow x)\ (\text{filtermap}\ ((\cup)F')\ (\text{finite\_subsets\_at\_top}\ (A - F)))$ 
  using tendsto_mono by blast
  have  $(\lambda G. \text{sum}\ f\ (F' \cup G)) \longrightarrow x\ (\text{finite\_subsets\_at\_top}\ (A - F))$ 
  if  $(\text{sum}\ f \circ (\cup)F') \longrightarrow x\ (\text{finite\_subsets\_at\_top}\ (A - F))$ 
  using that unfolding o_def by auto
  hence  $(\lambda G. \text{sum}\ f\ (F' \cup G)) \longrightarrow x\ (\text{finite\_subsets\_at\_top}\ (A - F))$ 

```

```

using tendsto_compose_filtermap [symmetric]
by (simp add: ⟨(sum f ⟶ x) (filtermap (( $\cup$ ) F') (finite_subsets_at_top (A - F)))⟩
    tendsto_compose_filtermap)
have  $\forall Y. \text{finite } Y \wedge Y \subseteq A - F \longrightarrow \text{sum } f \ (F' \cup Y) = \text{sum } f \ F' + \text{sum } f \ Y$ 
by (metis Diff_disjoint Int_Diff ⟨A - F = A - F'⟩ ⟨finite F'⟩ inf.orderE
    sum.union_disjoint)
hence  $\forall_F x \text{ in } \text{finite\_subsets\_at\_top } (A - F). \text{sum } f \ (F' \cup x) = \text{sum } f \ F' + \text{sum } f \ x$ 
unfolding eventually_finite_subsets_at_top
using exI [where x = {}]
by (simp add: ⟨ $\bigwedge P. P \ \{\} \implies \exists x. P \ x$ ⟩)
hence  $((\lambda G. \text{sum } f \ F' + \text{sum } f \ G) \longrightarrow x) (\text{finite\_subsets\_at\_top } (A - F))$ 
using tendsto_cong [THEN iffD1, rotated]
    ⟨ $((\lambda G. \text{sum } f \ (F' \cup G)) \longrightarrow x) (\text{finite\_subsets\_at\_top } (A - F))$ ⟩ by
fastforce
hence  $((\lambda G. \text{sum } f \ F' + \text{sum } f \ G) \longrightarrow \text{sum } f \ F' + (x - \text{sum } f \ F')) (\text{finite\_subsets\_at\_top } (A - F))$ 
by simp
hence  $(\text{sum } f \longrightarrow x - \text{sum } f \ F') (\text{finite\_subsets\_at\_top } (A - F))$ 
using tendsto_add_const_iff by blast
thus f summable_on (A - F)
unfolding summable_on_def has_sum_def by auto
qed

```

lemma

```

fixes f :: 'a  $\Rightarrow$  'b::{topological_ab_group_add}
assumes ⟨has_sum f B b⟩ and ⟨has_sum f A a⟩ and AB: A  $\subseteq$  B
shows has_sum_Diff: has_sum f (B - A) (b - a)
proof -
have finite_subsets1:
    finite_subsets_at_top (B - A)  $\leq$  filtermap ( $\lambda F. F - A$ ) (finite_subsets_at_top B)
proof (rule filter_leI)
fix P assume eventually P (filtermap ( $\lambda F. F - A$ ) (finite_subsets_at_top B))
then obtain X where finite X and X  $\subseteq$  B
and P: finite Y  $\wedge$  X  $\subseteq$  Y  $\wedge$  Y  $\subseteq$  B  $\longrightarrow$  P (Y - A) for Y
unfolding eventually_filtermap eventually_finite_subsets_at_top by auto

hence finite (X - A) and X - A  $\subseteq$  B - A
by auto
moreover have finite Y  $\wedge$  X - A  $\subseteq$  Y  $\wedge$  Y  $\subseteq$  B - A  $\longrightarrow$  P Y for Y
using P[where Y=Y $\cup$ X] ⟨finite X⟩ ⟨X  $\subseteq$  B⟩
by (metis Diff_subset Int_Diff Un_Diff finite_Un inf.orderE le_sup_iff
    sup.orderE sup_ge2)
ultimately show eventually P (finite_subsets_at_top (B - A))
unfolding eventually_finite_subsets_at_top by meson
qed
have finite_subsets2:

```

```

    filtermap ( $\lambda F. F \cap A$ ) (finite_subsets_at_top B)  $\leq$  finite_subsets_at_top A
  apply (rule filter_leI)
  using assms unfolding eventually_filtermap eventually_finite_subsets_at_top
    by (metis Int_subset_iff finite_Int inf_le2 subset_trans)

  from assms(1) have limB: ( $\text{sum } f \longrightarrow b$ ) (finite_subsets_at_top B)
    using has_sum_def by auto
  from assms(2) have limA: ( $\text{sum } f \longrightarrow a$ ) (finite_subsets_at_top A)
    using has_sum_def by blast
  have (( $\lambda F. \text{sum } f (F \cap A)$ )  $\longrightarrow a$ ) (finite_subsets_at_top B)
  proof (subst asm_rl [of ( $\lambda F. \text{sum } f (F \cap A)$ ) =  $\text{sum } f \circ (\lambda F. F \cap A)$ ])
    show ( $\lambda F. \text{sum } f (F \cap A)$ ) =  $\text{sum } f \circ (\lambda F. F \cap A)$ 
      unfolding o_def by auto
    show (( $\text{sum } f \circ (\lambda F. F \cap A)$ )  $\longrightarrow a$ ) (finite_subsets_at_top B)
      unfolding o_def
      using tendsto_compose_filtermap finite_subsets2 limA tendsto_mono
         $\langle (\lambda F. \text{sum } f (F \cap A)) = \text{sum } f \circ (\lambda F. F \cap A) \rangle$  by fastforce
  qed

  with limB have (( $\lambda F. \text{sum } f F - \text{sum } f (F \cap A)$ )  $\longrightarrow b - a$ ) (finite_subsets_at_top B)
    using tendsto_diff by blast
  have  $\text{sum } f X - \text{sum } f (X \cap A) = \text{sum } f (X - A)$  if finite X and  $X \subseteq B$  for
  X :: 'a set
    using that by (metis add_diff_cancel_left' sum.Int_Diff)
  hence  $\forall F. x \text{ in finite\_subsets\_at\_top } B. \text{sum } f x - \text{sum } f (x \cap A) = \text{sum } f (x - A)$ 
    by (rule eventually_finite_subsets_at_top_weakI)
  hence (( $\lambda F. \text{sum } f (F - A)$ )  $\longrightarrow b - a$ ) (finite_subsets_at_top B)
    using tendsto_cong [THEN iffD1, rotated]
       $\langle (\lambda F. \text{sum } f F - \text{sum } f (F \cap A)) \longrightarrow b - a \rangle$  (finite_subsets_at_top B)
  by fastforce
  hence ( $\text{sum } f \longrightarrow b - a$ ) (filtermap ( $\lambda F. F - A$ ) (finite_subsets_at_top B))
    by (subst tendsto_compose_filtermap[symmetric], simp add: o_def)
  hence limBA: ( $\text{sum } f \longrightarrow b - a$ ) (finite_subsets_at_top (B - A))
    apply (rule tendsto_mono[rotated])
    by (rule finite_subsets1)
  thus ?thesis
    by (simp add: has_sum_def)
  qed

```

lemma

```

  fixes f :: 'a  $\Rightarrow$  'b::{topological_ab_group_add}
  assumes f_summable_on B and f_summable_on A and  $A \subseteq B$ 
  shows summable_on_Diff: f_summable_on (B - A)
  by (meson assms summable_on_def has_sum_Diff)

```

lemma

```

fixes f :: 'a ⇒ 'b::{topological_ab_group_add,t2_space}
assumes f summable_on B and f summable_on A and AB: A ⊆ B
shows infsum_Diff: infsum f (B - A) = infsum f B - infsum f A
by (metis AB assms has_sum_Diff infsumI summable_on_def)

```

lemma *has_sum_mono_neutral*:

```

fixes f :: 'a⇒'b::{ordered_comm_monoid_add,linorder_topology}

```

```

assumes ⟨has_sum f A a⟩ and has_sum g B b
assumes ⟨⋀x. x ∈ A∩B ⇒ f x ≤ g x⟩
assumes ⟨⋀x. x ∈ A-B ⇒ f x ≤ 0⟩
assumes ⟨⋀x. x ∈ B-A ⇒ g x ≥ 0⟩
shows a ≤ b

```

proof –

```

define f' g' where ⟨f' x = (if x ∈ A then f x else 0)⟩ and ⟨g' x = (if x ∈ B
then g x else 0)⟩ for x

```

```

have [simp]: ⟨f summable_on A⟩ ⟨g summable_on B⟩
using assms(1,2) summable_on_def by auto
have ⟨has_sum f' (A∪B) a⟩
apply (subst has_sum_cong_neutral[where g=f and T=A])
by (auto simp: f'_def assms(1))
then have f'_lim: ⟨(sum f' ⟶ a) (finite_subsets_at_top (A∪B))⟩
by (meson has_sum_def)
have ⟨has_sum g' (A∪B) b⟩
apply (subst has_sum_cong_neutral[where g=g and T=B])
by (auto simp: g'_def assms(2))
then have g'_lim: ⟨(sum g' ⟶ b) (finite_subsets_at_top (A∪B))⟩
using has_sum_def by blast

```

```

have *: ⟨∀F x in finite_subsets_at_top (A ∪ B). sum f' x ≤ sum g' x⟩
apply (rule eventually_finite_subsets_at_top_weakI)
apply (rule sum_mono)
using assms by (auto simp: f'_def g'_def)
show ?thesis
apply (rule tendsto_le)
using * g'_lim f'_lim by auto

```

qed

lemma *infsum_mono_neutral*:

```

fixes f :: 'a⇒'b::{ordered_comm_monoid_add,linorder_topology}

```

```

assumes f summable_on A and g summable_on B

```

```

assumes ⟨⋀x. x ∈ A∩B ⇒ f x ≤ g x⟩

```

```

assumes ⟨⋀x. x ∈ A-B ⇒ f x ≤ 0⟩

```

```

assumes ⟨⋀x. x ∈ B-A ⇒ g x ≥ 0⟩

```

```

shows infsum f A ≤ infsum g B

```

```

by (rule has_sum_mono_neutral[of f A _ g B _]) (use assms in ⟨auto intro:
has_sum_infsum⟩)

```

lemma *has_sum_mono*:


```

fixes f :: 'a ⇒ 'b :: {ordered_comm_monoid_add, linorder_topology}
assumes has_sum f A x and has_sum g A y
assumes ⟨ $\bigwedge x. x \in A \implies f x \leq g x$ ⟩
shows x ≤ y
apply (rule has_sum_mono_neutral)
using assms by auto

```

```

lemma infsum_mono:
fixes f :: 'a ⇒ 'b :: {ordered_comm_monoid_add, linorder_topology}
assumes f summable_on A and g summable_on A
assumes ⟨ $\bigwedge x. x \in A \implies f x \leq g x$ ⟩
shows infsum f A ≤ infsum g A
apply (rule infsum_mono_neutral)
using assms by auto

```

```

lemma has_sum_finite[simp]:
assumes finite F
shows has_sum f F (sum f F)
using assms
by (auto intro: tendsto_Lim simp: finite_subsets_at_top_finite infsum_def has_sum_def
principal_eq_bot_iff)

```

```

lemma summable_on_finite[simp]:
fixes f :: 'a ⇒ 'b :: {comm_monoid_add, topological_space}
assumes finite F
shows f summable_on F
using assms summable_on_def has_sum_finite by blast

```

```

lemma infsum_finite[simp]:
assumes finite F
shows infsum f F = sum f F
using assms by (auto intro: tendsto_Lim simp: finite_subsets_at_top_finite
infsum_def principal_eq_bot_iff)

```

```

lemma has_sum_finite_approximation:
fixes f :: 'a ⇒ 'b :: {comm_monoid_add, metric_space}
assumes has_sum f A x and ε > 0
shows ∃ F. finite F ∧ F ⊆ A ∧ dist (sum f F) x ≤ ε
proof –
have (sum f ⟶ x) (finite_subsets_at_top A)
by (meson assms(1) has_sum_def)
hence *: ∀ F. F in (finite_subsets_at_top A). dist (sum f F) x < ε
using assms(2) by (rule tendstoD)
thus ?thesis
unfolding eventually_finite_subsets_at_top by fastforce
qed

```

```

lemma infsum_finite_approximation:
fixes f :: 'a ⇒ 'b :: {comm_monoid_add, metric_space}

```

```

    assumes  $f$  summable_on  $A$  and  $\varepsilon > 0$ 
    shows  $\exists F. \text{finite } F \wedge F \subseteq A \wedge \text{dist } (\text{sum } f F) (\text{infsum } f A) \leq \varepsilon$ 
  proof -
    from assms have has_sum  $f A$  (infsum  $f A$ )
    by (simp add: summable_iff_has_sum_infsum)
    from this and  $\langle \varepsilon > 0 \rangle$  show ?thesis
    by (rule has_sum_finite_approximation)
  qed

lemma abs_summable_summable:
  fixes  $f :: \langle 'a \Rightarrow 'b :: \text{banach} \rangle$ 
  assumes  $\langle f \text{ abs\_summable\_on } A \rangle$ 
  shows  $\langle f \text{ summable\_on } A \rangle$ 
  proof -
    from assms obtain  $L$  where  $\text{lim}: \langle (\text{sum } (\lambda x. \text{norm } (f x)) \longrightarrow L) (\text{finite\_subsets\_at\_top } A) \rangle$ 
    unfolding has_sum_def summable_on_def by blast
    then have *:  $\langle \text{cauchy\_filter } (\text{filtermap } (\text{sum } (\lambda x. \text{norm } (f x)))) (\text{finite\_subsets\_at\_top } A) \rangle$ 
    by (auto intro!: nhds_imp_cauchy_filter simp: filterlim_def)
    have  $\langle \exists P. \text{eventually } P (\text{finite\_subsets\_at\_top } A) \wedge$ 
       $(\forall F F'. P F \wedge P F' \longrightarrow \text{dist } (\text{sum } f F) (\text{sum } f F') < e) \rangle$  if  $\langle e > 0 \rangle$  for  $e$ 
    proof -
      define  $d$  where  $\langle d = e/4 \rangle$  and  $\langle P F \longleftrightarrow \text{finite } F \wedge F \subseteq A \wedge \text{dist } (\text{sum } (\lambda x. \text{norm } (f x)) F) L < d \rangle$  for  $F$ 
      then have  $\langle d > 0 \rangle$ 
      by (simp add: d_def that)
      have  $\text{ev\_}P: \langle \text{eventually } P (\text{finite\_subsets\_at\_top } A) \rangle$ 
      using lim
      by (auto simp add: P_def[abs_def]  $\langle 0 < d \rangle$  eventually_conj_iff eventually_finite_subsets_at_top_weakI tendsto_iff)

      moreover have  $\langle \text{dist } (\text{sum } f F1) (\text{sum } f F2) < e \rangle$  if  $\langle P F1 \rangle$  and  $\langle P F2 \rangle$  for  $F1 F2$ 
      proof -
        from ev_P
        obtain  $F'$  where  $\langle \text{finite } F' \rangle$  and  $\langle F' \subseteq A \rangle$  and  $P\_sup\_F': \langle \text{finite } F \wedge F \supseteq F' \wedge F \subseteq A \implies P F \rangle$  for  $F$ 
        by atomize_elim (simp add: eventually_finite_subsets_at_top)
        define  $F$  where  $\langle F = F' \cup F1 \cup F2 \rangle$ 
        have  $\langle \text{finite } F \rangle$  and  $\langle F \subseteq A \rangle$ 
        using F_def P_def[abs_def] that  $\langle \text{finite } F' \rangle \langle F' \subseteq A \rangle$  by auto
        have  $\text{dist\_}F: \langle \text{dist } (\text{sum } (\lambda x. \text{norm } (f x)) F) L < d \rangle$ 
        by (metis F_def  $\langle F \subseteq A \rangle P\_def P\_sup\_F' \langle \text{finite } F \rangle \text{le\_supE order_refl}$ )

        have  $\text{dist\_}F\_subset: \langle \text{dist } (\text{sum } f F) (\text{sum } f F') < 2*d \rangle$  if  $F': \langle F' \subseteq F \rangle \langle P F' \rangle$  for  $F'$ 
        proof -
          have  $\langle \text{dist } (\text{sum } f F) (\text{sum } f F') = \text{norm } (\text{sum } f (F - F')) \rangle$ 

```

```

    unfolding dist_norm using ‹finite F› F' by (subst sum_diff) auto
  also have ‹... ≤ norm (∑ x∈F-F'. norm (f x))›
    by (rule order.trans[OF sum_norm_le[OF order.refl]]) auto
  also have ‹... = dist (∑ x∈F. norm (f x)) (∑ x∈F'. norm (f x))›
    unfolding dist_norm using ‹finite F› F' by (subst sum_diff) auto
  also have ‹... < 2 * d›
    using dist_F F' unfolding P_def dist_norm real_norm_def by linarith
  finally show ‹dist (sum f F) (sum f F') < 2*d› .
qed

  have ‹dist (sum f F1) (sum f F2) ≤ dist (sum f F) (sum f F1) + dist (sum
f F) (sum f F2)›
    by (rule dist_triangle3)
  also have ‹... < 2 * d + 2 * d›
    by (intro add_strict_mono dist_F_subset that) (auto simp: F_def)
  also have ‹... ≤ e›
    by (auto simp: d_def)
  finally show ‹dist (sum f F1) (sum f F2) < e› .
qed
then show ?thesis
  using ev_P by blast
qed
then have ‹cauchy_filter (filtermap (sum f) (finite_subsets_at_top A))›
  by (simp add: cauchy_filter_metric_filtermap)
then obtain L' where ‹(sum f → L') (finite_subsets_at_top A)›
  apply atomize_elim unfolding filterlim_def
  apply (rule complete_uniform[where S=UNIV, simplified, THEN iffD1,
rule_format])
  apply (auto simp add: filtermap_bot_iff)
  by (meson Cauchy_convergent UNIV_I complete_def convergent_def)
then show ?thesis
  using summable_on_def has_sum_def by blast
qed

```

The converse of *abs_summable_summable* does not hold: Consider the Hilbert space of square-summable sequences. Let e_i denote the sequence with 1 in the i th position and 0 elsewhere. Let $f(i) := e_i/i$ for $i \geq 1$. We have $\neg f \text{ abs\_summable\_on } UNIV$ because $\|f(i)\| = 1/i$ and thus the sum over $\|f(i)\|$ diverges. On the other hand, we have $f \text{ summable\_on } UNIV$; the limit is the sequence with $1/i$ in the i th position.

(We have not formalized this separating example here because to the best of our knowledge, this Hilbert space has not been formalized in Isabelle/HOL yet.)

```

lemma norm_has_sum_bound:
  fixes f :: 'b ⇒ 'a::real_normed_vector
  and A :: 'b set
  assumes has_sum (λx. norm (f x)) A n
  assumes has_sum f A a

```

```

shows norm a ≤ n
proof -
  have norm a ≤ n + ε if ε > 0 for ε
  proof-
    have ∃ F. norm (a - sum f F) ≤ ε ∧ finite F ∧ F ⊆ A
    using has_sum_finite_approximation[where A=A and f=f and ε=ε] assms
  <0 < ε>
    by (metis dist_commute dist_norm)
  then obtain F where norm (a - sum f F) ≤ ε
    and finite F and F ⊆ A
    by (simp add: atomize_elim)
  hence norm a ≤ norm (sum f F) + ε
  by (metis add_commute diff_add_cancel dual_order.refl norm_triangle_mono)
  also have ... ≤ sum (λx. norm (f x)) F + ε
    using norm_sum by auto
  also have ... ≤ n + ε
    apply (rule add_right_mono)
    apply (rule has_sum_mono_neutral[where A=F and B=A and f=λx.
norm (f x) and g=λx. norm (f x)])
    using <finite F> <F ⊆ A> assms by auto
  finally show ?thesis
    by assumption
qed
thus ?thesis
  using linordered_field_class.field_le_epsilon by blast
qed

```

lemma norm_infsum_bound:

```

fixes f :: 'b ⇒ 'a::real_normed_vector
and A :: 'b set
assumes f_abs_summable_on A
shows norm (infsum f A) ≤ infsum (λx. norm (f x)) A
proof (cases f_summable_on A)
case True
show ?thesis
  apply (rule norm_has_sum_bound[where A=A and f=f and a=⟨infsum f
A⟩ and n=⟨infsum (λx. norm (f x)) A⟩])
  using assms True
  by (metis finite_subsets_at_top_neq_bot infsum_def summable_on_def has_sum_def
tendsto_Lim)+
next
case False
  obtain t where t_def: (sum (λx. norm (f x)) ⟶ t) (finite_subsets_at_top
A)
  using assms unfolding summable_on_def has_sum_def by blast
  have sumpos: sum (λx. norm (f x)) X ≥ 0
  for X
  by (simp add: sum_nonneg)
  have tgeq0: t ≥ 0

```

```

proof(rule ccontr)
  define  $S::\text{real set}$  where  $S = \{s. s < 0\}$ 
  assume  $\neg 0 \leq t$ 
  hence  $t < 0$  by simp
  hence  $t \in S$ 
  unfolding  $S\_def$  by blast
  moreover have open  $S$ 
  proof–
    have closed  $\{s::\text{real}. s \geq 0\}$ 
    using Elementary_Topology.closed_sequential_limits[where  $S = \{s::\text{real}.$ 
 $s \geq 0\}$ ]
    by (metis Lim_bounded2 mem_Collect_eq)
    moreover have  $\{s::\text{real}. s \geq 0\} = \text{UNIV} - S$ 
    unfolding  $S\_def$  by auto
    ultimately have closed  $(\text{UNIV} - S)$ 
    by simp
    thus ?thesis
    by (simp add: Compl_eq_Diff_UNIV open_closed)
  qed
  ultimately have  $\forall_F X$  in finite_subsets_at_top  $A. (\sum_{x \in X}. \text{norm } (f x)) \in S$ 
  using  $t\_def$  unfolding tendsto_def by blast
  hence  $\exists X. (\sum_{x \in X}. \text{norm } (f x)) \in S$ 
  by (metis (no_types, lifting) eventually_mono filterlim_iff finite_subsets_at_top_neq_bot
tendsto_Lim)
  then obtain  $X$  where  $(\sum_{x \in X}. \text{norm } (f x)) \in S$ 
  by blast
  hence  $(\sum_{x \in X}. \text{norm } (f x)) < 0$ 
  unfolding  $S\_def$  by auto
  thus False by (simp add: leD sumpos)
qed
  have  $\exists! h. (\text{sum } (\lambda x. \text{norm } (f x)) \longrightarrow h) (\text{finite\_subsets\_at\_top } A)$ 
  using  $t\_def$  finite_subsets_at_top_neq_bot tendsto_unique by blast
  hence  $t = (\text{Topological\_Spaces.Lim } (\text{finite\_subsets\_at\_top } A) (\text{sum } (\lambda x. \text{norm } (f x))))$ 
  using  $t\_def$  unfolding Topological_Spaces.Lim_def
  by (metis the_equality)
  hence  $\text{Lim } (\text{finite\_subsets\_at\_top } A) (\text{sum } (\lambda x. \text{norm } (f x))) \geq 0$ 
  using tgeq0 by blast
  thus ?thesis unfolding infsum_def
  using False by auto
qed

lemma infsum_tendsto:
  assumes  $\langle f \text{ summable\_on } S \rangle$ 
  shows  $\langle (\lambda F. \text{sum } f F) \longrightarrow \text{infsum } f S \rangle (\text{finite\_subsets\_at\_top } S)$ 
  using assms by (simp flip: has_sum_def)

```

lemma has_sum_0:

```

assumes  $\langle \bigwedge x. x \in M \implies f\ x = 0 \rangle$ 
shows  $\langle \text{has\_sum } f\ M\ 0 \rangle$ 
unfolding has_sum_def
apply (subst tendsto_cong [where  $g = \langle \lambda_. 0 \rangle$ ])
apply (rule eventually_finite_subsets_at_top_weakI)
using assms by (auto simp add: subset_iff)

```

```

lemma summable_on_0:
assumes  $\langle \bigwedge x. x \in M \implies f\ x = 0 \rangle$ 
shows  $\langle f\ \text{summable\_on } M \rangle$ 
using assms summable_on_def has_sum_0 by blast

```

```

lemma infsum_0:
assumes  $\langle \bigwedge x. x \in M \implies f\ x = 0 \rangle$ 
shows  $\langle \text{infsum } f\ M = 0 \rangle$ 
by (metis assms finite_subsets_at_top_neq_bot infsum_def has_sum_0 has_sum_def
tendsto_Lim)

```

Variants of *infsum_0* etc. suitable as simp-rules

```

lemma infsum_0_simp [simp]:  $\langle \text{infsum } (\lambda_. 0)\ M = 0 \rangle$ 
by (simp_all add: infsum_0)
lemma summable_on_0_simp [simp]:  $\langle (\lambda_. 0)\ \text{summable\_on } M \rangle$ 
by (simp_all add: summable_on_0)
lemma has_sum_0_simp [simp]:  $\langle \text{has\_sum } (\lambda_. 0)\ M\ 0 \rangle$ 
by (simp_all add: has_sum_0)

```

```

lemma has_sum_add:
fixes  $f\ g :: 'a \Rightarrow 'b :: \{\text{topological\_comm\_monoid\_add}\}$ 
assumes  $\langle \text{has\_sum } f\ A\ a \rangle$ 
assumes  $\langle \text{has\_sum } g\ A\ b \rangle$ 
shows  $\langle \text{has\_sum } (\lambda x. f\ x + g\ x)\ A\ (a + b) \rangle$ 
proof –
from assms have lim_f:  $\langle (\text{sum } f \longrightarrow a)\ (\text{finite\_subsets\_at\_top } A) \rangle$ 
and lim_g:  $\langle (\text{sum } g \longrightarrow b)\ (\text{finite\_subsets\_at\_top } A) \rangle$ 
by (simp_all add: has_sum_def)
then have lim:  $\langle (\text{sum } (\lambda x. f\ x + g\ x) \longrightarrow a + b)\ (\text{finite\_subsets\_at\_top } A) \rangle$ 
unfolding sum.distrib by (rule tendsto_add)
then show ?thesis
by (simp_all add: has_sum_def)
qed

```

```

lemma summable_on_add:
fixes  $f\ g :: 'a \Rightarrow 'b :: \{\text{topological\_comm\_monoid\_add}\}$ 
assumes  $\langle f\ \text{summable\_on } A \rangle$ 
assumes  $\langle g\ \text{summable\_on } A \rangle$ 
shows  $\langle (\lambda x. f\ x + g\ x)\ \text{summable\_on } A \rangle$ 
by (metis (full_types) assms(1) assms(2) summable_on_def has_sum_add)

```

```

lemma infsum_add:
  fixes f g :: 'a  $\Rightarrow$  'b::{topological_comm_monoid_add, t2_space}
  assumes <f summable_on A>
  assumes <g summable_on A>
  shows <infsum ( $\lambda x. f x + g x$ ) A = infsum f A + infsum g A>
proof -
  have <has_sum ( $\lambda x. f x + g x$ ) A (infsum f A + infsum g A)>
    by (simp add: assms(1) assms(2) has_sum_add)
  then show ?thesis
    using infsumI by blast
qed

```

```

lemma has_sum_Un_disjoint:
  fixes f :: 'a  $\Rightarrow$  'b::topological_comm_monoid_add
  assumes has_sum f A a
  assumes has_sum f B b
  assumes disj:  $A \cap B = \{\}$ 
  shows <has_sum f (A  $\cup$  B) (a + b)>
proof -
  define fA fB where <fA x = (if x  $\in$  A then f x else 0)>
    and <fB x = (if x  $\notin$  A then f x else 0)> for x
  have fA: <has_sum fA (A  $\cup$  B) a>
    apply (subst has_sum_cong_neutral[where T=A and g=f])
    using assms by (auto simp: fA_def)
  have fB: <has_sum fB (A  $\cup$  B) b>
    apply (subst has_sum_cong_neutral[where T=B and g=f])
    using assms by (auto simp: fB_def)
  have fAB: <f x = fA x + fB x> for x
    unfolding fA_def fB_def by simp
  show ?thesis
    unfolding fAB
    using fA fB by (rule has_sum_add)
qed

```

```

lemma summable_on_Un_disjoint:
  fixes f :: 'a  $\Rightarrow$  'b::topological_comm_monoid_add
  assumes f summable_on A
  assumes f summable_on B
  assumes disj:  $A \cap B = \{\}$ 
  shows <f summable_on (A  $\cup$  B)>
  by (meson assms(1) assms(2) disj summable_on_def has_sum_Un_disjoint)

```

```

lemma infsum_Un_disjoint:
  fixes f :: 'a  $\Rightarrow$  'b::{topological_comm_monoid_add, t2_space}
  assumes f summable_on A
  assumes f summable_on B
  assumes disj:  $A \cap B = \{\}$ 
  shows <infsum f (A  $\cup$  B) = infsum f A + infsum f B>

```

by (intro infsumI has_sum Un_disjoint has_sum infsum assms)

lemma (in uniform_space) *cauchy_filter_complete_converges*:
assumes *cauchy_filter F complete A F ≤ principal A F ≠ bot*
shows $\exists c. F \leq \text{nhds } c$
using *assms unfolding complete_uniform by blast*

The following lemma indeed needs a complete space (as formalized by the premise *complete UNIV*). The following two counterexamples show this:

- Consider the real vector space V of sequences with finite support, and with the ℓ_2 -norm (sum of squares). Let e_i denote the sequence with a 1 at position i . Let $f : \mathbb{Z} \rightarrow V$ be defined as $f(n) := e_{|n|}/n$ (with $f(0) := 0$). We have that $\sum_{n \in \mathbb{Z}} f(n) = 0$ (it even converges absolutely). But $\sum_{n \in \mathbb{N}} f(n)$ does not exist (it would converge against a sequence with infinite support).
- Let f be a positive rational valued function such that $\sum_{x \in B} f(x)$ is $\sqrt{2}$ and $\sum_{x \in A} f(x)$ is 1 (over the reals, with $A \subseteq B$). Then $\sum_{x \in B} f(x)$ does not exist over the rationals. But $\sum_{x \in A} f(x)$ exists.

The lemma also requires uniform continuity of the addition. And example of a topological group with continuous but not uniformly continuous addition would be the positive reals with the usual multiplication as the addition. We do not know whether the lemma would also hold for such topological groups.

lemma *summable_on_subset*:
fixes $A B$ **and** $f :: \langle 'a \Rightarrow 'b :: \{ab\_group\_add, uniform\_space\} \rangle$
assumes $\langle complete (UNIV :: 'b \text{ set}) \rangle$
assumes *plus_cont*: $\langle uniformly\_continuous\_on UNIV (\lambda(x::'b,y). x+y) \rangle$
assumes $\langle f \text{ summable\_on } A \rangle$
assumes $\langle B \subseteq A \rangle$
shows $\langle f \text{ summable\_on } B \rangle$
proof –
let $?filter\_fB = \langle filtermap (sum f) (finite\_subsets\_at\_top B) \rangle$
from $\langle f \text{ summable\_on } A \rangle$
obtain S **where** $\langle (sum f \longrightarrow S) (finite\_subsets\_at\_top A) \rangle$ **(is** $\langle (sum f \longrightarrow S) ?filter\_A \rangle$ **)**
using *summable_on_def has_sum_def by blast*
then have *cauchy_fA*: $\langle cauchy\_filter (filtermap (sum f) (finite\_subsets\_at\_top A)) \rangle$ **(is** $\langle cauchy\_filter ?filter\_fA \rangle$ **)**
by (auto intro!: nhds_imp_cauchy_filter simp: filterlim_def)

have $\langle cauchy\_filter (filtermap (sum f) (finite\_subsets\_at\_top B)) \rangle$
proof (unfold *cauchy_filter_def*, rule *filter_leI*)
fix $E :: \langle ('b \times 'b) \Rightarrow bool \rangle$ **assume** $\langle eventually E \text{ uniformity} \rangle$


```

then obtain  $E'$  where  $\langle \text{eventually } E' \text{ uniformity} \rangle$  and  $E'E'E$ :  $\langle E' (x, y) \longrightarrow$ 
 $E' (y, z) \longrightarrow E (x, z) \rangle$  for  $x \ y \ z$ 
using uniformity_trans by blast
obtain  $D$  where  $\langle \text{eventually } D \text{ uniformity} \rangle$  and  $DE$ :  $\langle D (x, y) \implies E' (x+c,$ 
 $y+c) \rangle$  for  $x \ y \ c$ 
using plus_cont  $\langle \text{eventually } E' \text{ uniformity} \rangle$ 
unfolding uniformly_continuous_on_uniformity filterlim_def le_filter_def
uniformity_prod_def
by (auto simp: case_prod_beta eventually_filtermap eventually_prod_same
uniformity_refl)
have  $DE'$ :  $E' (x, y)$  if  $D (x + c, y + c)$  for  $x \ y \ c$ 
using DE[of x + c y + c - c] that by simp

from  $\langle \text{eventually } D \text{ uniformity} \rangle$  and cauchy_fA have  $\langle \text{eventually } D (\text{?filter\_fA}$ 
 $\times_F \text{?filter\_fA}) \rangle$ 
unfolding cauchy_filter_def le_filter_def by simp
then obtain  $P1 \ P2$ 
where  $ev\_P1$ :  $\langle \text{eventually } (\lambda F. P1 \ (\text{sum } f \ F)) \ \text{?filter\_A} \rangle$ 
and  $ev\_P2$ :  $\langle \text{eventually } (\lambda F. P2 \ (\text{sum } f \ F)) \ \text{?filter\_A} \rangle$ 
and  $P1P2E$ :  $\langle P1 \ x \implies P2 \ y \implies D (x, y) \rangle$  for  $x \ y$ 
unfolding eventually_prod_filter eventually_filtermap
by auto
from  $ev\_P1$  obtain  $F1$  where  $F1$ :  $\langle \text{finite } F1 \rangle \ \langle F1 \subseteq A \rangle \ \langle \bigwedge F. F \supseteq F1 \implies$ 
 $\text{finite } F \implies F \subseteq A \implies P1 \ (\text{sum } f \ F) \rangle$ 
by (metis eventually_finite_subsets_at_top)
from  $ev\_P2$  obtain  $F2$  where  $F2$ :  $\langle \text{finite } F2 \rangle \ \langle F2 \subseteq A \rangle \ \langle \bigwedge F. F \supseteq F2 \implies$ 
 $\text{finite } F \implies F \subseteq A \implies P2 \ (\text{sum } f \ F) \rangle$ 
by (metis eventually_finite_subsets_at_top)
define  $F0 \ F0A \ F0B$  where  $\langle F0 \equiv F1 \cup F2 \rangle$  and  $\langle F0A \equiv F0 - B \rangle$  and  $\langle F0B$ 
 $\equiv F0 \cap B \rangle$ 
have [simp]:  $\langle \text{finite } F0 \rangle \ \langle F0 \subseteq A \rangle$ 
using  $\langle F1 \subseteq A \rangle \ \langle F2 \subseteq A \rangle \ \langle \text{finite } F1 \rangle \ \langle \text{finite } F2 \rangle$  unfolding F0_def by
blast+

have  $*$ :  $E' (\text{sum } f \ F1', \text{sum } f \ F2')$ 
if  $F1' \supseteq F0B \ F2' \supseteq F0B$   $\text{finite } F1' \ \text{finite } F2' \ F1' \subseteq B \ F2' \subseteq B$  for  $F1' \ F2'$ 
proof (intro DE'[where c = sum f F0A] P1P2E)
have  $P1 \ (\text{sum } f \ (F1' \cup F0A))$ 
using that assms  $F1(1,2) \ F2(1,2)$  by (intro F1) (auto simp: F0A_def
F0B_def F0_def)
thus  $P1 \ (\text{sum } f \ F1' + \text{sum } f \ F0A)$ 
by (subst (asm) sum.union_disjoint) (use that in <auto simp: F0A_def>)
next
have  $P2 \ (\text{sum } f \ (F2' \cup F0A))$ 
using that assms  $F1(1,2) \ F2(1,2)$  by (intro F2) (auto simp: F0A_def
F0B_def F0_def)
thus  $P2 \ (\text{sum } f \ F2' + \text{sum } f \ F0A)$ 
by (subst (asm) sum.union_disjoint) (use that in <auto simp: F0A_def>)

```

qed

```

show  $\langle \text{eventually } E \text{ } (?filter\_fB \times_F ?filter\_fB) \rangle$ 
  unfolding eventually_prod_filter
proof (safe intro!: exI)
show eventually  $(\lambda x. E' (x, \text{sum } f \text{ } F0B)) (\text{filtermap } (\text{sum } f) (\text{finite\_subsets\_at\_top } B))$ 
and eventually  $(\lambda x. E' (\text{sum } f \text{ } F0B, x)) (\text{filtermap } (\text{sum } f) (\text{finite\_subsets\_at\_top } B))$ 
  unfolding eventually_filtermap eventually_finite_subsets_at_top
  by (rule exI[of  $\_ F0B$ ]; use * in  $\langle \text{force } \text{simp: } F0B\_def \rangle$ ) +
next
  show  $E (x, y)$  if  $E' (x, \text{sum } f \text{ } F0B)$  and  $E' (\text{sum } f \text{ } F0B, y)$  for  $x \ y$ 
  using  $E' E' E$  that by blast
qed
qed

```

```

then obtain  $x$  where  $\langle ?filter\_fB \leq nhds \ x \rangle$ 
  using cauchy_filter_complete_converges[of  $?filter\_fB \ UNIV$ ]  $\langle \text{complete } (UNIV :: \_) \rangle$ 
  by (auto simp: filtermap_bot_iff)
then have  $\langle \text{sum } f \longrightarrow x \rangle (\text{finite\_subsets\_at\_top } B)$ 
  by (auto simp: filterlim_def)
then show ?thesis
  by (auto simp: summable_on_def has_sum_def)
qed

```

A special case of *summable_on_subset* for Banach spaces with less premises.

```

lemma summable_on_subset_banach:
  fixes  $A \ B$  and  $f :: \langle 'a \Rightarrow 'b :: \text{banach} \rangle$ 
  assumes  $\langle f \text{ summable\_on } A \rangle$ 
  assumes  $\langle B \subseteq A \rangle$ 
  shows  $\langle f \text{ summable\_on } B \rangle$ 
  by (rule summable_on_subset[OF  $\_ \_ \text{assms}$ ])
  (auto simp: complete_def convergent_def dest!: Cauchy_convergent)

```

```

lemma has_sum_empty[simp]:  $\langle \text{has\_sum } f \ \{ \} \ 0 \rangle$ 
  by (meson ex_in_conv has_sum_0)

```

```

lemma summable_on_empty[simp]:  $\langle f \text{ summable\_on } \{ \} \rangle$ 
  by auto

```

```

lemma infsum_empty[simp]:  $\langle \text{infsum } f \ \{ \} = 0 \rangle$ 
  by simp

```

```

lemma sum_has_sum:
  fixes  $f :: 'a \Rightarrow 'b :: \text{topological\_comm\_monoid\_add}$ 
  assumes finite:  $\langle \text{finite } A \rangle$ 
  assumes conv:  $\langle \bigwedge a. a \in A \implies \text{has\_sum } f \ (B \ a) \ (s \ a) \rangle$ 

```

```

assumes disj:  $\langle \bigwedge a\ a'.\ a \in A \implies a' \in A \implies a \neq a' \implies B\ a \cap B\ a' = \{\} \rangle$ 
shows  $\langle \text{has\_sum}\ f\ (\bigcup a \in A.\ B\ a)\ (\text{sum}\ s\ A) \rangle$ 
using assms
proof (insert finite conv disj, induction)
  case empty
  then show ?case
    by simp
next
  case (insert x A)
  have  $\langle \text{has\_sum}\ f\ (B\ x)\ (s\ x) \rangle$ 
    by (simp add: insert.premis)
  moreover have IH:  $\langle \text{has\_sum}\ f\ (\bigcup a \in A.\ B\ a)\ (\text{sum}\ s\ A) \rangle$ 
    using insert by simp
  ultimately have  $\langle \text{has\_sum}\ f\ (B\ x \cup (\bigcup a \in A.\ B\ a))\ (s\ x + \text{sum}\ s\ A) \rangle$ 
    apply (rule has_sum_Un_disjoint)
    using insert by auto
  then show ?case
    using insert.hyps by auto
qed

```

```

lemma summable_on_finite_union_disjoint:
  fixes f :: 'a  $\Rightarrow$  'b::topological_comm_monoid_add
  assumes finite:  $\langle \text{finite}\ A \rangle$ 
  assumes conv:  $\langle \bigwedge a.\ a \in A \implies f\ \text{summable\_on}\ (B\ a) \rangle$ 
  assumes disj:  $\langle \bigwedge a\ a'.\ a \in A \implies a' \in A \implies a \neq a' \implies B\ a \cap B\ a' = \{\} \rangle$ 
  shows  $\langle f\ \text{summable\_on}\ (\bigcup a \in A.\ B\ a) \rangle$ 
  using finite conv disj apply induction by (auto intro!: summable_on_Un_disjoint)

```

```

lemma sum_infsum:
  fixes f :: 'a  $\Rightarrow$  'b::{topological_comm_monoid_add, t2_space}
  assumes finite:  $\langle \text{finite}\ A \rangle$ 
  assumes conv:  $\langle \bigwedge a.\ a \in A \implies f\ \text{summable\_on}\ (B\ a) \rangle$ 
  assumes disj:  $\langle \bigwedge a\ a'.\ a \in A \implies a' \in A \implies a \neq a' \implies B\ a \cap B\ a' = \{\} \rangle$ 
  shows  $\langle \text{sum}\ (\lambda a.\ \text{infsum}\ f\ (B\ a))\ A = \text{infsum}\ f\ (\bigcup a \in A.\ B\ a) \rangle$ 
  by (rule sym, rule infsumI)
    (use sum_has_sum[of A f B  $\langle \lambda a.\ \text{infsum}\ f\ (B\ a) \rangle$ ] assms in auto)

```

The lemmas *infsum_comm_additive_general* and *infsum_comm_additive* (and variants) below both state that the infinite sum commutes with a continuous additive function. *infsum_comm_additive_general* is stated more for more general type classes at the expense of a somewhat less compact formulation of the premises. E.g., by avoiding the constant *additive* which introduces an additional sort constraint (group instead of monoid). For example, extended reals (*ereal*, *ennreal*) are not covered by *infsum_comm_additive*.

```

lemma has_sum_comm_additive_general:
  fixes f :: 'b :: {comm_monoid_add, topological_space}  $\Rightarrow$  'c :: {comm_monoid_add, topological_space}
  assumes f_sum:  $\langle \bigwedge F.\ \text{finite}\ F \implies F \subseteq S \implies \text{sum}\ (f\ o\ g)\ F = f\ (\text{sum}\ g\ F) \rangle$ 

```

— Not using *additive* because it would add sort constraint *ab_group_add*
assumes *cont*: $\langle f -x \rightarrow f x \rangle$
 — For *t2_space*, this is equivalent to *isCont f x* by *isCont_def*.
assumes *infsum*: $\langle \text{has\_sum } g \ S \ x \rangle$
shows $\langle \text{has\_sum } (f \ o \ g) \ S \ (f \ x) \rangle$
proof –
have $\langle \text{sum } g \longrightarrow x \rangle \ (finite\_subsets\_at\_top \ S)$
using *infsum has_sum_def* **by** *blast*
then have $\langle ((f \ o \ \text{sum } g) \longrightarrow f \ x) \ (finite\_subsets\_at\_top \ S) \rangle$
apply (*rule tendsto_compose_at*)
using *assms* **by** *auto*
then have $\langle \text{sum } (f \ o \ g) \longrightarrow f \ x \rangle \ (finite\_subsets\_at\_top \ S)$
apply (*rule tendsto_cong[THEN iffD1, rotated]*)
using *f_sum* **by** *fastforce*
then show $\langle \text{has\_sum } (f \ o \ g) \ S \ (f \ x) \rangle$
using *has_sum_def* **by** *blast*
qed

lemma *summable_on_comm_additive_general*:
fixes *f* :: $\langle 'b :: \{comm\_monoid\_add, topological\_space\} \Rightarrow 'c :: \{comm\_monoid\_add, topological\_space\} \rangle$
assumes $\langle \bigwedge F. finite \ F \Longrightarrow F \subseteq S \Longrightarrow \text{sum } (f \ o \ g) \ F = f \ (\text{sum } g \ F) \rangle$
 — Not using *additive* because it would add sort constraint *ab_group_add*
assumes $\langle \bigwedge x. \text{has\_sum } g \ S \ x \Longrightarrow f -x \rightarrow f x \rangle$
 — For *t2_space*, this is equivalent to *isCont f x* by *isCont_def*.
assumes $\langle g \text{ summable\_on } S \rangle$
shows $\langle (f \ o \ g) \text{ summable\_on } S \rangle$
by (*meson assms summable_on_def has_sum_comm_additive_general has_sum_def infsum_tendsto*)

lemma *infsum_comm_additive_general*:
fixes *f* :: $\langle 'b :: \{comm\_monoid\_add, t2\_space\} \Rightarrow 'c :: \{comm\_monoid\_add, t2\_space\} \rangle$
assumes *f_sum*: $\langle \bigwedge F. finite \ F \Longrightarrow F \subseteq S \Longrightarrow \text{sum } (f \ o \ g) \ F = f \ (\text{sum } g \ F) \rangle$
 — Not using *additive* because it would add sort constraint *ab_group_add*
assumes $\langle isCont \ f \ (\text{infsum } g \ S) \rangle$
assumes $\langle g \text{ summable\_on } S \rangle$
shows $\langle \text{infsum } (f \ o \ g) \ S = f \ (\text{infsum } g \ S) \rangle$
using *assms*
by (*intro infsumI has_sum_comm_additive_general has_sum_infsum*) (*auto simp: isCont_def*)

lemma *has_sum_comm_additive*:
fixes *f* :: $\langle 'b :: \{ab\_group\_add, topological\_space\} \Rightarrow 'c :: \{ab\_group\_add, topological\_space\} \rangle$
assumes $\langle additive \ f \rangle$
assumes $\langle f -x \rightarrow f x \rangle$
 — For *t2_space*, this is equivalent to *isCont f x* by *isCont_def*.
assumes *infsum*: $\langle \text{has\_sum } g \ S \ x \rangle$
shows $\langle \text{has\_sum } (f \ o \ g) \ S \ (f \ x) \rangle$
using *assms*
by (*intro has_sum_comm_additive_general has_sum_infsum*) (*auto simp: is-*

Cont_def additive.sum)

lemma *summable_on_comm_additive:*

fixes $f :: \langle 'b :: \{ab\_group\_add, t2\_space\} \Rightarrow 'c :: \{ab\_group\_add, topological\_space\} \rangle$
assumes $\langle additive\ f \rangle$
assumes $\langle isCont\ f\ (infsum\ g\ S) \rangle$
assumes $\langle g\ summable\_on\ S \rangle$
shows $\langle (f\ o\ g)\ summable\_on\ S \rangle$
by (*meson* *assms(1)* *assms(2)* *assms(3)* *summable_on_def* *has_sum_comm_additive* *has_sum_infsum* *isContD*)

lemma *infsum_comm_additive:*

fixes $f :: \langle 'b :: \{ab\_group\_add, t2\_space\} \Rightarrow 'c :: \{ab\_group\_add, t2\_space\} \rangle$
assumes $\langle additive\ f \rangle$
assumes $\langle isCont\ f\ (infsum\ g\ S) \rangle$
assumes $\langle g\ summable\_on\ S \rangle$
shows $\langle infsum\ (f\ o\ g)\ S = f\ (infsum\ g\ S) \rangle$
by (*rule* *infsum_comm_additive_general*; *auto* *simp*: *assms* *additive.sum*)

lemma *nonneg_bdd_above_has_sum:*

fixes $f :: \langle 'a \Rightarrow 'b :: \{conditionally\_complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology\} \rangle$
assumes $\langle \bigwedge x. x \in A \implies f\ x \geq 0 \rangle$
assumes $\langle bdd\_above\ (sum\ f\ \{F. F \subseteq A \wedge finite\ F\}) \rangle$
shows $\langle has\_sum\ f\ A\ (SUP\ F \in \{F. finite\ F \wedge F \subseteq A\}. sum\ f\ F) \rangle$

proof —

have $\langle (sum\ f \longrightarrow (SUP\ F \in \{F. finite\ F \wedge F \subseteq A\}. sum\ f\ F))\ (finite\_subsets\_at\_top\ A) \rangle$

proof (*rule* *order_tendstoI*)

fix a **assume** $\langle a < (SUP\ F \in \{F. finite\ F \wedge F \subseteq A\}. sum\ f\ F) \rangle$

then obtain F **where** $\langle a < sum\ f\ F \rangle$ **and** $\langle finite\ F \rangle$ **and** $\langle F \subseteq A \rangle$

by (*metis* (*mono_tags*, *lifting*) *Collect_cong* *Collect_empty_eq* *assms(2)* *empty_subsetI* *finite.emptyI* *less_cSUP_iff* *mem_Collect_eq*)

show $\langle \forall_F\ x\ in\ finite\_subsets\_at\_top\ A. a < sum\ f\ x \rangle$

unfolding *eventually_finite_subsets_at_top*

proof (*rule* *exI*[*of* $\_ F$], *safe*)

fix Y **assume** $Y: finite\ Y\ F \subseteq Y\ Y \subseteq A$

have $a < sum\ f\ F$

by *fact*

also have $\dots \leq sum\ f\ Y$

using *assms* Y **by** (*intro* *sum_mono2*) *auto*

finally show $a < sum\ f\ Y$.

qed (*use* $\langle finite\ F \rangle$ $\langle F \subseteq A \rangle$ **in** *auto*)

next

fix a **assume** $\ast: \langle (SUP\ F \in \{F. finite\ F \wedge F \subseteq A\}. sum\ f\ F) < a \rangle$

have $\langle sum\ f\ F < a \rangle$ **if** $\langle F \subseteq A \rangle$ **and** $\langle finite\ F \rangle$ **for** F

proof —

have $sum\ f\ F \leq (SUP\ F \in \{F. finite\ F \wedge F \subseteq A\}. sum\ f\ F)$

by (*rule* *cSUP_upper*) (*use* *that* *assms(2)* **in** $\langle auto\ simp: conj\_commute \rangle$)

```

    also have ... < a
    by fact
    finally show ?thesis .
qed
then show  $\langle \forall_F x \text{ in } \text{finite\_subsets\_at\_top } A. \text{ sum } f x < a \rangle$ 
  by (rule eventually_finite_subsets_at_top_weakI)
qed
then show ?thesis
  using has_sum_def by blast
qed

```

```

lemma nonneg_bdd_above_summable_on:
  fixes f ::  $\langle 'a \Rightarrow 'b :: \{\text{conditionally\_complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology}\} \rangle$ 
  assumes  $\langle \bigwedge x. x \in A \implies f x \geq 0 \rangle$ 
  assumes  $\langle \text{bdd\_above } (\text{sum } f \text{ ` } \{F. F \subseteq A \wedge \text{finite } F\}) \rangle$ 
  shows  $\langle f \text{ summable\_on } A \rangle$ 
  using assms(1) assms(2) summable_on_def nonneg_bdd_above_has_sum by blast

```

```

lemma nonneg_bdd_above_infsum:
  fixes f ::  $\langle 'a \Rightarrow 'b :: \{\text{conditionally\_complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology}\} \rangle$ 
  assumes  $\langle \bigwedge x. x \in A \implies f x \geq 0 \rangle$ 
  assumes  $\langle \text{bdd\_above } (\text{sum } f \text{ ` } \{F. F \subseteq A \wedge \text{finite } F\}) \rangle$ 
  shows  $\langle \text{infsum } f A = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{sum } f F) \rangle$ 
  using assms by (auto intro!: infsumI nonneg_bdd_above_has_sum)

```

```

lemma nonneg_has_sum_complete:
  fixes f ::  $\langle 'a \Rightarrow 'b :: \{\text{complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology}\} \rangle$ 
  assumes  $\langle \bigwedge x. x \in A \implies f x \geq 0 \rangle$ 
  shows  $\langle \text{has\_sum } f A (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{sum } f F) \rangle$ 
  using assms nonneg_bdd_above_has_sum by blast

```

```

lemma nonneg_summable_on_complete:
  fixes f ::  $\langle 'a \Rightarrow 'b :: \{\text{complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology}\} \rangle$ 
  assumes  $\langle \bigwedge x. x \in A \implies f x \geq 0 \rangle$ 
  shows  $\langle f \text{ summable\_on } A \rangle$ 
  using assms nonneg_bdd_above_summable_on by blast

```

```

lemma nonneg_infsum_complete:
  fixes f ::  $\langle 'a \Rightarrow 'b :: \{\text{complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology}\} \rangle$ 
  assumes  $\langle \bigwedge x. x \in A \implies f x \geq 0 \rangle$ 
  shows  $\langle \text{infsum } f A = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{sum } f F) \rangle$ 
  using assms nonneg_bdd_above_infsum by blast

```

```

lemma has_sum_nonneg:
  fixes f ::  $\langle 'a \Rightarrow 'b :: \{\text{ordered\_comm\_monoid\_add, linorder\_topology}\} \rangle$ 
  assumes has_sum f M a

```

```

    and  $\bigwedge x. x \in M \implies 0 \leq f x$ 
  shows  $a \geq 0$ 
  by (metis (no_types, lifting) DiffD1 assms(1) assms(2) empty_iff has_sum_0
    has_sum_mono_neutral order_refl)

```

lemma *infsum_nonneg*:

```

  fixes  $f :: 'a \Rightarrow 'b::\{\text{ordered\_comm\_monoid\_add}, \text{linorder\_topology}\}$ 
  assumes  $\bigwedge x. x \in M \implies 0 \leq f x$ 
  shows  $\text{infsum } f M \geq 0$  (is ?lhs  $\geq$  _)
  apply (cases  $\langle f \text{ summable\_on } M \rangle$ )
  apply (metis assms infsum_0_simp summable_on_0_simp infsum_mono)
  using assms by (auto simp add: infsum_not_exists)

```

lemma *has_sum_mono2*:

```

  fixes  $f :: 'a \Rightarrow 'b::\{\text{topological\_ab\_group\_add}, \text{ordered\_comm\_monoid\_add}, \text{linorder\_topology}\}$ 
  assumes  $\text{has\_sum } f A S \text{ has\_sum } f B S' A \subseteq B$ 
  assumes  $\bigwedge x. x \in B - A \implies f x \geq 0$ 
  shows  $S \leq S'$ 

```

proof –

```

  have  $\text{has\_sum } f (B - A) (S' - S)$ 
  by (rule has_sum_Diff) fact+
  hence  $S' - S \geq 0$ 
  by (rule has_sum_nonneg) (use assms(4) in auto)
  thus ?thesis
  by (metis add_0 add_mono_thms_linordered_semiring(3) diff_add_cancel)

```

qed

lemma *infsum_mono2*:

```

  fixes  $f :: 'a \Rightarrow 'b::\{\text{topological\_ab\_group\_add}, \text{ordered\_comm\_monoid\_add}, \text{linorder\_topology}\}$ 
  assumes  $f \text{ summable\_on } A \ f \text{ summable\_on } B A \subseteq B$ 
  assumes  $\bigwedge x. x \in B - A \implies f x \geq 0$ 
  shows  $\text{infsum } f A \leq \text{infsum } f B$ 
  by (rule has_sum_mono2[OF has_sum_infsum has_sum_infsum]) (use assms
    in auto)

```

lemma *finite_sum_le_has_sum*:

```

  fixes  $f :: 'a \Rightarrow 'b::\{\text{topological\_ab\_group\_add}, \text{ordered\_comm\_monoid\_add}, \text{linorder\_topology}\}$ 
  assumes  $\text{has\_sum } f A S \text{ finite } B B \subseteq A$ 
  assumes  $\bigwedge x. x \in A - B \implies f x \geq 0$ 
  shows  $\text{sum } f B \leq S$ 

```

proof (rule has_sum_mono2)

```

  show  $\text{has\_sum } f A S$ 
  by fact
  show  $\text{has\_sum } f B (\text{sum } f B)$ 
  by (rule has_sum_finite) fact+

```

qed (use assms in auto)

lemma *finite_sum_le_infsum*:

```

  fixes  $f :: 'a \Rightarrow 'b::\{\text{topological\_ab\_group\_add}, \text{ordered\_comm\_monoid\_add}, \text{linorder\_topology}\}$ 

```

```

assumes  $f$  summable_on  $A$  finite  $B$   $B \subseteq A$ 
assumes  $\bigwedge x. x \in A - B \implies f\ x \geq 0$ 
shows  $\text{sum } f\ B \leq \text{infsum } f\ A$ 
by (rule finite_sum_le_has_sum[OF has_sum_infsum]) (use assms in auto)

```

lemma has_sum_reindex:

```

assumes  $\langle \text{inj\_on } h\ A \rangle$ 
shows  $\langle \text{has\_sum } g\ (h\ 'A)\ x \longleftrightarrow \text{has\_sum } (g \circ h)\ A\ x \rangle$ 
proof -
  have  $\langle \text{has\_sum } g\ (h\ 'A)\ x \longleftrightarrow (\text{sum } g \longrightarrow x)\ (\text{finite\_subsets\_at\_top } (h\ 'A)) \rangle$ 
  by (simp add: has_sum_def)
  also have  $\langle \dots \longleftrightarrow ((\lambda F. \text{sum } g\ (h\ 'F)) \longrightarrow x)\ (\text{finite\_subsets\_at\_top } A) \rangle$ 
  apply (subst filtermap_image_finite_subsets_at_top[symmetric])
  using assms by (auto simp: filterlim_def filtermap_filtermap)
  also have  $\langle \dots \longleftrightarrow (\text{sum } (g \circ h) \longrightarrow x)\ (\text{finite\_subsets\_at\_top } A) \rangle$ 
  apply (rule tendsto_cong)
  apply (rule eventually_finite_subsets_at_top_weakI)
  apply (rule sum.reindex)
  using assms subset_inj_on by blast
  also have  $\langle \dots \longleftrightarrow \text{has\_sum } (g \circ h)\ A\ x \rangle$ 
  by (simp add: has_sum_def)
  finally show ?thesis .
qed

```

lemma summable_on_reindex:

```

assumes  $\langle \text{inj\_on } h\ A \rangle$ 
shows  $\langle g \text{ summable\_on } (h\ 'A) \longleftrightarrow (g \circ h) \text{ summable\_on } A \rangle$ 
by (simp add: assms summable_on_def has_sum_reindex)

```

lemma infsum_reindex:

```

assumes  $\langle \text{inj\_on } h\ A \rangle$ 
shows  $\langle \text{infsum } g\ (h\ 'A) = \text{infsum } (g \circ h)\ A \rangle$ 
by (metis (no_types, opaque_lifting) assms finite_subsets_at_top_neq_bot infsum_def
  summable_on_reindex has_sum_def has_sum_infsum has_sum_reindex tendsto_Lim)

```

lemma summable_on_reindex_bij_betw:

```

assumes bij_betw  $g\ A\ B$ 
shows  $(\lambda x. f\ (g\ x)) \text{ summable\_on } A \longleftrightarrow f \text{ summable\_on } B$ 
proof -
  thm summable_on_reindex
  have  $\langle (\lambda x. f\ (g\ x)) \text{ summable\_on } A \longleftrightarrow f \text{ summable\_on } g\ 'A \rangle$ 
  apply (rule summable_on_reindex[symmetric, unfolded o_def])
  using assms bij_betw_imp_inj_on by blast
  also have  $\langle \dots \longleftrightarrow f \text{ summable\_on } B \rangle$ 
  using assms bij_betw_imp_surj_on by blast
  finally show ?thesis .

```


qed

lemma *infsum_reindex_bij_betw*:

assumes *bij_betw* *g* *A* *B*

shows $\text{infsum } (\lambda x. f (g x)) A = \text{infsum } f B$

proof –

have $\langle \text{infsum } (\lambda x. f (g x)) A = \text{infsum } f (g \text{ ` } A) \rangle$

by (*metis* (*mono_tags*, *lifting*) *assms* *bij_betw_imp_inj_on* *infsum_cong* *infsum_reindex_o_def*)

also have $\langle \dots = \text{infsum } f B \rangle$

using *assms* *bij_betw_imp_surj_on* **by** *blast*

finally show *?thesis* .

qed

lemma *sum_uniformity*:

assumes *plus_cont*: $\langle \text{uniformly\_continuous\_on } UNIV (\lambda(x::'b::\{\text{uniform\_space}, \text{comm\_monoid\_add}\}, y). x+y) \rangle$

assumes $\langle \text{eventually } E \text{ uniformity} \rangle$

obtains *D* **where** $\langle \text{eventually } D \text{ uniformity} \rangle$

and $\langle \bigwedge M::'a \text{ set}. \bigwedge f f'::'a \Rightarrow 'b. \text{card } M \leq n \wedge (\forall m \in M. D (f m, f' m)) \Rightarrow E (\text{sum } f M, \text{sum } f' M) \rangle$

proof (*atomize_elim*, *insert* $\langle \text{eventually } E \text{ uniformity} \rangle$, *induction* *n* *arbitrary*: *E* *rule:nat_induct*)

case 0

then show *?case*

by (*metis* *card_eq_0_iff_equals0D* *le_zero_eq* *sum.infinite* *sum.not_neutral_contains_not_neutral* *uniformity_refl*)

next

case (*Suc* *n*)

from *plus_cont*[*unfolded* *uniformly_continuous_on_uniformity* *filterlim_def* *le_filter_def*, *rule_format*, *OF* *Suc.prem*s]

obtain *D1* *D2* **where** $\langle \text{eventually } D1 \text{ uniformity} \rangle$ **and** $\langle \text{eventually } D2 \text{ uniformity} \rangle$

and *D1D2E*: $\langle D1 (x, y) \Rightarrow D2 (x', y') \Rightarrow E (x + x', y + y') \rangle$ **for** *x* *y* *x'* *y'*

apply *atomize_elim*

by (*auto simp*: *eventually_prod_filter* *case_prod_beta* *uniformity_prod_def* *eventually_filtermap*)

from *Suc.IH*[*OF* $\langle \text{eventually } D2 \text{ uniformity} \rangle$]

obtain *D3* **where** $\langle \text{eventually } D3 \text{ uniformity} \rangle$ **and** *D3*: $\langle \text{card } M \leq n \Rightarrow (\forall m \in M. D3 (f m, f' m)) \Rightarrow D2 (\text{sum } f M, \text{sum } f' M) \rangle$

for *M* :: $\langle 'a \text{ set} \rangle$ **and** *f* *f'*

by *metis*

define *D* **where** $\langle D x \equiv D1 x \wedge D3 x \rangle$ **for** *x*

have $\langle \text{eventually } D \text{ uniformity} \rangle$

using *D_def* $\langle \text{eventually } D1 \text{ uniformity} \rangle$ $\langle \text{eventually } D3 \text{ uniformity} \rangle$ *eventually_elim2* **by** *blast*

```

have ⟨E (sum f M, sum f' M)⟩
  if ⟨card M ≤ Suc n⟩ and DM: ⟨∀ m ∈ M. D (f m, f' m)⟩
  for M :: 'a set and f f'
proof (cases ⟨card M = 0⟩)
  case True
  then show ?thesis
    by (metis Suc.premis card_eq_0_iff sum.empty sum.infinite uniformity_refl)
  next
  case False
  with ⟨card M ≤ Suc n⟩ obtain N x where ⟨card N ≤ n⟩ and ⟨x ∉ N⟩ and
    ⟨M = insert x N⟩
    by (metis card_Suc_eq less_Suc_eq_0_disj less_Suc_eq_le)

  from DM have ⟨∧ m. m ∈ N ⇒ D (f m, f' m)⟩
    using ⟨M = insert x N⟩ by blast
  with D3[OF ⟨card N ≤ n⟩]
  have D2_N: ⟨D2 (sum f N, sum f' N)⟩
    using D_def by blast

  from DM
  have ⟨D (f x, f' x)⟩
    using ⟨M = insert x N⟩ by blast
  then have ⟨D1 (f x, f' x)⟩
    by (simp add: D_def)

  with D2_N
  have ⟨E (f x + sum f N, f' x + sum f' N)⟩
    using D1D2E by presburger

  then show ⟨E (sum f M, sum f' M)⟩
    by (metis False ⟨M = insert x N⟩ ⟨x ∉ N⟩ card.infinite finite_insert
    sum.insert)
  qed
  with ⟨eventually D uniformity⟩
  show ?case
    by auto
  qed

```

lemma has_sum_Sigma:

```

fixes A :: 'a set and B :: 'a ⇒ 'b set
  and f :: 'a × 'b ⇒ 'c :: {comm_monoid_add, uniform_space}
assumes plus_cont: ⟨uniformly_continuous_on UNIV (λ(x::'c,y). x+y)⟩
assumes summableAB: has_sum f (Sigma A B) a
assumes summableB: ⟨∧ x. x ∈ A ⇒ has_sum (λy. f (x, y)) (B x) (b x)⟩
shows has_sum b A a
proof -
  define F FB FA where ⟨F = finite_subsets_at_top (Sigma A B)⟩ and ⟨FB x
  = finite_subsets_at_top (B x)⟩
  and ⟨FA = finite_subsets_at_top A⟩ for x

```

```

from summableB
have sum_b:  $\langle \text{sum } (\lambda y. f \ (x, y)) \longrightarrow b \ x \rangle \ (FB \ x) \rangle$  if  $\langle x \in A \rangle$  for  $x$ 
  using FB_def[abs_def] has_sum_def that by auto
from summableAB
have sum_S:  $\langle \text{sum } f \longrightarrow a \rangle \ F \rangle$ 
  using F_def has_sum_def by blast

have finite_proj:  $\langle \text{finite } \{b \mid b. (a, b) \in H\} \rangle$  if  $\langle \text{finite } H \rangle$  for  $H :: \langle 'a \times 'b \rangle \text{ set}$ 
and  $a$ 
  apply (subst asm_rl[of  $\langle \{b \mid b. (a, b) \in H\} = \text{snd } ' \{ab. ab \in H \wedge \text{fst } ab = a\} \rangle$ ])
  by (auto simp: image_iff that)

have  $\langle \text{sum } b \longrightarrow a \rangle \ FA \rangle$ 
proof (rule tendsto_iff_uniformity[THEN iffD2, rule_format])
  fix  $E :: \langle ('c \times 'c) \Rightarrow \text{bool} \rangle$ 
  assume  $\langle \text{eventually } E \text{ uniformity} \rangle$ 
  then obtain  $D$  where  $D\_uni: \langle \text{eventually } D \text{ uniformity} \rangle$  and  $DDE': \langle \bigwedge x \ y \ z. D \ (x, y) \Longrightarrow D \ (y, z) \Longrightarrow E \ (x, z) \rangle$ 
  by (metis (no_types, lifting)  $\langle \text{eventually } E \text{ uniformity} \rangle$  uniformity_transE)
  from sum_S obtain  $G$  where  $\langle \text{finite } G \rangle$  and  $\langle G \subseteq \text{Sigma } A \ B \rangle$ 
  and  $G\_sum: \langle G \subseteq H \Longrightarrow H \subseteq \text{Sigma } A \ B \Longrightarrow \text{finite } H \Longrightarrow D \ (\text{sum } f \ H, a) \rangle$  for  $H$ 
  unfolding tendsto_iff_uniformity
  by (metis (mono_tags, lifting)  $D\_uni$  F_def eventually_finite_subsets_at_top)
  have  $\langle \text{finite } (\text{fst } ' G) \rangle$  and  $\langle \text{fst } ' G \subseteq A \rangle$ 
  using  $\langle \text{finite } G \rangle \langle G \subseteq \text{Sigma } A \ B \rangle$  by auto
  thm uniformity_prod_def
  define  $Ga$  where  $\langle Ga \ a = \{b. (a, b) \in G\} \rangle$  for  $a$ 
  have  $Ga\_fin: \langle \text{finite } (Ga \ a) \rangle$  and  $Ga\_B: \langle Ga \ a \subseteq B \ a \rangle$  for  $a$ 
  using  $\langle \text{finite } G \rangle \langle G \subseteq \text{Sigma } A \ B \rangle$  finite_proj by (auto simp:  $Ga\_def$  finite_proj)

  have  $\langle E \ (\text{sum } b \ M, a) \rangle$  if  $\langle M \supseteq \text{fst } ' G \rangle$  and  $\langle \text{finite } M \rangle$  and  $\langle M \subseteq A \rangle$  for  $M$ 
  proof –
    define  $FMB$  where  $\langle FMB = \text{finite\_subsets\_at\_top } (\text{Sigma } M \ B) \rangle$ 
    have  $\langle \text{eventually } (\lambda H. D \ (\sum a \in M. b \ a, \sum (a, b) \in H. f \ (a, b))) \ FMB \rangle$ 
    proof –
      obtain  $D'$  where  $D'\_uni: \langle \text{eventually } D' \text{ uniformity} \rangle$ 
      and  $\langle \text{card } M' \leq \text{card } M \wedge (\forall m \in M'. D' \ (g \ m, g' \ m)) \Longrightarrow D \ (\text{sum } g \ M', \text{sum } g' \ M') \rangle$ 
      for  $M' :: \langle 'a \text{ set} \rangle$  and  $g \ g'$ 
      apply (rule sum_uniformity[OF plus_cont  $\langle \text{eventually } D \text{ uniformity} \rangle$ ,
where  $n = \langle \text{card } M \rangle$ ])
      by auto
      then have  $D'\_sum\_D: \langle (\forall m \in M. D' \ (g \ m, g' \ m)) \Longrightarrow D \ (\text{sum } g \ M, \text{sum } g' \ M) \rangle$  for  $g \ g'$ 
      by auto

```

```

    obtain Ha where ⟨Ha a ⊇ Ga a⟩ and Ha_fin: ⟨finite (Ha a)⟩ and Ha_B:
    ⟨Ha a ⊆ B a⟩
    and D'_sum_Ha: ⟨Ha a ⊆ L ⟹ L ⊆ B a ⟹ finite L ⟹ D' (b a, sum
    (λb. f (a,b)) L)⟩ if ⟨a ∈ A⟩ for a L
    proof -
    from sum_b[unfolded tendsto_iff_uniformity, rule_format, OF _ D'_uni[THEN
    uniformity_sym]]
    obtain Ha0 where ⟨finite (Ha0 a)⟩ and ⟨Ha0 a ⊆ B a⟩
    and ⟨Ha0 a ⊆ L ⟹ L ⊆ B a ⟹ finite L ⟹ D' (b a, sum (λb. f (a,b))
    L)⟩ if ⟨a ∈ A⟩ for a L
    unfolding FB_def eventually_finite_subsets_at_top unfolding prod.case
    by metis
    moreover define Ha where ⟨Ha a = Ha0 a ∪ Ga a⟩ for a
    ultimately show ?thesis
    using that[where Ha=Ha]
    using Ga_fin Ga_B by auto
  qed

  have ⟨D (∑ a∈M. b a, ∑ (a,b)∈H. f (a,b))⟩ if ⟨finite H⟩ and ⟨H ⊆ Sigma
  M B⟩ and ⟨H ⊇ Sigma M Ha⟩ for H
  proof -
    define Ha' where ⟨Ha' a = {b | b. (a,b) ∈ H}⟩ for a
    have [simp]: ⟨finite (Ha' a)⟩ and [simp]: ⟨Ha' a ⊇ Ha a⟩ and [simp]: ⟨Ha'
    a ⊆ B a⟩ if ⟨a ∈ M⟩ for a
    unfolding Ha'_def using ⟨finite H⟩ ⟨H ⊆ Sigma M B⟩ ⟨Sigma M Ha
    ⊆ H⟩ that finite_proj by auto
    have ⟨Sigma M Ha' = H⟩
    using that by (auto simp: Ha'_def)
    then have *: ⟨(∑ (a,b)∈H. f (a,b)) = (∑ a∈M. ∑ b∈Ha' a. f (a,b))⟩
    apply (subst sum.Sigma)
    using ⟨finite M⟩ by auto
    have ⟨D' (b a, sum (λb. f (a,b)) (Ha' a))⟩ if ⟨a ∈ M⟩ for a
    apply (rule D'_sum_Ha)
    using that ⟨M ⊆ A⟩ by auto
    then have ⟨D (∑ a∈M. b a, ∑ a∈M. sum (λb. f (a,b)) (Ha' a))⟩
    by (rule_tac D'_sum_D, auto)
    with * show ?thesis
    by auto
  qed
  moreover have ⟨Sigma M Ha ⊆ Sigma M B⟩
  using Ha_B ⟨M ⊆ A⟩ by auto
  ultimately show ?thesis
  unfolding FMB_def eventually_finite_subsets_at_top
  by (intro exI[of _ Sigma M Ha])
    (use Ha_fin that(2,3) in ⟨fastforce intro!: finite_SigmaI⟩)
  qed
  moreover have ⟨eventually (λH. D (∑ (a,b)∈H. f (a,b), a)) FMB⟩
  unfolding FMB_def eventually_finite_subsets_at_top
  proof (rule exI[of _ G], safe)

```

```

    fix Y assume Y: finite Y G ⊆ Y Y ⊆ Sigma M B
    have Y ⊆ Sigma A B
      using Y ⟨M ⊆ A⟩ by blast
    thus D (∑ (a,b)∈Y. f (a, b), a)
      using G_sum[of Y] Y by auto
  qed (use ⟨finite G⟩ ⟨G ⊆ Sigma A B⟩ that in auto)
  ultimately have ⟨∀F x in FMB. E (sum b M, a)⟩
    by eventually_elim (use DDE' in auto)
  then show ⟨E (sum b M, a)⟩
    by (rule eventually_const[THEN iffD1, rotated]) (force simp: FMB_def)
  qed
  then show ⟨∀F x in FA. E (sum b x, a)⟩
    using ⟨finite (fst 'G)⟩ and ⟨fst 'G ⊆ A⟩
    by (auto intro!: exI[of _ ⟨fst 'G⟩] simp add: FA_def eventually_finite_subsets_at_top)
  qed
  then show ?thesis
    by (simp add: FA_def has_sum_def)
  qed

```

lemma summable_on_Sigma:

```

  fixes A :: 'a set and B :: 'a ⇒ 'b set
  and f :: 'a ⇒ 'b ⇒ 'c::{comm_monoid_add, t2_space, uniform_space}
  assumes plus_cont: ⟨uniformly_continuous_on UNIV (λ(x::'c,y). x+y)⟩
  assumes summableAB: ⟨λ(x,y). f x y⟩ summable_on (Sigma A B)
  assumes summableB: ⟨λx. x∈A ⇒ (f x) summable_on (B x)⟩
  shows ⟨λx. infsum (f x) (B x)⟩ summable_on A
proof -
  from summableAB obtain a where a: ⟨has_sum (λ(x,y). f x y) (Sigma A B)
a⟩
  using has_sum_infsum by blast
  from summableB have b: ⟨λx. x∈A ⇒ has_sum (f x) (B x) (infsum (f x) (B
x))⟩
  by (auto intro!: has_sum_infsum)
  show ?thesis
  using plus_cont a b
  by (auto intro: has_sum_Sigma[where f=⟨λ(x,y). f x y⟩, simplified] simp:
summable_on_def)
  qed

```

lemma infsum_Sigma:

```

  fixes A :: 'a set and B :: 'a ⇒ 'b set
  and f :: 'a × 'b ⇒ 'c::{comm_monoid_add, t2_space, uniform_space}
  assumes plus_cont: ⟨uniformly_continuous_on UNIV (λ(x::'c,y). x+y)⟩
  assumes summableAB: f summable_on (Sigma A B)
  assumes summableB: ⟨λx. x∈A ⇒ (λy. f (x, y)) summable_on (B x)⟩
  shows infsum f (Sigma A B) = infsum (λx. infsum (λy. f (x, y)) (B x)) A
proof -
  from summableAB have a: ⟨has_sum f (Sigma A B) (infsum f (Sigma A B))⟩
  using has_sum_infsum by blast

```

```

from summableB have b:  $\langle \bigwedge x. x \in A \implies \text{has\_sum } (\lambda y. f(x, y)) (B\ x) \text{ (infsum } (\lambda y. f(x, y)) (B\ x)) \rangle$ 
  by (auto intro!: has_sum_infsum)
show ?thesis
  using plus_cont a b by (auto intro: infsumI[symmetric] has_sum_Sigma simp:
    summable_on_def)
qed

```

```

lemma infsum_Sigma':
  fixes A :: 'a set and B :: 'a  $\Rightarrow$  'b set
  and f ::  $\langle 'a \Rightarrow 'b \Rightarrow 'c :: \{ \text{comm\_monoid\_add}, \text{t2\_space}, \text{uniform\_space} \} \rangle$ 
  assumes plus_cont:  $\langle \text{uniformly\_continuous\_on UNIV } (\lambda(x::'c, y). x+y) \rangle$ 
  assumes summableAB:  $\langle \lambda(x, y). f\ x\ y \text{ summable\_on } (\text{Sigma } A\ B) \rangle$ 
  assumes summableB:  $\langle \bigwedge x. x \in A \implies (f\ x) \text{ summable\_on } (B\ x) \rangle$ 
  shows  $\langle \text{infsum } (\lambda x. \text{infsum } (f\ x) (B\ x))\ A = \text{infsum } (\lambda(x, y). f\ x\ y) (\text{Sigma } A\ B) \rangle$ 
  using infsum_Sigma[of  $\langle \lambda(x, y). f\ x\ y \rangle\ A\ B$ ]
  using assms by auto

```

A special case of *infsum_Sigma* etc. for Banach spaces. It has less premises.

```

lemma
  fixes A :: 'a set and B :: 'a  $\Rightarrow$  'b set
  and f ::  $\langle 'a \Rightarrow 'b \Rightarrow 'c :: \text{banach} \rangle$ 
  assumes [simp]:  $\langle \lambda(x, y). f\ x\ y \text{ summable\_on } (\text{Sigma } A\ B) \rangle$ 
  shows infsum_Sigma'_banach:  $\langle \text{infsum } (\lambda x. \text{infsum } (f\ x) (B\ x))\ A = \text{infsum } (\lambda(x, y). f\ x\ y) (\text{Sigma } A\ B) \rangle$  (is ?thesis1)
  and summable_on_Sigma_banach:  $\langle (\lambda x. \text{infsum } (f\ x) (B\ x)) \text{ summable\_on } A \rangle$ 
  (is ?thesis2)
proof -
  have [simp]:  $\langle (f\ x) \text{ summable\_on } (B\ x) \rangle$  if  $\langle x \in A \rangle$  for x
  proof -
    from assms
    have  $\langle (\lambda(x, y). f\ x\ y) \text{ summable\_on } (\text{Pair } x\ 'B\ x) \rangle$ 
    by (meson image_subset_iff summable_on_subset_banach mem_Sigma_iff
      that)
    then have  $\langle ((\lambda(x, y). f\ x\ y) \circ \text{Pair } x) \text{ summable\_on } (B\ x) \rangle$ 
    apply (rule_tac summable_on_reindex[THEN iffD1])
    by (simp add: inj_on_def)
    then show ?thesis
    by (auto simp: o_def)
  qed
show ?thesis1
  apply (rule infsum_Sigma')
  by auto
show ?thesis2
  apply (rule summable_on_Sigma)
  by auto
qed

```

```

lemma infsum_Sigma_banach:
  fixes A :: 'a set and B :: 'a  $\Rightarrow$  'b set
  and f :: 'a  $\times$  'b  $\Rightarrow$  'c::banach
  assumes [simp]: f summable_on (Sigma A B)
  shows  $\langle \text{infsum } (\lambda x. \text{infsum } (\lambda y. f (x,y)) (B x)) A = \text{infsum } f (Sigma A B) \rangle$ 
  using assms
  by (subst infsum_Sigma'_banach) auto

lemma infsum_swap:
  fixes A :: 'a set and B :: 'b set
  fixes f :: 'a  $\Rightarrow$  'b  $\Rightarrow$  'c::{comm_monoid_add,t2_space,uniform_space}
  assumes plus_cont:  $\langle \text{uniformly\_continuous\_on UNIV } (\lambda(x::'c,y). x+y) \rangle$ 
  assumes  $\langle \lambda(x,y). f x y \text{ summable\_on } (A \times B) \rangle$ 
  assumes  $\langle \bigwedge a. a \in A \implies (f a) \text{ summable\_on } B \rangle$ 
  assumes  $\langle \bigwedge b. b \in B \implies (\lambda a. f a b) \text{ summable\_on } A \rangle$ 
  shows  $\langle \text{infsum } (\lambda x. \text{infsum } (\lambda y. f x y) B) A = \text{infsum } (\lambda y. \text{infsum } (\lambda x. f x y) A) B \rangle$ 
proof -
  have [simp]:  $\langle \lambda(x,y). f y x \text{ summable\_on } (B \times A) \rangle$ 
  apply (subst product_swap[symmetric])
  apply (subst summable_on_reindex)
  using assms by (auto simp: o_def)
  have  $\langle \text{infsum } (\lambda x. \text{infsum } (\lambda y. f x y) B) A = \text{infsum } (\lambda(x,y). f x y) (A \times B) \rangle$ 
  apply (subst infsum_Sigma)
  using assms by auto
  also have  $\langle \dots = \text{infsum } (\lambda(x,y). f y x) (B \times A) \rangle$ 
  apply (subst product_swap[symmetric])
  apply (subst infsum_reindex)
  using assms by (auto simp: o_def)
  also have  $\langle \dots = \text{infsum } (\lambda y. \text{infsum } (\lambda x. f x y) A) B \rangle$ 
  apply (subst infsum_Sigma)
  using assms by auto
  finally show ?thesis .
qed

```

```

lemma infsum_swap_banach:
  fixes A :: 'a set and B :: 'b set
  fixes f :: 'a  $\Rightarrow$  'b  $\Rightarrow$  'c::banach
  assumes  $\langle \lambda(x,y). f x y \text{ summable\_on } (A \times B) \rangle$ 
  shows  $\text{infsum } (\lambda x. \text{infsum } (\lambda y. f x y) B) A = \text{infsum } (\lambda y. \text{infsum } (\lambda x. f x y) A) B$ 
proof -
  have [simp]:  $\langle \lambda(x,y). f y x \text{ summable\_on } (B \times A) \rangle$ 
  apply (subst product_swap[symmetric])
  apply (subst summable_on_reindex)
  using assms by (auto simp: o_def)
  have  $\langle \text{infsum } (\lambda x. \text{infsum } (\lambda y. f x y) B) A = \text{infsum } (\lambda(x,y). f x y) (A \times B) \rangle$ 
  apply (subst infsum_Sigma'_banach)
  using assms by auto

```

```

also have ⟨... = infsum (λ(x,y). f y x) (B × A)⟩
  apply (subst product_swap[symmetric])
  apply (subst infsum_reindex)
  using assms by (auto simp: o_def)
also have ⟨... = infsum (λy. infsum (λx. f x y) A) B⟩
  apply (subst infsum_Sigma'_banach)
  using assms by auto
finally show ?thesis .
qed

```

lemma *nonneg_infsum_le_0D*:

```

fixes f :: 'a ⇒ 'b::{topological_ab_group_add,ordered_ab_group_add,linorder_topology}
assumes infsum f A ≤ 0
  and abs_sum: f summable_on A
  and nneg: ∧x. x ∈ A ⇒ f x ≥ 0
  and x ∈ A
shows f x = 0
proof (rule ccontr)
  assume ⟨f x ≠ 0⟩
  have ex: ⟨f summable_on (A-⟨x⟩)⟩
    by (rule summable_on_cofin_subset) (use assms in auto)
  have pos: ⟨infsum f (A - ⟨x⟩) ≥ 0⟩
    by (rule infsum_nonneg) (use nneg in auto)

  have [trans]: ⟨x ≥ y ⇒ y > z ⇒ x > z⟩ for x y z :: 'b by auto

  have ⟨infsum f A = infsum f (A-⟨x⟩) + infsum f ⟨x⟩⟩
    by (subst infsum_Un_disjoint[symmetric]) (use assms ex in ⟨auto simp: insert_absorb⟩)
  also have ⟨... ≥ infsum f ⟨x⟩⟩ (is ⟨_ ≥ ...⟩)
    using pos by (rule add_increasing) simp
  also have ⟨... = f x⟩ (is ⟨_ = ...⟩)
    by (subst infsum_finite) auto
  also have ⟨... > 0⟩
    using ⟨f x ≠ 0⟩ assms(4) nneg by fastforce
  finally show False
    using assms by auto
qed

```

lemma *nonneg_has_sum_le_0D*:

```

fixes f :: 'a ⇒ 'b::{topological_ab_group_add,ordered_ab_group_add,linorder_topology}
assumes has_sum f A a ⟨a ≤ 0⟩
  and nneg: ∧x. x ∈ A ⇒ f x ≥ 0
  and x ∈ A
shows f x = 0
by (metis assms(1) assms(2) assms(4) infsumI nonneg_infsum_le_0D summable_on_def nneg)

```

lemma *has_sum_cmult_left*:


```

fixes f :: 'a  $\Rightarrow$  'b :: {topological_semigroup_mult, semiring_0}
assumes <has_sum f A a>
shows has_sum ( $\lambda x. f x * c$ ) A (a * c)
proof -
  from assms have <(sum f  $\longrightarrow$  a) (finite_subsets_at_top A)>
    using has_sum_def by blast
  then have <(( $\lambda F. \text{sum } f F * c$ )  $\longrightarrow$  a * c) (finite_subsets_at_top A)>
    by (simp add: tendsto_mult_right)
  then have <(sum ( $\lambda x. f x * c$ )  $\longrightarrow$  a * c) (finite_subsets_at_top A)>
    apply (rule tendsto_cong[THEN iffD1, rotated])
    apply (rule eventually_finite_subsets_at_top_weakI)
    using sum_distrib_right by blast
  then show ?thesis
    using infsumI has_sum_def by blast
qed

```

```

lemma infsum_cmult_left:
  fixes f :: 'a  $\Rightarrow$  'b :: {t2_space, topological_semigroup_mult, semiring_0}
  assumes <c  $\neq 0 \implies f$  summable_on A>
  shows infsum ( $\lambda x. f x * c$ ) A = infsum f A * c
proof (cases <c=0>)
  case True
  then show ?thesis by auto
next
  case False
  then have <has_sum f A (infsum f A)>
    by (simp add: assms)
  then show ?thesis
    by (auto intro!: infsumI has_sum_cmult_left)
qed

```

```

lemma summable_on_cmult_left:
  fixes f :: 'a  $\Rightarrow$  'b :: {t2_space, topological_semigroup_mult, semiring_0}
  assumes <f summable_on A>
  shows ( $\lambda x. f x * c$ ) summable_on A
  using assms summable_on_def has_sum_cmult_left by blast

```

```

lemma has_sum_cmult_right:
  fixes f :: 'a  $\Rightarrow$  'b :: {topological_semigroup_mult, semiring_0}
  assumes <has_sum f A a>
  shows has_sum ( $\lambda x. c * f x$ ) A (c * a)
proof -
  from assms have <(sum f  $\longrightarrow$  a) (finite_subsets_at_top A)>
    using has_sum_def by blast
  then have <(( $\lambda F. c * \text{sum } f F$ )  $\longrightarrow$  c * a) (finite_subsets_at_top A)>
    by (simp add: tendsto_mult_left)
  then have <(sum ( $\lambda x. c * f x$ )  $\longrightarrow$  c * a) (finite_subsets_at_top A)>
    apply (rule tendsto_cong[THEN iffD1, rotated])
    apply (rule eventually_finite_subsets_at_top_weakI)

```

```

    using sum_distrib_left by blast
  then show ?thesis
    using infsumI has_sum_def by blast
qed

```

```

lemma infsum_cmult_right:
  fixes f :: 'a  $\Rightarrow$  'b :: {t2_space, topological_semigroup_mult, semiring_0}
  assumes  $\langle c \neq 0 \implies f \text{ summable\_on } A \rangle$ 
  shows  $\langle \text{infsum } (\lambda x. c * f x) A = c * \text{infsum } f A \rangle$ 
proof (cases  $\langle c = 0 \rangle$ )
  case True
    then show ?thesis by auto
  next
    case False
    then have  $\langle \text{has\_sum } f A (\text{infsum } f A) \rangle$ 
      by (simp add: assms)
    then show ?thesis
      by (auto intro!: infsumI has_sum_cmult_right)
qed

```

```

lemma summable_on_cmult_right:
  fixes f :: 'a  $\Rightarrow$  'b :: {t2_space, topological_semigroup_mult, semiring_0}
  assumes  $\langle f \text{ summable\_on } A \rangle$ 
  shows  $(\lambda x. c * f x) \text{ summable\_on } A$ 
  using assms summable_on_def has_sum_cmult_right by blast

```

```

lemma summable_on_cmult_left':
  fixes f :: 'a  $\Rightarrow$  'b :: {t2_space, topological_semigroup_mult, division_ring}
  assumes  $\langle c \neq 0 \rangle$ 
  shows  $(\lambda x. f x * c) \text{ summable\_on } A \longleftrightarrow f \text{ summable\_on } A$ 
proof
  assume  $\langle f \text{ summable\_on } A \rangle$ 
  then show  $\langle (\lambda x. f x * c) \text{ summable\_on } A \rangle$ 
    by (rule summable_on_cmult_left)
  next
    assume  $\langle (\lambda x. f x * c) \text{ summable\_on } A \rangle$ 
    then have  $\langle (\lambda x. f x * c * \text{inverse } c) \text{ summable\_on } A \rangle$ 
      by (rule summable_on_cmult_left)
    then show  $\langle f \text{ summable\_on } A \rangle$ 
      by (metis (no_types, lifting) assms summable_on_cong mult.assoc mult.right_neutral right_inverse)
qed

```

```

lemma summable_on_cmult_right':
  fixes f :: 'a  $\Rightarrow$  'b :: {t2_space, topological_semigroup_mult, division_ring}
  assumes  $\langle c \neq 0 \rangle$ 
  shows  $(\lambda x. c * f x) \text{ summable\_on } A \longleftrightarrow f \text{ summable\_on } A$ 
proof
  assume  $\langle f \text{ summable\_on } A \rangle$ 

```

```

    then show  $\langle (\lambda x. c * f x) \text{ summable\_on } A \rangle$ 
      by (rule summable_on_cmult_right)
next
  assume  $\langle (\lambda x. c * f x) \text{ summable\_on } A \rangle$ 
  then have  $\langle (\lambda x. \text{inverse } c * (c * f x)) \text{ summable\_on } A \rangle$ 
    by (rule summable_on_cmult_right)
  then show  $\langle f \text{ summable\_on } A \rangle$ 
    by (metis (no_types, lifting) assms summable_on_cong left_inverse mult.assoc
mult.left_neutral)
qed

```

```

lemma infsum_cmult_left':
  fixes f :: 'a  $\Rightarrow$  'b :: {t2_space, topological_semigroup_mult, division_ring}
  shows infsum  $(\lambda x. f x * c) A = \text{infsum } f A * c$ 
proof (cases  $\langle c \neq 0 \longrightarrow f \text{ summable\_on } A \rangle$ )
  case True
  then show ?thesis
    apply (rule_tac infsum_cmult_left) by auto
next
  case False
  note asm = False
  then show ?thesis
  proof (cases  $\langle c = 0 \rangle$ )
    case True
    then show ?thesis by auto
  next
    case False
    with asm have nex:  $\langle \neg f \text{ summable\_on } A \rangle$ 
      by simp
    moreover have nex':  $\langle \neg (\lambda x. f x * c) \text{ summable\_on } A \rangle$ 
      using asm False apply (subst summable_on_cmult_left') by auto
    ultimately show ?thesis
      unfolding infsum_def by simp
  qed
qed

```

```

lemma infsum_cmult_right':
  fixes f :: 'a  $\Rightarrow$  'b :: {t2_space, topological_semigroup_mult, division_ring}
  shows infsum  $(\lambda x. c * f x) A = c * \text{infsum } f A$ 
proof (cases  $\langle c \neq 0 \longrightarrow f \text{ summable\_on } A \rangle$ )
  case True
  then show ?thesis
    apply (rule_tac infsum_cmult_right) by auto
next
  case False
  note asm = False
  then show ?thesis
  proof (cases  $\langle c = 0 \rangle$ )
    case True

```

```

    then show ?thesis by auto
  next
  case False
  with asm have nex:  $\langle \neg f \text{ summable\_on } A \rangle$ 
  by simp
  moreover have nex':  $\langle \neg (\lambda x. c * f x) \text{ summable\_on } A \rangle$ 
  using asm False apply (subst summable_on_cmult_right) by auto
  ultimately show ?thesis
  unfolding infsum_def by simp
qed
qed

```

```

lemma has_sum_constant[simp]:
  assumes  $\langle \text{finite } F \rangle$ 
  shows  $\langle \text{has\_sum } (\lambda_. c) F \text{ (of\_nat (card } F) * c) \rangle$ 
  by (metis assms has_sum_finite sum_constant)

```

```

lemma infsum_constant[simp]:
  assumes  $\langle \text{finite } F \rangle$ 
  shows  $\langle \text{infsum } (\lambda_. c) F = \text{of\_nat (card } F) * c \rangle$ 
  apply (subst infsum_finite[OF assms]) by simp

```

```

lemma infsum_diverge_constant:

```

— This probably does not really need all of *archimedean_field* but Isabelle/HOL has no type class such as, e.g., "archimedean ring".

```

  fixes c ::  $\langle 'a::\{\text{archimedean\_field, comm\_monoid\_add, linorder\_topology, topological\_semigroup\_mult}\} \rangle$ 

```

```

  assumes  $\langle \text{infinite } A \rangle$  and  $\langle c \neq 0 \rangle$ 

```

```

  shows  $\langle \neg (\lambda_. c) \text{ summable\_on } A \rangle$ 

```

```

proof (rule notI)

```

```

  assume  $\langle (\lambda_. c) \text{ summable\_on } A \rangle$ 

```

```

  then have  $\langle (\lambda_. \text{inverse } c * c) \text{ summable\_on } A \rangle$ 

```

```

  by (rule summable_on_cmult_right)

```

```

  then have [simp]:  $\langle (\lambda_. 1::'a) \text{ summable\_on } A \rangle$ 

```

```

  using assms by auto

```

```

  have  $\langle \text{infsum } (\lambda_. 1) A \geq d \rangle$  for  $d :: 'a$ 

```

```

proof -

```

```

  obtain n :: nat where  $\langle \text{of\_nat } n \geq d \rangle$ 

```

```

  by (meson real_arch_simple)

```

```

  from assms

```

```

  obtain F where  $\langle F \subseteq A \rangle$  and  $\langle \text{finite } F \rangle$  and  $\langle \text{card } F = n \rangle$ 

```

```

  by (meson infinite_arbitrarily_large)

```

```

  note  $\langle d \leq \text{of\_nat } n \rangle$ 

```

```

  also have  $\langle \text{of\_nat } n = \text{infsum } (\lambda_. 1::'a) F \rangle$ 

```

```

  by (simp add:  $\langle \text{card } F = n \rangle \langle \text{finite } F \rangle$ )

```

```

  also have  $\langle \dots \leq \text{infsum } (\lambda_. 1::'a) A \rangle$ 

```

```

  apply (rule infsum_mono_neutral)

```

```

  using  $\langle \text{finite } F \rangle \langle F \subseteq A \rangle$  by auto

```

```

    finally show ?thesis .
qed
then show False
  by (meson linordered_field_no_ub not_less)
qed

```

```

lemma has_sum_constant_archimedean[simp]:
  — This probably does not really need all of archimedean_field but Isabelle/HOL
  has no type class such as, e.g., "archimedean ring".
  fixes c :: ‹'a::{archimedean_field, comm_monoid_add, linorder_topology, topo-
  logical_semigroup_mult}›
  shows ‹infsum (λ_. c) A = of_nat (card A) * c›
  apply (cases ‹finite A›)
  apply simp
  apply (cases ‹c = 0›)
  apply simp
  by (simp add: infsum_diverge_constant infsum_not_exists)

```

```

lemma has_sum_uminus:
  fixes f :: ‹'a ⇒ 'b::topological_ab_group_add›
  shows ‹has_sum (λx. - f x) A a ⟷ has_sum f A (- a)›
  by (auto simp add: sum_negf[abs_def] tendsto_minus_cancel_left has_sum_def)

```

```

lemma summable_on_uminus:
  fixes f :: ‹'a ⇒ 'b::topological_ab_group_add›
  shows ‹(λx. - f x) summable_on A ⟷ f summable_on A›
  by (metis summable_on_def has_sum_uminus verit_minus_simplify(4))

```

```

lemma infsum_uminus:
  fixes f :: ‹'a ⇒ 'b::{topological_ab_group_add, t2_space}›
  shows ‹infsum (λx. - f x) A = - infsum f A›
  by (metis (full_types) add.inverse_inverse add.inverse_neutral infsumI infsum_def
  has_sum_infsum has_sum_uminus)

```

```

lemma has_sum_le_finite_sums:
  fixes a :: ‹'a::{comm_monoid_add, topological_space, linorder_topology}›
  assumes ‹has_sum f A a›
  assumes ‹⋀F. finite F ⟹ F ⊆ A ⟹ sum f F ≤ b›
  shows ‹a ≤ b›
proof -
  from assms(1)
  have 1: ‹(sum f ⟶ a) (finite_subsets_at_top A)›
    unfolding has_sum_def .
  from assms(2)
  have 2: ‹∀ F. F in finite_subsets_at_top A. sum f F ≤ b›
    by (rule_tac eventually_finite_subsets_at_top_weakI)
  show ‹a ≤ b›
    using 1 2
    apply (rule tendsto_le[where f=λ_. b])

```

by auto
qed

lemma *infsum_le_finite_sums*:
 fixes $b :: \langle 'a :: \{comm\_monoid\_add, topological\_space, linorder\_topology\} \rangle$
 assumes $\langle f \text{ summable\_on } A \rangle$
 assumes $\langle \bigwedge F. \text{finite } F \implies F \subseteq A \implies \text{sum } f F \leq b \rangle$
 shows $\langle \text{infsum } f A \leq b \rangle$
 by (meson assms(1) assms(2) has_sum_infsum has_sum_le_finite_sums)

lemma *summable_on_scaleR_left* [intro]:
 fixes $c :: \langle 'a :: \text{real\_normed\_vector} \rangle$
 assumes $c \neq 0 \implies f \text{ summable\_on } A$
 shows $(\lambda x. f x *_{\mathbb{R}} c) \text{ summable\_on } A$
 apply (cases $\langle c \neq 0 \rangle$)
 apply (subst asm_rl[of $\langle (\lambda x. f x *_{\mathbb{R}} c) = (\lambda y. y *_{\mathbb{R}} c) \circ f \rangle$], simp add: o_def)
 apply (rule summable_on_comm_additive)
 using assms by (auto simp add: scaleR_left.additive_axioms)

lemma *summable_on_scaleR_right* [intro]:
 fixes $f :: \langle 'a \Rightarrow 'b :: \text{real\_normed\_vector} \rangle$
 assumes $c \neq 0 \implies f \text{ summable\_on } A$
 shows $(\lambda x. c *_{\mathbb{R}} f x) \text{ summable\_on } A$
 apply (cases $\langle c \neq 0 \rangle$)
 apply (subst asm_rl[of $\langle (\lambda x. c *_{\mathbb{R}} f x) = (\lambda y. c *_{\mathbb{R}} y) \circ f \rangle$], simp add: o_def)
 apply (rule summable_on_comm_additive)
 using assms by (auto simp add: scaleR_right.additive_axioms)

lemma *infsum_scaleR_left*:
 fixes $c :: \langle 'a :: \text{real\_normed\_vector} \rangle$
 assumes $c \neq 0 \implies f \text{ summable\_on } A$
 shows $\text{infsum } (\lambda x. f x *_{\mathbb{R}} c) A = \text{infsum } f A *_{\mathbb{R}} c$
 apply (cases $\langle c \neq 0 \rangle$)
 apply (subst asm_rl[of $\langle (\lambda x. f x *_{\mathbb{R}} c) = (\lambda y. y *_{\mathbb{R}} c) \circ f \rangle$], simp add: o_def)
 apply (rule infsum_comm_additive)
 using assms by (auto simp add: scaleR_left.additive_axioms)

lemma *infsum_scaleR_right*:
 fixes $f :: \langle 'a \Rightarrow 'b :: \text{real\_normed\_vector} \rangle$
 shows $\text{infsum } (\lambda x. c *_{\mathbb{R}} f x) A = c *_{\mathbb{R}} \text{infsum } f A$
proof –
 consider (summable) $\langle f \text{ summable\_on } A \rangle \mid (c0) \langle c = 0 \rangle \mid (\text{not\_summable}) \langle \neg f \text{ summable\_on } A \rangle \langle c \neq 0 \rangle$
 by auto
 then show ?thesis
proof cases
 case summable

```

then show ?thesis
  apply (subst asm_rl[of  $\langle (\lambda x. c *_{\mathbb{R}} f x) = (\lambda y. c *_{\mathbb{R}} y) \circ f \rangle$ ], simp add: o_def)
  apply (rule infsum_comm_additive)
  using summable by (auto simp add: scaleR_right.additive_axioms)
next
case c0
then show ?thesis by auto
next
case not_summable
have  $\neg (\lambda x. c *_{\mathbb{R}} f x) \text{ summable\_on } A$ 
proof (rule notI)
  assume  $\langle (\lambda x. c *_{\mathbb{R}} f x) \text{ summable\_on } A \rangle$ 
  then have  $\langle (\lambda x. \text{inverse } c *_{\mathbb{R}} c *_{\mathbb{R}} f x) \text{ summable\_on } A \rangle$ 
    using summable_on_scaleR_right by blast
  then have  $\langle f \text{ summable\_on } A \rangle$ 
    using not_summable by auto
  with not_summable show False
  by simp
qed
then show ?thesis
  by (simp add: infsum_not_exists not_summable(1))
qed
qed

```

```

lemma infsum_Un_Int:
  fixes f :: 'a  $\Rightarrow$  'b::{topological_ab_group_add, t2_space}
  assumes [simp]: f summable_on A - B f summable_on B - A  $\langle f \text{ summable\_on } A \cap B \rangle$ 
  shows infsum f (A  $\cup$  B) = infsum f A + infsum f B - infsum f (A  $\cap$  B)
proof -
  have [simp]:  $\langle f \text{ summable\_on } A \rangle$ 
  apply (subst asm_rl[of  $\langle A = (A - B) \cup (A \cap B) \rangle$ ]) apply auto[1]
  apply (rule summable_on_Un_disjoint)
  by auto
  have  $\langle \text{infsum } f (A \cup B) = \text{infsum } f A + \text{infsum } f (B - A) \rangle$ 
  apply (subst infsum_Un_disjoint[symmetric])
  by auto
  moreover have  $\langle \text{infsum } f (B - A \cup A \cap B) = \text{infsum } f (B - A) + \text{infsum } f (A \cap B) \rangle$ 
  by (rule infsum_Un_disjoint) auto
  moreover have B - A  $\cup$  A  $\cap$  B = B
  by blast
  ultimately show ?thesis
  by auto
qed

```

```

lemma inj_combinator':
  assumes x  $\notin$  F

```

```

shows ⟨inj_on (λ(g, y). g(x := y)) (PiE F B × B x)⟩
proof -
  have inj_on ((λ(y, g). g(x := y)) ∘ prod.swap) (PiE F B × B x)
    using inj_combinator[of x F B] assms by (intro comp_inj_on) (auto simp:
product_swap)
  thus ?thesis
    by (simp add: o_def)
qed

```

lemma *infsum_prod_PiE*:

— See also *infsum_prod_PiE_abs* below with incomparable premises.

fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \{\text{comm\_monoid\_mult}, \text{topological\_semigroup\_mult}, \text{division\_ring}, \text{banach}\}$

assumes *finite*: *finite A*

assumes $\bigwedge x. x \in A \implies f\ x\ \text{summable\_on}\ B\ x$

assumes $(\lambda g. \prod_{x \in A}. f\ x\ (g\ x))\ \text{summable\_on}\ (PiE\ A\ B)$

shows $\text{infsum}\ (\lambda g. \prod_{x \in A}. f\ x\ (g\ x))\ (PiE\ A\ B) = (\prod_{x \in A}. \text{infsum}\ (f\ x)\ (B\ x))$

proof (use *finite* *assms*(2) **in** *induction*)

case *empty*

then show ?*case*

by *auto*

next

case (*insert x F*)

have *pi*: $\langle PiE\ (\text{insert}\ x\ F)\ B = (\lambda(g, y). g(x := y))\ ' (PiE\ F\ B \times B\ x) \rangle$

unfolding *PiE_insert_eq*

by (*subst* *swap_product* [*symmetric*]) (*simp* *add*: *image_image* *case_prod_unfold*)

have *prod*: $\langle (\prod_{x' \in F}. f\ x'\ ((p(x := y))\ x')) = (\prod_{x' \in F}. f\ x'\ (p\ x')) \rangle$ **for** *p y*

by (*rule* *prod.cong*) (*use* *insert.hyps* **in** *auto*)

have *inj*: $\langle \text{inj\_on}\ (\lambda(g, y). g(x := y))\ (PiE\ F\ B \times B\ x) \rangle$

using $\langle x \notin F \rangle$ **by** (*rule* *inj_combinator'*)

have *summable1*: $\langle (\lambda g. \prod_{x \in \text{insert}\ x\ F}. f\ x\ (g\ x))\ \text{summable\_on}\ PiE\ (\text{insert}\ x\ F)\ B \rangle$

using *insert.premis*(2) .

also have $\langle PiE\ (\text{insert}\ x\ F)\ B = (\lambda(g, y). g(x := y))\ ' (PiE\ F\ B \times B\ x) \rangle$

by (*simp* *only*: *pi*)

also have $(\lambda g. \prod_{x \in \text{insert}\ x\ F}. f\ x\ (g\ x))\ \text{summable\_on}\ \dots \longleftrightarrow$

$((\lambda g. \prod_{x \in \text{insert}\ x\ F}. f\ x\ (g\ x)) \circ (\lambda(g, y). g(x := y)))\ \text{summable\_on}\ (PiE\ F\ B \times B\ x)$

using *inj* **by** (*rule* *summable_on_reindex*)

also have $(\prod_{z \in F}. f\ z\ ((g(x := y))\ z)) = (\prod_{z \in F}. f\ z\ (g\ z))$ **for** *g y*

using *insert.hyps* **by** (*intro* *prod.cong*) *auto*

hence $((\lambda g. \prod_{x \in \text{insert}\ x\ F}. f\ x\ (g\ x)) \circ (\lambda(g, y). g(x := y))) =$

$(\lambda(p, y). f\ x\ y * (\prod_{x' \in F}. f\ x'\ (p\ x')))$

using *insert.hyps* **by** (*auto* *simp*: *fun_eq_iff* *cong*: *prod.cong_simp*)

finally have *summable2*: $\langle (\lambda(p, y). f\ x\ y * (\prod_{x' \in F}. f\ x'\ (p\ x')))\ \text{summable\_on}\ PiE\ F\ B \times B\ x \rangle$.


```

then have  $\langle (\lambda p. \sum_{\infty y \in B} x. f \ x \ y * (\prod_{x' \in F} f \ x' \ (p \ x'))) \text{ summable\_on } Pi_E \ F \ B \rangle$ 
by (rule summable_on_Sigma_banach)
then have  $\langle (\lambda p. (\sum_{\infty y \in B} x. f \ x \ y) * (\prod_{x' \in F} f \ x' \ (p \ x'))) \text{ summable\_on } Pi_E \ F \ B \rangle$ 
if  $\langle (\sum_{\infty y \in B} x. f \ x \ y) \neq 0 \rangle$ 
apply (subst infsum_cmult_left[symmetric])
using insert.premis(1) by blast
then have summable3:  $\langle (\lambda p. (\prod_{x' \in F} f \ x' \ (p \ x'))) \text{ summable\_on } Pi_E \ F \ B \rangle$ 
if  $\langle (\sum_{\infty y \in B} x. f \ x \ y) \neq 0 \rangle$ 
apply (subst (asm) summable_on_cmult_right')
using that by auto

have  $\langle (\sum_{\infty g \in Pi_E} (\text{insert } x \ F) \ B. \prod_{x \in \text{insert } x \ F} f \ x \ (g \ x)) = (\sum_{\infty (p,y) \in Pi_E \ F \ B \times B} x. \prod_{x' \in \text{insert } x \ F} f \ x' \ ((p(x:=y)) \ x')) \rangle$ 
apply (subst pi)
apply (subst infsum_reindex)
using inj by (auto simp: o_def case_prod_unfold)
also have  $\langle \dots = (\sum_{\infty (p, y) \in Pi_E \ F \ B \times B} x. f \ x \ y * (\prod_{x' \in F} f \ x' \ ((p(x:=y)) \ x'))) \rangle$ 
apply (subst prod.insert)
using insert by auto
also have  $\langle \dots = (\sum_{\infty (p, y) \in Pi_E \ F \ B \times B} x. f \ x \ y * (\prod_{x' \in F} f \ x' \ (p \ x'))) \rangle$ 
apply (subst prod) by rule
also have  $\langle \dots = (\sum_{\infty p \in Pi_E \ F \ B} \sum_{\infty y \in B} x. f \ x \ y * (\prod_{x' \in F} f \ x' \ (p \ x'))) \rangle$ 
apply (subst infsum_Sigma_banach[symmetric])
using summable2 apply blast
by fastforce
also have  $\langle \dots = (\sum_{\infty y \in B} x. f \ x \ y) * (\sum_{\infty p \in Pi_E \ F \ B} \prod_{x' \in F} f \ x' \ (p \ x')) \rangle$ 
apply (subst infsum_cmult_left')
apply (subst infsum_cmult_right')
by (rule refl)
also have  $\langle \dots = (\prod_{x \in \text{insert } x \ F} \text{infsum } (f \ x) \ (B \ x)) \rangle$ 
apply (subst prod.insert)
using  $\langle \text{finite } F \rangle \ \langle x \notin F \rangle$  apply auto[2]
apply (cases  $\langle \text{infsum } (f \ x) \ (B \ x) = 0 \rangle$ )
apply simp
apply (subst insert.IH)
apply (simp add: insert.premis(1))
apply (rule summable3)
by auto
finally show ?case
by simp
qed

```

lemma infsum_prod_PiE_abs:

— See also infsum_prod_PiE above with incomparable premises.

fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \{\text{banach, real\_normed\_div\_algebra, comm\_semiring\_1}\}$

assumes finite: finite A

assumes $\bigwedge x. x \in A \implies f \ x \ \text{abs\_summable\_on } B \ x$

```

shows infsum (λg. ∏ x∈A. f x (g x)) (PiE A B) = (∏ x∈A. infsum (f x) (B
x))
proof (use finite assms(2) in induction)
  case empty
  then show ?case
    by auto
next
case (insert x F)

  have pi: ⟨PiE (insert x F) B = (λ(g,y). g(x:=y)) ‘ (PiE F B × B x)⟩ for x F
and B :: 'a ⇒ 'b set
  unfolding PiE_insert_eq
  by (subst swap_product [symmetric]) (simp add: image_image case_prod_unfold)
  have prod: ⟨(∏ x'∈F. f x' ((p(x:=y)) x')) = (∏ x'∈F. f x' (p x'))⟩ for p y
  by (rule prod.cong) (use insert.hyps in auto)
  have inj: ⟨inj_on (λ(g, y). g(x := y)) (PiE F B × B x)⟩
  using ⟨x ∉ F⟩ by (rule inj_combinator')

define s where ⟨s x = infsum (λy. norm (f x y)) (B x)⟩ for x

have *: ⟨(∑ p∈P. norm (∏ x∈F. f x (p x))) ≤ prod s F⟩
  if P: ⟨P ⊆ PiE F B⟩ and [simp]: ⟨finite P⟩ ⟨finite F⟩
  and sum: ⟨∧ x. x ∈ F ⇒ f x abs_summable_on B x⟩ for P F
proof -
  define B' where ⟨B' x = {p x | p. p ∈ P}⟩ for x
  have [simp]: ⟨finite (B' x)⟩ for x
  using that by (auto simp: B'_def)
  have [simp]: ⟨finite (PiE F B')⟩
  by (simp add: finite_PiE)
  have [simp]: ⟨P ⊆ PiE F B'⟩
  using that by (auto simp: B'_def)
  have B'B: ⟨B' x ⊆ B x⟩ if ⟨x ∈ F⟩ for x
  unfolding B'_def using P that
  by auto
  have s_bound: ⟨(∑ y∈B' x. norm (f x y)) ≤ s x⟩ if ⟨x ∈ F⟩ for x
  apply (simp_all add: s_def flip: infsum_finite)
  apply (rule infsum_mono_neutral)
  using that sum B'B by auto
  have ⟨(∑ p∈P. norm (∏ x∈F. f x (p x))) ≤ (∑ p∈PiE F B'. norm (∏ x∈F. f
x (p x)))⟩
  apply (rule sum_mono2)
  by auto
  also have ⟨... = (∑ p∈PiE F B'. ∏ x∈F. norm (f x (p x)))⟩
  apply (subst prod_norm[symmetric])
  by simp
  also have ⟨... = (∏ x∈F. ∑ y∈B' x. norm (f x y))⟩
proof (use finite F in induction)
  case empty
  then show ?case by simp

```

```

next
case (insert x F)
have aux:  $\langle a = b \implies c * a = c * b \rangle$  for  $a \ b \ c :: \text{real}$ 
  by auto
have inj:  $\langle \text{inj\_on } (\lambda(g, y). g(x := y)) (Pi_E \ F \ B' \times \ B' \ x) \rangle$ 
  by (rule inj_combinator') (use insert.hyps in auto)
have  $\langle (\sum p \in Pi_E \ (insert \ x \ F) \ B'. \prod x \in insert \ x \ F. \text{norm } (f \ x \ (p \ x)))$ 
  =  $(\sum (p, y) \in Pi_E \ F \ B' \times \ B' \ x. \prod x' \in insert \ x \ F. \text{norm } (f \ x' \ ((p(x := y))$ 
 $x')) \rangle$ 
  apply (subst pi)
  apply (subst sum.reindex)
  using inj by (auto simp: case_prod_unfold)
also have  $\langle \dots = (\sum (p, y) \in Pi_E \ F \ B' \times \ B' \ x. \text{norm } (f \ x \ y) * (\prod x' \in F. \text{norm}$ 
 $(f \ x' \ ((p(x := y)) \ x')))) \rangle$ 
  apply (subst prod.insert)
  using insert.hyps by (auto simp: case_prod_unfold)
also have  $\langle \dots = (\sum (p, y) \in Pi_E \ F \ B' \times \ B' \ x. \text{norm } (f \ x \ y) * (\prod x' \in F. \text{norm}$ 
 $(f \ x' \ (p \ x')))) \rangle$ 
  apply (rule sum.cong)
  apply blast
  unfolding case_prod_unfold
  apply (rule aux)
  apply (rule prod.cong)
  using insert.hyps(2) by auto
also have  $\langle \dots = (\sum y \in B' \ x. \text{norm } (f \ x \ y)) * (\sum p \in Pi_E \ F \ B'. \prod x' \in F. \text{norm}$ 
 $(f \ x' \ (p \ x')) \rangle$ 
  apply (subst sum_product)
  apply (subst sum.swap)
  apply (subst sum.cartesian_product)
  by simp
also have  $\langle \dots = (\sum y \in B' \ x. \text{norm } (f \ x \ y)) * (\prod x \in F. \sum y \in B' \ x. \text{norm } (f \ x$ 
 $y)) \rangle$ 
  by (simp add: insert.IH)
also have  $\langle \dots = (\prod x \in insert \ x \ F. \sum y \in B' \ x. \text{norm } (f \ x \ y)) \rangle$ 
  using insert.hyps(1) insert.hyps(2) by force
finally show ?case .
qed
also have  $\langle \dots = (\prod x \in F. \sum_{\infty} y \in B' \ x. \text{norm } (f \ x \ y)) \rangle$ 
  by auto
also have  $\langle \dots \leq (\prod x \in F. s \ x) \rangle$ 
  apply (rule prod_mono)
  apply auto
  apply (simp add: sum_nonneg)
  using s_bound by presburger
finally show ?thesis .
qed
have  $\langle (\lambda g. \prod x \in insert \ x \ F. f \ x \ (g \ x)) \text{abs\_summable\_on } Pi_E \ (insert \ x \ F) \ B \rangle$ 
  apply (rule nonneg_bdd_above_summable_on)
  apply (simp; fail)

```

```

apply (rule bdd_aboveI[where  $M = \langle \prod x' \in \text{insert } x \ F. \ s \ x' \rangle$ ])
using * insert.hyps insert.prem by blast

also have  $\langle \text{Pi}_E \ (\text{insert } x \ F) \ B = (\lambda(g,y). \ g(x:=y)) \ ' \ (\text{Pi}_E \ F \ B \times B \ x) \rangle$ 
by (simp only: pi)
also have  $(\lambda g. \ \prod x \in \text{insert } x \ F. \ f \ x \ (g \ x)) \ \text{abs\_summable\_on} \ \dots \longleftrightarrow$ 
 $((\lambda g. \ \prod x \in \text{insert } x \ F. \ f \ x \ (g \ x)) \circ (\lambda(g,y). \ g(x:=y))) \ \text{abs\_summable\_on}$ 
 $(\text{Pi}_E \ F \ B \times B \ x))$ 
using inj by (subst summable_on_reindex) (auto simp: o_def)
also have  $(\prod z \in F. \ f \ z \ ((g(x := y)) \ z)) = (\prod z \in F. \ f \ z \ (g \ z))$  for  $g \ y$ 
using insert.hyps by (intro prod.cong) auto
hence  $((\lambda g. \ \prod x \in \text{insert } x \ F. \ f \ x \ (g \ x)) \circ (\lambda(g,y). \ g(x:=y))) =$ 
 $(\lambda(p, y). \ f \ x \ y * (\prod x' \in F. \ f \ x' \ (p \ x')))$ 
using insert.hyps by (auto simp: fun_eq_iff cong: prod.cong_simp)
finally have summable2:  $\langle (\lambda(p, y). \ f \ x \ y * (\prod x' \in F. \ f \ x' \ (p \ x'))) \ \text{abs\_summable\_on}$ 
 $\text{Pi}_E \ F \ B \times B \ x \rangle$  .

have  $\langle (\sum_{\infty} g \in \text{Pi}_E \ (\text{insert } x \ F) \ B. \ \prod x \in \text{insert } x \ F. \ f \ x \ (g \ x))$ 
 $= (\sum_{\infty} (p,y) \in \text{Pi}_E \ F \ B \times B \ x. \ \prod x' \in \text{insert } x \ F. \ f \ x' \ ((p(x:=y)) \ x')) \rangle$ 
apply (subst pi)
apply (subst infsum_reindex)
using inj by (auto simp: o_def case_prod_unfold)
also have  $\langle \dots = (\sum_{\infty} (p, y) \in \text{Pi}_E \ F \ B \times B \ x. \ f \ x \ y * (\prod x' \in F. \ f \ x' \ ((p(x:=y))$ 
 $x')) \rangle$ 
apply (subst prod.insert)
using insert by auto
also have  $\langle \dots = (\sum_{\infty} (p, y) \in \text{Pi}_E \ F \ B \times B \ x. \ f \ x \ y * (\prod x' \in F. \ f \ x' \ (p \ x')) \rangle$ 
apply (subst prod) by rule
also have  $\langle \dots = (\sum_{\infty} p \in \text{Pi}_E \ F \ B. \ \sum_{\infty} y \in B \ x. \ f \ x \ y * (\prod x' \in F. \ f \ x' \ (p \ x')) \rangle$ 
apply (subst infsum_Sigma_banach[symmetric])
using summable2 abs_summable_summable apply blast
by fastforce
also have  $\langle \dots = (\sum_{\infty} y \in B \ x. \ f \ x \ y) * (\sum_{\infty} p \in \text{Pi}_E \ F \ B. \ \prod x' \in F. \ f \ x' \ (p \ x')) \rangle$ 
apply (subst infsum_cmult_left')
apply (subst infsum_cmult_right')
by (rule refl)
also have  $\langle \dots = (\prod x \in \text{insert } x \ F. \ \text{infsum } (f \ x) \ (B \ x)) \rangle$ 
apply (subst prod.insert)
using  $\langle \text{finite } F \rangle \ \langle x \notin F \rangle$  apply auto[2]
apply (cases  $\langle \text{infsum } (f \ x) \ (B \ x) = 0 \rangle$ )
apply (simp; fail)
apply (subst insert.IH)
apply (auto simp add: insert.prem(1))
done
finally show ?case
by simp
qed

```

6.37.3 Absolute convergence

```

lemma abs_summable_countable:
  assumes ⟨f abs_summable_on A⟩
  shows ⟨countable {x∈A. f x ≠ 0}⟩
proof -
  have fin: ⟨finite {x∈A. norm (f x) ≥ t}⟩ if ⟨t > 0⟩ for t
  proof (rule ccontr)
    assume *: ⟨infinite {x ∈ A. t ≤ norm (f x)}⟩
    have ⟨infsum (λx. norm (f x)) A ≥ b⟩ for b
    proof -
      obtain b' where b': ⟨of_nat b' ≥ b / t⟩
      by (meson real_arch_simple)
      from *
      obtain F where cardF: ⟨card F ≥ b'⟩ and ⟨finite F⟩ and F: ⟨F ⊆ {x ∈ A.
t ≤ norm (f x)}⟩
      by (meson finite_if_finite_subsets_card_bdd_nle_le)
      have ⟨b ≤ of_nat b' * t⟩
      using b' ⟨t > 0⟩ by (simp add: field_simps split: if_splits)
      also have ⟨... ≤ of_nat (card F) * t⟩
      by (simp add: cardF that)
      also have ⟨... = sum (λx. t) F⟩
      by simp
      also have ⟨... ≤ sum (λx. norm (f x)) F⟩
      by (metis (mono_tags, lifting) F in_mono mem_Collect_eq sum_mono)
      also have ⟨... = infsum (λx. norm (f x)) F⟩
      using ⟨finite F⟩ by (rule infsum_finite[symmetric])
      also have ⟨... ≤ infsum (λx. norm (f x)) A⟩
      by (rule infsum_mono_neutral) (use ⟨finite F⟩ assms F in auto)
      finally show ?thesis .
    qed
    then show False
    by (meson gt_ex linorder_not_less)
  qed
  have ⟨countable (⋃ i∈{1..}. {x∈A. norm (f x) ≥ 1/of_nat i})⟩
  by (rule countable_UN) (use fin in ⟨auto intro!: countable_finite⟩)
  also have ⟨... = {x∈A. f x ≠ 0}⟩
  proof safe
    fix x assume x: x ∈ A f x ≠ 0
    define i where i = max 1 (nat (ceiling (1 / norm (f x))))
    have i ≥ 1
    by (simp add: i_def)
    moreover have real i ≥ 1 / norm (f x)
    unfolding i_def by linarith
    hence 1 / real i ≤ norm (f x) using ⟨f x ≠ 0⟩
    by (auto simp: divide_simps mult_ac)
    ultimately show x ∈ (⋃ i∈{1..}. {x ∈ A. 1 / real i ≤ norm (f x)})
    using ⟨x ∈ A⟩ by auto
  qed auto
  finally show ?thesis .

```

qed

```

lemma summable_on_iff_abs_summable_on_real:
  fixes f :: 'a  $\Rightarrow$  real
  shows  $\langle f \text{ summable\_on } A \longleftrightarrow f \text{ abs\_summable\_on } A \rangle$ 
proof (rule iffI)
  assume  $\langle f \text{ summable\_on } A \rangle$ 
  define n Ap An
    where  $\langle n \ x = \text{norm } (f \ x) \rangle$  and  $\langle A_p = \{x \in A. f \ x \geq 0\} \rangle$  and  $\langle A_n = \{x \in A. f \ x < 0\} \rangle$  for x
  have [simp]:  $\langle A_p \cup A_n = A \rangle$   $\langle A_p \cap A_n = \{\} \rangle$ 
    by (auto simp: Ap_def An_def)
  from  $\langle f \text{ summable\_on } A \rangle$  have [simp]:  $\langle f \text{ summable\_on } A_p \rangle$   $\langle f \text{ summable\_on } A_n \rangle$ 
    using Ap_def An_def summable_on_subset_banach by fastforce+
  then have [simp]:  $\langle n \text{ summable\_on } A_p \rangle$ 
    apply (subst summable_on_cong[where g=f])
    by (simp_all add: Ap_def n_def)
  moreover have [simp]:  $\langle n \text{ summable\_on } A_n \rangle$ 
    apply (subst summable_on_cong[where g= $\lambda x. - f \ x$ ])
    apply (simp add: An_def n_def [abs_def])
    by (simp add: summable_on_uminus)
  ultimately have [simp]:  $\langle n \text{ summable\_on } (A_p \cup A_n) \rangle$ 
    apply (rule summable_on_Un_disjoint) by simp
  then show  $\langle n \text{ summable\_on } A \rangle$ 
    by simp
next
  show  $\langle f \text{ abs\_summable\_on } A \Longrightarrow f \text{ summable\_on } A \rangle$ 
    using abs_summable_summable by blast
qed

```

```

lemma abs_summable_on_Sigma_iff:
  shows  $f \text{ abs\_summable\_on } \text{Sigma } A \ B \longleftrightarrow$ 
     $(\forall x \in A. (\lambda y. f \ (x, y)) \text{ abs\_summable\_on } B \ x) \wedge$ 
     $((\lambda x. \text{infsum } (\lambda y. \text{norm } (f \ (x, y))) \ (B \ x)) \text{ abs\_summable\_on } A)$ 
proof (intro iffI conjI ballI)
  assume asm:  $\langle f \text{ abs\_summable\_on } \text{Sigma } A \ B \rangle$ 
  then have  $\langle (\lambda x. \text{infsum } (\lambda y. \text{norm } (f \ (x, y))) \ (B \ x)) \text{ summable\_on } A \rangle$ 
    apply (rule_tac summable_on_Sigma_banach)
    by (auto simp: case_prod_unfold)
  then show  $\langle (\lambda x. \sum_{y \in B} x. \text{norm } (f \ (x, y))) \text{ abs\_summable\_on } A \rangle$ 
    using summable_on_iff_abs_summable_on_real by force

  show  $\langle (\lambda y. f \ (x, y)) \text{ abs\_summable\_on } B \ x \rangle$  if  $\langle x \in A \rangle$  for x
  proof -
    from asm have  $\langle f \text{ abs\_summable\_on } \text{Pair } x \ ' B \ x \rangle$ 
      apply (rule summable_on_subset_banach)
      using that by auto
  end

```

```

    then show ?thesis
      apply (subst (asm) summable_on_reindex)
      by (auto simp: o_def inj_on_def)
  qed
next
  assume asm:  $\langle (\forall x \in A. (\lambda xa. f(x, xa)) \text{ abs\_summable\_on } B \ x) \wedge$ 
     $(\lambda x. \sum_{y \in B} x. \text{ norm } (f(x, y))) \text{ abs\_summable\_on } A \rangle$ 
  have  $\langle (\sum_{xy \in F. \text{ norm } (f xy)) \leq (\sum_{x \in A. \sum_{y \in B} x. \text{ norm } (f(x, y))) \rangle$ 
    if  $\langle F \subseteq \text{Sigma } A \ B \rangle$  and [simp]:  $\langle \text{finite } F \rangle$  for  $F$ 
  proof -
    have [simp]:  $\langle (\text{SIGMA } x:\text{fst } 'F. \{y. (x, y) \in F\}) = F \rangle$ 
      by (auto intro!: set_eqI simp add: Domain.DomainI fst_eq_Domain)
    have [simp]:  $\langle \text{finite } \{y. (x, y) \in F\} \rangle$  for  $x$ 
      by (metis  $\langle \text{finite } F \rangle$  Range.intros finite_Range finite_subset mem_Collect_eq
        subsetI)
    have  $\langle (\sum_{xy \in F. \text{ norm } (f xy)) = (\sum_{x \in \text{fst } 'F. \sum_{y \in \{y. (x, y) \in F\}. \text{ norm } (f(x, y)) \rangle}$ 
      apply (subst sum.Sigma)
      by auto
    also have  $\langle \dots = (\sum_{x \in \text{fst } 'F. \sum_{y \in \{y. (x, y) \in F\}. \text{ norm } (f(x, y)) \rangle$ 
      apply (subst infsum_finite)
      by auto
    also have  $\langle \dots \leq (\sum_{x \in \text{fst } 'F. \sum_{y \in B} x. \text{ norm } (f(x, y)) \rangle$ 
      apply (rule infsum_mono)
      apply (simp; fail)
      apply (simp; fail)
      apply (rule infsum_mono_neutral)
      using asm that(1) by auto
    also have  $\langle \dots \leq (\sum_{x \in A. \sum_{y \in B} x. \text{ norm } (f(x, y)) \rangle$ 
      by (rule infsum_mono_neutral) (use asm that(1) in  $\langle \text{auto simp add: infsum\_nonneg} \rangle$ )
    finally show ?thesis .
  qed
  then show  $\langle f \text{ abs\_summable\_on } \text{Sigma } A \ B \rangle$ 
    by (intro nonneg_bdd_above_summable_on) (auto simp: bdd_above_def)
qed

lemma abs_summable_on_comparison_test:
  assumes  $g \text{ abs\_summable\_on } A$ 
  assumes  $\bigwedge x. x \in A \implies \text{norm } (f x) \leq \text{norm } (g x)$ 
  shows  $f \text{ abs\_summable\_on } A$ 
proof (rule nonneg_bdd_above_summable_on)
  show bdd_above (sum  $(\lambda x. \text{norm } (f x))$ )  $' \{F. F \subseteq A \wedge \text{finite } F\}$ 
  proof (rule bdd_aboveI2)
    fix  $F$  assume  $F: F \in \{F. F \subseteq A \wedge \text{finite } F\}$ 
    have  $\langle \text{sum } (\lambda x. \text{norm } (f x)) \ F \leq \text{sum } (\lambda x. \text{norm } (g x)) \ F \rangle$ 
      using assms  $F$  by (intro sum_mono) auto
    also have  $\langle \dots = \text{infsum } (\lambda x. \text{norm } (g x)) \ F \rangle$ 
      using  $F$  by simp
  end

```

```

    also have ⟨... ≤ infsum (λx. norm (g x)) A⟩
  proof (rule infsum_mono_neutral)
    show g abs_summable_on F
      by (rule summable_on_subset_banach[OF assms(1)]) (use F in auto)
    qed (use F assms in auto)
    finally show (∑ x∈F. norm (f x)) ≤ (∑∞ x∈A. norm (g x)) .
  qed
qed auto

```

```

lemma abs_summable_iff_bdd_above:
  fixes f :: ⟨'a ⇒ 'b::real_normed_vector⟩
  shows ⟨f abs_summable_on A ⟷ bdd_above (sum (λx. norm (f x))) ‘ {F. F⊆A
    ∧ finite F}⟩
proof (rule iffI)
  assume ⟨f abs_summable_on A⟩
  show ⟨bdd_above (sum (λx. norm (f x))) ‘ {F. F ⊆ A ∧ finite F}⟩
  proof (rule bdd_aboveI2)
    fix F assume F: F ∈ {F. F ⊆ A ∧ finite F}
    show (∑ x∈F. norm (f x)) ≤ (∑∞ x∈A. norm (f x))
      by (rule finite_sum_le_infsum) (use ⟨f abs_summable_on A⟩ F in auto)
    qed
  next
    assume ⟨bdd_above (sum (λx. norm (f x))) ‘ {F. F⊆A ∧ finite F}⟩
    then show ⟨f abs_summable_on A⟩
      by (simp add: nonneg_bdd_above_summable_on)
  qed

```

```

lemma abs_summable_product:
  fixes x :: 'a ⇒ 'b::{real_normed_div_algebra,banach,second_countable_topology}
  assumes x2_sum: (λi. (x i) * (x i)) abs_summable_on A
    and y2_sum: (λi. (y i) * (y i)) abs_summable_on A
  shows (λi. x i * y i) abs_summable_on A
proof (rule nonneg_bdd_above_summable_on)
  show bdd_above (sum (λxa. norm (x xa * y xa))) ‘ {F. F ⊆ A ∧ finite F}
  proof (rule bdd_aboveI2)
    fix F assume F: ⟨F ∈ {F. F ⊆ A ∧ finite F}⟩
    then have r1: finite F and b4: F ⊆ A
      by auto
    have a1: (∑∞ i∈F. norm (x i * x i)) ≤ (∑∞ i∈A. norm (x i * x i))
      apply (rule infsum_mono_neutral)
      using b4 r1 x2_sum by auto
    have norm (x i * y i) ≤ norm (x i * x i) + norm (y i * y i) for i
      unfolding norm_mult by (smt mult_left_mono mult_nonneg_nonneg mult_right_mono
        norm_ge_zero)
    hence (∑ i∈F. norm (x i * y i)) ≤ (∑ i∈F. norm (x i * x i) + norm (y i *
      y i))
      by (simp add: sum_mono)

```



```

    also have ... = ( $\sum_{i \in F}. \text{norm } (x \ i * x \ i)$ ) + ( $\sum_{i \in F}. \text{norm } (y \ i * y \ i)$ )
      by (simp add: sum.distrib)
    also have ... = ( $\sum_{\infty i \in F}. \text{norm } (x \ i * x \ i)$ ) + ( $\sum_{\infty i \in F}. \text{norm } (y \ i * y \ i)$ )
      by (simp add: ‹finite F›)
    also have ...  $\leq$  ( $\sum_{\infty i \in A}. \text{norm } (x \ i * x \ i)$ ) + ( $\sum_{\infty i \in A}. \text{norm } (y \ i * y \ i)$ )
      using F assms
      by (intro add_mono infsum_mono2) auto
    finally show ‹( $\sum_{x a \in F}. \text{norm } (x \ x a * y \ x a)$ )  $\leq$  ( $\sum_{\infty i \in A}. \text{norm } (x \ i * x \ i)$ )
      + ( $\sum_{\infty i \in A}. \text{norm } (y \ i * y \ i)$ )›
      by simp
  qed
qed auto

```

6.37.4 Extended reals and nats

```

lemma summable_on_enreal[simp]: ‹(f::_  $\Rightarrow$  ennreal) summable_on S›
  by (rule nonneg_summable_on_complete) simp

```

```

lemma summable_on_enat[simp]: ‹(f::_  $\Rightarrow$  enat) summable_on S›
  by (rule nonneg_summable_on_complete) simp

```

```

lemma has_sum_superconst_infinite_enreal:

```

```

  fixes f :: ‹'a  $\Rightarrow$  ennreal›
  assumes geqb: ‹ $\bigwedge x. x \in S \implies f \ x \geq b$ ›
  assumes b: ‹ $b > 0$ ›
  assumes ‹infinite S›
  shows has_sum f S  $\infty$ 

```

```

proof -

```

```

  have ‹(sum f  $\longrightarrow$   $\infty$ ) (finite_subsets_at_top S)›

```

```

  proof (rule order_tendstoI[rotated], simp)

```

```

    fix y :: ennreal assume ‹y <  $\infty$ ›

```

```

    then have ‹y / b <  $\infty$ ›

```

```

    by (metis b ennreal_divide_eq_top_iff gr_implies_not_zero infinity_enreal_def
top.not_eq_extremum)

```

```

    then obtain F where ‹finite F› and ‹F  $\subseteq$  S› and cardF: ‹card F > y / b›

```

```

      using ‹infinite S›

```

```

    by (metis ennreal_Ex_less_of_nat infinite_arbitrarily_large infinity_enreal_def)

```

```

    moreover have ‹sum f Y > y› if ‹finite Y› and ‹F  $\subseteq$  Y› and ‹Y  $\subseteq$  S› for

```

```

Y

```

```

  proof -

```

```

    have ‹y < b * card F›

```

```

      by (metis ‹y <  $\infty$ › b cardF divide_less_enreal ennreal_mult_eq_top_iff
gr_implies_not_zero infinity_enreal_def mult.commute top.not_eq_extremum)

```

```

    also have ‹...  $\leq$  b * card Y›

```

```

      by (meson b card_mono less_imp_le mult_left_mono of_nat_le_iff that(1)
that(2))

```

```

    also have ‹... = sum ( $\lambda \_. b$ ) Y›

```

```

      by (simp add: mult.commute)

```

```

    also have ‹...  $\leq$  sum f Y›

```

```

    using geqb by (meson subset_eq sum_mono that(3))
    finally show ?thesis .
  qed
  ultimately show  $\langle \forall_F x \text{ in } \text{finite\_subsets\_at\_top } S. y < \text{sum } f x \rangle$ 
    unfolding eventually_finite_subsets_at_top
    by auto
  qed
  then show ?thesis
    by (simp add: has_sum_def)
  qed

```

```

lemma infsum_superconst_infinite_ennreal:
  fixes f ::  $\langle 'a \Rightarrow \text{ennreal} \rangle$ 
  assumes  $\langle \bigwedge x. x \in S \implies f x \geq b \rangle$ 
  assumes  $\langle b > 0 \rangle$ 
  assumes  $\langle \text{infinite } S \rangle$ 
  shows  $\text{infsum } f S = \infty$ 
  using assms infsumI has_sum_superconst_infinite_ennreal by blast

```

```

lemma infsum_superconst_infinite_ereal:
  fixes f ::  $\langle 'a \Rightarrow \text{ereal} \rangle$ 
  assumes geqb:  $\langle \bigwedge x. x \in S \implies f x \geq b \rangle$ 
  assumes b:  $\langle b > 0 \rangle$ 
  assumes  $\langle \text{infinite } S \rangle$ 
  shows  $\text{infsum } f S = \infty$ 
proof -
  obtain b' where b':  $\langle \text{e2ennreal } b' = b \rangle$  and  $\langle b' > 0 \rangle$ 
    using b by blast
  have  $0 < \text{e2ennreal } b$ 
    using b' b
  by (metis dual_order.refl enn2ereal_e2ennreal gr_zeroI order_less_le zero_ennreal.abs_eq)
  hence *:  $\langle \text{infsum } (\text{e2ennreal } o f) S = \infty \rangle$ 
    using assms b'
  by (intro infsum_superconst_infinite_ennreal[where b=b']) (auto intro!: e2ennreal_mono)
  have  $\langle \text{infsum } f S = \text{infsum } (\text{enn2ereal } o (\text{e2ennreal } o f)) S \rangle$ 
    using geqb b by (intro infsum_cong) (fastforce simp: enn2ereal_e2ennreal)
  also have  $\langle \dots = \text{enn2ereal } \infty \rangle$ 
    apply (subst infsum_comm_additive_general)
    using * by (auto simp: continuous_at_enn2ereal)
  also have  $\langle \dots = \infty \rangle$ 
    by simp
  finally show ?thesis .
  qed

```

```

lemma has_sum_superconst_infinite_ereal:
  fixes f ::  $\langle 'a \Rightarrow \text{ereal} \rangle$ 
  assumes  $\langle \bigwedge x. x \in S \implies f x \geq b \rangle$ 
  assumes  $\langle b > 0 \rangle$ 

```

```

assumes  $\langle \text{infinite } S \rangle$ 
shows  $\text{has\_sum } f \ S \ \infty$ 
by (metis Infty_neq_0(1) assms infsum_def has_sum_infsum infsum_superconst_infinite_ereal)

lemma infsum_superconst_infinite_enat:
  fixes  $f :: \langle 'a \Rightarrow \text{enat} \rangle$ 
  assumes geqb:  $\langle \bigwedge x. x \in S \implies f \ x \geq b \rangle$ 
  assumes  $b: \langle b > 0 \rangle$ 
  assumes  $\langle \text{infinite } S \rangle$ 
  shows  $\text{infsum } f \ S = \infty$ 
proof –
  have  $\langle \text{ennreal\_of\_enat } (\text{infsum } f \ S) = \text{infsum } (\text{ennreal\_of\_enat } o \ f) \ S \rangle$ 
  apply (rule infsum_comm_additive_general[symmetric])
  by auto
  also have  $\langle \dots = \infty \rangle$ 
  by (metis assms(3) b comp_apply ennreal_of_enat_0 ennreal_of_enat_inj
ennreal_of_enat_le_iff_geqb_infsum_superconst_infinite_ennreal_not_gr_zero)
  also have  $\langle \dots = \text{ennreal\_of\_enat } \infty \rangle$ 
  by simp
  finally show ?thesis
  by (rule ennreal_of_enat_inj[THEN iffD1])
qed

```

```

lemma has_sum_superconst_infinite_enat:
  fixes  $f :: \langle 'a \Rightarrow \text{enat} \rangle$ 
  assumes  $\langle \bigwedge x. x \in S \implies f \ x \geq b \rangle$ 
  assumes  $\langle b > 0 \rangle$ 
  assumes  $\langle \text{infinite } S \rangle$ 
  shows  $\text{has\_sum } f \ S \ \infty$ 
  by (metis assms i0_lb has_sum_infsum infsum_superconst_infinite_enat nonneg_summable_on_complete)

```

This lemma helps to relate a real-valued infsum to a supremum over extended nonnegative reals.

```

lemma infsum_nonneg_is_SUPREMUM_ennreal:
  fixes  $f :: 'a \Rightarrow \text{real}$ 
  assumes summable:  $f \text{ summable\_on } A$ 
  and fnn:  $\bigwedge x. x \in A \implies f \ x \geq 0$ 
  shows  $\text{ennreal } (\text{infsum } f \ A) = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{ennreal } (\text{sum } f \ F)))$ 
proof –
  have  $\langle \text{ennreal } (\text{infsum } f \ A) = \text{infsum } (\text{ennreal } o \ f) \ A \rangle$ 
  apply (rule infsum_comm_additive_general[symmetric])
  apply (subst sum_ennreal[symmetric])
  using assms by auto
  also have  $\langle \dots = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{ennreal } (\text{sum } f \ F))) \rangle$ 
  apply (subst nonneg_infsum_complete, simp)
  apply (rule SUP_cong, blast)
  apply (subst sum_ennreal[symmetric])

```

```

    using fnn by auto
    finally show ?thesis .
qed

```

This lemma helps to related a real-valued infsum to a supremum over extended reals.

```

lemma infsum_nonneg_is_SUPREMUM_ereal:
  fixes f :: 'a  $\Rightarrow$  real
  assumes summable: f summable_on A
  and fnn:  $\bigwedge x. x \in A \implies f x \geq 0$ 
  shows ereal (infsum f A) = (SUP F $\in$ {F. finite F  $\wedge$  F  $\subseteq$  A}. (ereal (sum f F)))
proof -
  have  $\langle \text{ereal (infsum f A)} = \text{infsum (ereal o f) A} \rangle$ 
  apply (rule infsum_comm_additive_general[symmetric])
  using assms by auto
  also have  $\langle \dots = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{ereal (sum f F)})) \rangle$ 
  by (subst nonneg_infsum_complete) (simp_all add: assms)
  finally show ?thesis .
qed

```

6.37.5 Real numbers

Most lemmas in the general property section already apply to real numbers. A few ones that are specific to reals are given here.

```

lemma infsum_nonneg_is_SUPREMUM_real:
  fixes f :: 'a  $\Rightarrow$  real
  assumes summable: f summable_on A
  and fnn:  $\bigwedge x. x \in A \implies f x \geq 0$ 
  shows infsum f A = (SUP F $\in$ {F. finite F  $\wedge$  F  $\subseteq$  A}. (sum f F))
proof -
  have ereal (infsum f A) = (SUP F $\in$ {F. finite F  $\wedge$  F  $\subseteq$  A}. (ereal (sum f F)))
  using assms by (rule infsum_nonneg_is_SUPREMUM_ereal)
  also have  $\dots = \text{ereal (SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{sum f F}))$ 
  proof (subst ereal_SUP)
    show  $|\text{SUP } a \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{ereal (sum f a)}| \neq \infty$ 
    using calculation by fastforce
    show  $(\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{ereal (sum f F)}) = (\text{SUP } a \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{ereal (sum f a)})$ 
    by simp
  qed
  finally show ?thesis by simp
qed

```

```

lemma has_sum_nonneg_SUPREMUM_real:
  fixes f :: 'a  $\Rightarrow$  real
  assumes f summable_on A and  $\bigwedge x. x \in A \implies f x \geq 0$ 
  shows has_sum f A (SUP F $\in$ {F. finite F  $\wedge$  F  $\subseteq$  A}. (sum f F))

```

by (metis (mono_tags, lifting) assms has_sum_infsum infsum_nonneg_is_SUPREMUM_real)

lemma summable_countable_real:

fixes $f :: \langle 'a \Rightarrow \text{real} \rangle$

assumes $\langle f \text{ summable\_on } A \rangle$

shows $\langle \text{countable } \{x \in A. f\ x \neq 0\} \rangle$

using abs_summable_countable assms summable_on_iff_abs_summable_on_real

by blast

6.37.6 Complex numbers

lemma has_sum_cnj_iff[simp]:

fixes $f :: \langle 'a \Rightarrow \text{complex} \rangle$

shows $\langle \text{has\_sum } (\lambda x. \text{cnj } (f\ x))\ M\ (\text{cnj } a) \longleftrightarrow \text{has\_sum } f\ M\ a \rangle$

by (simp add: has_sum_def lim_cnj del: cnj_sum add: cnj_sum[symmetric, abs_def, of f])

lemma summable_on_cnj_iff[simp]:

$(\lambda i. \text{cnj } (f\ i)) \text{ summable\_on } A \longleftrightarrow f \text{ summable\_on } A$

by (metis complex_cnj_cnj summable_on_def has_sum_cnj_iff)

lemma infsum_cnj[simp]: $\langle \text{infsum } (\lambda x. \text{cnj } (f\ x))\ M = \text{cnj } (\text{infsum } f\ M) \rangle$

by (metis complex_cnj_zero infsumI has_sum_cnj_iff infsum_def summable_on_cnj_iff has_sum_infsum)

lemma infsum_Re:

assumes $f \text{ summable\_on } M$

shows $\text{infsum } (\lambda x. \text{Re } (f\ x))\ M = \text{Re } (\text{infsum } f\ M)$

apply (rule infsum_comm_additive[where f=Re, unfolded o_def])

using assms by (auto intro!: additive.intro)

lemma has_sum_Re:

assumes $\text{has\_sum } f\ M\ a$

shows $\text{has\_sum } (\lambda x. \text{Re } (f\ x))\ M\ (\text{Re } a)$

apply (rule has_sum_comm_additive[where f=Re, unfolded o_def])

using assms by (auto intro!: additive.intro tendsto_Re)

lemma summable_on_Re:

assumes $f \text{ summable\_on } M$

shows $(\lambda x. \text{Re } (f\ x)) \text{ summable\_on } M$

apply (rule summable_on_comm_additive[where f=Re, unfolded o_def])

using assms by (auto intro!: additive.intro)

lemma infsum_Im:

assumes $f \text{ summable\_on } M$

shows $\text{infsum } (\lambda x. \text{Im } (f\ x))\ M = \text{Im } (\text{infsum } f\ M)$

apply (rule infsum_comm_additive[where f=Im, unfolded o_def])

using assms by (auto intro!: additive.intro)

```

lemma has_sum_Im:
  assumes has_sum f M a
  shows has_sum ( $\lambda x. \text{Im } (f x)$ ) M (Im a)
  apply (rule has_sum_comm_additive[where f=Im, unfolded o_def])
  using assms by (auto intro!: additive.intro tendsto_Im)

lemma summable_on_Im:
  assumes f summable_on M
  shows ( $\lambda x. \text{Im } (f x)$ ) summable_on M
  apply (rule summable_on_comm_additive[where f=Im, unfolded o_def])
  using assms by (auto intro!: additive.intro)

lemma nonneg_infsum_le_0D_complex:
  fixes f :: 'a  $\Rightarrow$  complex
  assumes infsum f A  $\leq 0$ 
    and abs_sum: f summable_on A
    and nneg:  $\bigwedge x. x \in A \implies f x \geq 0$ 
    and x  $\in A$ 
  shows f x = 0
proof -
  have  $\langle \text{Im } (f x) = 0 \rangle$ 
    apply (rule nonneg_infsum_le_0D[where A=A])
    using assms
    by (auto simp add: infsum_Im summable_on_Im less_eq_complex_def)
  moreover have  $\langle \text{Re } (f x) = 0 \rangle$ 
    apply (rule nonneg_infsum_le_0D[where A=A])
    using assms by (auto simp add: summable_on_Re infsum_Re less_eq_complex_def)
  ultimately show ?thesis
    by (simp add: complex_eqI)
qed

lemma nonneg_has_sum_le_0D_complex:
  fixes f :: 'a  $\Rightarrow$  complex
  assumes has_sum f A a and  $\langle a \leq 0 \rangle$ 
    and  $\bigwedge x. x \in A \implies f x \geq 0$  and x  $\in A$ 
  shows f x = 0
  by (metis assms infsumI nonneg_infsum_le_0D_complex summable_on_def)

```

The lemma *infsum_mono_neutral* above applies to various linear ordered monoids such as the reals but not to the complex numbers. Thus we have a separate corollary for those:

```

lemma infsum_mono_neutral_complex:
  fixes f :: 'a  $\Rightarrow$  complex
  assumes [simp]: f summable_on A
    and [simp]: g summable_on B
  assumes  $\langle \bigwedge x. x \in A \cap B \implies f x \leq g x \rangle$ 
  assumes  $\langle \bigwedge x. x \in A - B \implies f x \leq 0 \rangle$ 
  assumes  $\langle \bigwedge x. x \in B - A \implies g x \geq 0 \rangle$ 
  shows  $\langle \text{infsum } f A \leq \text{infsum } g B \rangle$ 

```

proof –

```

have ⟨infsum (λx. Re (f x)) A ≤ infsum (λx. Re (g x)) B⟩
  apply (rule infsum_mono_neutral)
  using assms(3-5) by (auto simp add: summable_on_Re less_eq_complex_def)
then have Re: ⟨Re (infsum f A) ≤ Re (infsum g B)⟩
  by (metis assms(1-2) infsum_Re)
have ⟨infsum (λx. Im (f x)) A = infsum (λx. Im (g x)) B⟩
  apply (rule infsum_cong_neutral)
  using assms(3-5) by (auto simp add: summable_on_Re less_eq_complex_def)
then have Im: ⟨Im (infsum f A) = Im (infsum g B)⟩
  by (metis assms(1-2) infsum_Im)
from Re Im show ?thesis
  by (auto simp: less_eq_complex_def)

```

qed

lemma *infsum_mono_complex*:

— For real, *infsum_mono* can be used. But *complex* does not have the right typeclass.

```

fixes f g :: 'a ⇒ complex
assumes f_sum: f summable_on A and g_sum: g summable_on A
assumes leq: ∧x. x ∈ A ⇒ f x ≤ g x
shows infsum f A ≤ infsum g A
by (metis DiffE IntD1 f_sum g_sum infsum_mono_neutral_complex leq)

```

lemma *infsum_nonneg_complex*:

```

fixes f :: 'a ⇒ complex
assumes f_summable: f summable_on M
and f_nn: ∧x. x ∈ M ⇒ 0 ≤ f x
shows infsum f M ≥ 0 (is ?lhs ≥ _)
by (metis assms(1) assms(2) infsum_0_simp summable_on_0_simp infsum_mono_complex)

```

lemma *infsum_cmod*:

```

assumes f_summable: f summable_on M
and f_nn: ∧x. x ∈ M ⇒ 0 ≤ f x
shows infsum (λx. cmod (f x)) M = cmod (infsum f M)

```

proof –

```

have ⟨complex_of_real (infsum (λx. cmod (f x)) M) = infsum (λx. complex_of_real
(cmod (f x))) M⟩

```

```

proof (rule infsum_comm_additive[symmetric, unfolded o_def])

```

```

  have (λz. Re (f z)) summable_on M

```

```

    using assms summable_on_Re by blast

```

```

  also have ?this ⟷ f abs_summable_on M

```

```

    using f_nn by (intro summable_on_cong) (auto simp: less_eq_complex_def
cmod_def)

```

```

    finally show ... .

```

```

qed (auto simp: additive_def)

```

```

also have ⟨... = infsum f M⟩

```

```

  apply (rule infsum_cong)

```

```

    using fnn cmod_eq_Re complex_is_Real_iff less_eq_complex_def by force
  finally show ?thesis
    by (metis abs_of_nonneg infsum_def le_less_trans norm_ge_zero norm_infsum_bound
norm_of_real not_le order_refl)
qed

```

```

lemma summable_on_iff_abs_summable_on_complex:
  fixes f :: 'a  $\Rightarrow$  complex
  shows  $\langle f \text{ summable\_on } A \longleftrightarrow f \text{ abs\_summable\_on } A \rangle$ 
proof (rule iffI)
  assume  $\langle f \text{ summable\_on } A \rangle$ 
  define i r ni nr n where  $\langle i \ x = \text{Im } (f \ x) \rangle$  and  $\langle r \ x = \text{Re } (f \ x) \rangle$ 
    and  $\langle ni \ x = \text{norm } (i \ x) \rangle$  and  $\langle nr \ x = \text{norm } (r \ x) \rangle$  and  $\langle n \ x = \text{norm } (f \ x) \rangle$  for
  x
  from  $\langle f \text{ summable\_on } A \rangle$  have  $\langle i \text{ summable\_on } A \rangle$ 
    by (simp add: i_def[abs_def] summable_on_Im)
  then have [simp]:  $\langle ni \text{ summable\_on } A \rangle$ 
    using ni_def[abs_def] summable_on_iff_abs_summable_on_real by force

  from  $\langle f \text{ summable\_on } A \rangle$  have  $\langle r \text{ summable\_on } A \rangle$ 
    by (simp add: r_def[abs_def] summable_on_Re)
  then have [simp]:  $\langle nr \text{ summable\_on } A \rangle$ 
    by (metis nr_def summable_on_cong summable_on_iff_abs_summable_on_real)

  have n_sum:  $\langle n \ x \leq nr \ x + ni \ x \rangle$  for x
    by (simp add: n_def nr_def ni_def r_def i_def cmod_le)

  have *:  $\langle (\lambda x. nr \ x + ni \ x) \text{ summable\_on } A \rangle$ 
    apply (rule summable_on_add) by auto
  show  $\langle n \text{ summable\_on } A \rangle$ 
    apply (rule nonneg_bdd_above_summable_on)
    apply (simp add: n_def; fail)
    apply (rule bdd_aboveI[where M= $\langle \text{infsum } (\lambda x. nr \ x + ni \ x) \ A \rangle$ ])
    using * n_sum by (auto simp flip: infsum_finite simp: ni_def[abs_def]
nr_def[abs_def] intro!: infsum_mono_neutral)
next
  show  $\langle f \text{ abs\_summable\_on } A \Longrightarrow f \text{ summable\_on } A \rangle$ 
    using abs_summable_summable by blast
qed

```

```

lemma summable_countable_complex:
  fixes f :: 'a  $\Rightarrow$  complex
  assumes  $\langle f \text{ summable\_on } A \rangle$ 
  shows  $\langle \text{countable } \{x \in A. f \ x \neq 0\} \rangle$ 
  using abs_summable_countable assms summable_on_iff_abs_summable_on_complex
  by blast

end

```


6.38 Sums over Infinite Sets

```
theory Infinite_Set_Sum
  imports Set_Integral Infinite_Sum
begin
```

Conflicting notation from *HOL-Analysis.Infinite_Sum*

```
no_notation Infinite_Sum.abs_summable_on (infixr abs'_summable'_on 46)
```

```
lemma sets_eq_countable:
  assumes countable A space M = A  $\bigwedge x. x \in A \implies \{x\} \in \text{sets } M$ 
  shows sets M = Pow A
proof (intro equalityI subsetI)
  fix X assume X  $\in$  Pow A
  hence  $(\bigcup x \in X. \{x\}) \in \text{sets } M$ 
    by (intro sets.countable_UN' countable_subset[OF assms(1)]) (auto intro!:
assms(3))
  also have  $(\bigcup x \in X. \{x\}) = X$  by auto
  finally show X  $\in$  sets M .
next
  fix X assume X  $\in$  sets M
  from sets.sets_into_space[OF this] and assms
  show X  $\in$  Pow A by simp
qed
```

```
lemma measure_eqI_countable':
  assumes spaces: space M = A space N = A
  assumes sets:  $\bigwedge x. x \in A \implies \{x\} \in \text{sets } M \bigwedge x. x \in A \implies \{x\} \in \text{sets } N$ 
  assumes A: countable A
  assumes eq:  $\bigwedge a. a \in A \implies \text{emeasure } M \{a\} = \text{emeasure } N \{a\}$ 
  shows M = N
proof (rule measure_eqI_countable)
  show sets M = Pow A
    by (intro sets_eq_countable' assms)
  show sets N = Pow A
    by (intro sets_eq_countable' assms)
qed fact+
```

```
lemma count_space_PiM_finite:
  fixes B :: 'a  $\Rightarrow$  'b set
  assumes finite A  $\bigwedge i. \text{countable } (B i)$ 
  shows PiM A  $(\lambda i. \text{count\_space } (B i)) = \text{count\_space } (PiE A B)$ 
proof (rule measure_eqI_countable')
  show space (PiM A  $(\lambda i. \text{count\_space } (B i))$ ) = PiE A B
    by (simp add: space_PiM)
  show space (count_space (PiE A B)) = PiE A B by simp
next
  fix f assume f: f  $\in$  PiE A B
```

```

hence  $PiE\ A\ (\lambda x. \{f\ x\}) \in sets\ (Pi_M\ A\ (\lambda i. count\_space\ (B\ i)))$ 
  by (intro sets_PiM_I_finite assms) auto
also from  $f$  have  $PiE\ A\ (\lambda x. \{f\ x\}) = \{f\}$ 
  by (intro PiE_singleton) (auto simp: PiE_def)
finally show  $\{f\} \in sets\ (Pi_M\ A\ (\lambda i. count\_space\ (B\ i)))$  .
next
interpret product_sigma_finite  $(\lambda i. count\_space\ (B\ i))$ 
  by (intro product_sigma_finite.intro sigma_finite_measure_count_space_countable
assms)
thm sigma_finite_measure_count_space
fix  $f$  assume  $f: f \in PiE\ A\ B$ 
hence  $\{f\} = PiE\ A\ (\lambda x. \{f\ x\})$ 
  by (intro PiE_singleton [symmetric]) (auto simp: PiE_def)
also have  $emeasure\ (Pi_M\ A\ (\lambda i. count\_space\ (B\ i)))\ \dots =$ 
   $(\prod_{i \in A. emeasure\ (count\_space\ (B\ i))\ \{f\ i\})$ 
  using  $f$  assms by (subst emeasure_PiM) auto
also have  $\dots = (\prod_{i \in A. 1})$ 
  by (intro prod.cong refl, subst emeasure_count_space_finite) (use  $f$  in auto)
also have  $\dots = emeasure\ (count\_space\ (PiE\ A\ B))\ \{f\}$ 
  using  $f$  by (subst emeasure_count_space_finite) auto
finally show  $emeasure\ (Pi_M\ A\ (\lambda i. count\_space\ (B\ i)))\ \{f\} =$ 
   $emeasure\ (count\_space\ (Pi_E\ A\ B))\ \{f\}$  .
qed (simp_all add: countable_PiE assms)

```

definition *abs_summable_on* ::
 $('a \Rightarrow 'b :: \{banach, second\_countable\_topology\}) \Rightarrow 'a\ set \Rightarrow bool$
 $(infix\ abs\_summable'\_on\ 50)$
where
 $f\ abs\_summable\_on\ A \longleftrightarrow integrable\ (count\_space\ A)\ f$

definition *infsetsum* ::
 $('a \Rightarrow 'b :: \{banach, second\_countable\_topology\}) \Rightarrow 'a\ set \Rightarrow 'b$
where
 $infsetsum\ f\ A = lebesgue\_integral\ (count\_space\ A)\ f$

syntax (ASCII)
 $\_infsetsum :: ptnr \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow 'b::\{banach, second\_countable\_topology\}$
 $((3INFSETSUM\ \_::\_/\ \_) [0, 51, 10] 10)$

syntax
 $\_infsetsum :: ptnr \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow 'b::\{banach, second\_countable\_topology\}$
 $((2\sum_{a \in \_} \_/\ \_) [0, 51, 10] 10)$

translations — Beware of argument permutation!
 $\sum_{a \in A. b} \equiv CONST\ infsetsum\ (\lambda i. b)\ A$

syntax (ASCII)
 $\\_winfsetsum :: ptnr \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow 'b::\{banach, second\_countable\_topology\}$

```

  ((3INFSETSUM _:_. / _) [0, 51, 10] 10)
syntax
  _qinfsetsum :: pttrn  $\Rightarrow$  'b  $\Rightarrow$  'b::{banach, second_countable_topology}
  ((2 $\sum$  a_. / _) [0, 10] 10)
translations — Beware of argument permutation!
   $\sum a i. b \Rightarrow \text{CONST infsetsum } (\lambda i. b) (\text{CONST UNIV})$ 

syntax (ASCII)
  _qinfsetsum :: pttrn  $\Rightarrow$  bool  $\Rightarrow$  'a  $\Rightarrow$  'a::{banach, second_countable_topology}
  ((3INFSETSUM _ | / _ / _) [0, 0, 10] 10)
syntax
  _qinfsetsum :: pttrn  $\Rightarrow$  bool  $\Rightarrow$  'a  $\Rightarrow$  'a::{banach, second_countable_topology}
  ((2 $\sum$  a_ | ( _ ) . / _) [0, 0, 10] 10)
translations
   $\sum a x | P. t \Rightarrow \text{CONST infsetsum } (\lambda x. t) \{x. P\}$ 

print_translation <
  let
    fun sum_tr' [Abs (x, Tx, t), Const (const_syntax <Collect>, _) $ Abs (y, Ty,
P)] =
      if x <> y then raise Match
      else
        let
          val x' = Syntax_Trans.mark_bound_body (x, Tx);
          val t' = subst_bound (x', t);
          val P' = subst_bound (x', P);
        in
          Syntax.const syntax_const <_qinfsetsum> $ Syntax_Trans.mark_bound_abs
(x, Tx) $ P' $ t'
        end
        | sum_tr' _ = raise Match;
  in [(const_syntax <infsetsum>, K sum_tr')] end
  >

lemma restrict_count_space_subset:
   $A \subseteq B \Longrightarrow \text{restrict\_space } (\text{count\_space } B) A = \text{count\_space } A$ 
  by (subst restrict_count_space) (simp_all add: Int_absorb2)

lemma abs_summable_on_restrict:
  fixes f :: 'a  $\Rightarrow$  'b :: {banach, second_countable_topology}
  assumes  $A \subseteq B$ 
  shows  $f \text{ abs\_summable\_on } A \longleftrightarrow (\lambda x. \text{indicator } A \ x *_{\mathbb{R}} f \ x) \text{ abs\_summable\_on } B$ 
proof —
  have  $\text{count\_space } A = \text{restrict\_space } (\text{count\_space } B) A$ 
  by (rule restrict_count_space_subset [symmetric]) fact+
  also have  $\text{integrable } \dots f \longleftrightarrow \text{set\_integrable } (\text{count\_space } B) A f$ 
  by (simp add: integrable_restrict_space set_integrable_def)

```

2832

finally show *?thesis*
unfolding *abs_summable_on_def set_integrable_def* .
qed

lemma *abs_summable_on_altdef*: $f \text{ abs\_summable\_on } A \longleftrightarrow \text{set\_integrable } (\text{count\_space } UNIV) A f$
unfolding *abs_summable_on_def set_integrable_def*
by (*metis* (*no_types*) *inf_top.right_neutral integrable_restrict_space restrict_count_space sets_UNIV*)

lemma *abs_summable_on_altdef'*:
 $A \subseteq B \implies f \text{ abs\_summable\_on } A \longleftrightarrow \text{set\_integrable } (\text{count\_space } B) A f$
unfolding *abs_summable_on_def set_integrable_def*
by (*metis* (*no_types*) *Pow_iff abs_summable_on_def inf.orderE integrable_restrict_space restrict_count_space_subset sets_count_space space_count_space*)

lemma *abs_summable_on_norm_iff* [*simp*]:
 $(\lambda x. \text{norm } (f x)) \text{ abs\_summable\_on } A \longleftrightarrow f \text{ abs\_summable\_on } A$
by (*simp* *add: abs_summable_on_def integrable_norm_iff*)

lemma *abs_summable_on_normI*: $f \text{ abs\_summable\_on } A \implies (\lambda x. \text{norm } (f x)) \text{ abs\_summable\_on } A$
by *simp*

lemma *abs_summable_complex_of_real* [*simp*]: $(\lambda n. \text{complex\_of\_real } (f n)) \text{ abs\_summable\_on } A \longleftrightarrow f \text{ abs\_summable\_on } A$
by (*simp* *add: abs_summable_on_def complex_of_real_integrable_eq*)

lemma *abs_summable_on_comparison_test*:
assumes $g \text{ abs\_summable\_on } A$
assumes $\bigwedge x. x \in A \implies \text{norm } (f x) \leq \text{norm } (g x)$
shows $f \text{ abs\_summable\_on } A$
using *assms Bochner_Integration.integrable_bound*[*of count_space A g f*]
unfolding *abs_summable_on_def* **by** (*auto simp: AE_count_space*)

lemma *abs_summable_on_comparison_test'*:
assumes $g \text{ abs\_summable\_on } A$
assumes $\bigwedge x. x \in A \implies \text{norm } (f x) \leq g x$
shows $f \text{ abs\_summable\_on } A$
proof (*rule abs_summable_on_comparison_test*[*OF assms(1), of f*])
fix x **assume** $x \in A$
with *assms(2)* **have** $\text{norm } (f x) \leq g x$.
also have $\dots \leq \text{norm } (g x)$ **by** *simp*
finally show $\text{norm } (f x) \leq \text{norm } (g x)$.
qed

lemma *abs_summable_on_cong* [*cong*]:
 $(\bigwedge x. x \in A \implies f x = g x) \implies A = B \implies (f \text{ abs\_summable\_on } A) \longleftrightarrow (g \text{ abs\_summable\_on } B)$

unfolding *abs_summable_on_def* **by** (*intro integrable_cong*) *auto*

lemma *abs_summable_on_cong_neutral*:

assumes $\bigwedge x. x \in A - B \implies f\ x = 0$

assumes $\bigwedge x. x \in B - A \implies g\ x = 0$

assumes $\bigwedge x. x \in A \cap B \implies f\ x = g\ x$

shows $f\ \text{abs\_summable\_on}\ A \longleftrightarrow g\ \text{abs\_summable\_on}\ B$

unfolding *abs_summable_on_altdef set_integrable_def* **using** *assms*

by (*intro Bochner_Integration.integrable_cong refl*)

(*auto simp: indicator_def split: if_splits*)

lemma *abs_summable_on_restrict'*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second\_countable\_topology}\}$

assumes $A \subseteq B$

shows $f\ \text{abs\_summable\_on}\ A \longleftrightarrow (\lambda x. \text{if } x \in A \text{ then } f\ x \text{ else } 0)\ \text{abs\_summable\_on}\ B$

by (*subst abs_summable_on_restrict[OF assms]*) (*intro abs_summable_on_cong, auto*)

lemma *abs_summable_on_nat_iff*:

$f\ \text{abs\_summable\_on}\ (A :: \text{nat set}) \longleftrightarrow \text{summable } (\lambda n. \text{if } n \in A \text{ then } \text{norm } (f\ n) \text{ else } 0)$

proof –

have $f\ \text{abs\_summable\_on}\ A \longleftrightarrow \text{summable } (\lambda x. \text{norm } (\text{if } x \in A \text{ then } f\ x \text{ else } 0))$

by (*subst abs_summable_on_restrict'[of UNIV]*)

(*simp_all add: abs_summable_on_def integrable_count_space_nat_iff*)

also have $(\lambda x. \text{norm } (\text{if } x \in A \text{ then } f\ x \text{ else } 0)) = (\lambda x. \text{if } x \in A \text{ then } \text{norm } (f\ x) \text{ else } 0)$

by *auto*

finally show *?thesis* .

qed

lemma *abs_summable_on_nat_iff'*:

$f\ \text{abs\_summable\_on}\ (\text{UNIV} :: \text{nat set}) \longleftrightarrow \text{summable } (\lambda n. \text{norm } (f\ n))$

by (*subst abs_summable_on_nat_iff*) *auto*

lemma *nat_abs_summable_on_comparison_test*:

fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, second\_countable\_topology}\}$

assumes $g\ \text{abs\_summable\_on}\ I$

assumes $\bigwedge n. \llbracket n \geq N; n \in I \rrbracket \implies \text{norm } (f\ n) \leq g\ n$

shows $f\ \text{abs\_summable\_on}\ I$

using *assms* **by** (*fastforce simp add: abs_summable_on_nat_iff intro: summable_comparison_test'*)

lemma *abs_summable_comparison_test_ev*:

assumes $g\ \text{abs\_summable\_on}\ I$

assumes *eventually* $(\lambda x. x \in I \longrightarrow \text{norm } (f\ x) \leq g\ x)$ *sequentially*

shows $f\ \text{abs\_summable\_on}\ I$

by (*metis (no_types, lifting) nat_abs_summable_on_comparison_test eventu-*

2834

ally_at_top_linorder assms)

lemma *abs_summable_on_Cauchy*:

f abs_summable_on (UNIV :: nat set) $\longleftrightarrow (\forall e > 0. \exists N. \forall m \geq N. \forall n. (\sum_{m..<n} \text{norm } (f x)) < e)$

by (*simp add: abs_summable_on_nat_iff' summable_Cauchy sum_nonneg*)

lemma *abs_summable_on_finite* [*simp*]: *finite A $\implies f \text{ abs\_summable\_on } A$*

unfolding *abs_summable_on_def* **by** (*rule integrable_count_space*)

lemma *abs_summable_on_empty* [*simp, intro*]: *f abs_summable_on {}*

by *simp*

lemma *abs_summable_on_subset*:

assumes *f abs_summable_on B* **and** *A $\subseteq B$*

shows *f abs_summable_on A*

unfolding *abs_summable_on_altdef*

by (*rule set_integrable_subset*) (*insert assms, auto simp: abs_summable_on_altdef*)

lemma *abs_summable_on_union* [*intro*]:

assumes *f abs_summable_on A* **and** *f abs_summable_on B*

shows *f abs_summable_on (A $\cup$ B)*

using *assms* **unfolding** *abs_summable_on_altdef* **by** (*intro set_integrable_Un*)

auto

lemma *abs_summable_on_insert_iff* [*simp*]:

f abs_summable_on insert x A $\longleftrightarrow f \text{ abs\_summable\_on } A$

proof *safe*

assume *f abs_summable_on insert x A*

thus *f abs_summable_on A*

by (*rule abs_summable_on_subset*) *auto*

next

assume *f abs_summable_on A*

from *abs_summable_on_union* [*OF this, of {x}*]

show *f abs_summable_on insert x A* **by** *simp*

qed

lemma *abs_summable_sum*:

assumes $\bigwedge x. x \in A \implies f x \text{ abs\_summable\_on } B$

shows $(\lambda y. \sum_{x \in A} f x y) \text{ abs\_summable\_on } B$

using *assms* **unfolding** *abs_summable_on_def* **by** (*intro Bochner_Integration.integrable_sum*)

lemma *abs_summable_Re*: *f abs_summable_on A $\implies (\lambda x. \text{Re } (f x)) \text{ abs\_summable\_on } A$*

by (*simp add: abs_summable_on_def*)

lemma *abs_summable_Im*: *f abs_summable_on A $\implies (\lambda x. \text{Im } (f x)) \text{ abs\_summable\_on } A$*

by (*simp add: abs_summable_on_def*)

```

lemma abs_summable_on_finite_diff:
  assumes f abs_summable_on A  $A \subseteq B$  finite  $(B - A)$ 
  shows f abs_summable_on B
proof -
  have f abs_summable_on  $(A \cup (B - A))$ 
    by (intro abs_summable_on_union assms abs_summable_on_finite)
  also from assms have  $A \cup (B - A) = B$  by blast
  finally show ?thesis .
qed

lemma abs_summable_on_reindex_bij_betw:
  assumes bij_betw g A B
  shows  $(\lambda x. f (g x))$  abs_summable_on A  $\longleftrightarrow$  f abs_summable_on B
proof -
  have *: count_space B = distr (count_space A) (count_space B) g
    by (rule distr_bij_count_space [symmetric]) fact
  show ?thesis unfolding abs_summable_on_def
    by (subst *, subst integrable_distr_eq[of _ _ count_space B])
      (insert assms, auto simp: bij_betw_def)
qed

lemma abs_summable_on_reindex:
  assumes  $(\lambda x. f (g x))$  abs_summable_on A
  shows f abs_summable_on  $(g \text{ ` } A)$ 
proof -
  define g' where  $g' = \text{inv\_into } A \text{ } g$ 
  from assms have  $(\lambda x. f (g x))$  abs_summable_on  $(g' \text{ ` } g \text{ ` } A)$ 
    by (rule abs_summable_on_subset) (auto simp: g'_def inv_into_into)
  also have ?this  $\longleftrightarrow$   $(\lambda x. f (g (g' x)))$  abs_summable_on  $(g \text{ ` } A)$  unfolding
    g'_def
    by (intro abs_summable_on_reindex_bij_betw [symmetric] inj_on_imp_bij_betw
      inj_on_inv_into) auto
  also have  $\dots \longleftrightarrow$  f abs_summable_on  $(g \text{ ` } A)$ 
    by (intro abs_summable_on_cong refl) (auto simp: g'_def f_inv_into_f)
  finally show ?thesis .
qed

lemma abs_summable_on_reindex_iff:
  inj_on g A  $\implies$   $(\lambda x. f (g x))$  abs_summable_on A  $\longleftrightarrow$  f abs_summable_on  $(g \text{ ` } A)$ 
  by (intro abs_summable_on_reindex_bij_betw inj_on_imp_bij_betw)

lemma abs_summable_on_Sigma_project2:
  fixes A :: 'a set and B :: 'a  $\Rightarrow$  'b set
  assumes f abs_summable_on  $(\text{Sigma } A \text{ } B)$   $x \in A$ 
  shows  $(\lambda y. f (x, y))$  abs_summable_on  $(B x)$ 
proof -
  from assms(2) have f abs_summable_on  $(\text{Sigma } \{x\} \text{ } B)$ 

```

```

    by (intro abs_summable_on_subset [OF assms(1)]) auto
  also have ?this  $\longleftrightarrow$   $(\lambda z. f(x, \text{snd } z)) \text{ abs\_summable\_on } (\text{Sigma } \{x\} B)$ 
    by (rule abs_summable_on_cong) auto
  finally have  $(\lambda y. f(x, y)) \text{ abs\_summable\_on } (\text{snd } ' \text{Sigma } \{x\} B)$ 
    by (rule abs_summable_on_reindex)
  also have  $\text{snd } ' \text{Sigma } \{x\} B = B \ x$ 
    using assms by (auto simp: image_iff)
  finally show ?thesis .
qed

```

```

lemma abs_summable_on_Times_swap:
   $f \text{ abs\_summable\_on } A \times B \longleftrightarrow (\lambda(x,y). f(y,x)) \text{ abs\_summable\_on } B \times A$ 
proof -
  have bij: bij_betw  $(\lambda(x,y). (y,x)) (B \times A) (A \times B)$ 
    by (auto simp: bij_betw_def inj_on_def)
  show ?thesis
    by (subst abs_summable_on_reindex_bij_betw[OF bij, of f, symmetric])
      (simp_all add: case_prod_unfold)
qed

```

```

lemma abs_summable_on_0 [simp, intro]:  $(\lambda_. 0) \text{ abs\_summable\_on } A$ 
  by (simp add: abs_summable_on_def)

```

```

lemma abs_summable_on_uminus [intro]:
   $f \text{ abs\_summable\_on } A \implies (\lambda x. -f x) \text{ abs\_summable\_on } A$ 
unfolding abs_summable_on_def by (rule Bochner_Integration.integrable_minus)

```

```

lemma abs_summable_on_add [intro]:
  assumes  $f \text{ abs\_summable\_on } A$  and  $g \text{ abs\_summable\_on } A$ 
  shows  $(\lambda x. f x + g x) \text{ abs\_summable\_on } A$ 
  using assms unfolding abs_summable_on_def by (rule Bochner_Integration.integrable_add)

```

```

lemma abs_summable_on_diff [intro]:
  assumes  $f \text{ abs\_summable\_on } A$  and  $g \text{ abs\_summable\_on } A$ 
  shows  $(\lambda x. f x - g x) \text{ abs\_summable\_on } A$ 
  using assms unfolding abs_summable_on_def by (rule Bochner_Integration.integrable_diff)

```

```

lemma abs_summable_on_scaleR_left [intro]:
  assumes  $c \neq 0 \implies f \text{ abs\_summable\_on } A$ 
  shows  $(\lambda x. f x *_R c) \text{ abs\_summable\_on } A$ 
  using assms unfolding abs_summable_on_def by (intro Bochner_Integration.integrable_scaleR_left)

```

```

lemma abs_summable_on_scaleR_right [intro]:
  assumes  $c \neq 0 \implies f \text{ abs\_summable\_on } A$ 
  shows  $(\lambda x. c *_R f x) \text{ abs\_summable\_on } A$ 
  using assms unfolding abs_summable_on_def by (intro Bochner_Integration.integrable_scaleR_right)

```

```

lemma abs_summable_on_cmult_right [intro]:
  fixes  $f :: 'a \Rightarrow 'b :: \{\text{banach, real\_normed\_algebra, second\_countable\_topology}\}$ 

```


assumes $c \neq 0 \implies f \text{ abs\_summable\_on } A$
shows $(\lambda x. c * f x) \text{ abs\_summable\_on } A$
using *assms* **unfolding** *abs_summable_on_def* **by** (intro Bochner_Integration.integrable_mult_right)

lemma *abs_summable_on_cmult_left* [intro]:
fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, real\_normed\_algebra, second\_countable\_topology}\}$
assumes $c \neq 0 \implies f \text{ abs\_summable\_on } A$
shows $(\lambda x. f x * c) \text{ abs\_summable\_on } A$
using *assms* **unfolding** *abs_summable_on_def* **by** (intro Bochner_Integration.integrable_mult_left)

lemma *abs_summable_on_prod_PiE*:
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \{\text{real\_normed\_field, banach, second\_countable\_topology}\}$
assumes *finite*: $\text{finite } A$ **and** *countable*: $\bigwedge x. x \in A \implies \text{countable } (B x)$
assumes *summable*: $\bigwedge x. x \in A \implies f x \text{ abs\_summable\_on } B x$
shows $(\lambda g. \prod_{x \in A}. f x (g x)) \text{ abs\_summable\_on } \text{PiE } A B$
proof –
define B' **where** $B' = (\lambda x. \text{if } x \in A \text{ then } B x \text{ else } \{\})$
from *assms* **have** [simp]: $\text{countable } (B' x)$ **for** x
by (auto simp: *B'_def*)
then interpret *product_sigma_finite_count_space* $\circ B'$
unfolding *o_def* **by** (intro *product_sigma_finite.intro* *sigma_finite_measure_count_space_countable*)
from *assms* **have** *integrable* (*PiM* A (*count_space* $\circ B'$)) $(\lambda g. \prod_{x \in A}. f x (g x))$
by (intro *product_integrable_prod*) (auto simp: *abs_summable_on_def* *B'_def*)
also have *PiM* A (*count_space* $\circ B'$) = *count_space* (*PiE* $A B'$)
unfolding *o_def* **using** *finite* **by** (intro *count_space_PiM_finite*) *simp_all*
also have *PiE* $A B' = \text{PiE } A B$ **by** (intro *PiE_cong*) (*simp_all* add: *B'_def*)
finally show ?thesis **by** (*simp* add: *abs_summable_on_def*)
qed

lemma *not_summable_infsetsum_eq*:
 $\neg f \text{ abs\_summable\_on } A \implies \text{infsetsum } f A = 0$
by (*simp* add: *abs_summable_on_def* *infsetsum_def* *not_integrable_integral_eq*)

lemma *infsetsum_altdef*:
 $\text{infsetsum } f A = \text{set\_lebesgue\_integral } (\text{count\_space } \text{UNIV}) A f$
unfolding *set_lebesgue_integral_def*
by (*subst* *integral_restrict_space* [*symmetric*])
 (auto simp: *restrict_count_space_subset_infsetsum_def*)

lemma *infsetsum_altdef'*:
 $A \subseteq B \implies \text{infsetsum } f A = \text{set\_lebesgue\_integral } (\text{count\_space } B) A f$
unfolding *set_lebesgue_integral_def*
by (*subst* *integral_restrict_space* [*symmetric*])
 (auto simp: *restrict_count_space_subset_infsetsum_def*)

lemma *nn_integral_conv_infsetsum*:
assumes $f \text{ abs\_summable\_on } A \wedge x. x \in A \implies f x \geq 0$

shows $nn\_integral\ (count\_space\ A)\ f = ennreal\ (infsetsum\ f\ A)$
using *assms* **unfolding** *infsetsum_def abs_summable_on_def*
by (*subst nn_integral_eq_integral*) *auto*

lemma *infsetsum_conv_nn_integral*:
assumes $nn\_integral\ (count\_space\ A)\ f \neq \infty \wedge x. x \in A \implies f\ x \geq 0$
shows $infsetsum\ f\ A = enn2real\ (nn\_integral\ (count\_space\ A)\ f)$
unfolding *infsetsum_def* **using** *assms*
by (*subst integral_eq_nn_integral*) *auto*

lemma *infsetsum_cong* [*cong*]:
 $(\wedge x. x \in A \implies f\ x = g\ x) \implies A = B \implies infsetsum\ f\ A = infsetsum\ g\ B$
unfolding *infsetsum_def* **by** (*intro Bochner_Integration.integral_cong*) *auto*

lemma *infsetsum_0* [*simp*]: $infsetsum\ (\lambda_.\ 0)\ A = 0$
by (*simp add: infsetsum_def*)

lemma *infsetsum_all_0*: $(\wedge x. x \in A \implies f\ x = 0) \implies infsetsum\ f\ A = 0$
by *simp*

lemma *infsetsum_nonneg*: $(\wedge x. x \in A \implies f\ x \geq (0::real)) \implies infsetsum\ f\ A \geq 0$
unfolding *infsetsum_def* **by** (*rule Bochner_Integration.integral_nonneg*) *auto*

lemma *sum_infsetsum*:
assumes $\wedge x. x \in A \implies f\ x\ abs\_summable\_on\ B$
shows $(\sum_{x \in A}. \sum_{a \in B}. f\ x\ a) = (\sum_{a \in B}. \sum_{x \in A}. f\ x\ a)$
using *assms* **by** (*simp add: infsetsum_def abs_summable_on_def Bochner_Integration.integral_sum*)

lemma *Re_infsetsum*: $f\ abs\_summable\_on\ A \implies Re\ (infsetsum\ f\ A) = (\sum_{a \in A}. Re\ (f\ a))$
by (*simp add: infsetsum_def abs_summable_on_def*)

lemma *Im_infsetsum*: $f\ abs\_summable\_on\ A \implies Im\ (infsetsum\ f\ A) = (\sum_{a \in A}. Im\ (f\ a))$
by (*simp add: infsetsum_def abs_summable_on_def*)

lemma *infsetsum_of_real*:
shows $infsetsum\ (\lambda x. of\_real\ (f\ x))$
 $:: 'a :: \{real\_normed\_algebra\_1, banach, second\_countable\_topology, real\_inner\}$
 $A =$
 $of\_real\ (infsetsum\ f\ A)$
unfolding *infsetsum_def*
by (*rule integral_bounded_linear'[OF bounded_linear_of_real bounded_linear_inner_left[of 1]]*) *auto*

lemma *infsetsum_finite* [*simp*]: $finite\ A \implies infsetsum\ f\ A = (\sum_{x \in A}. f\ x)$
by (*simp add: infsetsum_def lebesgue_integral_count_space_finite*)

```

lemma infsetsum_nat:
  assumes f abs_summable_on A
  shows infsetsum f A = ( $\sum n.$  if  $n \in A$  then  $f n$  else 0)
proof -
  from assms have infsetsum f A = ( $\sum n.$  indicator A n *R f n)
  unfolding infsetsum_altdef abs_summable_on_altdef set_lebesgue_integral_def
  set_integrable_def
  by (subst integral_count_space_nat) auto
  also have ( $\lambda n.$  indicator A n *R f n) = ( $\lambda n.$  if  $n \in A$  then  $f n$  else 0)
  by auto
  finally show ?thesis .
qed

```

```

lemma infsetsum_nat':
  assumes f abs_summable_on UNIV
  shows infsetsum f UNIV = ( $\sum n.$  f n)
  using assms by (subst infsetsum_nat) auto

```

```

lemma sums_infsetsum_nat:
  assumes f abs_summable_on A
  shows ( $\lambda n.$  if  $n \in A$  then  $f n$  else 0) sums infsetsum f A
proof -
  from assms have summable ( $\lambda n.$  if  $n \in A$  then norm (f n) else 0)
  by (simp add: abs_summable_on_nat_iff)
  also have ( $\lambda n.$  if  $n \in A$  then norm (f n) else 0) = ( $\lambda n.$  norm (if  $n \in A$  then f
n else 0))
  by auto
  finally have summable ( $\lambda n.$  if  $n \in A$  then f n else 0)
  by (rule summable_norm_cancel)
  with assms show ?thesis
  by (auto simp: sums_iff infsetsum_nat)
qed

```

```

lemma sums_infsetsum_nat':
  assumes f abs_summable_on UNIV
  shows f sums infsetsum f UNIV
  using sums_infsetsum_nat [OF assms] by simp

```

```

lemma infsetsum_Un_disjoint:
  assumes f abs_summable_on A f abs_summable_on B A  $\cap$  B = {}
  shows infsetsum f (A  $\cup$  B) = infsetsum f A + infsetsum f B
  using assms unfolding infsetsum_altdef abs_summable_on_altdef
  by (subst set_integral_Un) auto

```

```

lemma infsetsum_Diff:
  assumes f abs_summable_on B A  $\subseteq$  B
  shows infsetsum f (B - A) = infsetsum f B - infsetsum f A
proof -
  have infsetsum f ((B - A)  $\cup$  A) = infsetsum f (B - A) + infsetsum f A

```

```

    using assms(2) by (intro infsetsum_Un_disjoint abs_summable_on_subset[OF
assms(1)]) auto
    also from assms(2) have  $(B - A) \cup A = B$ 
    by auto
    ultimately show ?thesis
    by (simp add: algebra_simps)
qed

```

```

lemma infsetsum_Un_Int:
  assumes f_abs_summable_on  $(A \cup B)$ 
  shows  $\text{infsetsum } f (A \cup B) = \text{infsetsum } f A + \text{infsetsum } f B - \text{infsetsum } f (A \cap B)$ 
proof -
  have  $A \cup B = A \cup (B - A \cap B)$ 
  by auto
  also have  $\text{infsetsum } f \dots = \text{infsetsum } f A + \text{infsetsum } f (B - A \cap B)$ 
  by (intro infsetsum_Un_disjoint abs_summable_on_subset[OF assms]) auto
  also have  $\text{infsetsum } f (B - A \cap B) = \text{infsetsum } f B - \text{infsetsum } f (A \cap B)$ 
  by (intro infsetsum_Diff abs_summable_on_subset[OF assms]) auto
  finally show ?thesis
  by (simp add: algebra_simps)
qed

```

```

lemma infsetsum_reindex_bij_betw:
  assumes bij_betw g A B
  shows  $\text{infsetsum } (\lambda x. f (g x)) A = \text{infsetsum } f B$ 
proof -
  have *:  $\text{count\_space } B = \text{distr } (\text{count\_space } A) (\text{count\_space } B) g$ 
  by (rule distr_bij_count_space [symmetric]) fact
  show ?thesis unfolding infsetsum_def
  by (subst *, subst integral_distr[of _ _ count_space B])
    (insert assms, auto simp: bij_betw_def)
qed

```

```

theorem infsetsum_reindex:
  assumes inj_on g A
  shows  $\text{infsetsum } f (g ` A) = \text{infsetsum } (\lambda x. f (g x)) A$ 
by (intro infsetsum_reindex_bij_betw [symmetric] inj_on_imp_bij_betw assms)

```

```

lemma infsetsum_cong_neutral:
  assumes  $\bigwedge x. x \in A - B \implies f x = 0$ 
  assumes  $\bigwedge x. x \in B - A \implies g x = 0$ 
  assumes  $\bigwedge x. x \in A \cap B \implies f x = g x$ 
  shows  $\text{infsetsum } f A = \text{infsetsum } g B$ 
unfolding infsetsum_altdef set_lebesgue_integral_def using assms
by (intro Bochner_Integration.integral_cong refl)
  (auto simp: indicator_def split: if_splits)

```

```

lemma infsetsum_mono_neutral:

```

```

fixes  $f\ g :: 'a \Rightarrow \text{real}$ 
assumes  $f\ \text{abs\_summable\_on}\ A$  and  $g\ \text{abs\_summable\_on}\ B$ 
assumes  $\bigwedge x. x \in A \implies f\ x \leq g\ x$ 
assumes  $\bigwedge x. x \in A - B \implies f\ x \leq 0$ 
assumes  $\bigwedge x. x \in B - A \implies g\ x \geq 0$ 
shows  $\text{infsetsum}\ f\ A \leq \text{infsetsum}\ g\ B$ 
using assms unfolding infsetsum_altdef set_lebesgue_integral_def abs\_summable\_on\_altdef
set_integrable_def
by (intro Bochner_Integration.integral_mono) (auto simp: indicator_def)

```

```

lemma infsetsum_mono_neutral_left:
fixes  $f\ g :: 'a \Rightarrow \text{real}$ 
assumes  $f\ \text{abs\_summable\_on}\ A$  and  $g\ \text{abs\_summable\_on}\ B$ 
assumes  $\bigwedge x. x \in A \implies f\ x \leq g\ x$ 
assumes  $A \subseteq B$ 
assumes  $\bigwedge x. x \in B - A \implies g\ x \geq 0$ 
shows  $\text{infsetsum}\ f\ A \leq \text{infsetsum}\ g\ B$ 
using  $\langle A \subseteq B \rangle$  by (intro infsetsum_mono_neutral assms) auto

```

```

lemma infsetsum_mono_neutral_right:
fixes  $f\ g :: 'a \Rightarrow \text{real}$ 
assumes  $f\ \text{abs\_summable\_on}\ A$  and  $g\ \text{abs\_summable\_on}\ B$ 
assumes  $\bigwedge x. x \in A \implies f\ x \leq g\ x$ 
assumes  $B \subseteq A$ 
assumes  $\bigwedge x. x \in A - B \implies f\ x \leq 0$ 
shows  $\text{infsetsum}\ f\ A \leq \text{infsetsum}\ g\ B$ 
using  $\langle B \subseteq A \rangle$  by (intro infsetsum_mono_neutral assms) auto

```

```

lemma infsetsum_mono:
fixes  $f\ g :: 'a \Rightarrow \text{real}$ 
assumes  $f\ \text{abs\_summable\_on}\ A$  and  $g\ \text{abs\_summable\_on}\ A$ 
assumes  $\bigwedge x. x \in A \implies f\ x \leq g\ x$ 
shows  $\text{infsetsum}\ f\ A \leq \text{infsetsum}\ g\ A$ 
by (intro infsetsum_mono_neutral assms) auto

```

```

lemma norm_infsetsum_bound:
 $\text{norm} (\text{infsetsum}\ f\ A) \leq \text{infsetsum} (\lambda x. \text{norm} (f\ x))\ A$ 
unfolding abs\_summable\_on_def infsetsum_def
by (rule Bochner_Integration.integral_norm_bound)

```

```

theorem infsetsum_Sigma:
fixes  $A :: 'a\ \text{set}$  and  $B :: 'a \Rightarrow 'b\ \text{set}$ 
assumes [simp]: countable  $A$  and  $\bigwedge i. \text{countable}\ (B\ i)$ 
assumes summable:  $f\ \text{abs\_summable\_on}\ (\text{Sigma}\ A\ B)$ 
shows  $\text{infsetsum}\ f\ (\text{Sigma}\ A\ B) = \text{infsetsum}\ (\lambda x. \text{infsetsum}\ (\lambda y. f\ (x, y))\ (B\ x))\ A$ 
proof –
define  $B'$  where  $B' = (\bigcup_{i \in A} B\ i)$ 
have [simp]: countable  $B'$ 

```

```

    unfolding B'_def by (intro countable_UN assms)
  interpret pair_sigma_finite count_space A count_space B'
  by (intro pair_sigma_finite.intro sigma_finite_measure_count_space_countable)
fact+

```

```

  have integrable (count_space (A × B')) (λz. indicator (Sigma A B) z *R f z)
    using summable
  by (metis (mono_tags, lifting) abs_summable_on_altdef abs_summable_on_def
    integrable_cong integrable_mult indicator set_integrable_def sets_UNIV)
  also have ?this ⟷ integrable (count_space A ⊗M count_space B') (λ(x, y).
    indicator (B x) y *R f (x, y))
  by (intro Bochner_Integration.integrable_cong)
    (auto simp: pair_measure_countable indicator_def split: if_splits)
  finally have integrable: ... .

```

```

  have infsetsum (λx. infsetsum (λy. f (x, y)) (B x)) A =
    (∫ x. infsetsum (λy. f (x, y)) (B x) ∂count_space A)
  unfolding infsetsum_def by simp
  also have ... = (∫ x. ∫ y. indicator (B x) y *R f (x, y) ∂count_space B'
    ∂count_space A)
  proof (rule Bochner_Integration.integral_cong [OF refl])
    show ∧x. x ∈ space (count_space A) ⟹
      (∑a y ∈ B x. f (x, y)) = LINT y | count_space B'. indicat_real (B x) y *R f
      (x, y)
    using infsetsum_altdef'[of _ B']
  unfolding set_lebesgue_integral_def B'_def
  by auto
qed
  also have ... = (∫ (x, y). indicator (B x) y *R f (x, y) ∂(count_space A ⊗M
    count_space B'))
  by (subst integral_fst [OF integrable]) auto
  also have ... = (∫ z. indicator (Sigma A B) z *R f z ∂count_space (A × B'))
  by (intro Bochner_Integration.integral_cong)
    (auto simp: pair_measure_countable indicator_def split: if_splits)
  also have ... = infsetsum f (Sigma A B)
  unfolding set_lebesgue_integral_def [symmetric]
  by (rule infsetsum_altdef' [symmetric]) (auto simp: B'_def)
  finally show ?thesis ..
qed

```

```

lemma infsetsum_Sigma':
  fixes A :: 'a set and B :: 'a ⇒ 'b set
  assumes [simp]: countable A and ∧i. countable (B i)
  assumes summable: (λ(x, y). f x y) abs_summable_on (Sigma A B)
  shows infsetsum (λx. infsetsum (λy. f x y) (B x)) A = infsetsum (λ(x, y). f x
    y) (Sigma A B)
  using assms by (subst infsetsum_Sigma) auto

```

```

lemma infsetsum_Times:

```

```

fixes  $A :: 'a \text{ set}$  and  $B :: 'b \text{ set}$ 
assumes [simp]: countable  $A$  and countable  $B$ 
assumes summable:  $f \text{ abs\_summable\_on } (A \times B)$ 
shows  $\text{infsetsum } f (A \times B) = \text{infsetsum } (\lambda x. \text{infsetsum } (\lambda y. f (x, y)) B) A$ 
using assms by (subst infsetsum_Sigma) auto

```

lemma *infsetsum_Times'*:

```

fixes  $A :: 'a \text{ set}$  and  $B :: 'b \text{ set}$ 
fixes  $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \{\text{banach, second\_countable\_topology}\}$ 
assumes [simp]: countable  $A$  and [simp]: countable  $B$ 
assumes summable:  $(\lambda(x,y). f x y) \text{ abs\_summable\_on } (A \times B)$ 
shows  $\text{infsetsum } (\lambda x. \text{infsetsum } (\lambda y. f x y) B) A = \text{infsetsum } (\lambda(x,y). f x y) (A \times B)$ 
using assms by (subst infsetsum_Times) auto

```

lemma *infsetsum_swap*:

```

fixes  $A :: 'a \text{ set}$  and  $B :: 'b \text{ set}$ 
fixes  $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \{\text{banach, second\_countable\_topology}\}$ 
assumes [simp]: countable  $A$  and [simp]: countable  $B$ 
assumes summable:  $(\lambda(x,y). f x y) \text{ abs\_summable\_on } A \times B$ 
shows  $\text{infsetsum } (\lambda x. \text{infsetsum } (\lambda y. f x y) B) A = \text{infsetsum } (\lambda y. \text{infsetsum } (\lambda x. f x y) A) B$ 
proof –
  from summable have summable':  $(\lambda(x,y). f y x) \text{ abs\_summable\_on } B \times A$ 
  by (subst abs\_summable\_on_Times_swap) auto
  have bij:  $\text{bij\_betw } (\lambda(x, y). (y, x)) (B \times A) (A \times B)$ 
  by (auto simp: bij\_betw\_def inj\_on\_def)
  have  $\text{infsetsum } (\lambda x. \text{infsetsum } (\lambda y. f x y) B) A = \text{infsetsum } (\lambda(x,y). f x y) (A \times B)$ 
  using summable by (subst infsetsum_Times) auto
  also have  $\dots = \text{infsetsum } (\lambda(x,y). f y x) (B \times A)$ 
  by (subst infsetsum_reindex_bij_betw[OF bij, of  $\lambda(x,y). f x y$ , symmetric])
  (simp_all add: case\_prod\_unfold)
  also have  $\dots = \text{infsetsum } (\lambda y. \text{infsetsum } (\lambda x. f x y) A) B$ 
  using summable' by (subst infsetsum_Times) auto
  finally show ?thesis .
qed

```

theorem *abs_summable_on_Sigma_iff*:

```

assumes [simp]: countable  $A$  and  $\bigwedge x. x \in A \implies \text{countable } (B x)$ 
shows  $f \text{ abs\_summable\_on Sigma } A B \longleftrightarrow$ 
   $(\forall x \in A. (\lambda y. f (x, y)) \text{ abs\_summable\_on } B x) \wedge$ 
   $((\lambda x. \text{infsetsum } (\lambda y. \text{norm } (f (x, y))) (B x)) \text{ abs\_summable\_on } A)$ 

```

proof *safe*

```

define  $B'$  where  $B' = (\bigcup x \in A. B x)$ 
have [simp]: countable  $B'$ 
  unfolding  $B'_\text{def}$  using assms by auto
interpret pair_sigma_finite count_space  $A$  count_space  $B'$ 
by (intro pair_sigma_finite.intro sigma_finite_measure_count_space_countable)

```

```

fact+
{
  assume *:  $f$  abs_summable_on Sigma  $A$   $B$ 
  thus  $(\lambda y. f(x, y))$  abs_summable_on  $B$   $x$  if  $x \in A$  for  $x$ 
    using that by (rule abs_summable_on_Sigma_project2)

  have set_integrable (count_space  $(A \times B')$ ) (Sigma  $A$   $B$ )  $(\lambda z. \text{norm}(f z))$ 
    using abs_summable_on_normI[OF *]
    by (subst abs_summable_on_altdef' [symmetric]) (auto simp: B'_def)
  also have count_space  $(A \times B') = \text{count\_space } A \otimes_M \text{count\_space } B'$ 
    by (simp add: pair_measure_countable)
  finally have integrable (count_space  $A$ )
     $(\lambda x. \text{lebesgue\_integral } (\text{count\_space } B') \quad$ 
       $(\lambda y. \text{indicator } (\text{Sigma } A \ B) (x, y) *_R \text{norm } (f(x, y))))$ 
    unfolding set_integrable_def by (rule integrable_fst')
  also have  $?this \longleftrightarrow \text{integrable } (\text{count\_space } A)$ 
     $(\lambda x. \text{lebesgue\_integral } (\text{count\_space } B') \quad$ 
       $(\lambda y. \text{indicator } (B \ x) \ y *_R \text{norm } (f(x, y))))$ 
    by (intro integrable_cong refl) (simp_all add: indicator_def)
  also have  $\dots \longleftrightarrow \text{integrable } (\text{count\_space } A) \ (\lambda x. \text{infsetsum } (\lambda y. \text{norm } (f(x, y))) (B \ x))$ 
     $y))) (B \ x))$ 
    unfolding set_lebesgue_integral_def [symmetric]
    by (intro integrable_cong refl infsetsum_altdef' [symmetric]) (auto simp: B'_def)
  also have  $\dots \longleftrightarrow (\lambda x. \text{infsetsum } (\lambda y. \text{norm } (f(x, y))) (B \ x)) \text{abs\_summable\_on}$ 
     $A$ 
    by (simp add: abs_summable_on_def)
  finally show  $\dots$  .
}
{
  assume *:  $\forall x \in A. (\lambda y. f(x, y))$  abs_summable_on  $B$   $x$ 
  assume  $(\lambda x. \sum_{y \in B} x. \text{norm } (f(x, y)))$  abs_summable_on  $A$ 
  also have  $?this \longleftrightarrow (\lambda x. \int y \in B \ x. \text{norm } (f(x, y)) \ \partial \text{count\_space } B') \text{abs\_summable\_on}$ 
     $A$ 
    by (intro abs_summable_on_cong refl infsetsum_altdef') (auto simp: B'_def)
  also have  $\dots \longleftrightarrow (\lambda x. \int y. \text{indicator } (\text{Sigma } A \ B) (x, y) *_R \text{norm } (f(x, y)) \ \partial \text{count\_space } B')$ 
     $\text{abs\_summable\_on } A$  (is  $\_ \longleftrightarrow ?h \text{abs\_summable\_on } \_$ )
    unfolding set_lebesgue_integral_def
    by (intro abs_summable_on_cong) (auto simp: indicator_def)
  also have  $\dots \longleftrightarrow \text{integrable } (\text{count\_space } A) \ ?h$ 
    by (simp add: abs_summable_on_def)
  finally have **:  $\dots$  .
}

have integrable (count_space  $A \otimes_M \text{count\_space } B'$ )  $(\lambda z. \text{indicator } (\text{Sigma } A \ B) \ z *_R f \ z)$ 
proof (rule Fubini_integrable, goal_cases)
  case 3
  {

```



```

fix  $x$  assume  $x: x \in A$ 
with  $*$  have  $(\lambda y. f(x, y)) \text{ abs\_summable\_on } B$   $x$ 
  by blast
also have  $?this \longleftrightarrow \text{integrable } (\text{count\_space } B')$ 
   $(\lambda y. \text{indicator } (B \ x) \ y \ *_R f(x, y))$ 
  unfolding set_integrable_def [symmetric]
using  $x$  by (intro abs\_summable\_on\_altdef') (auto simp: B'_def)
also have  $(\lambda y. \text{indicator } (B \ x) \ y \ *_R f(x, y)) =$ 
   $(\lambda y. \text{indicator } (\text{Sigma } A \ B) (x, y) \ *_R f(x, y))$ 
  using  $x$  by (auto simp: indicator_def)
finally have  $\text{integrable } (\text{count\_space } B')$ 
   $(\lambda y. \text{indicator } (\text{Sigma } A \ B) (x, y) \ *_R f(x, y)) .$ 
}
thus  $?case$  by (auto simp: AE_count_space)
qed (insert **, auto simp: pair_measure_countable)
moreover have  $\text{count\_space } A \otimes_M \text{count\_space } B' = \text{count\_space } (A \times B')$ 
  by (simp add: pair_measure_countable)
moreover have  $\text{set\_integrable } (\text{count\_space } (A \times B')) (\text{Sigma } A \ B) f \longleftrightarrow$ 
   $f \text{ abs\_summable\_on Sigma } A \ B$ 
  by (rule abs\_summable\_on\_altdef' [symmetric]) (auto simp: B'_def)
ultimately show  $f \text{ abs\_summable\_on Sigma } A \ B$ 
  by (simp add: set_integrable_def)
}
qed

```

```

lemma abs\_summable\_on\_Sigma\_project1:
  assumes  $(\lambda(x,y). f \ x \ y) \text{ abs\_summable\_on Sigma } A \ B$ 
  assumes [simp]: countable  $A$  and  $\bigwedge x. x \in A \implies \text{countable } (B \ x)$ 
  shows  $(\lambda x. \text{infsetsum } (\lambda y. \text{norm } (f \ x \ y)) (B \ x)) \text{ abs\_summable\_on } A$ 
  using assms by (subst (asm) abs\_summable\_on\_Sigma\_iff) auto

```

```

lemma abs\_summable\_on\_Sigma\_project1':
  assumes  $(\lambda(x,y). f \ x \ y) \text{ abs\_summable\_on Sigma } A \ B$ 
  assumes [simp]: countable  $A$  and  $\bigwedge x. x \in A \implies \text{countable } (B \ x)$ 
  shows  $(\lambda x. \text{infsetsum } (\lambda y. f \ x \ y) (B \ x)) \text{ abs\_summable\_on } A$ 
  by (intro abs\_summable\_on\_comparison\_test') [OF abs\_summable\_on\_Sigma\_project1 [OF
assms]]
  norm\_infsetsum\_bound)

```

```

theorem infsetsum\_prod\_PiE:
  fixes  $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \{\text{real\_normed\_field}, \text{banach}, \text{second\_countable\_topology}\}$ 
  assumes finite: finite  $A$  and countable:  $\bigwedge x. x \in A \implies \text{countable } (B \ x)$ 
  assumes summable:  $\bigwedge x. x \in A \implies f \ x \text{ abs\_summable\_on } B \ x$ 
  shows  $\text{infsetsum } (\lambda g. \prod_{x \in A}. f \ x \ (g \ x)) (\text{PiE } A \ B) = (\prod_{x \in A}. \text{infsetsum } (f \ x) (B \ x))$ 
proof –
  define  $B'$  where  $B' = (\lambda x. \text{if } x \in A \text{ then } B \ x \text{ else } \{\})$ 
  from assms have [simp]: countable  $(B' \ x)$  for  $x$ 
  by (auto simp: B'_def)

```

```

then interpret product_sigma_finite count_space ∘ B'
unfolding o_def by (intro product_sigma_finite.intro sigma_finite_measure_count_space_countable)
have infsetsum (λg. ∏ x∈A. f x (g x)) (PiE A B) =
  (∫ g. (∏ x∈A. f x (g x)) ∂count_space (PiE A B))
  by (simp add: infsetsum_def)
also have PiE A B = PiE A B'
  by (intro PiE_cong) (simp_all add: B'_def)
hence count_space (PiE A B) = count_space (PiE A B')
  by simp
also have ... = PiM A (count_space ∘ B')
  unfolding o_def using finite by (intro count_space_PiM_finite [symmetric])
simp_all
also have (∫ g. (∏ x∈A. f x (g x)) ∂...) = (∏ x∈A. infsetsum (f x) (B' x))
  by (subst product_integral_prod)
  (insert summable_finite, simp_all add: infsetsum_def B'_def abs_summable_on_def)
also have ... = (∏ x∈A. infsetsum (f x) (B x))
  by (intro prod.cong refl) (simp_all add: B'_def)
finally show ?thesis .
qed

```

```

lemma infsetsum_uminus: infsetsum (λx. -f x) A = -infsetsum f A
unfolding infsetsum_def abs_summable_on_def
by (rule Bochner_Integration.integral_minus)

```

```

lemma infsetsum_add:
assumes f abs_summable_on A and g abs_summable_on A
shows infsetsum (λx. f x + g x) A = infsetsum f A + infsetsum g A
using assms unfolding infsetsum_def abs_summable_on_def
by (rule Bochner_Integration.integral_add)

```

```

lemma infsetsum_diff:
assumes f abs_summable_on A and g abs_summable_on A
shows infsetsum (λx. f x - g x) A = infsetsum f A - infsetsum g A
using assms unfolding infsetsum_def abs_summable_on_def
by (rule Bochner_Integration.integral_diff)

```

```

lemma infsetsum_scaleR_left:
assumes c ≠ 0 ⇒ f abs_summable_on A
shows infsetsum (λx. f x *R c) A = infsetsum f A *R c
using assms unfolding infsetsum_def abs_summable_on_def
by (rule Bochner_Integration.integral_scaleR_left)

```

```

lemma infsetsum_scaleR_right:
infsetsum (λx. c *R f x) A = c *R infsetsum f A
unfolding infsetsum_def abs_summable_on_def
by (subst Bochner_Integration.integral_scaleR_right) auto

```

```

lemma infsetsum_cmult_left:
fixes f :: 'a ⇒ 'b :: {banach, real_normed_algebra, second_countable_topology}

```

```

assumes  $c \neq 0 \implies f \text{ abs\_summable\_on } A$ 
shows  $\text{infsetsum } (\lambda x. f x * c) A = \text{infsetsum } f A * c$ 
using assms unfolding infsetsum_def abs_summable_on_def
by (rule Bochner_Integration.integral_mult_left)

```

```

lemma infsetsum_cmult_right:
fixes  $f :: 'a \Rightarrow 'b :: \{\text{banach, real\_normed\_algebra, second\_countable\_topology}\}$ 
assumes  $c \neq 0 \implies f \text{ abs\_summable\_on } A$ 
shows  $\text{infsetsum } (\lambda x. c * f x) A = c * \text{infsetsum } f A$ 
using assms unfolding infsetsum_def abs_summable_on_def
by (rule Bochner_Integration.integral_mult_right)

```

```

lemma infsetsum_cdiv:
fixes  $f :: 'a \Rightarrow 'b :: \{\text{banach, real\_normed\_field, second\_countable\_topology}\}$ 
assumes  $c \neq 0 \implies f \text{ abs\_summable\_on } A$ 
shows  $\text{infsetsum } (\lambda x. f x / c) A = \text{infsetsum } f A / c$ 
using assms unfolding infsetsum_def abs_summable_on_def by auto

```

```

lemma
fixes  $f :: 'a \Rightarrow 'c :: \{\text{banach, real\_normed\_field, second\_countable\_topology}\}$ 
assumes [simp]: countable  $A$  and [simp]: countable  $B$ 
assumes  $f \text{ abs\_summable\_on } A$  and  $g \text{ abs\_summable\_on } B$ 
shows  $\text{abs\_summable\_on\_product}: (\lambda(x,y). f x * g y) \text{ abs\_summable\_on } A \times B$ 
and  $\text{infsetsum\_product}: \text{infsetsum } (\lambda(x,y). f x * g y) (A \times B) =$ 
 $\text{infsetsum } f A * \text{infsetsum } g B$ 

```

```

proof –
from assms show  $(\lambda(x,y). f x * g y) \text{ abs\_summable\_on } A \times B$ 
by (subst abs_summable_on_Sigma_iff)
  (auto intro!: abs_summable_on_cmult_right simp: norm_mult infsetsum_cmult_right)
with assms show  $\text{infsetsum } (\lambda(x,y). f x * g y) (A \times B) = \text{infsetsum } f A *$ 
 $\text{infsetsum } g B$ 
by (subst infsetsum_Sigma)
  (auto simp: infsetsum_cmult_left infsetsum_cmult_right)
qed

```

```

lemma abs_summable_finite_sumsI:
assumes  $\bigwedge F. \text{finite } F \implies F \subseteq S \implies \text{sum } (\lambda x. \text{norm } (f x)) F \leq B$ 
shows  $f \text{ abs\_summable\_on } S$ 

```

```

proof –
have main:  $f \text{ abs\_summable\_on } S \wedge \text{infsetsum } (\lambda x. \text{norm } (f x)) S \leq B$  if  $\langle B \geq 0 \rangle$  and  $\langle S \neq \{\} \rangle$ 
proof –
define  $M \text{ normf}$  where  $M = \text{count\_space } S$  and  $\text{normf } x = \text{ennreal } (\text{norm } (f x))$  for  $x$ 
have  $\text{sum normf } F \leq \text{ennreal } B$ 

```

```

    if finite F and F ⊆ S and
      ∧ F. finite F ⇒ F ⊆ S ⇒ (∑ i∈F. ennreal (norm (f i))) ≤ ennreal B
and
  ennreal 0 ≤ ennreal B for F
  using that unfolding normf_def[symmetric] by simp
  hence normf_B: finite F ⇒ F ⊆ S ⇒ sum normf F ≤ ennreal B for F
  using assms[THEN ennreal_leI]
  by auto
  have integralS M g ≤ B if simple_function M g and g ≤ normf for g
  proof -
    define gS where gS = g ` S
    have finite gS
      using that unfolding gS_def M_def simple_function_count_space by simp
    have gS ≠ {} unfolding gS_def using ⟨S ≠ {}⟩ by auto
    define part where part r = g - ` {r} ∩ S for r
    have r_finite: r < ∞ if r : gS for r
      using ⟨g ≤ normf⟩ that unfolding gS_def le_fun_def normf_def apply
    auto
      using ennreal_less_top neq_top_trans top.not_eq_extremum by blast
    define B' where B' r = (SUP F∈{F. finite F ∧ F ⊆ part r}. sum normf F)
  for r
    have B'fin: B' r < ∞ for r
    proof -
      have B' r ≤ (SUP F∈{F. finite F ∧ F ⊆ part r}. sum normf F)
        unfolding B'_def
        by (metis (mono_tags, lifting) SUP_least SUP_upper)
      also have ... ≤ B
        using normf_B unfolding part_def
        by (metis (no_types, lifting) Int_subset_iff SUP_least mem_Collect_eq)
      also have ... < ∞
        by simp
      finally show ?thesis by simp
    qed
    have sumB': sum B' gS ≤ ennreal B + ε if ε>0 for ε
    proof -
      define N εN where N = card gS and εN = ε / N
      have N > 0
        unfolding N_def using ⟨gS≠{}⟩ ⟨finite gS⟩
        by (simp add: card_gt_0_iff)
      from εN_def that have εN > 0
        by (simp add: ennreal_of_nat_eq_real_of_nat ennreal_zero_less_divide)
      have c1: ∃ y. B' r ≤ sum normf y + εN ∧ finite y ∧ y ⊆ part r
        if B' r = 0 for r
        using that by auto
      have c2: ∃ y. B' r ≤ sum normf y + εN ∧ finite y ∧ y ⊆ part r if B' r ≠
    0 for r
    proof -
      have B' r - εN < B' r
        using B'fin ⟨0 < εN⟩ ennreal_between that by fastforce

```

```

    have  $B' r - \varepsilon N < \text{Sup } (\text{sum normf } \{F. \text{finite } F \wedge F \subseteq \text{part } r\}) \implies$ 
       $\exists F. B' r - \varepsilon N \leq \text{sum normf } F \wedge \text{finite } F \wedge F \subseteq \text{part } r$ 
    by (metis (no_types, lifting) leD le_cases less_SUP_iff mem_Collect_eq)
    hence  $B' r - \varepsilon N < B' r \implies \exists F. B' r - \varepsilon N \leq \text{sum normf } F \wedge \text{finite } F$ 
 $\wedge F \subseteq \text{part } r$ 
      by (subst (asm) (2) B'_def)
    then obtain  $F$  where  $B' r - \varepsilon N \leq \text{sum normf } F$  and  $\text{finite } F$  and  $F$ 
 $\subseteq \text{part } r$ 
      using  $\langle B' r - \varepsilon N < B' r \rangle$  by auto
    thus  $\exists F. B' r \leq \text{sum normf } F + \varepsilon N \wedge \text{finite } F \wedge F \subseteq \text{part } r$ 
      by (metis add_commute ennreal_minus_le_iff)
  qed
  have  $\forall x. \exists y. B' x \leq \text{sum normf } y + \varepsilon N \wedge$ 
     $\text{finite } y \wedge y \subseteq \text{part } x$ 
    using c1 c2
    by blast
  hence  $\exists F. \forall x. B' x \leq \text{sum normf } (F x) + \varepsilon N \wedge \text{finite } (F x) \wedge F x \subseteq \text{part}$ 
 $x$ 
    by metis
  then obtain  $F$  where  $F: \text{sum normf } (F r) + \varepsilon N \geq B' r$  and  $F\text{fin}: \text{finite}$ 
 $(F r)$  and  $F\text{partr}: F r \subseteq \text{part } r$  for  $r$ 
    using atomize_elim by auto
  have  $w1: \text{finite } gS$ 
    by (simp add:  $\langle \text{finite } gS \rangle$ )
  have  $w2: \forall i \in gS. \text{finite } (F i)$ 
    by (simp add:  $F\text{fin}$ )
  have  $\text{False}$ 
    if  $\bigwedge r. F r \subseteq g - \{r\} \wedge F r \subseteq S$ 
      and  $i \in gS$  and  $j \in gS$  and  $i \neq j$  and  $x \in F i$  and  $x \in F j$ 
    for  $i j x$ 
    by (metis subsetD that(1) that(4) that(5) that(6) vimage_singleton_eq)

  hence  $w3: \forall i \in gS. \forall j \in gS. i \neq j \longrightarrow F i \cap F j = \{\}$ 
    using  $F\text{partr}[\text{unfolded part\_def}]$  by auto
  have  $w4: \text{sum normf } (\bigcup (F ` gS)) + \varepsilon = \text{sum normf } (\bigcup (F ` gS)) + \varepsilon$ 
    by simp
  have  $\text{sum } B' gS \leq (\sum r \in gS. \text{sum normf } (F r) + \varepsilon N)$ 
    using  $F$  by (simp add: sum_mono)
  also have  $\dots = (\sum r \in gS. \text{sum normf } (F r)) + (\sum r \in gS. \varepsilon N)$ 
    by (simp add: sum.distrib)
  also have  $\dots = (\sum r \in gS. \text{sum normf } (F r)) + (\text{card } gS * \varepsilon N)$ 
    by auto
  also have  $\dots = (\sum r \in gS. \text{sum normf } (F r)) + \varepsilon$ 
    unfolding  $\varepsilon N\_def N\_def[\text{symmetric}]$  using  $\langle N > 0 \rangle$ 
  by (simp add: ennreal_times_divide_mult_commute_mult_divide_eq_ennreal)
  also have  $\dots = \text{sum normf } (\bigcup (F ` gS)) + \varepsilon$ 
    using w1 w2 w3 w4
    by (subst sum.UNION_disjoint[symmetric])
  also have  $\dots \leq B + \varepsilon$ 

```

```

      using ⟨finite gS⟩ normf_B add_right_mono Ffin Fpartr unfolding
part_def
  by (simp add: ⟨gS ≠ {}⟩ cSUP_least)
  finally show ?thesis
  by auto
qed
hence sumB': sum B' gS ≤ B
  using ennreal_le_epsilon ennreal_less_zero_iff by blast
have ∀ r. ∃ y. r ∈ gS ⟶ B' r = ennreal y
  using B'_fin less_top_ennreal by auto
hence ∃ B''. ∀ r. r ∈ gS ⟶ B' r = ennreal (B'' r)
  by (rule_tac choice)
then obtain B'' where B'': B' r = ennreal (B'' r) if r ∈ gS for r
  by atomize_elim
have cases[case_names zero finite infinite]: P if r=0 ⟹ P and finite (part
r) ⟹ P
  and infinite (part r) ⟹ r≠0 ⟹ P for P r
  using that by metis
have emeasure_B': r * emeasure M (part r) ≤ B' r if r : gS for r
proof (cases rule:cases[of r])
  case zero
  thus ?thesis by simp
next
  case finite
  have s1: sum g F ≤ sum normf F
    if F ∈ {F. finite F ∧ F ⊆ part r}
    for F
    using ⟨g ≤ normf⟩
    by (simp add: le_fun_def sum_mono)

  have r * of_nat (card (part r)) = r * (∑ x∈part r. 1) by simp
  also have ... = (∑ x∈part r. r)
    using mult.commute by auto
  also have ... = (∑ x∈part r. g x)
    unfolding part_def by auto
  also have ... ≤ (SUP F∈{F. finite F ∧ F⊆part r}. sum g F)
    using finite
    by (simp add: Sup_upper)
  also have ... ≤ B' r
    unfolding B'_def
    using s1 SUP_subset_mono by blast
  finally have r * of_nat (card (part r)) ≤ B' r by assumption
  thus ?thesis
    unfolding M_def
    using part_def finite by auto
next
  case infinite
  from r_finite[OF ⟨r : gS⟩] obtain r' where r': r = ennreal r'
    using ennreal_cases by auto

```

```

with infinite have r' > 0
  using ennreal_less_zero_iff not_gr_zero by blast
obtain N::nat where N:N > B / r' and real N > 0 apply atomize_elim
  using reals_Archimedean2
  by (metis less_trans linorder_negE linordered_idom)
obtain F where finite F and card F = N and F ⊆ part r
  using infinite(1) infinite_arbitrarily_large by blast
from ⟨F ⊆ part r⟩ have F ⊆ S unfolding part_def by simp
have B < r * N
  unfolding r' ennreal_of_nat_eq_real_of_nat
  using N ⟨0 < r'⟩ ⟨B ≥ 0⟩ r'
  by (metis enn2real_ennreal enn2real_less_iff ennreal_less_top ennreal_mult' less_le mult_less_cancel_left pos nonzero_mult_div_cancel_left times_divide_eq_right)
also have r * N = (∑ x∈F. r)
  using ⟨card F = N⟩ by (simp add: mult.commute)
also have (∑ x∈F. r) = (∑ x∈F. g x)
  using ⟨F ⊆ part r⟩ part_def sum.cong subsetD by fastforce
also have (∑ x∈F. g x) ≤ (∑ x∈F. ennreal (norm (f x)))
  by (metis (mono_tags, lifting) ⟨g ≤ normf⟩ ⟨normf ≡ λx. ennreal (norm (f x))⟩ le_fun_def sum_mono)
also have (∑ x∈F. ennreal (norm (f x))) ≤ B
  using ⟨F ⊆ S⟩ ⟨finite F⟩ ⟨normf ≡ λx. ennreal (norm (f x))⟩ normf_B
by blast
  finally have B < B by auto
  thus ?thesis by simp
qed

have integralS M g = (∑ r ∈ gS. r * emeasure M (part r))
  unfolding simple_integral_def gS_def M_def part_def by simp
also have ... ≤ (∑ r ∈ gS. B' r)
  by (simp add: emeasure_B' sum_mono)
also have ... ≤ B
  using sumB' by blast
finally show ?thesis by assumption
qed

hence int_leq_B: integralN M normf ≤ B
unfolding nn_integral_def by (metis (no_types, lifting) SUP_least mem_Collect_eq)
hence integralN M normf < ∞
  using le_less_trans by fastforce
hence integrable M f
  unfolding M_def normf_def by (rule integrableI_bounded[rotated], simp)
hence v1: f abs_summable_on S
  unfolding abs_summable_on_def M_def by simp

have (λx. norm (f x)) abs_summable_on S
  using v1 Infinite_Set_Sum.abs_summable_on_norm_iff[where A = S and f = f]
  by auto

```

```

moreover have  $0 \leq \text{norm } (f\ x)$ 
  if  $x \in S$  for  $x$ 
  by simp
moreover have  $(\int^+ x. \text{ennreal } (\text{norm } (f\ x))\ \partial \text{count\_space } S) \leq \text{ennreal } B$ 
  using  $M\_def\ \langle \text{norm} f \equiv \lambda x. \text{ennreal } (\text{norm } (f\ x)) \rangle\ \text{int\_leq\_} B$  by auto
ultimately have  $\text{ennreal } (\sum_{a \in S}. \text{norm } (f\ a)) \leq \text{ennreal } B$ 
  by (simp add: nn\_integral\_conv\_infsetsum)
hence  $v2: (\sum_{a \in S}. \text{norm } (f\ a)) \leq B$ 
  by (subst ennreal\_le\_iff[symmetric], simp add: assms  $\langle B \geq 0 \rangle$ )
show ?thesis
  using  $v1\ v2$  by auto
qed
then show  $f\ \text{abs\_summable\_on } S$ 
  by (metis abs\_summable\_on\_finite assms empty_subsetI finite.emptyI sum\_clauses(1))
qed

```

```

lemma infsetsum_nonneg_is_SUPREMUM_ennreal:
  fixes  $f :: 'a \Rightarrow \text{real}$ 
  assumes summable:  $f\ \text{abs\_summable\_on } A$ 
  and fnn:  $\bigwedge x. x \in A \implies f\ x \geq 0$ 
  shows  $\text{ennreal } (\text{infsetsum } f\ A) = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{ennreal } (\text{sum } f\ F)))$ 
proof –
  have  $\text{sum } f\ A: \text{sum } f\ F \leq \text{infsetsum } f\ A$ 
    if  $F \in \{F. \text{finite } F \wedge F \subseteq A\}$ 
    for  $F$ 
  proof –
    from that have finite F and  $F \subseteq A$  by auto
    from  $\langle \text{finite } F \rangle$  have  $\text{sum } f\ F = \text{infsetsum } f\ F$  by auto
    also have  $\dots \leq \text{infsetsum } f\ A$ 
  proof (rule infsetsum_mono_neutral_left)
    show  $f\ \text{abs\_summable\_on } F$ 
      by (simp add:  $\langle \text{finite } F \rangle$ )
    show  $f\ \text{abs\_summable\_on } A$ 
      by (simp add: local.summable)
    show  $f\ x \leq f\ x$ 
      if  $x \in F$ 
      for  $x :: 'a$ 
      by simp
    show  $F \subseteq A$ 
      by (simp add:  $\langle F \subseteq A \rangle$ )
    show  $0 \leq f\ x$ 
      if  $x \in A - F$ 
      for  $x :: 'a$ 
      using that fnn by auto
  qed
  finally show ?thesis by assumption
qed

```



```

hence geg: ennreal (infsetsum f A) ≥ (SUP F∈{G. finite G ∧ G ⊆ A}. (ennreal
(sum f F)))
by (meson SUP_least ennreal_leI)

define fe where fe x = ennreal (f x) for x

have sum_f_int: infsetsum f A = ∫+ x. fe x ∂(count_space A)
unfolding infsetsum_def fe_def
proof (rule nn_integral_eq_integral [symmetric])
show integrable (count_space A) f
using abs_summable_on_def local.summable by blast
show AE x in count_space A. 0 ≤ f x
using fnn by auto
qed
also have ... = (SUP g ∈ {g. finite (g'A) ∧ g ≤ fe}. integralS (count_space A)
g)
unfolding nn_integral_def simple_function_count_space by simp
also have ... ≤ (SUP F∈{F. finite F ∧ F ⊆ A}. (ennreal (sum f F)))
proof (rule Sup_least)
fix x assume x ∈ integralS (count_space A) ' {g. finite (g'A) ∧ g ≤ fe}
then obtain g where xg: x = integralS (count_space A) g and fin_gA: finite
(g'A)
and g_fe: g ≤ fe by auto
define F where F = {z:A. g z ≠ 0}
hence F ⊆ A by simp

have fin: finite {z:A. g z = t} if t ≠ 0 for t
proof (rule ccontr)
assume inf: infinite {z:A. g z = t}
hence tgA: t ∈ g'A
by (metis (mono_tags, lifting) image_eqI not_finite_existsD)
have x = (∑ x ∈ g'A. x * emeasure (count_space A) (g -' {x} ∩ A))
unfolding xg simple_integral_def space_count_space by simp
also have ... ≥ (∑ x ∈ {t}. x * emeasure (count_space A) (g -' {x} ∩ A))
(is _ ≥ ...)
proof (rule sum_mono2)
show finite (g'A)
by (simp add: fin_gA)
show {t} ⊆ g'A
by (simp add: tgA)
show 0 ≤ b * emeasure (count_space A) (g -' {b} ∩ A)
if b ∈ g'A - {t}
for b :: ennreal
using that
by simp
qed
also have ... = t * emeasure (count_space A) (g -' {t} ∩ A)
by auto
also have ... = t * ∞

```

```

proof (subst emeasure_count_space_infinite)
  show  $g - \{t\} \cap A \subseteq A$ 
    by simp
  have  $\{a \in A. g \ a = t\} = \{a \in g - \{t\}. a \in A\}$ 
    by auto
  thus infinite  $(g - \{t\} \cap A)$ 
    by (metis (full_types) Int_def inf)
  show  $t * \infty = t * \infty$ 
    by simp
qed
also have  $\dots = \infty$  using  $\langle t \neq 0 \rangle$ 
  by (simp add: ennreal_mult_eq_top_iff)
finally have  $x\_inf: x = \infty$ 
  using neq_top_trans by auto
have  $x = \text{integral}^S (\text{count\_space } A) \ g$  by (fact xg)
also have  $\dots = \text{integral}^N (\text{count\_space } A) \ g$ 
  by (simp add: fin_gA nn_integral_eq_simple_integral)
also have  $\dots \leq \text{integral}^N (\text{count\_space } A) \ fe$ 
  using g_fe
  by (simp add: le_funD nn_integral_mono)
also have  $\dots < \infty$ 
  by (metis sum_f_int ennreal_less_top infinity_ennreal_def)
finally have  $x\_fin: x < \infty$  by simp
from  $x\_inf \ x\_fin$  show False by simp
qed
have  $F: F = (\bigcup t \in g'A - \{0\}. \{z \in A. g \ z = t\})$ 
  unfolding F_def by auto
hence finite F
  unfolding F using fin_gA fin by auto
have  $x = \text{integral}^N (\text{count\_space } A) \ g$ 
  unfolding xg
  by (simp add: fin_gA nn_integral_eq_simple_integral)
also have  $\dots = \text{set\_nn\_integral} (\text{count\_space } UNIV) \ A \ g$ 
  by (simp add: nn_integral_restrict_space[symmetric] restrict_count_space)
also have  $\dots = \text{set\_nn\_integral} (\text{count\_space } UNIV) \ F \ g$ 
proof -
  have  $\forall a. g \ a * (\text{if } a \in \{a \in A. g \ a \neq 0\} \text{ then } 1 \text{ else } 0) = g \ a * (\text{if } a \in A$ 
then 1 else 0)
    by auto
  hence  $(\int^+ a. g \ a * (\text{if } a \in A \text{ then } 1 \text{ else } 0) \ \partial \text{count\_space } UNIV)$ 
     $= (\int^+ a. g \ a * (\text{if } a \in \{a \in A. g \ a \neq 0\} \text{ then } 1 \text{ else } 0) \ \partial \text{count\_space } UNIV)$ 
    by presburger
  thus ?thesis unfolding F_def indicator_def
    using mult.right_neutral mult_zero_right nn_integral_cong
    by (simp add: of_bool_def)
qed
also have  $\dots = \text{integral}^N (\text{count\_space } F) \ g$ 
  by (simp add: nn_integral_restrict_space[symmetric] restrict_count_space)

```

```

also have ... = sum g F
  using ⟨finite F⟩ by (rule nn_integral_count_space_finite)
also have sum g F ≤ sum fe F
  using g_fe unfolding le_fun_def
  by (simp add: sum_mono)
also have ... ≤ (SUP F ∈ {G. finite G ∧ G ⊆ A}. (sum fe F))
  using ⟨finite F⟩ ⟨F ⊆ A⟩
  by (simp add: SUP_upper)
also have ... = (SUP F ∈ {F. finite F ∧ F ⊆ A}. (ennreal (sum f F)))
proof (rule SUP_cong [OF refl])
  have finite x ⇒ x ⊆ A ⇒ (∑ x ∈ x. ennreal (f x)) = ennreal (sum f x)
    for x
    by (metis fnn_subsetCE sum_ennreal)
  thus sum fe x = ennreal (sum f x)
    if x ∈ {G. finite G ∧ G ⊆ A}
    for x :: 'a set
    using that unfolding fe_def by auto
qed
finally show x ≤ ... by simp
qed
finally have leq: ennreal (infsetsum f A) ≤ (SUP F ∈ {F. finite F ∧ F ⊆ A}.
  (ennreal (sum f F)))
  by assumption
from leq geq show ?thesis by simp
qed

lemma infsetsum_nonneg_is_SUPREMUM_ereal:
  fixes f :: 'a ⇒ real
  assumes summable: f abs_summable_on A
  and fnn: ⋀ x. x ∈ A ⇒ f x ≥ 0
  shows ereal (infsetsum f A) = (SUP F ∈ {F. finite F ∧ F ⊆ A}. (ereal (sum f
  F)))
proof -
  have ereal (infsetsum f A) = enn2ereal (ennreal (infsetsum f A))
    by (simp add: fnn infsetsum_nonneg)
  also have ... = enn2ereal (SUP F ∈ {F. finite F ∧ F ⊆ A}. ennreal (sum f F))
    apply (subst infsetsum_nonneg_is_SUPREMUM_ennreal)
    using fnn by (auto simp add: local.summable)
  also have ... = (SUP F ∈ {F. finite F ∧ F ⊆ A}. (ereal (sum f F)))
  proof (simp add: image_def Sup_ennreal.rep_eq)
    have 0 ≤ Sup {y. ∃ x. (∃ xa. finite xa ∧ xa ⊆ A ∧ x = ennreal (sum f xa)) ∧
      y = enn2ereal x}
      by (metis (mono_tags, lifting) Sup_upper empty_subsetI ennreal_0 fi-
      nite.emptyI
        mem_Collect_eq sum.empty zero_ennreal.rep_eq)
    moreover have (∃ x. (∃ y. finite y ∧ y ⊆ A ∧ x = ennreal (sum f y)) ∧ y =
      enn2ereal x) =
      (∃ x. finite x ∧ x ⊆ A ∧ y = ereal (sum f x)) for y
    proof -

```

```

have  $(\exists x. (\exists y. \text{finite } y \wedge y \subseteq A \wedge x = \text{ennreal } (\text{sum } f \ y)) \wedge y = \text{enn2ereal } x) \longleftrightarrow$ 
 $(\exists X x. \text{finite } X \wedge X \subseteq A \wedge x = \text{ennreal } (\text{sum } f \ X) \wedge y = \text{enn2ereal } x)$ 
by blast
also have  $\dots \longleftrightarrow (\exists X. \text{finite } X \wedge X \subseteq A \wedge y = \text{ereal } (\text{sum } f \ X))$ 
by  $(\text{rule } \text{arg\_cong}[of \_ \_ Ex])$ 
 $(\text{auto simp: fun\_eq\_iff intro!: enn2ereal\_ennreal sum\_nonneg enn2ereal\_ennreal[symmetric] fnn})$ 
finally show ?thesis .
qed
hence  $\text{Sup } \{y. \exists x. (\exists y. \text{finite } y \wedge y \subseteq A \wedge x = \text{ennreal } (\text{sum } f \ y)) \wedge y = \text{enn2ereal } x\} =$ 
 $\text{Sup } \{y. \exists x. \text{finite } x \wedge x \subseteq A \wedge y = \text{ereal } (\text{sum } f \ x)\}$ 
by simp
ultimately show  $\max 0 (\text{Sup } \{y. \exists x. (\exists xa. \text{finite } xa \wedge xa \subseteq A \wedge x = \text{ennreal } (\text{sum } f \ xa)) \wedge y = \text{enn2ereal } x\})$ 
 $= \text{Sup } \{y. \exists x. \text{finite } x \wedge x \subseteq A \wedge y = \text{ereal } (\text{sum } f \ x)\}$ 
by linarith
qed
finally show ?thesis
by simp
qed

```

The following theorem relates (abs_summable_on) with $\text{Infinite\_Sum.abs\_summable\_on}$. Note that while this theorem expresses an equivalence, the notion on the lhs is more general nonetheless because it applies to a wider range of types. (The rhs requires second-countable Banach spaces while the lhs is well-defined on arbitrary real vector spaces.)

lemma *abs_summable_equivalent*: $\langle \text{Infinite\_Sum.abs\_summable\_on } f \ A \longleftrightarrow f \text{ abs\_summable\_on } A \rangle$

proof $(\text{rule } \text{iffI})$

define *n* **where** $\langle n \ x = \text{norm } (f \ x) \rangle$ **for** *x*

assume $\langle n \text{ summable\_on } A \rangle$

then have $\langle \text{sum } n \ F \leq \text{infsum } n \ A \rangle$ **if** $\langle \text{finite } F \rangle$ **and** $\langle F \subseteq A \rangle$ **for** *F*

using *that* **by** $(\text{auto simp flip: infsum\_finite simp: n\_def[abs\_def] intro!: infsum\_mono\_neutral})$

then show $\langle f \text{ abs\_summable\_on } A \rangle$

by $(\text{auto intro!: abs\_summable\_finite\_sumsI simp: n\_def})$

next

define *n* **where** $\langle n \ x = \text{norm } (f \ x) \rangle$ **for** *x*

assume $\langle f \text{ abs\_summable\_on } A \rangle$

then have $\langle n \text{ abs\_summable\_on } A \rangle$

by $(\text{simp add: } \langle f \text{ abs\_summable\_on } A \rangle \text{ n\_def})$

then have $\langle \text{sum } n \ F \leq \text{infsetsum } n \ A \rangle$ **if** $\langle \text{finite } F \rangle$ **and** $\langle F \subseteq A \rangle$ **for** *F*

using *that* **by** $(\text{auto simp flip: infsetsum\_finite simp: n\_def[abs\_def] intro!: infsetsum\_mono\_neutral})$

then show $\langle n \text{ summable\_on } A \rangle$

apply $(\text{rule\_tac nonneg\_bdd\_above\_summable\_on})$

```

    by (auto simp add: n_def bdd_above_def)
qed

lemma infsetsum_infsum:
  assumes f_abs_summable_on A
  shows infsetsum f A = infsum f A
proof -
  have conv_sum_norm[simp]:  $(\lambda x. \text{norm } (f x)) \text{ summable\_on } A$ 
    using abs_summable_equivalent assms by blast
  have norm (infsetsum f A - infsum f A)  $\leq \varepsilon$  if  $\varepsilon > 0$  for  $\varepsilon$ 
  proof -
    define  $\delta$  where  $\delta = \varepsilon / 2$ 
    with that have [simp]:  $\delta > 0$  by simp
    obtain F1 where F1A:  $F1 \subseteq A$  and finF1: finite F1 and leq_eps: infsetsum
       $(\lambda x. \text{norm } (f x)) (A - F1) \leq \delta$ 
  proof -
    have sum_SUP:  $\text{ereal } (\text{infsetsum } (\lambda x. \text{norm } (f x)) A) = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{ereal } (\text{sum } (\lambda x. \text{norm } (f x)) F))$ 
      (is  $\_ = ?\text{SUP}$ )
    apply (rule infsetsum_nonneg_is_SUPREMUM_ereal)
    using assms by auto

    have  $(\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{ereal } (\sum_{x \in F} \text{norm } (f x))) - \text{ereal } \delta$ 
       $< (\text{SUP } i \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{ereal } (\sum_{x \in i} \text{norm } (f x)))$ 
    using  $\langle \delta > 0 \rangle$ 
    by (metis diff_strict_left_mono diff_zero ereal_less_eq(3) ereal_minus(1)
      not_le sum_SUP)
    then obtain F where  $F \in \{F. \text{finite } F \wedge F \subseteq A\}$  and  $\text{ereal } (\text{sum } (\lambda x. \text{norm } (f x)) F) > ?\text{SUP} - \text{ereal } (\delta)$ 
    by (meson less_SUP_iff)

    hence  $\text{sum } (\lambda x. \text{norm } (f x)) F > \text{infsetsum } (\lambda x. \text{norm } (f x)) A - (\delta)$ 
    unfolding sum_SUP[symmetric] by auto
    hence  $\delta > \text{infsetsum } (\lambda x. \text{norm } (f x)) (A - F)$ 
  proof (subst infsetsum_Diff)
    show  $(\lambda x. \text{norm } (f x)) \text{ abs\_summable\_on } A$ 
    if  $(\sum_{a \in A} \text{norm } (f a)) - \delta < (\sum_{x \in F} \text{norm } (f x))$ 
    using that
    by (simp add: assms)
    show  $F \subseteq A$ 
    if  $(\sum_{a \in A} \text{norm } (f a)) - \delta < (\sum_{x \in F} \text{norm } (f x))$ 
    using that  $\langle F \in \{F. \text{finite } F \wedge F \subseteq A\} \rangle$  by blast
    show  $(\sum_{a \in A} \text{norm } (f a)) - (\sum_{a \in F} \text{norm } (f a)) < \delta$ 
    if  $(\sum_{a \in A} \text{norm } (f a)) - \delta < (\sum_{x \in F} \text{norm } (f x))$ 
    using that  $\langle F \in \{F. \text{finite } F \wedge F \subseteq A\} \rangle$  by auto
  qed
  thus ?thesis using that
  apply atomize_elim
  using  $\langle F \in \{F. \text{finite } F \wedge F \subseteq A\} \rangle \text{ less\_imp\_le}$  by blast

```

```

qed
obtain F2 where F2A: F2 ⊆ A and finF2: finite F2
  and dist: dist (sum (λx. norm (f x)) F2) (infsum (λx. norm (f x)) A) ≤ δ
  apply atomize_elim
  by (metis ‹0 < δ› conv_sum_norm infsum_finite_approximation)
have leq_eps': infsum (λx. norm (f x)) (A - F2) ≤ δ
  apply (subst infsum_Diff)
  using finF2 F2A dist by (auto simp: dist_norm)
define F where F = F1 ∪ F2
have FA: F ⊆ A and finF: finite F
  unfolding F_def using F1A F2A finF1 finF2 by auto

have (∑a x ∈ A - (F1 ∪ F2). norm (f x)) ≤ (∑a x ∈ A - F1. norm (f x))
  apply (rule infsetsum_mono_neutral_left)
  using abs_summable_on_subset assms by fastforce+
hence leq_eps: infsetsum (λx. norm (f x)) (A - F) ≤ δ
  unfolding F_def
  using leq_eps by linarith
have infsum (λx. norm (f x)) (A - (F1 ∪ F2))
  ≤ infsum (λx. norm (f x)) (A - F2)
  apply (rule infsum_mono_neutral)
  using finF by (auto simp add: finF2 summable_on_cofin_subset F_def)
hence leq_eps': infsum (λx. norm (f x)) (A - F) ≤ δ
  unfolding F_def
  by (rule order.trans[OF leq_eps'])
have norm (infsetsum f A - infsetsum f F) = norm (infsetsum f (A - F))
  apply (subst infsetsum_Diff [symmetric])
  by (auto simp: FA assms)
also have ... ≤ infsetsum (λx. norm (f x)) (A - F)
  using norm_infsetsum_bound by blast
also have ... ≤ δ
  using leq_eps by simp
finally have diff1: norm (infsetsum f A - infsetsum f F) ≤ δ
  by assumption
have norm (infsum f A - infsum f F) = norm (infsum f (A - F))
  apply (subst infsum_Diff [symmetric])
  by (auto simp: assms abs_summable_summable finF FA)
also have ... ≤ infsum (λx. norm (f x)) (A - F)
  by (simp add: finF summable_on_cofin_subset norm_infsum_bound)
also have ... ≤ δ
  using leq_eps' by simp
finally have diff2: norm (infsum f A - infsum f F) ≤ δ
  by assumption

have x1: infsetsum f F = infsum f F
  using finF by simp
have norm (infsetsum f A - infsum f A) ≤ norm (infsetsum f A - infsetsum
f F) + norm (infsum f A - infsum f F)
  apply (rule_tac norm_diff_triangle_le)

```

```

    apply auto
  by (simp_all add: x1 norm_minus_commute)
also have ...  $\leq \varepsilon$ 
  using diff1 diff2  $\delta\_def$  by linarith
finally show ?thesis
  by assumption
qed
hence  $\text{norm } (\text{infsetsum } f \ A - \text{infsum } f \ A) = 0$ 
  by (meson antisym_conv1 dense_ge norm_not_less_zero)
thus ?thesis
  by auto
qed
end

```

6.39 Faces, Extreme Points, Polytopes, Polyhedra etc

Ported from HOL Light by L C Paulson

```

theory Polytope
imports Cartesian_Euclidean_Space Path_Connected
begin

```

6.39.1 Faces of a (usually convex) set

definition $\text{face\_of} :: ['a::\text{real\_vector\_set}, 'a \text{ set}] \Rightarrow \text{bool}$ (**infixr** (face'_of) 50)

where

```

 $T \text{ face\_of } S \longleftrightarrow$ 
 $T \subseteq S \wedge \text{convex } T \wedge$ 
 $(\forall a \in S. \forall b \in S. \forall x \in T. x \in \text{open\_segment } a \ b \longrightarrow a \in T \wedge b \in T)$ 

```

lemma face_ofD : $\llbracket T \text{ face\_of } S; x \in \text{open\_segment } a \ b; a \in S; b \in S; x \in T \rrbracket \Longrightarrow a \in T \wedge b \in T$

unfolding face_of_def **by** blast

lemma $\text{face\_of\_translation\_eq}$ [simp]:

$((+) \ a \ ' \ T \text{ face\_of } (+) \ a \ ' \ S) \longleftrightarrow T \text{ face\_of } S$

proof –

have $*$: $\bigwedge a \ T \ S. T \text{ face\_of } S \Longrightarrow ((+) \ a \ ' \ T \text{ face\_of } (+) \ a \ ' \ S)$

by (simp add: face_of_def)

show ?thesis

by (force simp : $\text{image\_comp } o\_def \text{ dest: } * [\text{where } a = -a] \text{ intro: } *$)

qed

lemma $\text{face\_of\_linear\_image}$:

assumes $\text{linear } f \text{ inj } f$

shows $(f \ ' \ c \text{ face\_of } f \ ' \ S) \longleftrightarrow c \text{ face\_of } S$

2860

by (*simp add: face_of_def inj_image_subset_iff inj_image_mem_iff open_segment_linear_image assms*)

lemma *face_of_refl*: $\text{convex } S \implies S \text{ face\_of } S$
by (*auto simp: face_of_def*)

lemma *face_of_refl_eq*: $S \text{ face\_of } S \longleftrightarrow \text{convex } S$
by (*auto simp: face_of_def*)

lemma *empty_face_of [iff]*: $\{\} \text{ face\_of } S$
by (*simp add: face_of_def*)

lemma *face_of_empty [simp]*: $S \text{ face\_of } \{\} \longleftrightarrow S = \{\}$
by (*meson empty_face_of face_of_def subset_empty*)

lemma *face_of_trans [trans]*: $\llbracket S \text{ face\_of } T; T \text{ face\_of } u \rrbracket \implies S \text{ face\_of } u$
unfolding *face_of_def* **by** (*safe; blast*)

lemma *face_of_face*: $T \text{ face\_of } S \implies (f \text{ face\_of } T \longleftrightarrow f \text{ face\_of } S \wedge f \subseteq T)$
unfolding *face_of_def* **by** (*safe; blast*)

lemma *face_of_subset*: $\llbracket F \text{ face\_of } S; F \subseteq T; T \subseteq S \rrbracket \implies F \text{ face\_of } T$
unfolding *face_of_def* **by** (*safe; blast*)

lemma *face_of_slice*: $\llbracket F \text{ face\_of } S; \text{convex } T \rrbracket \implies (F \cap T) \text{ face\_of } (S \cap T)$
unfolding *face_of_def* **by** (*blast intro: convex_Int*)

lemma *face_of_Int*: $\llbracket t1 \text{ face\_of } S; t2 \text{ face\_of } S \rrbracket \implies (t1 \cap t2) \text{ face\_of } S$
unfolding *face_of_def* **by** (*blast intro: convex_Int*)

lemma *face_of_Inter*: $\llbracket A \neq \{\}; \bigwedge T. T \in A \implies T \text{ face\_of } S \rrbracket \implies (\bigcap A) \text{ face\_of } S$
unfolding *face_of_def* **by** (*blast intro: convex_Inter*)

lemma *face_of_Int_Int*: $\llbracket F \text{ face\_of } T; F' \text{ face\_of } t' \rrbracket \implies (F \cap F') \text{ face\_of } (T \cap t')$
unfolding *face_of_def* **by** (*blast intro: convex_Int*)

lemma *face_of_imp_subset*: $T \text{ face\_of } S \implies T \subseteq S$
unfolding *face_of_def* **by** *blast*

proposition *face_of_imp_eq_affine_Int*:
fixes $S :: 'a::\text{euclidean\_space}$ *set*
assumes $S: \text{convex } S$ **and** $T: T \text{ face\_of } S$
shows $T = (\text{affine hull } T) \cap S$

proof –

have *convex* T **using** T **by** (*simp add: face_of_def*)

have $*$: *False* **if** $x: x \in \text{affine hull } T$ **and** $x \in S$ $x \notin T$ **and** $y: y \in \text{rel\_interior } T$ **for** $x \ y$


```

proof -
  obtain e where e>0 and e: cball y e  $\cap$  affine hull T  $\subseteq$  T
    using y by (auto simp: rel_interior_cball)
  have y  $\neq$  x y  $\in$  S y  $\in$  T
    using face_of_imp_subset rel_interior_subset T that by blast+
  then have zne:  $\bigwedge u. \llbracket u \in \{0 <..< 1\}; (1 - u) *_R y + u *_R x \in T \rrbracket \implies \text{False}$ 
    using  $\langle x \in S \rangle \langle x \notin T \rangle \langle T \text{ face\_of } S \rangle$  unfolding face_of_def
    by (meson greaterThanLessThan_iff in_segment(2))
  have in01:  $\min (1/2) (e / \text{norm } (x - y)) \in \{0 <..< 1\}$ 
    using  $\langle y \neq x \rangle \langle e > 0 \rangle$  by simp
  have §:  $\text{norm } (\min (1/2) (e / \text{norm } (x - y)) *_R y - \min (1/2) (e / \text{norm } (x - y)) *_R x) \leq e$ 
    using  $\langle e > 0 \rangle$ 
    by (simp add: scaleR_diff_right [symmetric] norm_minus_commute min_mult_distrib_right)
  show False
    apply (rule zne [OF in01 e [THEN subsetD]])
    using  $\langle y \in T \rangle$ 
    apply (simp add: hull_inc mem_affine x)
    by (simp add: dist_norm algebra_simps §)
qed
show ?thesis
proof (rule subset_antisym)
  show T  $\subseteq$  affine hull T  $\cap$  S
    using assms by (simp add: hull_subset face_of_imp_subset)
  show affine hull T  $\cap$  S  $\subseteq$  T
    using *  $\langle \text{convex } T \rangle$  rel_interior_eq_empty by fastforce
qed
qed

lemma face_of_imp_closed:
  fixes S :: 'a::euclidean_space set
  assumes convex S closed S T face_of S shows closed T
  by (metis affine_affine_hull affine_closed closed_Int face_of_imp_eq_affine_Int assms)

lemma face_of_Int_supporting_hyperplane_le_strong:
  assumes convex (S  $\cap$  {x. a  $\cdot$  x = b}) and aleb:  $\bigwedge x. x \in S \implies a \cdot x \leq b$ 
  shows (S  $\cap$  {x. a  $\cdot$  x = b}) face_of S
proof -
  have *: a  $\cdot$  u = a  $\cdot$  x if x  $\in$  open_segment u v u  $\in$  S v  $\in$  S and b: b = a  $\cdot$  x
    for u v x
  proof (rule antisym)
    show a  $\cdot$  u  $\leq$  a  $\cdot$  x
      using aleb  $\langle u \in S \rangle \langle b = a \cdot x \rangle$  by blast
  next
    obtain  $\xi$  where b = a  $\cdot$  ((1 -  $\xi$ ) *_R u +  $\xi$  *_R v) 0 <  $\xi$  < 1
      using  $\langle b = a \cdot x \rangle \langle x \in \text{open\_segment } u \ v \rangle$  in_segment
      by (auto simp: open_segment_image_interval split: if_split_asm)
    then have b +  $\xi$  * (a  $\cdot$  u)  $\leq$  a  $\cdot$  u +  $\xi$  * b

```

```

    using aleb [OF ⟨v ∈ S⟩] by (simp add: algebra_simps)
  then have  $(1 - \xi) * b \leq (1 - \xi) * (a \cdot u)$ 
    by (simp add: algebra_simps)
  then have  $b \leq a \cdot u$ 
    using ⟨ξ < 1⟩ by auto
  with b show  $a \cdot x \leq a \cdot u$  by simp
qed
show ?thesis
  using * open_segment_commute by (fastforce simp add: face_of_def assms)
qed

```

```

lemma face_of_Int_supporting_hyperplane_ge_strong:
   $\llbracket \text{convex}(S \cap \{x. a \cdot x = b\}); \bigwedge x. x \in S \implies a \cdot x \geq b \rrbracket$ 
 $\implies (S \cap \{x. a \cdot x = b\}) \text{ face\_of } S$ 
  using face_of_Int_supporting_hyperplane_le_strong [of S -a -b] by simp

```

```

lemma face_of_Int_supporting_hyperplane_le:
   $\llbracket \text{convex } S; \bigwedge x. x \in S \implies a \cdot x \leq b \rrbracket \implies (S \cap \{x. a \cdot x = b\}) \text{ face\_of } S$ 
  by (simp add: convex_Int convex_hyperplane face_of_Int_supporting_hyperplane_le_strong)

```

```

lemma face_of_Int_supporting_hyperplane_ge:
   $\llbracket \text{convex } S; \bigwedge x. x \in S \implies a \cdot x \geq b \rrbracket \implies (S \cap \{x. a \cdot x = b\}) \text{ face\_of } S$ 
  by (simp add: convex_Int convex_hyperplane face_of_Int_supporting_hyperplane_ge_strong)

```

```

lemma face_of_imp_convex:  $T \text{ face\_of } S \implies \text{convex } T$ 
  using face_of_def by blast

```

```

lemma face_of_imp_compact:
  fixes S :: 'a::euclidean_space set
  shows  $\llbracket \text{convex } S; \text{compact } S; T \text{ face\_of } S \rrbracket \implies \text{compact } T$ 
  by (meson bounded_subset compact_eq_bounded_closed face_of_imp_closed
    face_of_imp_subset)

```

```

lemma face_of_Int_subface:
   $\llbracket A \cap B \text{ face\_of } A; A \cap B \text{ face\_of } B; C \text{ face\_of } A; D \text{ face\_of } B \rrbracket$ 
 $\implies (C \cap D) \text{ face\_of } C \wedge (C \cap D) \text{ face\_of } D$ 
  by (meson face_of_Int_Int face_of_face inf_le1 inf_le2)

```

```

lemma subset_of_face_of:
  fixes S :: 'a::real_normed_vector set
  assumes  $T \text{ face\_of } S \wedge u \subseteq S \wedge T \cap (\text{rel\_interior } u) \neq \{\}$ 
  shows  $u \subseteq T$ 
proof
  fix c
  assume  $c \in u$ 
  obtain b where  $b \in T \wedge b \in \text{rel\_interior } u$  using assms by auto
  then obtain e where  $e > 0 \wedge b \in u$  and  $e: \text{cball } b \cap \text{affine hull } u \subseteq u$ 
    by (auto simp: rel_interior_cball)
  show  $c \in T$ 

```

```

proof (cases b=c)
  case True with ⟨b ∈ T⟩ show ?thesis by blast
next
  case False
  define d where d = b + (e / norm(b - c)) *R (b - c)
  have d ∈ cball b e ∩ affine hull u
    using ⟨e > 0⟩ ⟨b ∈ u⟩ ⟨c ∈ u⟩
    by (simp add: d_def dist_norm hull_inc mem_affine_3_minus False)
  with e have d ∈ u by blast
  have nbc: norm (b - c) + e > 0 using ⟨e > 0⟩
  by (metis add.commute le_less_trans less_add_same_cancel2 norm_ge_zero)
  then have [simp]: d ≠ c using False scaleR_cancel_left [of 1 + (e / norm (b
- c)) b c]
    by (simp add: algebra_simps d_def) (simp add: field_split_simps)
  have [simp]: ((e - e * e / (e + norm (b - c))) / norm (b - c)) = (e / (e +
norm (b - c)))
    using False nbc
    by (simp add: divide_simps) (simp add: algebra_simps)
  have b ∈ open_segment d c
    apply (simp add: open_segment_image_interval)
    apply (simp add: d_def algebra_simps image_def)
    apply (rule_tac x=e / (e + norm (b - c)) in bexI)
    using False nbc ⟨0 < e⟩ by (auto simp: algebra_simps)
  then have d ∈ T ∧ c ∈ T
    by (meson ⟨b ∈ T⟩ ⟨c ∈ u⟩ ⟨d ∈ u⟩ assms face_ofD subset_iff)
  then show ?thesis ..
qed
qed

lemma face_of_eq:
  fixes S :: 'a::real_normed_vector set
  assumes T face_of S U face_of S (rel_interior T) ∩ (rel_interior U) ≠ {}
  shows T = U
  using assms
  unfolding disjoint_iff_not_equal
  by (metis IntI empty_iff face_of_imp_subset mem_rel_interior_ball subset_antisym
subset_of_face_of)

lemma face_of_disjoint_rel_interior:
  fixes S :: 'a::real_normed_vector set
  assumes T face_of S T ≠ S
  shows T ∩ rel_interior S = {}
  by (meson assms subset_of_face_of face_of_imp_subset order_refl subset_antisym)

lemma face_of_disjoint_interior:
  fixes S :: 'a::real_normed_vector set
  assumes T face_of S T ≠ S
  shows T ∩ interior S = {}
proof –

```

```

have  $T \cap \text{interior } S \subseteq \text{rel\_interior } S$ 
  by (meson inf_sup_ord(2) interior_subset_rel_interior order.trans)
thus ?thesis
  by (metis (no_types) Int_greatest assms face_of_disjoint_rel_interior inf_sup_ord(1)
subset_empty)
qed

```

```

lemma face_of_subset_rel_boundary:
  fixes  $S :: 'a::\text{real\_normed\_vector\_set}$ 
  assumes  $T \text{ face\_of } S$   $T \neq S$ 
  shows  $T \subseteq (S - \text{rel\_interior } S)$ 
by (meson DiffI assms disjoint_iff_not_equal face_of_disjoint_rel_interior face_of_imp_subset
rev_subsetD subsetI)

```

```

lemma face_of_subset_rel_frontier:
  fixes  $S :: 'a::\text{real\_normed\_vector\_set}$ 
  assumes  $T \text{ face\_of } S$   $T \neq S$ 
  shows  $T \subseteq \text{rel\_frontier } S$ 
  using assms closure_subset face_of_disjoint_rel_interior face_of_imp_subset
rel_frontier_def by fastforce

```

```

lemma face_of_aff_dim_lt:
  fixes  $S :: 'a::\text{euclidean\_space\_set}$ 
  assumes  $\text{convex } S$   $T \text{ face\_of } S$   $T \neq S$ 
  shows  $\text{aff\_dim } T < \text{aff\_dim } S$ 
proof -
  have  $\text{aff\_dim } T \leq \text{aff\_dim } S$ 
    by (simp add: face_of_imp_subset aff_dim_subset assms)
  moreover have  $\text{aff\_dim } T \neq \text{aff\_dim } S$ 
  proof (cases  $T = \{\}$ )
    case True then show ?thesis
      by (metis aff_dim_empty  $\langle T \neq S \rangle$ )
    next case False then show ?thesis
      by (metis Set.set_insert assms convex_rel_frontier_aff_dim dual_order.irrefl
face_of_imp_convex face_of_subset_rel_frontier insert_not_empty subsetI)
  qed
  ultimately show ?thesis
    by simp
qed

```

```

lemma subset_of_face_of_affine_hull:
  fixes  $S :: 'a::\text{euclidean\_space\_set}$ 
  assumes  $T: T \text{ face\_of } S$  and  $\text{convex } S$   $U \subseteq S$  and  $\text{dis: } \neg \text{disjnt } (\text{affine hull } T)$ 
  ( $\text{rel\_interior } U$ )
  shows  $U \subseteq T$ 
proof (rule subset_of_face_of [OF  $T \langle U \subseteq S \rangle$ ])
  show  $T \cap \text{rel\_interior } U \neq \{\}$ 
    using face_of_imp_eq_affine_Int [OF  $\langle \text{convex } S \rangle T$ ] rel_interior_subset [of
 $U$ ]  $\text{dis } \langle U \subseteq S \rangle \text{disjnt\_def}$ 

```

by fastforce
qed

lemma *affine_hull_face_of_disjoint_rel_interior*:

fixes $S :: 'a::\text{euclidean\_space}$ set
assumes $\text{convex } S$ $\text{face\_of } S \ F \neq S$
shows $\text{affine\_hull } F \cap \text{rel\_interior } S = \{\}$
by (metis assms disjnt_def face_of_imp_subset order_refl subset_antisym subset_of_face_of_affine_hull)

lemma *affine_diff_divide*:

assumes $\text{affine } S$ $k \neq 0$ $k \neq 1$ and $xy: x \in S$ $y \ /_R (1 - k) \in S$
shows $(x - y) \ /_R k \in S$
proof –
have $\text{inverse}(k) *_{\text{R}} (x - y) = (1 - \text{inverse } k) *_{\text{R}} \text{inverse}(1 - k) *_{\text{R}} y + \text{inverse}(k) *_{\text{R}} x$
using assms
by (simp add: algebra_simps) (simp add: scaleR_left_distrib [symmetric] field_split_simps)
then show ?thesis
using $\langle \text{affine } S \rangle xy$ by (auto simp: affine_alt)
qed

proposition *face_of_convex_hulls*:

assumes $S: \text{finite } S$ $T \subseteq S$ and $\text{disj: } \text{affine\_hull } T \cap \text{convex\_hull } (S - T) = \{\}$
shows $(\text{convex\_hull } T) \text{ face\_of } (\text{convex\_hull } S)$
proof –
have $\text{fin: } \text{finite } T$ $\text{finite } (S - T)$ using assms
by (auto simp: finite_subset)
have *: $x \in \text{convex\_hull } T$
if $x: x \in \text{convex\_hull } S$ and $y: y \in \text{convex\_hull } S$ and $w: w \in \text{convex\_hull } T$ $w \in \text{open\_segment } x \ y$
for $x \ y \ w$
proof –
have $\text{waff: } w \in \text{affine\_hull } T$
using convex_hull_subset_affine_hull w by blast
obtain $a \ b$ where $a: \bigwedge i. i \in S \implies 0 \leq a \ i$ and $\text{asum: } \text{sum } a \ S = 1$ and $\text{aeqx: } (\sum_{i \in S} a \ i *_{\text{R}} i) = x$
and $b: \bigwedge i. i \in S \implies 0 \leq b \ i$ and $\text{bsum: } \text{sum } b \ S = 1$ and $\text{begy: } (\sum_{i \in S} b \ i *_{\text{R}} i) = y$
using $x \ y$ by (auto simp: assms convex_hull_finite)
obtain u where $(1 - u) *_{\text{R}} x + u *_{\text{R}} y \in \text{convex\_hull } T$ $x \neq y$ and $\text{weq: } w = (1 - u) *_{\text{R}} x + u *_{\text{R}} y$
and $u01: 0 < u < 1$
using w by (auto simp: open_segment_image_interval split: if_split_asm)
define c where $c \ i = (1 - u) * a \ i + u * b \ i$ for i
have $\text{cge0: } \bigwedge i. i \in S \implies 0 \leq c \ i$
using $a \ b \ u01$ by (simp add: c_def)

```

have sumc1: sum c S = 1
  by (simp add: c_def sum.distrib sum_distrib_left [symmetric] asum bsum)
have sumci_xy:  $(\sum_{i \in S} c \ i *_{\mathbb{R}} i) = (1 - u) *_{\mathbb{R}} x + u *_{\mathbb{R}} y$ 
  apply (simp add: c_def sum.distrib scaleR_left_distrib)
by (simp only: scaleR_scaleR [symmetric] Real_Vector_Spaces.scaleR_right.sum
[symmetric] aeqx beqy)
show ?thesis
proof (cases sum c (S - T) = 0)
case True
  have ci0:  $\bigwedge i. i \in (S - T) \implies c \ i = 0$ 
    using True cge0 fin(2) sum_nonneg_eq_0_iff by auto
  have a0:  $a \ i = 0$  if  $i \in (S - T)$  for  $i$ 
    using ci0 [OF that] u01 a [of i] b [of i] that
  by (simp add: c_def Groups.ordered_comm_monoid_add_class.add_nonneg_eq_0_iff)
  have [simp]: sum a T = 1
    using assms by (metis sum.mono_neutral_cong_right a0 asum)
  show ?thesis
    apply (simp add: convex_hull_finite ⟨finite T⟩)
    apply (rule_tac x=a in exI)
    using a0 assms
    apply (auto simp: cge0 a aeqx [symmetric] sum.mono_neutral_right)
    done
next
case False
  define k where  $k = \text{sum } c \ (S - T)$ 
  have  $k > 0$  using False
  unfolding k_def by (metis DiffD1 antisym_conv cge0 sum_nonneg not_less)
  have weq_sumsum:  $w = \text{sum } (\lambda x. c \ x *_{\mathbb{R}} x) \ T + \text{sum } (\lambda x. c \ x *_{\mathbb{R}} x) \ (S - T)$ 
    by (metis (no_types) add.commute S(1) S(2) sum.subset_diff sumci_xy weq)
  show ?thesis
  proof (cases k = 1)
  case True
    then have sum c T = 0
      by (simp add: S k_def sum_diff sumc1)
    then have [simp]: sum c (S - T) = 1
      by (simp add: S sum_diff sumc1)
    have ci0:  $\bigwedge i. i \in T \implies c \ i = 0$ 
      by (meson ⟨finite T⟩ ⟨sum c T = 0⟩ ⟨T ⊆ S⟩ cge0 sum_nonneg_eq_0_iff subsetCE)
    then have [simp]:  $(\sum_{i \in S - T} c \ i *_{\mathbb{R}} i) = w$ 
      by (simp add: weq_sumsum)
    have  $w \in \text{convex hull } (S - T)$ 
      apply (simp add: convex_hull_finite fin)
      apply (rule_tac x=c in exI)
      apply (auto simp: cge0 weq True k_def)
      done
    then show ?thesis

```

```

      using disj waff by blast
    next
    case False
    then have sumcf:  $\sum c \ T = 1 - k$ 
      by (simp add: S k_def sum_diff sumc1)
    have ge0:  $\bigwedge x. x \in T \implies 0 \leq \text{inverse } (1 - k) * c \ x$ 
    by (metis  $\langle T \subseteq S \rangle$  cge0 inverse_nonnegative_iff_nonnegative mult_nonneg_nonneg
subsetD sum_nonneg sumcf)
    have eq1:  $(\sum_{x \in T} \text{inverse } (1 - k) * c \ x) = 1$ 
    by (metis False eq_iff_diff_eq_0 mult.commute right_inverse sum_distrib_left
sumcf)
    have  $(\sum_{i \in T} c \ i *_{\mathbb{R}} i) /_{\mathbb{R}} (1 - k) \in \text{convex hull } T$ 
    apply (simp add: convex_hull_finite fin)
    apply (rule_tac  $x = \lambda i. \text{inverse } (1 - k) * c \ i$  in exI)
    by (metis (mono_tags, lifting) eq1 ge0 scaleR_scaleR scale_sum_right
sum.cong)
    with  $\langle 0 < k \rangle$  have  $\text{inverse}(k) *_{\mathbb{R}} (w - \sum (\lambda i. c \ i *_{\mathbb{R}} i) \ T) \in \text{affine hull } T$ 
    by (simp add: affine_diff_divide [OF affine_affine_hull] False waff
convex_hull_subset_affine_hull [THEN subsetD])
    moreover have  $\text{inverse}(k) *_{\mathbb{R}} (w - \sum (\lambda x. c \ x *_{\mathbb{R}} x) \ T) \in \text{convex hull } (S - T)$ 
    apply (simp add: weq_sumsum convex_hull_finite fin)
    apply (rule_tac  $x = \lambda i. \text{inverse } k * c \ i$  in exI)
    using  $\langle k > 0 \rangle$  cge0
    apply (auto simp: scaleR_right.sum sum_distrib_left [symmetric] k_def
[symmetric])
    done
    ultimately show ?thesis
    using disj by blast
  qed
qed
qed
have [simp]:  $\text{convex hull } T \subseteq \text{convex hull } S$ 
  by (simp add:  $\langle T \subseteq S \rangle$  hull_mono)
show ?thesis
  using open_segment_commute by (auto simp: face_of_def intro: *)
qed

proposition face_of_convex_hull_insert:
  assumes finite S a  $\notin$  affine hull S and T: T face_of convex hull S
  shows T face_of convex hull insert a S
proof -
  have convex hull S face_of convex hull insert a S
    by (simp add: asms face_of_convex_hulls insert_Diff_if subset_insertI)
  then show ?thesis
    using T face_of_trans by blast
qed

```

proposition *face_of_affine_trivial*:
assumes *affine* S T *face_of* S
shows $T = \{\}$ $\vee$ $T = S$
proof (*rule ccontr, clarsimp*)
assume $T \neq \{\}$ $T \neq S$
then obtain a **where** $a \in T$ **by** *auto*
then have $a \in S$
using $\langle T \text{ face\_of } S \rangle$ *face_of_imp_subset* **by** *blast*
have $S \subseteq T$
proof
fix b **assume** $b \in S$
show $b \in T$
proof (*cases* $a = b$)
case *True* **with** $\langle a \in T \rangle$ **show** *?thesis* **by** *auto*
next
case *False*
then have $a \neq 2 *_{\mathbb{R}} a - b$
by (*simp add: scaleR_2*)
with *False* **have** $a \in \text{open\_segment } (2 *_{\mathbb{R}} a - b) b$
apply (*clarsimp simp: open_segment_def closed_segment_def*)
apply (*rule_tac x=1/2 in exI*)
by (*simp add: algebra_simps*)
moreover have $2 *_{\mathbb{R}} a - b \in S$
by (*rule mem_affine [OF \langle affine S \rangle \langle a \in S \rangle \langle b \in S \rangle, of 2 -1, simplified]*)
moreover note $\langle b \in S \rangle \langle a \in T \rangle$
ultimately show *?thesis*
by (*rule face_ofD [OF \langle T face_of S \rangle, THEN conjunct2]*)
qed
qed
then show *False*
using $\langle T \neq S \rangle \langle T \text{ face\_of } S \rangle$ *face_of_imp_subset* **by** *blast*
qed

lemma *face_of_affine_eq*:
 $\text{affine } S \implies (T \text{ face\_of } S \iff T = \{\} \vee T = S)$
using *affine_imp_convex* *face_of_affine_trivial* *face_of_refl* **by** *auto*

proposition *Inter_faces_finite_altbound*:
fixes $T :: 'a::\text{euclidean\_space}$ *set set*
assumes *cfaI*: $\bigwedge c. c \in T \implies c \text{ face\_of } S$
shows $\exists F'. \text{finite } F' \wedge F' \subseteq T \wedge \text{card } F' \leq \text{DIM}('a) + 2 \wedge \bigcap F' = \bigcap T$
proof (*cases* $\forall F'. \text{finite } F' \wedge F' \subseteq T \wedge \text{card } F' \leq \text{DIM}('a) + 2 \longrightarrow (\exists c. c \in T \wedge c \cap (\bigcap F') \subset (\bigcap F'))$)
case *True*
then obtain c **where** c :
 $\bigwedge F'. \llbracket \text{finite } F'; F' \subseteq T; \text{card } F' \leq \text{DIM}('a) + 2 \rrbracket \implies c \in F' \in T \wedge c \in F' \cap (\bigcap F') \subset (\bigcap F')$


```

    by metis
  define d where d  $\equiv \lambda n. ((\lambda r. \text{insert } (c \ r) \ r) \sim^n) \{c\}$ 
  note d_def [simp]
  have dSuc:  $\bigwedge n. d \ (\text{Suc } n) = \text{insert } (c \ (d \ n)) \ (d \ n)$ 
    by simp
  have dn_notempty:  $d \ n \neq \{\}$  for n
    by (induction n) auto
  have dn_le_Suc:  $d \ n \subseteq T \wedge \text{finite}(d \ n) \wedge \text{card}(d \ n) \leq \text{Suc } n$  if  $n \leq \text{DIM}('a) + 2$  for n
  using that
  proof (induction n)
    case 0
    then show ?case by (simp add: c)
  next
    case (Suc n)
    then show ?case by (auto simp: c card_insert_if)
  qed
  have aff_dim_le:  $\text{aff\_dim}(\bigcap (d \ n)) \leq \text{DIM}('a) - \text{int } n$  if  $n \leq \text{DIM}('a) + 2$  for n
  using that
  proof (induction n)
    case 0
    then show ?case
      by (simp add: aff_dim_le_DIM)
  next
    case (Suc n)
    have fs:  $\bigcap (d \ (\text{Suc } n)) \text{ face\_of } S$ 
      by (meson Suc.prem cfaI dn_le_Suc dn_notempty face_of_Inter subsetCE)
    have condn:  $\text{convex} (\bigcap (d \ n))$ 
      using Suc.prem nat_le_linear not_less_eq_eq
      by (blast intro: face_of_imp_convex cfaI convex_Inter dest: dn_le_Suc)
    have fdn:  $\bigcap (d \ (\text{Suc } n)) \text{ face\_of } \bigcap (d \ n)$ 
      by (metis (no_types, lifting) Inter_anti_mono Suc.prem dSuc cfaI dn_le_Suc dn_notempty face_of_Inter face_of_imp_subset face_of_subset subset_iff subset_insertI)
    have ne:  $\bigcap (d \ (\text{Suc } n)) \neq \bigcap (d \ n)$ 
      by (metis (no_types, lifting) Suc.prem Suc_leD c complete_lattice_class.Inf_insert dSuc dn_le_Suc less_irrefl order.trans)
    have *:  $\bigwedge m::\text{int}. \bigwedge d. \bigwedge d'::\text{int}. d < d' \wedge d' \leq m - n \implies d \leq m - \text{of\_nat}(n+1)$ 
      by arith
    have aff_dim ( $\bigcap (d \ (\text{Suc } n))$ )  $< \text{aff\_dim} (\bigcap (d \ n))$ 
      by (rule face_of_aff_dim_lt [OF condn fdn ne])
    moreover have  $\text{aff\_dim} (\bigcap (d \ n)) \leq \text{int} (\text{DIM}('a)) - \text{int } n$ 
      using Suc by auto
    ultimately
    have  $\text{aff\_dim} (\bigcap (d \ (\text{Suc } n))) \leq \text{int} (\text{DIM}('a)) - (n+1)$  by arith
    then show ?case by linarith
  qed
  have  $\text{aff\_dim} (\bigcap (d \ (\text{DIM}('a) + 2))) \leq -2$ 

```

```

    using aff_dim_le [OF order_refl] by simp
  with aff_dim_geq [of  $\bigcap (d (DIM('a) + 2))$ ] show ?thesis
    using order.trans by fastforce
next
  case False
  then show ?thesis
    apply simp
    apply (erule ex_forward)
    by blast
qed

```

lemma *faces_of_translation*:

```

  {F. F face_of image ( $\lambda x. a + x$ ) S} = image (image ( $\lambda x. a + x$ )) {F. F face_of S}
proof -
  have  $\bigwedge F. F \text{ face\_of } (+) a ' S \implies \exists G. G \text{ face\_of } S \wedge F = (+) a ' G$ 
    by (metis face_of_imp_subset face_of_translation_eq subset_imageE)
  then show ?thesis
    by (auto simp: image_iff)
qed

```

proposition *face_of_Times*:

```

  assumes F face_of S and F' face_of S'
  shows (F  $\times$  F') face_of (S  $\times$  S')
proof -
  have F  $\times$  F'  $\subseteq$  S  $\times$  S'
    using assms [unfolded face_of_def] by blast
  moreover
  have convex (F  $\times$  F')
    using assms [unfolded face_of_def] by (blast intro: convex_Times)
  moreover
  have a  $\in$  F  $\wedge$  a'  $\in$  F'  $\wedge$  b  $\in$  F  $\wedge$  b'  $\in$  F'
    if a  $\in$  S b  $\in$  S a'  $\in$  S' b'  $\in$  S' x  $\in$  F  $\times$  F' x  $\in$  open_segment (a,a') (b,b')
    for a b a' b' x
  proof (cases b=a  $\vee$  b'=a')
    case True with that show ?thesis
      using assms
      by (force simp: in_segment dest: face_ofD)
    next
      case False with assms [unfolded face_of_def] that show ?thesis
        by (blast dest!: open_segment_PairD)
  qed
  ultimately show ?thesis
    unfolding face_of_def by blast
qed

```

corollary *face_of_Times_decomp*:

```

  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  shows C face_of (S  $\times$  S')  $\longleftrightarrow$  ( $\exists F F'. F \text{ face\_of } S \wedge F' \text{ face\_of } S' \wedge C =$ 

```

```

 $F \times F')$ 
  (is ?lhs = ?rhs)
proof
  assume C: ?lhs
  show ?rhs
  proof (cases C = {})
    case True then show ?thesis by auto
  next
    case False
    have 1: fst ' C  $\subseteq$  S snd ' C  $\subseteq$  S'
      using C face_of_imp_subset by fastforce+
    have convex C
      using C by (metis face_of_imp_convex)
    have conv: convex (fst ' C) convex (snd ' C)
      by (simp_all add: <convex C> convex_linear_image linear_fst linear_snd)
    have fstab:  $a \in \text{fst ' C} \wedge b \in \text{fst ' C}$ 
      if  $a \in S \wedge b \in S \wedge x \in \text{open\_segment } a \ b \wedge (x, x') \in C$  for  $a \ b \ x \ x'$ 
    proof -
      have *:  $(x, x') \in \text{open\_segment } (a, x') \ (b, x')$ 
        using that by (auto simp: in_segment)
      show ?thesis
        using face_ofD [OF C *] that face_of_imp_subset [OF C] by force
    qed
    have fst: fst ' C face_of S
      by (force simp: face_of_def 1 conv fstab)
    have sndab:  $a' \in \text{snd ' C} \wedge b' \in \text{snd ' C}$ 
      if  $a' \in S' \wedge b' \in S' \wedge x' \in \text{open\_segment } a' \ b' \wedge (x, x') \in C$  for  $a' \ b' \ x \ x'$ 
    proof -
      have *:  $(x, x') \in \text{open\_segment } (x, a') \ (x, b')$ 
        using that by (auto simp: in_segment)
      show ?thesis
        using face_ofD [OF C *] that face_of_imp_subset [OF C] by force
    qed
    have snd: snd ' C face_of S'
      by (force simp: face_of_def 1 conv sndab)
    have cc:  $\text{rel\_interior } C \subseteq \text{rel\_interior } (\text{fst ' C}) \times \text{rel\_interior } (\text{snd ' C})$ 
      by (force simp: face_of_Times rel_interior_Times conv fst snd <convex C>
        linear_fst linear_snd rel_interior_convex_linear_image [symmetric])
    have C = fst ' C  $\times$  snd ' C
    proof (rule face_of_eq [OF C])
      show fst ' C  $\times$  snd ' C face_of S  $\times$  S'
        by (simp add: face_of_Times rel_interior_Times conv fst snd)
      show  $\text{rel\_interior } C \cap \text{rel\_interior } (\text{fst ' C} \times \text{snd ' C}) \neq \{\}$ 
        using False rel_interior_eq_empty <convex C> cc
        by (auto simp: face_of_Times rel_interior_Times conv fst)
    qed
    with fst snd show ?thesis by metis
  qed
next

```

assume ?rhs with face_of_Times show ?lhs by auto
qed

lemma face_of_Times_eq:
fixes $S :: 'a::euclidean\_space\ set$ and $S' :: 'b::euclidean\_space\ set$
shows $(F \times F')\ face\_of\ (S \times S') \longleftrightarrow$
 $F = \{\} \vee F' = \{\} \vee F\ face\_of\ S \wedge F'\ face\_of\ S'$
by (auto simp: face_of_Times_decomp times_eq_iff)

lemma hyperplane_face_of_halfspace_le: $\{x.\ a \cdot x = b\}\ face\_of\ \{x.\ a \cdot x \leq b\}$
proof -
have $\{x.\ a \cdot x \leq b\} \cap \{x.\ a \cdot x = b\} = \{x.\ a \cdot x = b\}$
by auto
with face_of_Int_supporting_hyperplane_le [OF convex_halfspace_le [of a b],
of a b]
show ?thesis by auto
qed

lemma hyperplane_face_of_halfspace_ge: $\{x.\ a \cdot x = b\}\ face\_of\ \{x.\ a \cdot x \geq b\}$
proof -
have $\{x.\ a \cdot x \geq b\} \cap \{x.\ a \cdot x = b\} = \{x.\ a \cdot x = b\}$
by auto
with face_of_Int_supporting_hyperplane_ge [OF convex_halfspace_ge [of b a],
of b a]
show ?thesis by auto
qed

lemma face_of_halfspace_le:
fixes $a :: 'n::euclidean\_space$
shows $F\ face\_of\ \{x.\ a \cdot x \leq b\} \longleftrightarrow$
 $F = \{\} \vee F = \{x.\ a \cdot x = b\} \vee F = \{x.\ a \cdot x \leq b\}$
(is ?lhs = ?rhs)
proof (cases $a = 0$)
case True then show ?thesis
using face_of_affine_eq affine_UNIV by auto
next
case False
then have ine: $\text{interior}\ \{x.\ a \cdot x \leq b\} \neq \{\}$
using halfspace_eq_empty_lt interior_halfspace_le by blast
show ?thesis
proof
assume L: ?lhs
have $F\ face\_of\ \{x.\ a \cdot x = b\}$ if $F \neq \{x.\ a \cdot x \leq b\}$
proof -
have $F\ face\_of\ rel\_frontier\ \{x.\ a \cdot x \leq b\}$
proof (rule face_of_subset [OF L])
show $F \subseteq rel\_frontier\ \{x.\ a \cdot x \leq b\}$
by (simp add: L face_of_subset_rel_frontier that)
qed (force simp: rel_frontier_def closed_halfspace_le)

```

    then show ?thesis
    using False
    by (simp add: frontier_halfspace_le rel_frontier_nonempty_interior [OF
ine])
  qed
  with L show ?rhs
  using affine_hyperplane_face_of_affine_eq by blast
next
  assume ?rhs
  then show ?lhs
  by (metis convex_halfspace_le empty_face_of face_of_refl_hyperplane_face_of_halfspace_le)
qed
qed

lemma face_of_halfspace_ge:
  fixes a :: 'n::euclidean_space
  shows  $F \text{ face\_of } \{x. a \cdot x \geq b\} \longleftrightarrow$ 
 $F = \{\} \vee F = \{x. a \cdot x = b\} \vee F = \{x. a \cdot x \geq b\}$ 
using face_of_halfspace_le [of F -a -b] by simp

```

6.39.2 Exposed faces

That is, faces that are intersection with supporting hyperplane

definition $\text{exposed\_face\_of} :: ['a::\text{euclidean\_space set}, 'a \text{ set}] \Rightarrow \text{bool}$
 $(\text{infixr } (\text{exposed\_face\_of}) \ 50)$

where $T \text{ exposed\_face\_of } S \longleftrightarrow$
 $T \text{ face\_of } S \wedge (\exists a b. S \subseteq \{x. a \cdot x \leq b\} \wedge T = S \cap \{x. a \cdot x = b\})$

```

lemma empty_exposed_face_of [iff]:  $\{\} \text{ exposed\_face\_of } S$ 
  apply (simp add: exposed_face_of_def)
  apply (rule_tac x=0 in exI)
  apply (rule_tac x=1 in exI, force)
  done

```

lemma $\text{exposed\_face\_of\_refl\_eq} \ [simp]: S \text{ exposed\_face\_of } S \longleftrightarrow \text{convex } S$
proof

```

  assume S: convex S
  have  $S \subseteq \{x. 0 \cdot x \leq 0\} \wedge S = S \cap \{x. 0 \cdot x = 0\}$ 
  by auto
  with S show  $S \text{ exposed\_face\_of } S$ 
  using exposed_face_of_def face_of_refl_eq by blast
qed (simp add: exposed_face_of_def face_of_refl_eq)

```

lemma $\text{exposed\_face\_of\_refl}: \text{convex } S \Longrightarrow S \text{ exposed\_face\_of } S$
 by simp

```

lemma exposed_face_of:
   $T \text{ exposed\_face\_of } S \longleftrightarrow$ 
 $T \text{ face\_of } S \wedge$ 

```

```

      (T = {} ∨ T = S ∨
      (∃ a b. a ≠ 0 ∧ S ⊆ {x. a · x ≤ b} ∧ T = S ∩ {x. a · x = b}))
proof (cases T = {})
  case True then show ?thesis
    by simp
next
  case False
  show ?thesis
  proof (cases T = S)
    case True then show ?thesis
      by (simp add: face_of_refl_eq)
  next
  case False
  with ⟨T ≠ {}⟩ show ?thesis
    apply (auto simp: exposed_face_of_def)
    apply (metis inner_zero_left)
    done
  qed
qed

lemma exposed_face_of_Int_supporting_hyperplane_le:
  ⌊convex S; ∧x. x ∈ S ⇒ a · x ≤ b⌋ ⇒ (S ∩ {x. a · x = b}) exposed_face_of
  S
by (force simp: exposed_face_of_def face_of_Int_supporting_hyperplane_le)

lemma exposed_face_of_Int_supporting_hyperplane_ge:
  ⌊convex S; ∧x. x ∈ S ⇒ a · x ≥ b⌋ ⇒ (S ∩ {x. a · x = b}) exposed_face_of
  S
using exposed_face_of_Int_supporting_hyperplane_le [of S -a -b] by simp

proposition exposed_face_of_Int:
  assumes T exposed_face_of S
  and u exposed_face_of S
  shows (T ∩ u) exposed_face_of S
proof -
  obtain a b where T: S ∩ {x. a · x = b} face_of S
  and S: S ⊆ {x. a · x ≤ b}
  and teq: T = S ∩ {x. a · x = b}
  using assms by (auto simp: exposed_face_of_def)
  obtain a' b' where u: S ∩ {x. a' · x = b'} face_of S
  and s': S ⊆ {x. a' · x ≤ b'}
  and ueq: u = S ∩ {x. a' · x = b'}
  using assms by (auto simp: exposed_face_of_def)
  have tu: T ∩ u face_of S
  using T teq u ueq by (simp add: face_of_Int)
  have ss: S ⊆ {x. (a + a') · x ≤ b + b'}
  using S s' by (force simp: inner_left_distrib)
  show ?thesis
  apply (simp add: exposed_face_of_def tu)

```

```

    apply (rule_tac x=a+a' in exI)
    apply (rule_tac x=b+b' in exI)
    using S s'
    apply (fastforce simp: ss inner_left_distrib teq ueq)
    done
qed

proposition exposed_face_of_Inter:
  fixes P :: 'a::euclidean_space set set
  assumes P ≠ {}
    and  $\bigwedge T. T \in P \implies T \text{ exposed\_face\_of } S$ 
    shows  $\bigcap P \text{ exposed\_face\_of } S$ 
proof -
  obtain Q where finite Q and QsubP:  $Q \subseteq P$  card Q ≤ DIM('a) + 2 and IntQ:
 $\bigcap Q = \bigcap P$ 
    using Inter_faces_finite_altbound [of P S] assms [unfolded exposed_face_of]
    by force
  show ?thesis
  proof (cases Q = {})
  case True then show ?thesis
    by (metis IntQ Inter_UNIV_conv(2) assms(1) assms(2) ex_in_conv)
  next
  case False
  have  $Q \subseteq \{T. T \text{ exposed\_face\_of } S\}$ 
    using QsubP assms by blast
  moreover have  $Q \subseteq \{T. T \text{ exposed\_face\_of } S\} \implies \bigcap Q \text{ exposed\_face\_of } S$ 
    using ⟨finite Q⟩ False
    by (induction Q rule: finite_induct; use exposed_face_of_Int in fastforce)
  ultimately show ?thesis
    by (simp add: IntQ)
qed
qed

proposition exposed_face_of_sums:
  assumes convex S and convex T
    and F exposed_face_of {x + y | x y. x ∈ S ∧ y ∈ T}
    (is F exposed_face_of ?ST)
  obtains k l
    where k exposed_face_of S l exposed_face_of T
       $F = \{x + y \mid x y. x \in k \wedge y \in l\}$ 
proof (cases F = {})
  case True then show ?thesis
    using that by blast
  next
  case False
  show ?thesis
  proof (cases F = ?ST)
  case True then show ?thesis
    using assms exposed_face_of_refl_eq that by blast

```

```

next
  case False
  obtain  $p$  where  $p \in F$  using  $\langle F \neq \{\} \rangle$  by blast
  moreover
  obtain  $u\ z$  where  $T: ?ST \cap \{x. u \cdot x = z\}$  face_of  $?ST$ 
    and  $S: ?ST \subseteq \{x. u \cdot x \leq z\}$ 
    and  $feq: F = ?ST \cap \{x. u \cdot x = z\}$ 
    using assms by (auto simp: exposed_face_of_def)
  ultimately obtain  $a0\ b0$ 
    where  $p: p = a0 + b0$  and  $a0 \in S\ b0 \in T$  and  $z: u \cdot p = z$ 
    by auto
  have  $lez: u \cdot (x + y) \leq z$  if  $x \in S\ y \in T$  for  $x\ y$ 
    using S that by auto
  have sef:  $S \cap \{x. u \cdot x = u \cdot a0\}$  exposed_face_of  $S$ 
  proof (rule exposed_face_of_Int_supporting_hyperplane_le [OF  $\langle convex\ S \rangle$ ])
    show  $\bigwedge x. x \in S \implies u \cdot x \leq u \cdot a0$ 
      by (metis p z add_le_cancel_right inner_right_distrib lez [OF  $\_ \langle b0 \in T \rangle$ ])
    qed
  have tef:  $T \cap \{x. u \cdot x = u \cdot b0\}$  exposed_face_of  $T$ 
  proof (rule exposed_face_of_Int_supporting_hyperplane_le [OF  $\langle convex\ T \rangle$ ])
    show  $\bigwedge x. x \in T \implies u \cdot x \leq u \cdot b0$ 
      by (metis p z add.commute add_le_cancel_right inner_right_distrib lez [OF  $\langle a0 \in S \rangle$ ])
    qed
  have  $\{x + y \mid x\ y. x \in S \wedge u \cdot x = u \cdot a0 \wedge y \in T \wedge u \cdot y = u \cdot b0\} \subseteq F$ 
    by (auto simp: feq) (metis inner_right_distrib p z)
  moreover have  $F \subseteq \{x + y \mid x\ y. x \in S \wedge u \cdot x = u \cdot a0 \wedge y \in T \wedge u \cdot y = u \cdot b0\}$ 
  proof -
    have  $\bigwedge x\ y. \llbracket z = u \cdot (x + y); x \in S; y \in T \rrbracket \implies u \cdot x = u \cdot a0 \wedge u \cdot y = u \cdot b0$ 
      using  $z\ p \langle a0 \in S \rangle \langle b0 \in T \rangle$ 
      apply (simp add: inner_right_distrib)
      apply (metis add_le_cancel_right antisym lez [unfolded inner_right_distrib add.commute])
    done
    then show ?thesis
      using feq by blast
    qed
  ultimately have  $F = \{x + y \mid x\ y. x \in S \cap \{x. u \cdot x = u \cdot a0\} \wedge y \in T \cap \{x. u \cdot x = u \cdot b0\}\}$ 
    by blast
  then show ?thesis
    by (rule that [OF sef tef])
  qed
qed

```

proposition *exposed_face_of_parallel*:


```

T exposed_face_of S  $\longleftrightarrow$ 
  T face_of S  $\wedge$ 
  ( $\exists a\ b. S \subseteq \{x. a \cdot x \leq b\} \wedge T = S \cap \{x. a \cdot x = b\} \wedge$ 
    ( $T \neq \{\} \longrightarrow T \neq S \longrightarrow a \neq 0$ )  $\wedge$ 
    ( $T \neq S \longrightarrow (\forall w \in \text{affine hull } S. (w + a) \in \text{affine hull } S)$ ))
(is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
  proof (clarsimp simp: exposed_face_of_def)
    fix a b
    assume faceS:  $S \cap \{x. a \cdot x = b\} = \text{face\_of } S$  and Ssub:  $S \subseteq \{x. a \cdot x \leq b\}$ 
    show  $\exists c\ d. S \subseteq \{x. c \cdot x \leq d\} \wedge$ 
       $S \cap \{x. a \cdot x = b\} = S \cap \{x. c \cdot x = d\} \wedge$ 
      ( $S \cap \{x. a \cdot x = b\} \neq \{\} \longrightarrow S \cap \{x. a \cdot x = b\} \neq S \longrightarrow c \neq 0$ )  $\wedge$ 
      ( $S \cap \{x. a \cdot x = b\} \neq S \longrightarrow (\forall w \in \text{affine hull } S. w + c \in \text{affine hull } S)$ )
  proof (cases affine hull  $S \cap \{x. -a \cdot x \leq -b\} = \{\} \vee \text{affine hull } S \subseteq \{x. -a \cdot x \leq -b\}$ )
    case True
    then show ?thesis
    proof
      assume affine hull  $S \cap \{x. -a \cdot x \leq -b\} = \{\}$ 
      then show ?thesis
      apply (rule_tac x=0 in exI)
      apply (rule_tac x=1 in exI)
      using hull_subset by fastforce
    next
      assume affine hull  $S \subseteq \{x. -a \cdot x \leq -b\}$ 
      then show ?thesis
      apply (rule_tac x=0 in exI)
      apply (rule_tac x=0 in exI)
      using Ssub hull_subset by fastforce
    qed
  next
    case False
    then obtain a' b' where a'  $\neq 0$ 
    and le: affine hull  $S \cap \{x. a' \cdot x \leq b'\} = \text{affine hull } S \cap \{x. -a \cdot x \leq -b\}$ 
    and eq: affine hull  $S \cap \{x. a' \cdot x = b'\} = \text{affine hull } S \cap \{x. -a \cdot x = -b\}$ 
    and mem:  $\bigwedge w. w \in \text{affine hull } S \implies w + a' \in \text{affine hull } S$ 
    using affine_parallel_slice affine_affine_hull by metis
    show ?thesis
    proof (intro conjI impI allI ballI exI)
      have *:  $S \subseteq -(\text{affine hull } S \cap \{x. P\ x\}) \cup \text{affine hull } S \cap \{x. Q\ x\} \implies S \subseteq \{x. \neg P\ x \vee Q\ x\}$ 
      for P Q
      using hull_subset by fastforce
      have  $S \subseteq \{x. \neg (a' \cdot x \leq b') \vee a' \cdot x = b'\}$ 
      by (rule *) (use le eq Ssub in auto)
      then show  $S \subseteq \{x. -a' \cdot x \leq -b'\}$ 
    qed
  qed

```

```

    by auto
  show  $S \cap \{x. a \cdot x = b\} = S \cap \{x. -a' \cdot x = -b'\}$ 
    using eq_hull_subset [of  $S$  affine] by force
  show  $\llbracket S \cap \{x. a \cdot x = b\} \neq \{\}; S \cap \{x. a \cdot x = b\} \neq S \rrbracket \implies -a' \neq 0$ 
    using  $\langle a' \neq 0 \rangle$  by auto
  show  $w + -a' \in \text{affine hull } S$ 
    if  $S \cap \{x. a \cdot x = b\} \neq S$   $w \in \text{affine hull } S$  for  $w$ 
  proof -
    have  $w + 1 *_R (w - (w + a')) \in \text{affine hull } S$ 
      using affine_affine_hull mem mem_affine_3_minus that(2) by blast
    then show ?thesis by simp
  qed
qed
qed
qed
next
  assume ?rhs then show ?lhs
    unfolding exposed_face_of_def by blast
qed

```

6.39.3 Extreme points of a set: its singleton faces

definition *extreme_point_of* :: $[a::\text{real\_vector}, 'a \text{ set}] \Rightarrow \text{bool}$
 (infixr (*extreme'_point'_of*) 50)
 where x *extreme_point_of* $S \longleftrightarrow$
 $x \in S \wedge (\forall a \in S. \forall b \in S. x \notin \text{open\_segment } a \ b)$

lemma *extreme_point_of_stillconvex*:
 $\text{convex } S \implies (x \text{ extreme\_point\_of } S \longleftrightarrow x \in S \wedge \text{convex}(S - \{x\}))$
 by (fastforce simp add: convex_contains_segment extreme_point_of_def open_segment_def)

lemma *face_of_singleton*:
 $\{x\} \text{ face\_of } S \longleftrightarrow x \text{ extreme\_point\_of } S$
 by (fastforce simp add: extreme_point_of_def face_of_def)

lemma *extreme_point_not_in_REL_INTERIOR*:
 fixes $S :: 'a::\text{real\_normed\_vector set}$
 shows $\llbracket x \text{ extreme\_point\_of } S; S \neq \{x\} \rrbracket \implies x \notin \text{rel\_interior } S$
 by (metis disjoint_iff face_of_disjoint_rel_interior face_of_singleton insertI1)

lemma *extreme_point_not_in_interior*:
 fixes $S :: 'a::\{\text{real\_normed\_vector}, \text{perfect\_space}\} \text{ set}$
 assumes $x \text{ extreme\_point\_of } S$ shows $x \notin \text{interior } S$
proof (cases $S = \{x\}$)
 case False
 then show ?thesis
 by (meson asms subsetD extreme_point_not_in_REL_INTERIOR interior_subset_rel_interior)
qed (simp add: empty_interior_finite)

lemma *extreme_point_of_face*:

$F \text{ face\_of } S \implies v \text{ extreme\_point\_of } F \longleftrightarrow v \text{ extreme\_point\_of } S \wedge v \in F$

by (meson empty_subsetI face_of_face face_of_singleton insert_subset)

lemma *extreme_point_of_convex_hull*:

$x \text{ extreme\_point\_of } (\text{convex hull } S) \implies x \in S$

using hull_minimal [of S (convex hull S) - $\{x\}$ convex]

using hull_subset [of S convex]

by (force simp add: extreme_point_of_stillconvex)

proposition *extreme_points_of_convex_hull*:

$\{x. x \text{ extreme\_point\_of } (\text{convex hull } S)\} \subseteq S$

using extreme_point_of_convex_hull **by** auto

lemma *extreme_point_of_empty* [simp]: $\neg (x \text{ extreme\_point\_of } \{\})$

by (simp add: extreme_point_of_def)

lemma *extreme_point_of_singleton* [iff]: $x \text{ extreme\_point\_of } \{a\} \longleftrightarrow x = a$

using extreme_point_of_stillconvex **by** auto

lemma *extreme_point_of_translation_eq*:

$(a + x) \text{ extreme\_point\_of } (\text{image } (\lambda x. a + x) S) \longleftrightarrow x \text{ extreme\_point\_of } S$

by (auto simp: extreme_point_of_def)

lemma *extreme_points_of_translation*:

$\{x. x \text{ extreme\_point\_of } (\text{image } (\lambda x. a + x) S)\} =$

$(\lambda x. a + x) ' \{x. x \text{ extreme\_point\_of } S\}$

using extreme_point_of_translation_eq

by auto (metis (no_types, lifting) image_iff mem_Collect_eq minus_add_cancel)

lemma *extreme_points_of_translation_subtract*:

$\{x. x \text{ extreme\_point\_of } (\text{image } (\lambda x. x - a) S)\} =$

$(\lambda x. x - a) ' \{x. x \text{ extreme\_point\_of } S\}$

using extreme_points_of_translation [of $- a$ S]

by simp

lemma *extreme_point_of_Int*:

$\llbracket x \text{ extreme\_point\_of } S; x \text{ extreme\_point\_of } T \rrbracket \implies x \text{ extreme\_point\_of } (S \cap T)$

by (simp add: extreme_point_of_def)

lemma *extreme_point_of_Int_supporting_hyperplane_le*:

$\llbracket S \cap \{x. a \cdot x = b\} = \{c\}; \bigwedge x. x \in S \implies a \cdot x \leq b \rrbracket \implies c \text{ extreme\_point\_of } S$

by (metis convex_singleton face_of_Int_supporting_hyperplane_le_strong face_of_singleton)

lemma *extreme_point_of_Int_supporting_hyperplane_ge*:

$\llbracket S \cap \{x. a \cdot x = b\} = \{c\}; \bigwedge x. x \in S \implies a \cdot x \geq b \rrbracket \implies c \text{ extreme\_point\_of } S$

using extreme_point_of_Int_supporting_hyperplane_le [of $S -a -b$ c]

by simp

lemma *exposed_point_of_Int_supporting_hyperplane_le*:

$\llbracket S \cap \{x. a \cdot x = b\} = \{c\}; \bigwedge x. x \in S \implies a \cdot x \leq b \rrbracket \implies \{c\} \text{ exposed\_face\_of } S$

unfolding *exposed_face_of_def*

by (*force simp: face_of_singleton extreme_point_of_Int_supporting_hyperplane_le*)

lemma *exposed_point_of_Int_supporting_hyperplane_ge*:

$\llbracket S \cap \{x. a \cdot x = b\} = \{c\}; \bigwedge x. x \in S \implies a \cdot x \geq b \rrbracket \implies \{c\} \text{ exposed\_face\_of } S$

using *exposed_point_of_Int_supporting_hyperplane_le* [*of S -a -b c*]

by *simp*

lemma *extreme_point_of_convex_hull_insert*:

assumes *finite S a* $\notin \text{convex hull } S$

shows *a extreme_point_of* (*convex hull (insert a S)*)

proof (*cases a* $\in S$)

case *False*

then show *?thesis*

using *face_of_convex_hulls* [*of insert a S {a}*] *assms*

by (*auto simp: face_of_singleton hull_same*)

qed (*use assms in <simp add: hull_inc>*)

6.39.4 Facets

definition *facet_of* :: [*'a::euclidean_space set, 'a set*] $\Rightarrow$ *bool*

(**infixr** (*facet'_of*) 50)

where $F \text{ facet\_of } S \iff F \text{ face\_of } S \wedge F \neq \{\} \wedge \text{aff\_dim } F = \text{aff\_dim } S - 1$

lemma *facet_of_empty* [*simp*]: $\neg S \text{ facet\_of } \{\}$

by (*simp add: facet_of_def*)

lemma *facet_of_irrefl* [*simp*]: $\neg S \text{ facet\_of } S$

by (*simp add: facet_of_def*)

lemma *facet_of_imp_face_of*: $F \text{ facet\_of } S \implies F \text{ face\_of } S$

by (*simp add: facet_of_def*)

lemma *facet_of_imp_subset*: $F \text{ facet\_of } S \implies F \subseteq S$

by (*simp add: face_of_imp_subset facet_of_def*)

lemma *hyperplane_facet_of_halfspace_le*:

$a \neq 0 \implies \{x. a \cdot x = b\} \text{ facet\_of } \{x. a \cdot x \leq b\}$

unfolding *facet_of_def hyperplane_eq_empty*

by (*auto simp: hyperplane_face_of_halfspace_ge hyperplane_face_of_halfspace_le*

Suc_leI of_nat_diff aff_dim_halfspace_le)

lemma *hyperplane_facet_of_halfspace_ge*:

$a \neq 0 \implies \{x. a \cdot x = b\} \text{ facet\_of } \{x. a \cdot x \geq b\}$

unfolding *facet_of_def hyperplane_eq_empty*

by (auto simp: hyperplane_face_of_halfspace_le hyperplane_face_of_halfspace_ge
 Suc_leI of_nat_diff aff_dim_halfspace_ge)

lemma facet_of_halfspace_le:

$F \text{ facet\_of } \{x. a \cdot x \leq b\} \longleftrightarrow a \neq 0 \wedge F = \{x. a \cdot x = b\}$
 (is ?lhs = ?rhs)

proof

assume c: ?lhs

with c facet_of_irrefl show ?rhs

by (force simp: aff_dim_halfspace_le facet_of_def face_of_halfspace_le cong:
 conj_cong split: if_split_asm)

next

assume ?rhs then show ?lhs

by (simp add: hyperplane_facet_of_halfspace_le)

qed

lemma facet_of_halfspace_ge:

$F \text{ facet\_of } \{x. a \cdot x \geq b\} \longleftrightarrow a \neq 0 \wedge F = \{x. a \cdot x = b\}$

using facet_of_halfspace_le [of $F - a - b$] by simp

6.39.5 Edges: faces of affine dimension 1

definition edge_of :: [$'a::\text{euclidean\_space}$ set, $'a$ set] $\Rightarrow$ bool (infixr (edge'_of)
 50)

where $e \text{ edge\_of } S \longleftrightarrow e \text{ face\_of } S \wedge \text{aff\_dim } e = 1$

lemma edge_of_imp_subset:

$S \text{ edge\_of } T \implies S \subseteq T$

by (simp add: edge_of_def face_of_imp_subset)

6.39.6 Existence of extreme points

proposition different_norm_3_collinear_points:

fixes $a :: 'a::\text{euclidean\_space}$

assumes $x \in \text{open\_segment } a \ b \ \text{norm}(a) = \text{norm}(b) \ \text{norm}(x) = \text{norm}(b)$

shows False

proof -

obtain u where $\text{norm } ((1 - u) *_{\mathbb{R}} a + u *_{\mathbb{R}} b) = \text{norm } b$

and $a \neq b$

and $u01: 0 < u \ u < 1$

using assms by (auto simp: open_segment_image_interval if_splits)

then have $(1 - u) *_{\mathbb{R}} a \cdot (1 - u) *_{\mathbb{R}} a + ((1 - u) * 2) *_{\mathbb{R}} a \cdot u *_{\mathbb{R}} b =$
 $(1 - u * u) *_{\mathbb{R}} (a \cdot a)$

using assms by (simp add: norm_eq algebra_simps inner_commute)

then have $(1 - u) *_{\mathbb{R}} ((1 - u) *_{\mathbb{R}} a \cdot a + (2 * u) *_{\mathbb{R}} a \cdot b) =$
 $(1 - u) *_{\mathbb{R}} ((1 + u) *_{\mathbb{R}} (a \cdot a))$

by (simp add: algebra_simps)

then have $(1 - u) *_{\mathbb{R}} (a \cdot a) + (2 * u) *_{\mathbb{R}} (a \cdot b) = (1 + u) *_{\mathbb{R}} (a \cdot a)$

using $u01$ by auto

then have $a \cdot b = a \cdot a$

```

    using u01 by (simp add: algebra_simps)
  then have a = b
    using ⟨norm(a) = norm(b)⟩ norm_eq vector_eq by fastforce
  then show ?thesis
    using ⟨a ≠ b⟩ by force
qed

```

proposition *extreme_point_exists_convex*:

```

  fixes S :: 'a::euclidean_space set
  assumes compact S convex S S ≠ {}
  obtains x where x extreme_point_of S
proof -
  obtain x where x ∈ S and xsup:  $\bigwedge y. y \in S \implies \text{norm } y \leq \text{norm } x$ 
    using distance_attains_sup [of S 0] assms by auto
  have False if a ∈ S b ∈ S and x: x ∈ open_segment a b for a b
  proof -
    have noax: norm a ≤ norm x and nobx: norm b ≤ norm x using xsup that
  by auto
    have a ≠ b
      using empty_iff open_segment_idem x by auto
    show False
      by (metis dist_0_norm dist_decreases_open_segment noax nobx not_le x)
  qed
  then show ?thesis
    by (meson ⟨x ∈ S⟩ extreme_point_of_def that)
qed

```

6.39.7 Krein-Milman, the weaker form

proposition *Krein_Milman*:

```

  fixes S :: 'a::euclidean_space set
  assumes compact S convex S
  shows S = closure(convex hull {x. x extreme_point_of S})
proof (cases S = {})
  case True then show ?thesis by simp
next
  case False
  have closed S
    by (simp add: ⟨compact S⟩ compact_imp_closed)
  have closure (convex hull {x. x extreme_point_of S}) ⊆ S
    by (simp add: ⟨closed S⟩ assms closure_minimal extreme_point_of_def hull_minimal)
  moreover have u ∈ closure (convex hull {x. x extreme_point_of S})
    if u ∈ S for u
  proof (rule ccontr)
    assume unot: u ∉ closure (convex hull {x. x extreme_point_of S})
    then obtain a b where a · u < b
      and ab:  $\bigwedge x. x \in \text{closure}(\text{convex hull } \{x. x \text{ extreme\_point\_of } S\}) \implies b < a \cdot x$ 
    using separating_hyperplane_closed_point [of closure (convex hull {x. x ex-

```

```

treme_point_of S}]]
  by blast
have continuous_on S (( $\cdot$ ) a)
  by (rule continuous_intros)+
then obtain m where m  $\in$  S and m:  $\bigwedge y. y \in S \implies a \cdot m \leq a \cdot y$ 
  using continuous_attains_inf [of S  $\lambda x. a \cdot x$ ] <compact S> <u  $\in$  S>
  by auto
define T where T = S  $\cap$  {x. a  $\cdot$  x = a  $\cdot$  m}
have m  $\in$  T
  by (simp add: T_def <m  $\in$  S>)
moreover have compact T
  by (simp add: T_def compact_Int_closed [OF <compact S> closed_hyperplane])
moreover have convex T
  by (simp add: T_def convex_Int [OF <convex S> convex_hyperplane])
ultimately obtain v where v: v extreme_point_of T
  using extreme_point_exists_convex [of T] by auto
then have {v} face_of T
  by (simp add: face_of_singleton)
also have T face_of S
  by (simp add: T_def m face_of_Int_supporting_hyperplane_ge [OF <convex
S>])
finally have v extreme_point_of S
  by (simp add: face_of_singleton)
then have b < a  $\cdot$  v
  using closure_subset by (simp add: closure_hull hull_inc ab)
then show False
  using <a  $\cdot$  u < b> <{v} face_of T> face_of_imp_subset m T_def that by
fastforce
qed
ultimately show ?thesis
  by blast
qed

```

Now the sharper form.

```

lemma Krein_Milman_Minkowski_aux:
  fixes S :: 'a::euclidean_space set
  assumes n: dim S = n and S: compact S convex S 0  $\in$  S
  shows 0  $\in$  convex hull {x. x extreme_point_of S}
using n S
proof (induction n arbitrary: S rule: less_induct)
  case (less n S) show ?case
  proof (cases 0  $\in$  rel_interior S)
    case True with Krein_Milman less.premis
    show ?thesis
      by (metis subsetD convex_convex_hull convex_rel_interior_closure rel_interior_subset)
  next
    case False
    have rel_interior S  $\neq$  {}
      by (simp add: rel_interior_convex_nonempty_aux less)
  end

```

```

then obtain c where c: c ∈ rel_interior S by blast
obtain a where a ≠ 0
  and le_ay:  $\bigwedge y. y \in S \implies a \cdot 0 \leq a \cdot y$ 
  and less_ay:  $\bigwedge y. y \in \text{rel\_interior } S \implies a \cdot 0 < a \cdot y$ 
  by (blast intro: supporting_hyperplane_rel_boundary intro!: less False)
have face:  $S \cap \{x. a \cdot x = 0\}$  face_of S
  using face_of_Int_supporting_hyperplane_ge le_ay ⟨convex S⟩ by auto
then have co: compact ( $S \cap \{x. a \cdot x = 0\}$ ) convex ( $S \cap \{x. a \cdot x = 0\}$ )
  using less.prem by (blast intro: face_of_imp_compact face_of_imp_convex)+
have a · y = 0 if y ∈ span ( $S \cap \{x. a \cdot x = 0\}$ ) for y
proof -
  have y ∈ span {x. a · x = 0}
    by (metis inf.cobounded2 span_mono subsetCE that)
  then show ?thesis
    by (blast intro: span_induct [OF __ subspace_hyperplane])
qed
then have dim ( $S \cap \{x. a \cdot x = 0\}$ ) < n
  by (metis (no_types) less_ay c subsetD dim_eq_span inf.strict_order_iff
    inf_le1 ⟨dim S = n⟩ not_le rel_interior_subset span_0 span_base)
then have 0 ∈ convex hull {x. x extreme_point_of ( $S \cap \{x. a \cdot x = 0\}$ )}
  by (rule less.IH) (auto simp: co less.prem)
then show ?thesis
  by (metis (mono_tags, lifting) Collect_mono_iff face_extreme_point_of_face
    hull_mono subset_iff)
qed
qed

```

```

theorem Krein_Milman_Minkowski:
  fixes S :: 'a::euclidean_space set
  assumes compact S convex S
  shows S = convex hull {x. x extreme_point_of S}
proof
  show S ⊆ convex hull {x. x extreme_point_of S}
  proof
    fix a assume [simp]: a ∈ S
    have 1: compact ((+) (- a) ` S)
      by (simp add: ⟨compact S⟩ compact_translation_subtract cong: image_cong_simp)
    have 2: convex ((+) (- a) ` S)
      by (simp add: ⟨convex S⟩ compact_translation_subtract)
    show a_inver: a ∈ convex hull {x. x extreme_point_of S}
      using Krein_Milman_Minkowski_aux [OF refl 1 2]
        convex_hull_translation [of -a]
        by (auto simp: extreme_points_of_translation_subtract translation_assoc
          cong: image_cong_simp)
    qed
  next
    show convex hull {x. x extreme_point_of S} ⊆ S
    proof -

```



```

  have {a. a extreme_point_of S}  $\subseteq$  S
    using extreme_point_of_def by blast
  then show ?thesis
    by (simp add: convex S hull_minimal)
qed
qed

```

6.39.8 Applying it to convex hulls of explicitly indicated finite sets

corollary *Krein_Milman_polytope:*

fixes $S :: 'a::\text{euclidean\_space}$ set

shows

finite S

$\implies \text{convex hull } S =$

$\text{convex hull } \{x. x \text{ extreme\_point\_of } (\text{convex hull } S)\}$

by (simp add: Krein_Milman_Minkowski finite_imp_compact_convex_hull)

lemma *extreme_points_of_convex_hull_eq:*

fixes $S :: 'a::\text{euclidean\_space}$ set

shows

$\llbracket \text{compact } S; \bigwedge T. T \subset S \implies \text{convex hull } T \neq \text{convex hull } S \rrbracket$

$\implies \{x. x \text{ extreme\_point\_of } (\text{convex hull } S)\} = S$

by (metis (full_types) Krein_Milman_Minkowski compact_convex_hull convex_convex_hull extreme_points_of_convex_hull psubsetI)

lemma *extreme_point_of_convex_hull_eq:*

fixes $S :: 'a::\text{euclidean\_space}$ set

shows

$\llbracket \text{compact } S; \bigwedge T. T \subset S \implies \text{convex hull } T \neq \text{convex hull } S \rrbracket$

$\implies (x \text{ extreme\_point\_of } (\text{convex hull } S) \longleftrightarrow x \in S)$

using extreme_points_of_convex_hull_eq **by** auto

lemma *extreme_point_of_convex_hull_convex_independent:*

fixes $S :: 'a::\text{euclidean\_space}$ set

assumes compact S **and** $S: \bigwedge a. a \in S \implies a \notin \text{convex hull } (S - \{a\})$

shows $(x \text{ extreme\_point\_of } (\text{convex hull } S) \longleftrightarrow x \in S)$

proof –

have convex hull T $\neq$ convex hull S **if** $T \subset S$ **for** T

proof –

obtain a **where** $T \subseteq S$ $a \in S$ $a \notin T$ **using** $\langle T \subset S \rangle$ **by** blast

then show ?thesis

by (metis (full_types) Diff_eq_empty_iff Diff_insert0 S hull_mono hull_subset

insert_Diff_single subsetCE)

qed

then show ?thesis

by (rule extreme_point_of_convex_hull_eq [OF $\langle \text{compact } S \rangle$])

qed

```

lemma extreme_point_of_convex_hull_affine_independent:
  fixes S :: 'a::euclidean_space set
  shows
     $\neg$  affine_dependent S
     $\implies (x \text{ extreme\_point\_of } (\text{convex hull } S) \longleftrightarrow x \in S)$ 
  by (metis aff_independent_finite affine_dependent_def affine_hull_convex_hull
    extreme_point_of_convex_hull_convex_independent_finite_imp_compact_hull_inc)

```

Elementary proofs exist, not requiring Euclidean spaces and all this development

```

lemma extreme_point_of_convex_hull_2:
  fixes x :: 'a::euclidean_space
  shows x extreme_point_of (convex hull {a,b})  $\longleftrightarrow$  x = a  $\vee$  x = b
proof -
  have x extreme_point_of (convex hull {a,b})  $\longleftrightarrow$  x  $\in$  {a,b}
  by (intro extreme_point_of_convex_hull_affine_independent affine_independent_2)
  then show ?thesis
  by simp
qed

```

```

lemma extreme_point_of_segment:
  fixes x :: 'a::euclidean_space
  shows
    x extreme_point_of closed_segment a b  $\longleftrightarrow$  x = a  $\vee$  x = b
  by (simp add: extreme_point_of_convex_hull_2 segment_convex_hull)

```

```

lemma face_of_convex_hull_subset:
  fixes S :: 'a::euclidean_space set
  assumes compact S and T: T face_of (convex hull S)
  obtains s' where s'  $\subseteq$  S T = convex hull s'
proof
  show {x. x extreme_point_of T}  $\subseteq$  S
  using T extreme_point_of_convex_hull extreme_point_of_face by blast
  show T = convex hull {x. x extreme_point_of T}
  proof (rule Krein_Milman_Minkowski)
    show compact T
    using T assms compact_convex_hull face_of_imp_compact by auto
    show convex T
    using T face_of_imp_convex by blast
  qed
qed

```

```

lemma face_of_convex_hull_aux:
  assumes eq: x *_R p = u *_R a + v *_R b + w *_R c
  and x: u + v + w = x x  $\neq$  0 and S: affine S a  $\in$  S b  $\in$  S c  $\in$  S
  shows p  $\in$  S
proof -

```

```

have  $p = (u *_{\mathbb{R}} a + v *_{\mathbb{R}} b + w *_{\mathbb{R}} c) /_{\mathbb{R}} x$ 
  by (metis  $\langle x \neq 0 \rangle$  eq mult.commute right_inverse scaleR_one scaleR_scaleR)
moreover have  $\text{affine hull } \{a, b, c\} \subseteq S$ 
  by (simp add: S hull_minimal)
moreover have  $(u *_{\mathbb{R}} a + v *_{\mathbb{R}} b + w *_{\mathbb{R}} c) /_{\mathbb{R}} x \in \text{affine hull } \{a, b, c\}$ 
  apply (simp add: affine_hull_3)
  apply (rule_tac  $x=u/x$  in exI)
  apply (rule_tac  $x=v/x$  in exI)
  apply (rule_tac  $x=w/x$  in exI)
  using x apply (auto simp: field_split_simps)
done
ultimately show ?thesis by force
qed

proposition face_of_convex_hull_insert_eq:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  assumes finite S and  $a: a \notin \text{affine hull } S$ 
  shows  $(F \text{ face\_of } (\text{convex hull } (\text{insert } a \ S))) \longleftrightarrow$ 
     $F \text{ face\_of } (\text{convex hull } S) \vee$ 
     $(\exists F'. F' \text{ face\_of } (\text{convex hull } S) \wedge F = \text{convex hull } (\text{insert } a \ F'))$ 
    (is  $F \text{ face\_of } ?CAS \longleftrightarrow \_$ )
proof safe
  assume  $F: F \text{ face\_of } ?CAS$ 
  and *:  $\nexists F'. F' \text{ face\_of } \text{convex hull } S \wedge F = \text{convex hull } \text{insert } a \ F'$ 
  obtain T where  $T: T \subseteq \text{insert } a \ S$  and  $\text{FeqT}: F = \text{convex hull } T$ 
  by (metis F  $\langle \text{finite } S \rangle$  compact_insert finite_imp_compact face_of_convex_hull_subset)
  show  $F \text{ face\_of } \text{convex hull } S$ 
  proof (cases  $a \in T$ )
  case True
  have  $F = \text{convex hull } \text{insert } a \ (\text{convex hull } T \cap \text{convex hull } S)$ 
  proof
  have  $T \subseteq \text{insert } a \ (\text{convex hull } T \cap \text{convex hull } S)$ 
  using T hull_subset by fastforce
  then show  $F \subseteq \text{convex hull } \text{insert } a \ (\text{convex hull } T \cap \text{convex hull } S)$ 
  by (simp add: FeqT hull_mono)
  show  $\text{convex hull } \text{insert } a \ (\text{convex hull } T \cap \text{convex hull } S) \subseteq F$ 
  by (simp add: FeqT True hull_inc hull_minimal)
  qed
  moreover have  $\text{convex hull } T \cap \text{convex hull } S \text{ face\_of } \text{convex hull } S$ 
  by (metis F FeqT convex_convex_hull face_of_slice hull_mono inf.absorb_iff2
subset_insertI)
  ultimately show ?thesis
  using * by force
next
case False
  then show ?thesis
  by (metis FeqT F T face_of_subset hull_mono subset_insert subset_insertI)
qed
next

```

```

    assume F face_of convex hull S
    show F face_of ?CAS
    by (simp add: ⟨F face_of convex hull S⟩ a face_of_convex_hull_insert ⟨finite
S⟩)
next
  fix F
  assume F: F face_of convex hull S
  show convex hull insert a F face_of ?CAS
  proof (cases S = {})
    case True
    then show ?thesis
    using F face_of_affine_eq by auto
  next
    case False
    have anotc: a ∉ convex hull S
    by (metis (no_types) a affine_hull_convex_hull hull_inc)
    show ?thesis
    proof (cases F = {})
      case True show ?thesis
      using anotc by (simp add: ⟨F = {}⟩ ⟨finite S⟩ extreme_point_of_convex_hull_insert
face_of_singleton)
    next
      case False
      have convex_hull_insert_a_F ⊆ ?CAS
      by (simp add: F a ⟨finite S⟩ convex_hull_subset face_of_convex_hull_insert
face_of_imp_subset hull_inc)
      moreover
      have (∃ y v. (1 - ub) *R a + ub *R b = (1 - v) *R a + v *R y ∧
0 ≤ v ∧ v ≤ 1 ∧ y ∈ F) ∧
(∃ x u. (1 - uc) *R a + uc *R c = (1 - u) *R a + u *R x ∧
0 ≤ u ∧ u ≤ 1 ∧ x ∈ F)
      if *: (1 - ux) *R a + ux *R x
      ∈ open_segment ((1 - ub) *R a + ub *R b) ((1 - uc) *R a + uc *R
c)
      and 0 ≤ ub ub ≤ 1 0 ≤ uc uc ≤ 1 0 ≤ ux ux ≤ 1
      and b: b ∈ convex hull S and c: c ∈ convex hull S and x ∈ F
      for b c ub uc ux x
      proof -
        have xah: x ∈ affine hull S
        using F convex_hull_subset_affine_hull face_of_imp_subset ⟨x ∈ F⟩ by
blast
        have ah: b ∈ affine hull S c ∈ affine hull S
        using b c convex_hull_subset_affine_hull by blast+
        obtain v where ne: (1 - ub) *R a + ub *R b ≠ (1 - uc) *R a + uc *R c
        and eq: (1 - ux) *R a + ux *R x =
(1 - v) *R ((1 - ub) *R a + ub *R b) + v *R ((1 - uc) *R a +
uc *R c)
        and 0 < v v < 1
        using * by (auto simp: in_segment)

```

```

then have 0:  $((1 - ux) - ((1 - v) * (1 - ub) + v * (1 - uc))) *_R a +$ 
 $(ux *_R x - (((1 - v) * ub) *_R b + (v * uc) *_R c)) = 0$ 
by (auto simp: algebra_simps)
then have  $((1 - ux) - ((1 - v) * (1 - ub) + v * (1 - uc))) *_R a =$ 
 $((1 - v) * ub) *_R b + (v * uc) *_R c + (-ux) *_R x$ 
by (auto simp: algebra_simps)
then have  $a \in \text{affine hull } S$  if  $1 - ux - ((1 - v) * (1 - ub) + v * (1 -$ 
 $uc)) \neq 0$ 
by (rule face_of_convex_hull_aux) (use b c xah ah that in ⟨auto simp:
algebra_simps⟩)
then have  $1 - ux - ((1 - v) * (1 - ub) + v * (1 - uc)) = 0$ 
using a by blast
with 0 have equx:  $(1 - v) * ub + v * uc = ux$ 
and uxx:  $ux *_R x = (((1 - v) * ub) *_R b + (v * uc) *_R c)$ 
by auto (auto simp: algebra_simps)
show ?thesis
proof (cases uc = 0)
case True
then show ?thesis
using equx  $\langle 0 \leq ub \rangle \langle ub \leq 1 \rangle \langle v < 1 \rangle$  uxx  $\langle x \in F \rangle$  by force
next
case False
show ?thesis
proof (cases ub = 0)
case True
then show ?thesis
using equx  $\langle 0 \leq uc \rangle \langle uc \leq 1 \rangle \langle 0 < v \rangle$  uxx  $\langle x \in F \rangle$  by force
next
case False
then have  $0 < ub$   $0 < uc$ 
using  $\langle uc \neq 0 \rangle \langle 0 \leq ub \rangle \langle 0 \leq uc \rangle$  by auto
then have  $(1 - v) * ub > 0$   $v * uc > 0$ 
by (simp_all add:  $\langle 0 < uc \rangle \langle 0 < v \rangle \langle v < 1 \rangle$ )
then have  $ux \neq 0$ 
using equx  $\langle 0 < v \rangle$  by auto
have  $b \in F \wedge c \in F$ 
proof (cases b = c)
case True
then show ?thesis
by (metis  $\langle ux \neq 0 \rangle$  equx real_vector.scale_cancel_left scaleR_add_left
uxx  $\langle x \in F \rangle$ )
next
case False
have  $x = (((1 - v) * ub) *_R b + (v * uc) *_R c) /_R ux$ 
by (metis  $\langle ux \neq 0 \rangle$  uxx mult.commute right_inverse scaleR_one
scaleR_scaleR)
also have  $\dots = (1 - v * uc / ux) *_R b + (v * uc / ux) *_R c$ 
using  $\langle ux \neq 0 \rangle$  equx apply (auto simp: field_split_simps)
by (metis add.commute add_diff_eq add_divide_distrib diff_add_cancel

```

```

scaleR_add_left)
  finally have  $x = (1 - v * uc / ux) *_R b + (v * uc / ux) *_R c$  .
  then have  $x \in \text{open\_segment } b \ c$ 
  apply (simp add: in_segment  $\langle b \neq c \rangle$ )
  apply (rule_tac  $x = (v * uc) / ux$  in exI)
  using  $\langle 0 \leq ux \rangle \langle ux \neq 0 \rangle \langle 0 < uc \rangle \langle 0 < v \rangle \langle 0 < ub \rangle \langle v < 1 \rangle \text{ equx}$ 
  apply (force simp: field_split_simps)
  done
  then show ?thesis
  by (rule face_ofD [OF F _ b c  $\langle x \in F \rangle$ ])
qed
with  $\langle 0 \leq ub \rangle \langle ub \leq 1 \rangle \langle 0 \leq uc \rangle \langle uc \leq 1 \rangle$  show ?thesis by blast
qed
qed
qed
moreover have  $\text{convex hull } F = F$ 
  by (meson F convex_hull_eq face_of_imp_convex)
ultimately show ?thesis
  unfolding face_of_def by (fastforce simp: convex_hull_insert_alt  $\langle S \neq \{\} \rangle \langle F \neq \{\} \rangle$ )
qed
qed
qed

```

lemma *face_of_convex_hull_insert2*:

```

fixes  $a :: 'a :: \text{euclidean\_space}$ 
assumes  $S$ : finite  $S$  and  $a$ :  $a \notin \text{affine hull } S$  and  $F$ :  $F$  face_of  $\text{convex hull } S$ 
shows  $\text{convex hull } (\text{insert } a \ F) \text{ face\_of } \text{convex hull } (\text{insert } a \ S)$ 
by (metis F face_of_convex_hull_insert_eq [OF S a])

```

proposition *face_of_convex_hull_affine_independent*:

```

fixes  $S :: 'a :: \text{euclidean\_space}$  set
assumes  $\neg \text{affine\_dependent } S$ 
shows  $(T \text{ face\_of } (\text{convex hull } S)) \longleftrightarrow (\exists c. c \subseteq S \wedge T = \text{convex hull } c)$ 
(is ?lhs = ?rhs)

```

proof

```

  assume ?lhs
  then show ?rhs
  by (meson  $\langle T \text{ face\_of } \text{convex hull } S \rangle \text{ aff\_independent\_finite assms face\_of\_convex\_hull\_subset finite\_imp\_compact}$ )
next
  assume ?rhs
  then obtain  $c$  where  $c \subseteq S$  and  $T$ :  $T = \text{convex hull } c$ 
  by blast
  have  $\text{affine hull } c \cap \text{affine hull } (S - c) = \{\}$ 
  by (intro disjoint_affine_hull [OF assms  $\langle c \subseteq S \rangle$ , auto])
  then have  $\text{affine hull } c \cap \text{convex hull } (S - c) = \{\}$ 
  using convex_hull_subset_affine_hull by fastforce
  then show ?lhs

```

by (metis face_of_convex_hulls $\langle c \subseteq S \rangle$ aff_independent_finite assms T)
qed

lemma facet_of_convex_hull_affine_independent:

fixes $S :: 'a::euclidean\_space$ set

assumes $\neg$ affine_dependent S

shows T facet_of (convex hull S) $\longleftrightarrow$

$T \neq \{\}$ $\wedge$ ($\exists u. u \in S \wedge T = \text{convex hull } (S - \{u\})$)
(is ?lhs = ?rhs)

proof

assume ?lhs

then have T face_of (convex hull S) $T \neq \{\}$

and afft: aff_dim T = aff_dim (convex hull S) - 1

by (auto simp: facet_of_def)

then obtain c where $c \subseteq S$ and c: $T = \text{convex hull } c$

by (auto simp: face_of_convex_hull_affine_independent [OF assms])

then have affs: aff_dim S = aff_dim c + 1

by (metis aff_dim_convex_hull afft eq_diff_eq)

have $\neg$ affine_dependent c

using $\langle c \subseteq S \rangle$ affine_dependent_subset assms by blast

with affs have card (S - c) = 1

apply (simp add: aff_dim_affine_independent [symmetric] aff_dim_convex_hull)

by (metis aff_dim_affine_independent aff_independent_finite One_nat_def

$\langle c \subseteq S \rangle$ add commute

add_diff_cancel_right' assms card_Diff_subset card_mono of_nat_1

of_nat_diff of_nat_eq_iff)

then obtain u where $u: u \in S - c$

by (metis DiffI $\langle c \subseteq S \rangle$ aff_independent_finite assms cancel_comm_monoid_add_class.diff_cancel
card_Diff_subset subsetI subset_antisym zero_neq_one)

then have $u: S = \text{insert } u \ c$

by (metis Diff_subset $\langle c \subseteq S \rangle$ $\langle \text{card } (S - c) = 1 \rangle$ card_1_singletonE double_diff_insert_Diff insert_subset_singletonD)

have $T = \text{convex hull } (c - \{u\})$

by (metis Diff_empty Diff_insert0 $\langle T \text{ facet\_of convex hull } S \rangle$ c facet_of_irrefl insert_absorb u)

with $\langle T \neq \{\} \rangle$ show ?rhs

using c u by auto

next

assume ?rhs

then obtain u where $T \neq \{\}$ $u \in S$ and $u: T = \text{convex hull } (S - \{u\})$

by (force simp: facet_of_def)

then have $\neg S \subseteq \{u\}$

using $\langle T \neq \{\} \rangle$ u by auto

have aff_dim (S - {u}) = aff_dim S - 1

using assms $\langle u \in S \rangle$

unfolding affine_dependent_def

by (metis add_diff_cancel_right' aff_dim_insert insert_Diff [of u S])

then have aff_dim (convex hull (S - {u})) = aff_dim (convex hull S) - 1

by (simp add: aff_dim_convex_hull)

```

then show ?lhs
  by (metis Diff_subset ‹ $T \neq \{\}$ › assms face_of_convex_hull_affine_independent
facet_of_def u)
qed

```

```

lemma facet_of_convex_hull_affine_independent_alt:
  fixes  $S :: 'a::euclidean\_space$  set
  assumes  $\neg$  affine_dependent  $S$ 
  shows  $(T \text{ facet\_of } (\text{convex hull } S) \longleftrightarrow 2 \leq \text{card } S \wedge (\exists u. u \in S \wedge T = \text{convex hull } (S - \{u\})))$ 
    (is ?lhs = ?rhs)

```

```

proof
  assume  $L$ : ?lhs
  then obtain  $x$  where
     $x \in S$  and  $x$ :  $T = \text{convex hull } (S - \{x\})$  and finite  $S$ 
  using assms facet_of_convex_hull_affine_independent aff_independent_finite
by blast
  moreover have  $\text{Suc } (\text{Suc } 0) \leq \text{card } S$ 
  using  $L$   $x$  ‹ $x \in S$ › ‹finite  $S$ ›
  by (metis Suc_leI assms card.remove convex_hull_eq_empty card_gt_0_iff
facet_of_convex_hull_affine_independent finite_Diff not_less_eq_eq)
  ultimately show ?rhs
    by auto
next
  assume ?rhs then show ?lhs
    using assms
    by (auto simp: facet_of_convex_hull_affine_independent Set.subset_singleton_iff)
qed

```

```

lemma segment_face_of:
  assumes (closed_segment  $a$   $b$ ) face_of  $S$ 
  shows  $a$  extreme_point_of  $S$   $b$  extreme_point_of  $S$ 
proof –
  have  $as$ :  $\{a\}$  face_of  $S$ 
  by (metis (no_types) assms convex_hull_singleton empty_iff extreme_point_of_convex_hull_insert
face_of_face face_of_singleton finite.emptyI finite.insertI insert_absorb insert_iff
segment_convex_hull)
  moreover have  $\{b\}$  face_of  $S$ 
  proof –
    have  $b \in \text{convex hull } \{a\} \vee b$  extreme_point_of  $\text{convex hull } \{b, a\}$ 
    by (meson extreme_point_of_convex_hull_insert finite.emptyI finite.insertI)
    moreover have closed_segment  $a$   $b = \text{convex hull } \{b, a\}$ 
    using closed_segment_commute segment_convex_hull by blast
    ultimately show ?thesis
    by (metis  $as$  assms face_of_face convex_hull_singleton empty_iff face_of_singleton
insertE)
  qed
  ultimately show  $a$  extreme_point_of  $S$   $b$  extreme_point_of  $S$ 
    using face_of_singleton by blast+

```


qed

proposition *Krein_Milman_frontier*:

fixes $S :: 'a::euclidean\_space\ set$

assumes $convex\ S\ compact\ S$

shows $S = convex\_hull\ (frontier\ S)$
 $(is\ ?lhs = ?rhs)$

proof

have $?lhs \subseteq convex\_hull\ \{x. x\ extreme\_point\_of\ S\}$

using *Krein_Milman_Minkowski assms* **by** *blast*

also have $\dots \subseteq ?rhs$

proof (*rule hull_mono*)

show $\{x. x\ extreme\_point\_of\ S\} \subseteq frontier\ S$

using *closure_subset*

by (*auto simp: frontier_def extreme_point_not_in_interior extreme_point_of_def*)

qed

finally show $?lhs \subseteq ?rhs$.

next

have $?rhs \subseteq convex\_hull\ S$

by (*metis Diff_subset <compact S> closure_closed compact_eq_bounded_closed*

frontier_def hull_mono)

also have $\dots \subseteq ?lhs$

by (*simp add: <convex S> hull_same*)

finally show $?rhs \subseteq ?lhs$.

qed

6.39.9 Polytopes

definition *polytope where*

$polytope\ S \equiv \exists v. finite\ v \wedge S = convex\_hull\ v$

lemma *polytope_translation_eq*: $polytope\ (image\ (\lambda x. a + x)\ S) \longleftrightarrow polytope\ S$

proof –

have $\bigwedge a\ A. polytope\ A \implies polytope\ ((+) a\ ` A)$

by (*metis (no_types) convex_hull_translation finite_imageI polytope_def*)

then show *?thesis*

by (*metis (no_types) add.left_inverse image_add_0 translation_assoc*)

qed

lemma *polytope_linear_image*: $\llbracket linear\ f; polytope\ p \rrbracket \implies polytope\ (image\ f\ p)$

unfolding *polytope_def* **using** *convex_hull_linear_image* **by** *blast*

lemma *polytope_empty*: $polytope\ \{\}$

using *convex_hull_empty polytope_def* **by** *blast*

lemma *polytope_convex_hull*: $finite\ S \implies polytope\ (convex\_hull\ S)$

using *polytope_def* **by** *auto*

lemma *polytope_Times*: $\llbracket \text{polytope } S; \text{polytope } T \rrbracket \implies \text{polytope}(S \times T)$
unfolding *polytope_def*
by (*metis finite_cartesian_product convex_hull_Times*)

lemma *face_of_polytope_polytope*:
fixes $S :: 'a::\text{euclidean\_space set}$
shows $\llbracket \text{polytope } S; F \text{ face\_of } S \rrbracket \implies \text{polytope } F$
unfolding *polytope_def*
by (*meson face_of_convex_hull_subset finite_imp_compact finite_subset*)

lemma *finite_polytope_faces*:
fixes $S :: 'a::\text{euclidean\_space set}$
assumes *polytope S*
shows *finite {F. F face_of S}*
proof –
obtain v **where** *finite v S = convex hull v*
using *assms polytope_def* **by** *auto*
have *finite ((hull) convex ' {T. T $\subseteq$ v})*
by (*simp add: <finite v>*)
moreover **have** $\{F. F \text{ face\_of } S\} \subseteq ((\text{hull}) \text{ convex } ' \{T. T \subseteq v\})$
by (*metis (no_types, lifting) <finite v> <S = convex hull v> face_of_convex_hull_subset*
finite_imp_compact image_eqI mem_Collect_eq subsetI)
ultimately show *?thesis*
by (*blast intro: finite_subset*)
qed

lemma *finite_polytope_facets*:
assumes *polytope S*
shows *finite {T. T facet_of S}*
by (*simp add: assms facet_of_def finite_polytope_faces*)

lemma *polytope_scaling*:
assumes *polytope S* **shows** *polytope (image ($\lambda x. c *_R x$) S)*
by (*simp add: assms polytope_linear_image*)

lemma *polytope_imp_compact*:
fixes $S :: 'a::\text{real\_normed\_vector set}$
shows *polytope S $\implies$ compact S*
by (*metis finite_imp_compact_convex_hull polytope_def*)

lemma *polytope_imp_convex*: *polytope S $\implies$ convex S*
by (*metis convex_convex_hull polytope_def*)

lemma *polytope_imp_closed*:
fixes $S :: 'a::\text{real\_normed\_vector set}$
shows *polytope S $\implies$ closed S*
by (*simp add: compact_imp_closed polytope_imp_compact*)

lemma *polytope_imp_bounded*:

```

fixes  $S :: 'a::real\_normed\_vector\ set$ 
shows  $polytope\ S \implies bounded\ S$ 
by (simp add: compact_imp_bounded polytope_imp_compact)

lemma  $polytope\_interval$ :  $polytope(cbox\ a\ b)$ 
unfolding  $polytope\_def$  by (meson closed_interval_as_convex_hull)

lemma  $polytope\_sing$ :  $polytope\ \{a\}$ 
using  $polytope\_def$  by force

lemma  $face\_of\_polytope\_insert$ :
 $\llbracket polytope\ S; a \notin affine\ hull\ S; F\ face\_of\ S \rrbracket \implies F\ face\_of\ convex\ hull\ (insert\ a\ S)$ 
by (metis (no_types, lifting) affine_hull_convex_hull face_of_convex_hull_insert hull_insert polytope_def)

proposition  $face\_of\_polytope\_insert2$ :
fixes  $a :: 'a :: euclidean\_space$ 
assumes  $polytope\ S\ a \notin affine\ hull\ S\ F\ face\_of\ S$ 
shows  $convex\ hull\ (insert\ a\ F)\ face\_of\ convex\ hull\ (insert\ a\ S)$ 
proof –
obtain  $V$  where  $finite\ V\ S = convex\ hull\ V$ 
using  $assms$  by (auto simp: polytope_def)
then have  $convex\ hull\ (insert\ a\ F)\ face\_of\ convex\ hull\ (insert\ a\ V)$ 
using  $affine\_hull\_convex\_hull\ assms\ face\_of\_convex\_hull\_insert2$  by blast
then show ?thesis
by (metis  $\langle S = convex\ hull\ V \rangle\ hull\_insert$ )
qed

```

6.39.10 Polyhedra

definition $polyhedron$ **where**

$$\begin{aligned}
 polyhedron\ S \equiv & \\
 & \exists F. finite\ F \wedge \\
 & S = \bigcap F \wedge \\
 & (\forall h \in F. \exists a\ b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\})
 \end{aligned}$$

```

lemma  $polyhedron\_Int$  [intro,simp]:
 $\llbracket polyhedron\ S; polyhedron\ T \rrbracket \implies polyhedron\ (S \cap T)$ 
apply (clarsimp simp add: polyhedron_def)
subgoal for  $F\ G$ 
by (rule_tac  $x=F \cup G$  in exI, auto)
done

```

```

lemma  $polyhedron\_UNIV$  [iff]:  $polyhedron\ UNIV$ 
unfolding  $polyhedron\_def$ 
by (rule_tac  $x=\{\}$  in exI) auto

```

```

lemma  $polyhedron\_Inter$  [intro,simp]:

```

$\llbracket \text{finite } F; \bigwedge S. S \in F \implies \text{polyhedron } S \rrbracket \implies \text{polyhedron}(\bigcap F)$
 by (induction F rule: finite_induct) auto

lemma polyhedron_empty [iff]: polyhedron ({ } :: 'a :: euclidean_space set)
 proof -
 define i::'a where (i $\equiv$ SOME i. i $\in$ Basis)
 have $\exists a. a \neq 0 \wedge (\exists b. \{x. i \cdot x \leq -1\} = \{x. a \cdot x \leq b\})$
 by (rule_tac x=i in exI) (force simp: i_def SOME_Basis nonzero_Basis)
 moreover have $\exists a b. a \neq 0 \wedge \{x. -i \cdot x \leq -1\} = \{x. a \cdot x \leq b\}$
 apply (rule_tac x=-i in exI)
 apply (rule_tac x=-1 in exI)
 apply (simp add: i_def SOME_Basis nonzero_Basis)
 done
 ultimately show ?thesis
 unfolding polyhedron_def
 by (rule_tac x={x. i $\cdot$ x $\leq$ -1}, {x. -i $\cdot$ x $\leq$ -1} in exI) force
 qed

lemma polyhedron_halfspace_le:
 fixes a :: 'a :: euclidean_space
 shows polyhedron {x. a $\cdot$ x $\leq$ b}
 proof (cases a = 0)
 case True then show ?thesis by auto
 next
 case False
 then show ?thesis
 unfolding polyhedron_def
 by (rule_tac x={x. a $\cdot$ x $\leq$ b} in exI) auto
 qed

lemma polyhedron_halfspace_ge:
 fixes a :: 'a :: euclidean_space
 shows polyhedron {x. a $\cdot$ x $\geq$ b}
 using polyhedron_halfspace_le [of -a -b] by simp

lemma polyhedron_hyperplane:
 fixes a :: 'a :: euclidean_space
 shows polyhedron {x. a $\cdot$ x = b}
 proof -
 have {x. a $\cdot$ x = b} = {x. a $\cdot$ x $\leq$ b} $\cap$ {x. a $\cdot$ x $\geq$ b}
 by force
 then show ?thesis
 by (simp add: polyhedron_halfspace_ge polyhedron_halfspace_le)
 qed

lemma affine_imp_polyhedron:
 fixes S :: 'a :: euclidean_space set
 shows affine S $\implies$ polyhedron S

by (metis affine_hull_eq polyhedron_Inter polyhedron_hyperplane affine_hull_finite_intersection_hyperplanes [of S])

lemma polyhedron_imp_closed:
 fixes $S :: 'a :: \text{euclidean\_space set}$
 shows $\text{polyhedron } S \implies \text{closed } S$
 by (metis closed_Inter closed_halfspace_le polyhedron_def)

lemma polyhedron_imp_convex:
 fixes $S :: 'a :: \text{euclidean\_space set}$
 shows $\text{polyhedron } S \implies \text{convex } S$
 by (metis convex_Inter convex_halfspace_le polyhedron_def)

lemma polyhedron_affine_hull:
 fixes $S :: 'a :: \text{euclidean\_space set}$
 shows $\text{polyhedron}(\text{affine hull } S)$
 by (simp add: affine_imp_polyhedron)

6.39.11 Canonical polyhedron representation making facial structure explicit

proposition polyhedron_Int_affine:
 fixes $S :: 'a :: \text{euclidean\_space set}$
 shows $\text{polyhedron } S \longleftrightarrow$
 $(\exists F. \text{finite } F \wedge S = (\text{affine hull } S) \cap \bigcap F \wedge$
 $(\forall h \in F. \exists a b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}))$
 (is ?lhs = ?rhs)
 proof
 assume ?lhs then show ?rhs
 using hull_subset polyhedron_def by fastforce
 next
 assume ?rhs then show ?lhs
 by (metis polyhedron_Int polyhedron_Inter polyhedron_affine_hull polyhedron_halfspace_le)
 qed

proposition rel_interior_polyhedron_explicit:
 assumes finite F
 and seq: $S = \text{affine hull } S \cap \bigcap F$
 and faceq: $\bigwedge h. h \in F \implies a h \neq 0 \wedge h = \{x. a h \cdot x \leq b h\}$
 and psub: $\bigwedge F'. F' \subset F \implies S \subset \text{affine hull } S \cap \bigcap F'$
 shows $\text{rel\_interior } S = \{x \in S. \forall h \in F. a h \cdot x < b h\}$
 proof -
 have rels: $\bigwedge x. x \in \text{rel\_interior } S \implies x \in S$
 by (meson IntE mem_rel_interior)
 moreover have $a i \cdot x < b i$ if $x: x \in \text{rel\_interior } S$ and $i \in F$ for $x i$
 proof -
 have fif: $F - \{i\} \subset F$
 using $\langle i \in F \rangle$ Diff_insert_absorb Diff_subset set_insert psubsetI by blast

```

then have  $S \subset \text{affine hull } S \cap \bigcap (F - \{i\})$ 
  by (rule psub)
then obtain  $z$  where  $ssub: S \subseteq \bigcap (F - \{i\})$  and  $zint: z \in \bigcap (F - \{i\})$ 
  and  $z \notin S$  and  $zaff: z \in \text{affine hull } S$ 
  by auto
have  $z \neq x$ 
  using  $\langle z \notin S \rangle$  rels  $x$  by blast
have  $z \notin \text{affine hull } S \cap \bigcap F$ 
  using  $\langle z \notin S \rangle$  seq by auto
then have  $aiz: a \cdot i \cdot z > b \cdot i$ 
  using faceq  $zint$   $zaff$  by fastforce
obtain  $e$  where  $e > 0$   $x \in S$  and  $e: \text{ball } x \cap \text{affine hull } S \subseteq S$ 
  using  $x$  by (auto simp: mem_rel_interior_ball)
then have  $ins: \bigwedge y. \llbracket \text{norm } (x - y) < e; y \in \text{affine hull } S \rrbracket \implies y \in S$ 
  by (metis IntI subsetD dist_norm mem_ball)
define  $\xi$  where  $\xi = \min (1/2) (e / 2 / \text{norm}(z - x))$ 
have  $\text{norm } (\xi * x - \xi * z) = \text{norm } (\xi * (x - z))$ 
  by (simp add:  $\xi\_def$  algebra_simps norm_mult)
also have  $\dots = \xi * \text{norm } (x - z)$ 
  using  $\langle e > 0 \rangle$  by (simp add:  $\xi\_def$ )
also have  $\dots < e$ 
  using  $\langle z \neq x \rangle \langle e > 0 \rangle$  by (simp add:  $\xi\_def$  min_def field_split_simps
norm_minus_commute)
finally have  $lte: \text{norm } (\xi * x - \xi * z) < e$ 
have  $\xi\_aff: \xi * z + (1 - \xi) * x \in \text{affine hull } S$ 
  by (metis  $\langle x \in S \rangle$  add commute affine_affine_hull diff_add_cancel hull_inc
mem_affine  $zaff$ )
have  $\xi * z + (1 - \xi) * x \in S$ 
  using  $ins$  [ $OF$   $\xi\_aff$ ] by (simp add: algebra_simps lte)
then obtain  $l$  where  $l: 0 < l < 1$  and  $ls: (l * z + (1 - l) * x) \in S$ 
  using  $\langle e > 0 \rangle \langle z \neq x \rangle$ 
  by (rule_tac  $l = \xi$  in that) (auto simp:  $\xi\_def$ )
then have  $i: l * z + (1 - l) * x \in i$ 
  using seq  $\langle i \in F \rangle$  by auto
have  $b \cdot i \cdot l + (a \cdot i \cdot x) * (1 - l) < a \cdot i \cdot (l * z + (1 - l) * x)$ 
  using  $l$  by (simp add: algebra_simps aiz)
also have  $\dots \leq b \cdot i$  using  $i$ 
  using faceq mem_Collect_eq  $\langle i \in F \rangle$  by blast
finally have  $(a \cdot i \cdot x) * (1 - l) < b \cdot i * (1 - l)$ 
  by (simp add: algebra_simps)
with  $l$  show ?thesis
  by simp
qed
moreover have  $x \in \text{rel\_interior } S$ 
  if  $x \in S$  and less:  $\bigwedge h. h \in F \implies a \cdot h \cdot x < b \cdot h$  for  $x$ 
proof -
  have  $1: \bigwedge h. h \in F \implies x \in \text{interior } h$ 
    by (metis interior_halfspace_le mem_Collect_eq less faceq)
  have  $2: \bigwedge y. \llbracket \forall h \in F. y \in \text{interior } h; y \in \text{affine hull } S \rrbracket \implies y \in S$ 

```

```

    by (metis IntI Inter_iff subsetD interior_subset seq)
  show ?thesis
    apply (simp add: rel_interior ⟨x ∈ S⟩)
    apply (rule_tac x=⋂ h∈F. interior h in exI)
    apply (auto simp: ⟨finite F⟩ open_INT 1 2)
    done
  qed
  ultimately show ?thesis by blast
qed

lemma polyhedron_Int_affine_parallel:
  fixes S :: 'a :: euclidean_space set
  shows polyhedron S ⟷
    (∃ F. finite F ∧
      S = (affine hull S) ∩ (⋂ F) ∧
      (∀ h ∈ F. ∃ a b. a ≠ 0 ∧ h = {x. a • x ≤ b} ∧
        (∀ x ∈ affine hull S. (x + a) ∈ affine hull S)))
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain F where finite F and seq: S = (affine hull S) ∩ ⋂ F
    and faces: ⋀ h. h ∈ F ⟹ ∃ a b. a ≠ 0 ∧ h = {x. a • x ≤ b}
  by (fastforce simp add: polyhedron_Int_affine)
  then obtain a b where ab: ⋀ h. h ∈ F ⟹ a h ≠ 0 ∧ h = {x. a h • x ≤ b h}
  by metis
  show ?rhs
  proof -
    have ∃ a' b'. a' ≠ 0 ∧
      affine hull S ∩ {x. a' • x ≤ b'} = affine hull S ∩ h ∧
      (∀ w ∈ affine hull S. (w + a') ∈ affine hull S)
    if h ∈ F ¬(affine hull S ⊆ h) for h
  proof -
    have a h ≠ 0 and h = {x. a h • x ≤ b h} h ∩ ⋂ F = ⋂ F
    using ⟨h ∈ F⟩ ab by auto
    then have (affine hull S) ∩ {x. a h • x ≤ b h} ≠ {}
    by (metis (no_types) affine_hull_eq_empty inf.absorb_iff2 inf_assoc
      inf_bot_right inf_commute seq that(2))
    moreover have ¬(affine hull S ⊆ {x. a h • x ≤ b h})
    using ⟨h = {x. a h • x ≤ b h}⟩ that(2) by blast
    ultimately show ?thesis
    using affine_parallel_slice [of affine hull S]
    by (metis ⟨h = {x. a h • x ≤ b h}⟩ affine_affine_hull)
  qed
  then obtain a b
  where ab: ⋀ h. [h ∈ F; ¬(affine hull S ⊆ h)]
    ⟹ a h ≠ 0 ∧
      affine hull S ∩ {x. a h • x ≤ b h} = affine hull S ∩ h ∧
      (∀ w ∈ affine hull S. (w + a h) ∈ affine hull S)

```

```

    by metis
    have seq2:  $S = \text{affine hull } S \cap (\bigcap h \in \{h \in F. \neg \text{affine hull } S \subseteq h\}. \{x. a \cdot h \cdot x \leq b \cdot h\})$ 
    by (subst seq) (auto simp: ab INT_extend_simps)
    show ?thesis
    apply (rule_tac x=( $\lambda h. \{x. a \cdot h \cdot x \leq b \cdot h\}$ ) '  $\{h. h \in F \wedge \neg(\text{affine hull } S \subseteq h)\}$  in exI)
    apply (intro conjI seq2)
    using ‹finite F› apply force
    using ab apply blast
    done
qed
next
  assume ?rhs then show ?lhs
    by (metis polyhedron_Int_affine)
qed

```

proposition *polyhedron_Int_affine_parallel_minimal*:

fixes $S :: 'a :: \text{euclidean\_space set}$

shows $\text{polyhedron } S \longleftrightarrow$

$(\exists F. \text{finite } F \wedge$
 $S = (\text{affine hull } S) \cap (\bigcap F) \wedge$
 $(\forall h \in F. \exists a \ b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\} \wedge$
 $(\forall x \in \text{affine hull } S. (x + a) \in \text{affine hull } S)) \wedge$
 $(\forall F'. F' \subset F \longrightarrow S \subset (\text{affine hull } S) \cap (\bigcap F'))))$
(is $?lhs = ?rhs)$

proof

assume $?lhs$

then obtain $f0$

where $f0: \text{finite } f0$

$S = (\text{affine hull } S) \cap (\bigcap f0)$

(is $?P f0)$

$\forall h \in f0. \exists a \ b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\} \wedge$

$(\forall x \in \text{affine hull } S. (x + a) \in \text{affine hull } S)$

(is $?Q f0)$

by (*force simp: polyhedron_Int_affine_parallel*)

define n **where** $n = (\text{LEAST } n. \exists F. \text{card } F = n \wedge \text{finite } F \wedge ?P F \wedge ?Q F)$

have $nf: \exists F. \text{card } F = n \wedge \text{finite } F \wedge ?P F \wedge ?Q F$

apply (*simp add: n_def*)

apply (*rule LeastI [where k = card f0]*)

using $f0$ **apply** *auto*

done

then obtain F **where** $F: \text{card } F = n$ **finite** F **and** $seq: ?P F$ **and** $aff: ?Q F$

by *blast*

then have $\neg (\text{finite } g \wedge ?P g \wedge ?Q g)$ **if** $\text{card } g < n$ **for** g

using *that* **by** (*auto simp: n_def dest!: not_less_Least*)

then have $*$: $\neg (?P g \wedge ?Q g)$ **if** $g \subset F$ **for** g

using *that* ‹finite F› *psubset_card_mono* ‹card F = n›


```

    by (metis finite_Int_inf.strict_order_iff)
  have 1:  $\bigwedge F'. F' \subset F \implies S \subseteq \text{affine hull } S \cap \bigcap F'$ 
    by (subst seq) blast
  have 2:  $S \neq \text{affine hull } S \cap \bigcap F' \text{ if } F' \subset F \text{ for } F'$ 
    using * [OF that] by (metis IntE aff_inf.strict_order_iff that)
  show ?rhs
    by (metis ⟨finite F⟩ seq aff_psubsetI 1 2)
next
  assume ?rhs then show ?lhs
    by (auto simp: polyhedron_Int_affine_parallel)
qed

```

lemma *polyhedron_Int_affine_minimal*:

fixes $S :: 'a :: \text{euclidean\_space set}$

shows $\text{polyhedron } S \longleftrightarrow$

$(\exists F. \text{finite } F \wedge S = (\text{affine hull } S) \cap \bigcap F \wedge$
 $(\forall h \in F. \exists a b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}) \wedge$
 $(\forall F'. F' \subset F \longrightarrow S \subset (\text{affine hull } S) \cap \bigcap F'))$
 (is ?lhs = ?rhs)

proof

assume ?lhs

then show ?rhs

by (force simp: polyhedron_Int_affine_parallel_minimal elim!: ex_forward)

qed (auto simp: polyhedron_Int_affine elim!: ex_forward)

proposition *facet_of_polyhedron_explicit*:

assumes *finite F*

and *seq: S = affine hull S ∩ ∩ F*

and *faceq: ∩ h. h ∈ F ⟹ a h ≠ 0 ∧ h = {x. a h · x ≤ b h}*

and *psub: ∩ F'. F' ⊂ F ⟹ S ⊂ affine hull S ∩ ∩ F'*

shows $C \text{ facet\_of } S \longleftrightarrow (\exists h. h \in F \wedge C = S \cap \{x. a h \cdot x = b h\})$

proof (cases $S = \{\}$)

case True **with** psub **show** ?thesis **by** force

next

case False

have *polyhedron S*

unfolding *polyhedron_Int_affine* **by** (metis ⟨finite F⟩ faceq seq)

then have *convex S*

by (rule polyhedron_imp_convex)

with False *rel_interior_eq_empty* **have** *rel_interior S ≠ {}* **by** blast

then obtain *x* **where** $x \in \text{rel\_interior } S$ **by** auto

then obtain *T* **where** open T $x \in T$ $x \in S$ $T \cap \text{affine hull } S \subseteq S$

by (force simp: mem_rel_interior)

then have *xaff: x ∈ affine hull S* **and** *xint: x ∈ ∩ F*

using *seq hull_inc* **by** auto

have *rel_interior S = {x ∈ S. ∀ h ∈ F. a h · x < b h}*

by (rule *rel_interior_polyhedron_explicit* [OF ⟨finite F⟩ seq faceq psub])

with $\langle x \in \text{rel\_interior } S \rangle$

```

have [simp]:  $\bigwedge h. h \in F \implies a \cdot h \cdot x < b \cdot h$  by blast
have *:  $(S \cap \{x. a \cdot h \cdot x = b \cdot h\}) \text{ facet\_of } S$  if  $h \in F$  for  $h$ 
proof -
  have  $S \subset \text{affine hull } S \cap \bigcap (F - \{h\})$ 
  using psub that by (metis Diff_disjoint Diff_subset insert_disjoint(2) psub-
setI)
  then obtain  $z$  where  $z_{\text{aff}}: z \in \text{affine hull } S$  and  $z_{\text{int}}: z \in \bigcap (F - \{h\})$  and
 $z \notin S$ 
  by force
  then have  $z \neq x \wedge z \notin h$  using seq  $\langle x \in S \rangle$  by auto
  have  $x \in h$  using that  $z_{\text{int}}$  by auto
  then have  $ab: a \cdot h \cdot x \leq b \cdot h$ 
  using faceq that by blast
  also have  $\dots < a \cdot h \cdot z$  using  $\langle z \notin h \rangle$  faceq [OF that]  $z_{\text{int}}$  by auto
  finally have  $xltz: a \cdot h \cdot x < a \cdot h \cdot z$  .
  define  $l$  where  $l = (b \cdot h - a \cdot h \cdot x) / (a \cdot h \cdot z - a \cdot h \cdot x)$ 
  define  $w$  where  $w = (1 - l) * x + l * z$ 
  have  $0 < l < 1$ 
  using ab  $xltz$   $\langle b \cdot h < a \cdot h \cdot z \rangle$   $\langle h \in F \rangle$ 
  by (auto simp: l_def field_split_simps)
  have  $awl: a \cdot i \cdot w < b \cdot i$  if  $i \in F \wedge i \neq h$  for  $i$ 
  proof -
    have  $(1 - l) * (a \cdot i \cdot x) < (1 - l) * b \cdot i$ 
    by (simp add:  $\langle l < 1 \rangle$   $\langle i \in F \rangle$ )
    moreover have  $l * (a \cdot i \cdot z) \leq l * b \cdot i$ 
    proof (rule mult_left_mono)
      show  $a \cdot i \cdot z \leq b \cdot i$ 
      by (metis Diff_insert_absorb Inter_iff Set.set_insert  $\langle h \in F \rangle$  faceq insertE
mem_Collect_eq that  $z_{\text{int}}$ )
    qed (use  $\langle 0 < l \rangle$  in auto)
    ultimately show ?thesis by (simp add: w_def algebra_simps)
  qed
  have  $wq: a \cdot h \cdot w = b \cdot h$ 
  using  $xltz$  unfolding w_def l_def
  by (simp add: algebra_simps) (simp add: field_simps)
  have  $faceS: S \cap \{x. a \cdot h \cdot x = b \cdot h\} \text{ face\_of } S$ 
  proof (rule face_of_Int_supporting_hyperplane_le)
    show  $\bigwedge x. x \in S \implies a \cdot h \cdot x \leq b \cdot h$ 
    using faceq seq that by fastforce
  qed fact
  have  $w \in \text{affine hull } S$ 
  by (simp add: w_def mem_affine xaff zaff)
  moreover have  $w \in \bigcap F$ 
  using  $\langle a \cdot h \cdot w = b \cdot h \rangle$  awlt faceq less_eq_real_def by blast
  ultimately have  $w \in S$ 
  using seq by blast
  with  $wq$  have  $ne: S \cap \{x. a \cdot h \cdot x = b \cdot h\} \neq \{\}$  by blast
  moreover have  $\text{affine hull } (S \cap \{x. a \cdot h \cdot x = b \cdot h\}) = (\text{affine hull } S) \cap \{x. a$ 
 $h \cdot x = b \cdot h\}$ 

```

```

proof
  show affine hull (S ∩ {x. a h · x = b h}) ⊆ affine hull S ∩ {x. a h · x = b h}
    apply (intro Int_greatest hull_mono Int_lower1)
    apply (metis affine_hull_eq affine_hyperplane hull_mono inf_le2)
    done
next
show affine hull S ∩ {x. a h · x = b h} ⊆ affine hull (S ∩ {x. a h · x = b h})
proof
  fix y
  assume ya: y ∈ affine hull S ∩ {y. a h · y = b h}
  obtain T where 0 < T
    and T: ⋀j. [j ∈ F; j ≠ h] ⟹ T * (a j · y - a j · w) ≤ b j - a j · w
  proof (cases F - {h} = {})
    case True then show ?thesis
      by (rule_tac T=1 in that) auto
  next
    case False
    then obtain h' where h': h' ∈ F - {h} by auto
    let ?body = (λj. if 0 < a j · y - a j · w
      then (b j - a j · w) / (a j · y - a j · w) else 1) ' (F - {h})
    define inff where inff = Inf ?body
    from ⟨finite F⟩ have finite ?body
      by blast
    moreover from h' have ?body ≠ {}
      by blast
    moreover have j > 0 if j ∈ ?body for j
    proof -
      from that obtain x where x ∈ F and x ≠ h and *: j =
        (if 0 < a x · y - a x · w
          then (b x - a x · w) / (a x · y - a x · w) else 1)
        by blast
      with awlt [of x] have a x · w < b x
        by simp
      with * show ?thesis
        by simp
    qed
    ultimately have 0 < inff
      by (simp_all add: finite_less_Inf_iff inff_def)
    moreover have inff * (a j · y - a j · w) ≤ b j - a j · w
      if j ∈ F j ≠ h for j
    proof (cases a j · w < a j · y)
      case True
      then have inff ≤ (b j - a j · w) / (a j · y - a j · w)
        unfolding inff_def
        using ⟨finite F⟩ by (auto intro: cInf_le_finite simp add: that split:
if_split_asm)
      then show ?thesis
        using ⟨0 < inff⟩ awlt [OF that] mult_strict_left_mono
        by (fastforce simp add: field_split_simps split: if_split_asm)
    qed
  qed

```

```

next
  case False
  with  $\langle 0 < \text{inff} \rangle$  have  $\text{inff} * (a \cdot j \cdot y - a \cdot j \cdot w) \leq 0$ 
    by (simp add: mult_le_0_iff)
  also have  $\dots < b \cdot j - a \cdot j \cdot w$ 
    by (simp add: awlt that)
  finally show ?thesis by simp
qed
ultimately show ?thesis
  by (blast intro: that)
qed
define C where  $C = (1 - T) *_{\mathbb{R}} w + T *_{\mathbb{R}} y$ 
have  $(1 - T) *_{\mathbb{R}} w + T *_{\mathbb{R}} y \in j$  if  $j \in F$  for  $j$ 
proof (cases  $j = h$ )
  case True
  have  $(1 - T) *_{\mathbb{R}} w + T *_{\mathbb{R}} y \in \{x. a \cdot h \cdot x \leq b \cdot h\}$ 
    using weq yaff by (auto simp: algebra_simps)
  with True faceq [OF that] show ?thesis by metis
next
  case False
  with T that have  $(1 - T) *_{\mathbb{R}} w + T *_{\mathbb{R}} y \in \{x. a \cdot j \cdot x \leq b \cdot j\}$ 
    by (simp add: algebra_simps)
  with faceq [OF that] show ?thesis by simp
qed
moreover have  $(1 - T) *_{\mathbb{R}} w + T *_{\mathbb{R}} y \in \text{affine hull } S$ 
  using yaff  $\langle w \in \text{affine hull } S \rangle$  affine_affine_hull affine_alt by blast
ultimately have  $C \in S$ 
  using seq by (force simp: C_def)
moreover have  $a \cdot h \cdot C = b \cdot h$ 
  using yaff by (force simp: C_def algebra_simps weq)
ultimately have caff:  $C \in \text{affine hull } (S \cap \{y. a \cdot h \cdot y = b \cdot h\})$ 
  by (simp add: hull_inc)
have waff:  $w \in \text{affine hull } (S \cap \{y. a \cdot h \cdot y = b \cdot h\})$ 
  using  $\langle w \in S \rangle$  weq by (blast intro: hull_inc)
have yeq:  $y = (1 - \text{inverse } T) *_{\mathbb{R}} w + C /_{\mathbb{R}} T$ 
  using  $\langle 0 < T \rangle$  by (simp add: C_def algebra_simps)
show  $y \in \text{affine hull } (S \cap \{y. a \cdot h \cdot y = b \cdot h\})$ 
  by (metis yeq affine_affine_hull [simplified affine_alt, rule_format, OF
waff caff])
qed
qed
ultimately have  $\text{aff\_dim } (\text{affine hull } (S \cap \{x. a \cdot h \cdot x = b \cdot h\})) = \text{aff\_dim } S - 1$ 
  using  $\langle b \cdot h < a \cdot h \cdot z \rangle$  zaff by (force simp: aff_dim_affine_Int_hyperplane)
then show ?thesis
  by (simp add: ne_faceS facet_of_def)
qed
show ?thesis
proof

```

```

    show  $\exists h. h \in F \wedge C = S \cap \{x. a \cdot h \cdot x = b\} \implies C \text{ facet\_of } S$ 
      using * by blast
  next
    assume  $C \text{ facet\_of } S$ 
    then have  $C \text{ face\_of } S \text{ convex } C \neq \{\}$  and affc:  $\text{aff\_dim } C = \text{aff\_dim } S$ 
  - 1
    by (auto simp: facet_of_def face_of_imp_convex)
    then obtain  $x$  where  $x \in \text{rel\_interior } C$ 
    by (force simp: rel_interior_eq_empty)
    then have  $x \in C$ 
    by (meson subsetD rel_interior_subset)
    then have  $x \in S$ 
    using  $\langle C \text{ facet\_of } S \rangle \text{ facet\_of\_imp\_subset}$  by blast
    have rels:  $\text{rel\_interior } S = \{x \in S. \forall h \in F. a \cdot h \cdot x < b\}$ 
    by (rule rel_interior_polyhedron_explicit [OF assms])
    have  $C \neq S$ 
    using  $\langle C \text{ facet\_of } S \rangle \text{ facet\_of\_irrefl}$  by blast
    then have  $x \notin \text{rel\_interior } S$ 
    by (metis IntI empty_iff  $\langle x \in C \rangle \langle C \neq S \rangle \langle C \text{ face\_of } S \rangle \text{ face\_of\_disjoint\_rel\_interior}$ )
    with rels  $\langle x \in S \rangle$  obtain  $i$  where  $i \in F$  and  $i: a \cdot i \cdot x \geq b$ 
    by force
    have  $x \in \{u. a \cdot i \cdot u \leq b\}$ 
    by (metis IntD2 InterE  $\langle i \in F \rangle \langle x \in S \rangle \text{ faceq seq}$ )
    then have  $a \cdot i \cdot x \leq b$  by simp
    then have  $a \cdot i \cdot x = b$  using  $i$  by auto
    have  $C \subseteq S \cap \{x. a \cdot i \cdot x = b\}$ 
    proof (rule subset_of_face_of [of _ S])
      show  $S \cap \{x. a \cdot i \cdot x = b\} \text{ face\_of } S$ 
      by (simp add: *  $\langle i \in F \rangle \text{ facet\_of\_imp\_face\_of}$ )
      show  $C \subseteq S$ 
      by (simp add:  $\langle C \text{ face\_of } S \rangle \text{ face\_of\_imp\_subset}$ )
      show  $S \cap \{x. a \cdot i \cdot x = b\} \cap \text{rel\_interior } C \neq \{\}$ 
      using  $\langle a \cdot i \cdot x = b \rangle \langle x \in S \rangle x$  by blast
    qed
    then have cface:  $C \text{ face\_of } (S \cap \{x. a \cdot i \cdot x = b\})$ 
    by (meson  $\langle C \text{ face\_of } S \rangle \text{ face\_of\_subset inf\_le1}$ )
    have con:  $\text{convex } (S \cap \{x. a \cdot i \cdot x = b\})$ 
    by (simp add:  $\langle \text{convex } S \rangle \text{ convex\_Int convex\_hyperplane}$ )
    show  $\exists h. h \in F \wedge C = S \cap \{x. a \cdot h \cdot x = b\}$ 
    apply (rule_tac  $x=i$  in exI)
    by (metis (no_types) *  $\langle i \in F \rangle \text{ affc facet\_of\_def less\_irrefl face\_of\_aff\_dim\_lt}$ 
    [OF con cface])
  qed
qed

```

```

lemma face_of_polyhedron_subset_explicit:
  fixes  $S :: 'a :: \text{euclidean\_space set}$ 
  assumes  $\text{finite } F$ 

```

```

    and seq:  $S = \text{affine hull } S \cap \bigcap F$ 
    and faceq:  $\bigwedge h. h \in F \implies a \cdot h \neq 0 \wedge h = \{x. a \cdot h \cdot x \leq b \cdot h\}$ 
    and psub:  $\bigwedge F'. F' \subset F \implies S \subset \text{affine hull } S \cap \bigcap F'$ 
    and C:  $C \text{ face\_of } S \text{ and } C \neq \{\} \implies C \neq S$ 
    obtains h where  $h \in F \wedge C \subseteq S \cap \{x. a \cdot h \cdot x = b \cdot h\}$ 
  proof -
    have  $C \subseteq S$  using  $\langle C \text{ face\_of } S \rangle$ 
    by (simp add: face_of_imp_subset)
    have polyhedron S
    by (metis  $\langle \text{finite } F \rangle$  faceq polyhedron_Int polyhedron_Inter polyhedron_affine_hull
    polyhedron_halfspace_le seq)
    then have convex S
    by (simp add: polyhedron_imp_convex)
    then have *:  $(S \cap \{x. a \cdot h \cdot x = b \cdot h\}) \text{ face\_of } S$  if  $h \in F$  for h
    using faceq seq face_of_Int_supporting_hyperplane_le that by fastforce
    have rel_interior  $C \neq \{\}$ 
    using  $C \neq \{\}$  face_of_imp_convex rel_interior_eq_empty by blast
    then obtain x where  $x \in \text{rel\_interior } C$  by auto
    have rels:  $\text{rel\_interior } S = \{x \in S. \forall h \in F. a \cdot h \cdot x < b \cdot h\}$ 
    by (rule rel_interior_polyhedron_explicit [OF  $\langle \text{finite } F \rangle$  seq faceq psub])
    then have xnot:  $x \notin \text{rel\_interior } S$ 
    by (metis IntI  $\langle x \in \text{rel\_interior } C \rangle C \neq S$  contra_subsetD empty_iff
    face_of_disjoint_rel_interior_subset)
    then have  $x \in S$ 
    using  $\langle C \subseteq S \rangle \langle x \in \text{rel\_interior } C \rangle \text{rel\_interior\_subset}$  by auto
    then have xint:  $x \in \bigcap F$ 
    using seq by blast
    have  $F \neq \{\}$  using assms
    by (metis affine_Int affine_Inter affine_affine_hull ex_in_conv face_of_affine_trivial)
    then obtain i where  $i \in F \wedge (a \cdot i \cdot x < b \cdot i)$ 
    using  $\langle x \in S \rangle \text{rels } xnot$  by auto
    with xint have  $a \cdot i \cdot x = b \cdot i$ 
    by (metis eq_iff mem_Collect_eq not_le Inter_iff faceq)
    have face:  $S \cap \{x. a \cdot i \cdot x = b \cdot i\} \text{ face\_of } S$ 
    by (simp add:  $\langle i \in F \rangle$ )
    show ?thesis
  proof
    show  $C \subseteq S \cap \{x. a \cdot i \cdot x = b \cdot i\}$ 
    using subset_of_face_of [OF face  $\langle C \subseteq S \rangle \langle a \cdot i \cdot x = b \cdot i \rangle \langle x \in \text{rel\_interior } C \rangle \langle x \in S \rangle$ ] by blast
    qed fact
  qed

```

Initial part of proof duplicates that above

proposition *face_of_polyhedron_explicit*:

fixes $S :: 'a :: \text{euclidean\_space}$ set

assumes *finite F*

and seq: $S = \text{affine hull } S \cap \bigcap F$

and faceq: $\bigwedge h. h \in F \implies a \cdot h \neq 0 \wedge h = \{x. a \cdot h \cdot x \leq b \cdot h\}$

```

    and psub:  $\bigwedge F'. F' \subseteq F \implies S \subseteq \text{affine hull } S \cap \bigcap F'$ 
    and C:  $C \text{ face\_of } S \text{ and } C \neq \{\}$   $C \neq S$ 
    shows  $C = \bigcap \{S \cap \{x. a \cdot h \cdot x = b\} \mid h. h \in F \wedge C \subseteq S \cap \{x. a \cdot h \cdot x = b\}\}$ 
  proof -
    let ?ab =  $\lambda h. \{x. a \cdot h \cdot x = b\}$ 
    have  $C \subseteq S$  using  $\langle C \text{ face\_of } S \rangle$ 
    by (simp add: face_of_imp_subset)
    have polyhedron S
    by (metis  $\langle \text{finite } F \rangle$  faceq polyhedron_Int polyhedron_Inter polyhedron_affine_hull
    polyhedron_halfspace_le seq)
    then have convex S
    by (simp add: polyhedron_imp_convex)
    then have *:  $(S \cap ?ab h) \text{ face\_of } S$  if  $h \in F$  for  $h$ 
    using faceq seq face_of_Int_supporting_hyperplane_le that by fastforce
    have rel_interior  $C \neq \{\}$ 
    using  $C \langle C \neq \{\} \rangle$  face_of_imp_convex rel_interior_eq_empty by blast
    then obtain z where z:  $z \in \text{rel\_interior } C$  by auto
    have rels:  $\text{rel\_interior } S = \{z \in S. \forall h \in F. a \cdot h \cdot z < b\}$ 
    by (rule rel_interior_polyhedron_explicit [OF  $\langle \text{finite } F \rangle$  seq faceq psub])
    then have xnot:  $z \notin \text{rel\_interior } S$ 
    by (metis IntI  $\langle z \in \text{rel\_interior } C \rangle$   $C \langle C \neq S \rangle$  contra_subsetD empty_iff
    face_of_disjoint_rel_interior rel_interior_subset)
    then have  $z \in S$ 
    using  $\langle C \subseteq S \rangle$   $\langle z \in \text{rel\_interior } C \rangle$  rel_interior_subset by auto
    with seq have xint:  $z \in \bigcap F$  by blast
    have open  $(\bigcap_{h \in \{h \in F. a \cdot h \cdot z < b\}} \{w. a \cdot h \cdot w < b\})$ 
    by (auto simp:  $\langle \text{finite } F \rangle$  open_halfspace_lt open_INT)
    then obtain e where  $0 < e$ 
    ball  $z \in \bigcap_{h \in \{h \in F. a \cdot h \cdot z < b\}} \{w. a \cdot h \cdot w < b\}$ 
    by (auto intro: openE [of _ z])
    then have e:  $\bigwedge h. [h \in F; a \cdot h \cdot z < b] \implies \text{ball } z \in \{w. a \cdot h \cdot w < b\}$ 
    by blast
    have  $C \subseteq (S \cap ?ab h) \longleftrightarrow z \in S \cap ?ab h$  if  $h \in F$  for  $h$ 
  proof
    show  $z \in S \cap ?ab h \implies C \subseteq S \cap ?ab h$ 
    by (metis * Collect_cong IntI  $\langle C \subseteq S \rangle$  empty_iff subset_of_face_of that z)
  next
    show  $C \subseteq S \cap ?ab h \implies z \in S \cap ?ab h$ 
    using  $\langle z \in \text{rel\_interior } C \rangle$  rel_interior_subset by force
  qed
  then have **:  $\{S \cap ?ab h \mid h. h \in F \wedge C \subseteq S \wedge C \subseteq ?ab h\} =$ 
     $\{S \cap ?ab h \mid h. h \in F \wedge z \in S \cap ?ab h\}$ 
    by blast
  have bsub:  $\text{ball } z \in \text{affine hull } \bigcap \{S \cap ?ab h \mid h. h \in F \wedge a \cdot h \cdot z = b\}$ 
     $\subseteq \text{affine hull } S \cap \bigcap F \cap \bigcap \{?ab h \mid h. h \in F \wedge a \cdot h \cdot z = b\}$ 
    if  $i \in F$  and  $i: a \cdot i \cdot z = b$  for  $i$ 
  proof -
    have sub:  $\text{ball } z \in \bigcap \{?ab h \mid h. h \in F \wedge a \cdot h \cdot z = b\} \subseteq j$ 

```

```

      if  $j \in F$  for  $j$ 
    proof -
      have  $a \cdot j \cdot z \leq b \cdot j$  using faceq that xint by auto
      then consider  $a \cdot j \cdot z < b \cdot j \mid a \cdot j \cdot z = b \cdot j$  by linarith
      then have  $\exists G. G \in \{?ab \ h \mid h. h \in F \wedge a \cdot h \cdot z = b \cdot h\} \wedge ball \ z \ e \cap G \subseteq j$ 
      proof cases
        assume  $a \cdot j \cdot z < b \cdot j$ 
        then have  $ball \ z \ e \cap \{x. a \cdot i \cdot x = b \cdot i\} \subseteq j$ 
          using  $e \ [OF \ \langle j \in F \rangle]$  faceq that
          by (fastforce simp: ball_def)
        then show ?thesis
          by (rule_tac x={x. a · i · x = b · i} in exI) (force simp: ⟨i ∈ F⟩ i)
      next
        assume  $eq: a \cdot j \cdot z = b \cdot j$ 
        with faceq that show ?thesis
          by (rule_tac x={x. a · j · x = b · j} in exI) (fastforce simp add: ⟨j ∈ F⟩)
      qed
      then show ?thesis by blast
    qed
  have 1: affine hull  $\bigcap \{S \cap ?ab \ h \mid h. h \in F \wedge a \cdot h \cdot z = b \cdot h\} \subseteq \text{affine hull } S$ 
    using that ⟨z ∈ S⟩ by (intro hull_mono) auto
  have 2: affine hull  $\bigcap \{S \cap ?ab \ h \mid h. h \in F \wedge a \cdot h \cdot z = b \cdot h\}$ 
     $\subseteq \bigcap \{?ab \ h \mid h. h \in F \wedge a \cdot h \cdot z = b \cdot h\}$ 
    by (rule hull_minimal) (auto intro: affine_hyperplane)
  have 3:  $ball \ z \ e \cap \bigcap \{?ab \ h \mid h. h \in F \wedge a \cdot h \cdot z = b \cdot h\} \subseteq \bigcap F$ 
    by (iprover intro: sub Inter_greatest)
  have *:  $\llbracket A \subseteq (B :: 'a \text{ set}); A \subseteq C; E \cap C \subseteq D \rrbracket \implies E \cap A \subseteq (B \cap D) \cap C$ 
    for  $A \ B \ C \ D \ E$  by blast
  show ?thesis by (intro * 1 2 3)
qed
have  $\exists h. h \in F \wedge C \subseteq ?ab \ h$ 
  using assms
  by (metis face_of_polyhedron_subset_explicit [OF ⟨finite F⟩ seq faceq psub]
le_inf_iff)
then have fac:  $\bigcap \{S \cap ?ab \ h \mid h. h \in F \wedge C \subseteq S \cap ?ab \ h\}$  face_of S
  using * by (force simp: ⟨C ⊆ S⟩ intro: face_of_Inter)
have red:  $(\bigwedge a. P \ a \implies T \subseteq S \cap \bigcap \{F \ X \mid X. P \ X\}) \implies T \subseteq \bigcap \{S \cap F \ X \mid X :: 'a \text{ set}. P \ X\}$  for  $P \ T \ F$ 
  by blast
have  $ball \ z \ e \cap \text{affine hull } \bigcap \{S \cap ?ab \ h \mid h. h \in F \wedge a \cdot h \cdot z = b \cdot h\}$ 
   $\subseteq \bigcap \{S \cap ?ab \ h \mid h. h \in F \wedge a \cdot h \cdot z = b \cdot h\}$ 
  by (rule red) (metis seq bsub)
with  $\langle 0 < e \rangle$  have zinrel:  $z \in \text{rel\_interior}$ 
   $(\bigcap \{S \cap ?ab \ h \mid h. h \in F \wedge z \in S \wedge a \cdot h \cdot z = b \cdot h\})$ 
  by (auto simp: mem_rel_interior_ball ⟨z ∈ S⟩)
show ?thesis
  using z zinrel
  by (intro face_of_eq [OF C fac]) (force simp: **)
qed

```


6.39.12 More general corollaries from the explicit representation

corollary *facet_of_polyhedron*:

assumes *polyhedron S* **and** *C facet_of S*

obtains *a b* **where** $a \neq 0$ $S \subseteq \{x. a \cdot x \leq b\}$ $C = S \cap \{x. a \cdot x = b\}$

proof –

obtain *F* **where** *finite F* **and** *seq: S = affine hull S $\cap$ $\bigcap$ F*

and *faces: $\bigwedge h. h \in F \implies \exists a b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}$*

and *min: $\bigwedge F'. F' \subset F \implies S \subset (\text{affine hull } S) \cap \bigcap F'$*

using *assms* **by** (*simp add: polyhedron_Int_affine_minimal*) *meson*

then obtain *a b* **where** *ab: $\bigwedge h. h \in F \implies a h \neq 0 \wedge h = \{x. a h \cdot x \leq b h\}$*

by *metis*

obtain *i* **where** $i \in F$ **and** *C: C = S $\cap$ $\{x. a i \cdot x = b i\}$*

using *facet_of_polyhedron_explicit [OF $\langle$ finite F $\rangle$ seq ab min]* *assms*

by *force*

moreover have *ssub: S $\subseteq$ $\{x. a i \cdot x \leq b i\}$*

using $\langle i \in F \rangle$ *ab* **by** (*subst seq*) *auto*

ultimately show *?thesis*

by (*rule_tac a = a i and b = b i in that*) (*simp_all add: ab*)

qed

corollary *face_of_polyhedron*:

assumes *polyhedron S* **and** *C face_of S* **and** $C \neq \{\}$ **and** $C \neq S$

shows $C = \bigcap \{F. F \text{ facet\_of } S \wedge C \subseteq F\}$

proof –

obtain *F* **where** *finite F* **and** *seq: S = affine hull S $\cap$ $\bigcap$ F*

and *faces: $\bigwedge h. h \in F \implies \exists a b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}$*

and *min: $\bigwedge F'. F' \subset F \implies S \subset (\text{affine hull } S) \cap \bigcap F'$*

using *assms* **by** (*simp add: polyhedron_Int_affine_minimal*) *meson*

then obtain *a b* **where** *ab: $\bigwedge h. h \in F \implies a h \neq 0 \wedge h = \{x. a h \cdot x \leq b h\}$*

by *metis*

show *?thesis*

apply (*subst face_of_polyhedron_explicit [OF $\langle$ finite F $\rangle$ seq ab min]*)

apply (*auto simp: assms facet_of_polyhedron_explicit [OF $\langle$ finite F $\rangle$ seq ab min]* *cong: Collect_cong*)

done

qed

lemma *face_of_polyhedron_subset_facet*:

assumes *polyhedron S* **and** *C face_of S* **and** $C \neq \{\}$ **and** $C \neq S$

obtains *F* **where** $F \text{ facet\_of } S$ $C \subseteq F$

using *face_of_polyhedron assms*

by (*metis (no_types, lifting) Inf_greatest antisym_conv face_of_imp_subset mem_Collect_eq*)

lemma *exposed_face_of_polyhedron*:

assumes *polyhedron S*

shows $F \text{ exposed\_face\_of } S \iff F \text{ face\_of } S$

```

proof
  show  $F \text{ exposed\_face\_of } S \implies F \text{ face\_of } S$ 
    by (simp add: exposed_face_of_def)
next
  assume  $F \text{ face\_of } S$ 
  show  $F \text{ exposed\_face\_of } S$ 
  proof (cases  $F = \{\} \vee F = S$ )
    case True then show ?thesis
      using  $\langle F \text{ face\_of } S \rangle \text{ exposed\_face\_of }$  by blast
  next
  case False
  then have  $\{g. g \text{ facet\_of } S \wedge F \subseteq g\} \neq \{\}$ 
  by (metis Collect_empty_eq_bot  $\langle F \text{ face\_of } S \rangle \text{ assms empty\_def face\_of\_polyhedron\_subset\_facet}$ )
  moreover have  $\bigwedge T. \llbracket T \text{ facet\_of } S; F \subseteq T \rrbracket \implies T \text{ exposed\_face\_of } S$ 
  by (metis assms exposed_face_of_facet_of_imp_face_of_facet_of_polyhedron)
  ultimately have  $\bigcap \{G. G \text{ facet\_of } S \wedge F \subseteq G\} \text{ exposed\_face\_of } S$ 
  by (metis (no_types, lifting) mem_Collect_eq exposed_face_of_Inter)
  then show ?thesis
    using False  $\langle F \text{ face\_of } S \rangle \text{ assms face\_of\_polyhedron }$  by fastforce
qed
qed

lemma face_of_polyhedron_polyhedron:
  fixes  $S :: 'a :: \text{euclidean\_space set}$ 
  assumes  $\text{polyhedron } S$   $c \text{ face\_of } S$  shows  $\text{polyhedron } c$ 
by (metis assms face_of_imp_eq_affine_Int polyhedron_Int polyhedron_affine_hull polyhedron_imp_convex)

lemma finite_polyhedron_faces:
  fixes  $S :: 'a :: \text{euclidean\_space set}$ 
  assumes  $\text{polyhedron } S$ 
  shows finite  $\{F. F \text{ face\_of } S\}$ 
proof –
  obtain  $F$  where finite  $F$  and seq:  $S = \text{affine hull } S \cap \bigcap F$ 
    and faces:  $\bigwedge h. h \in F \implies \exists a b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}$ 
    and min:  $\bigwedge F'. F' \subset F \implies S \subset (\text{affine hull } S) \cap \bigcap F'$ 
  using assms by (simp add: polyhedron_Int_affine_minimal) meson
  then obtain  $a b$  where ab:  $\bigwedge h. h \in F \implies a h \neq 0 \wedge h = \{x. a h \cdot x \leq b h\}$ 
  by metis
  have finite  $\{\bigcap \{S \cap \{x. a h \cdot x = b h\} \mid h. h \in F'\} \mid F'. F' \in \text{Pow } F\}$ 
  by (simp add: finite_Pow)
  moreover have  $\{F. F \text{ face\_of } S\} - \{\{\}, S\} \subseteq \{\bigcap \{S \cap \{x. a h \cdot x = b h\} \mid h. h \in F'\} \mid F'. F' \in \text{Pow } F\}$ 
  apply clarify
  apply (rename_tac c)
  apply (drule face_of_polyhedron_explicit [OF  $\langle \text{finite } F \rangle \text{ seq } ab \text{ min, simplified}$ ], simp_all)
  apply (rule_tac  $x = \{h \in F. c \subseteq S \cap \{x. a h \cdot x = b h\}\}$  in exI, auto)
  done

```

```

ultimately show ?thesis
  by (meson finite.emptyI finite.insertI finite_Diff2 finite_subset)
qed

lemma finite_polyhedron_exposed_faces:
  polyhedron S  $\implies$  finite {F. F exposed_face_of S}
using exposed_face_of_polyhedron finite_polyhedron_faces by fastforce

lemma finite_polyhedron_extreme_points:
  fixes S :: 'a :: euclidean_space set
  assumes polyhedron S shows finite {v. v extreme_point_of S}
proof -
  have finite {v. {v} face_of S}
  using assms by (intro finite_subset [OF _ finite_vimageI [OF finite_polyhedron_faces]],
auto)
  then show ?thesis
    by (simp add: face_of_singleton)
qed

lemma finite_polyhedron_facets:
  fixes S :: 'a :: euclidean_space set
  shows polyhedron S  $\implies$  finite {F. F facet_of S}
  unfolding facet_of_def
  by (blast intro: finite_subset [OF _ finite_polyhedron_faces])

proposition rel_interior_of_polyhedron:
  fixes S :: 'a :: euclidean_space set
  assumes polyhedron S
  shows rel_interior S = S -  $\bigcup$  {F. F facet_of S}
proof -
  obtain F where finite F and seq: S = affine hull S  $\cap$   $\bigcap$  F
    and faces:  $\bigwedge h. h \in F \implies \exists a b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}$ 
    and min:  $\bigwedge F'. F' \subset F \implies S \subset (\text{affine hull } S) \cap \bigcap F'$ 
  using assms by (simp add: polyhedron_Int_affine_minimal) meson
  then obtain a b where ab:  $\bigwedge h. h \in F \implies a h \neq 0 \wedge h = \{x. a h \cdot x \leq b h\}$ 
    by metis
  have facet: (c facet_of S)  $\longleftrightarrow$  ( $\exists h. h \in F \wedge c = S \cap \{x. a h \cdot x = b h\}$ ) for c
    by (rule facet_of_polyhedron_explicit [OF <finite F> seq ab min])
  have rel: rel_interior S = {x  $\in$  S.  $\forall h \in F. a h \cdot x < b h$ }
    by (rule rel_interior_polyhedron_explicit [OF <finite F> seq ab min])
  have a h  $\cdot$  x < b h if x  $\in$  S h  $\in$  F and xnot: x  $\notin$   $\bigcup$  {F. F facet_of S} for x h
  proof -
    have x  $\in$   $\bigcap$  F using seq that by force
    with <h  $\in$  F> ab have a h  $\cdot$  x  $\leq$  b h by auto
    then consider a h  $\cdot$  x < b h | a h  $\cdot$  x = b h by linarith
    then show ?thesis
  proof cases
    case 1 then show ?thesis .

```

```

next
case 2
have Collect (( $\in$ )  $x$ )  $\notin$  Collect (( $\in$ ) ( $\bigcup \{A. A \text{ facet\_of } S\}$ ))
  using xnot by fastforce
then have  $F \notin$  Collect (( $\in$ )  $h$ )
  using 2  $\langle x \in S \rangle$  facet by blast
with 2 that  $\langle x \in \bigcap F \rangle$  show ?thesis
  by blast
qed
qed
moreover have  $\exists h \in F. a \cdot h \cdot x \geq b \cdot h$  if  $x \in \bigcup \{F. F \text{ facet\_of } S\}$  for  $x$ 
  using that by (force simp: facet)
ultimately show ?thesis
  by (force simp: rel)
qed

```

```

lemma rel_boundary_of_polyhedron:
  fixes  $S :: 'a :: \text{euclidean\_space}$  set
  assumes polyhedron  $S$ 
  shows  $S - \text{rel\_interior } S = \bigcup \{F. F \text{ facet\_of } S\}$ 
using facet_of_imp_subset by (fastforce simp add: rel_interior_of_polyhedron
  assms)

```

```

lemma rel_frontier_of_polyhedron:
  fixes  $S :: 'a :: \text{euclidean\_space}$  set
  assumes polyhedron  $S$ 
  shows  $\text{rel\_frontier } S = \bigcup \{F. F \text{ facet\_of } S\}$ 
by (simp add: assms rel_frontier_def polyhedron_imp_closed rel_boundary_of_polyhedron)

```

```

lemma rel_frontier_of_polyhedron_alt:
  fixes  $S :: 'a :: \text{euclidean\_space}$  set
  assumes polyhedron  $S$ 
  shows  $\text{rel\_frontier } S = \bigcup \{F. F \text{ face\_of } S \wedge F \neq S\}$ 
proof
  show  $\text{rel\_frontier } S \subseteq \bigcup \{F. F \text{ face\_of } S \wedge F \neq S\}$ 
    by (force simp: rel_frontier_of_polyhedron facet_of_def assms)
qed (use face_of_subset_rel_frontier in fastforce)

```

A characterization of polyhedra as having finitely many faces

```

proposition polyhedron_eq_finite_exposed_faces:
  fixes  $S :: 'a :: \text{euclidean\_space}$  set
  shows  $\text{polyhedron } S \longleftrightarrow \text{closed } S \wedge \text{convex } S \wedge \text{finite } \{F. F \text{ exposed\_face\_of } S\}$ 
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (auto simp: polyhedron_imp_closed polyhedron_imp_convex finite_polyhedron_exposed_faces)
next
  assume ?rhs

```

```

then have closed  $S$  convex  $S$  and fin: finite  $\{F. F \text{ exposed\_face\_of } S\}$  by auto
show ?lhs
proof (cases  $S = \{\}$ )
  case True then show ?thesis by auto
next
  case False
  define  $F$  where  $F = \{h. h \text{ exposed\_face\_of } S \wedge h \neq \{\} \wedge h \neq S\}$ 
  have finite  $F$  by (simp add: fin F_def)
  have hface:  $h \text{ face\_of } S$ 
    and  $\exists a b. a \neq 0 \wedge S \subseteq \{x. a \cdot x \leq b\} \wedge h = S \cap \{x. a \cdot x = b\}$ 
    if  $h \in F$  for  $h$ 
    using exposed_face_of F_def that by blast+
  then obtain  $a b$  where ab:
     $\wedge h. h \in F \implies a h \neq 0 \wedge S \subseteq \{x. a h \cdot x \leq b h\} \wedge h = S \cap \{x. a h \cdot x = b h\}$ 
  by metis
  have *: False
    if paff:  $p \in \text{affine hull } S$  and  $p \notin S$ 
    and pint:  $p \in \bigcap \{\{x. a h \cdot x \leq b h\} \mid h. h \in F\}$  for  $p$ 
  proof –
    have rel_interior  $S \neq \{\}$ 
      by (simp add:  $\langle S \neq \{\} \rangle \langle \text{convex } S \rangle \text{rel\_interior\_eq\_empty}$ )
    then obtain  $c$  where  $c: c \in \text{rel\_interior } S$  by auto
    with rel_interior_subset have  $c \in S$  by blast
    have ccp: closed_segment  $c p \subseteq \text{affine hull } S$ 
      by (meson affine_affine_hull affine_imp_convex c closed_segment_subset
hull_subset paff rel_interior_subset subsetCE)
    have oS: openin (top_of_set (closed_segment  $c p$ )) (closed_segment  $c p \cap \text{rel\_interior } S$ )
      by (force simp: openin_rel_interior openin_Int intro: openin_subtopology_Int_subset
[OF _ ccp])
    obtain  $x$  where xcl:  $x \in \text{closed\_segment } c p$  and  $x \in S$  and xnot:  $x \notin \text{rel\_interior } S$ 
      using connected_openin [of closed_segment c p]
      apply simp
      apply (drule_tac  $x = \text{closed\_segment } c p \cap \text{rel\_interior } S$  in spec)
      apply (drule mp [OF _ oS])
      apply (drule_tac  $x = \text{closed\_segment } c p \cap (- S)$  in spec)
      using rel_interior_subset  $\langle \text{closed } S \rangle c \langle p \notin S \rangle$  apply blast
    done
  then obtain  $\mu$  where  $0 \leq \mu \leq 1$  and req:  $x = (1 - \mu) *_R c + \mu *_R p$ 
    by (auto simp: in_segment)
  show False
proof (cases  $\mu = 0 \vee \mu = 1$ )
  case True with req  $c$  xnot  $\langle x \in S \rangle \langle p \notin S \rangle$ 
    show False by auto
next
  case False
  then have xos:  $x \in \text{open\_segment } c p$ 

```

```

    using ⟨x ∈ S⟩ c open_segment_def that(2) xcl xnot by auto
  have xclo: x ∈ closure S
    using ⟨x ∈ S⟩ closure_subset by blast
  obtain d where d ≠ 0
    and dle:  $\bigwedge y. y \in \text{closure } S \implies d \cdot x \leq d \cdot y$ 
    and dless:  $\bigwedge y. y \in \text{rel\_interior } S \implies d \cdot x < d \cdot y$ 
    by (metis supporting_hyperplane_relative_frontier [OF ⟨convex S⟩ xclo
xnot])
  have sex:  $S \cap \{y. d \cdot y = d \cdot x\}$  exposed_face_of S
  by (simp add: ⟨closed S⟩ dle exposed_face_of_Int_supporting_hyperplane_ge
[OF ⟨convex S⟩])
  have sne:  $S \cap \{y. d \cdot y = d \cdot x\} \neq \{\}$ 
    using ⟨x ∈ S⟩ by blast
  have sns:  $S \cap \{y. d \cdot y = d \cdot x\} \neq S$ 
    by (metis (mono_tags) Int_Collect c subsetD dless not_le order_refl
rel_interior_subset)
  obtain h where h ∈ F x ∈ h
    using F_def ⟨x ∈ S⟩ sex sns by blast
  have abface:  $\{y. a \cdot h \cdot y = b \cdot h\}$  face_of {y. a · h · y ≤ b · h}
    using hyperplane_face_of_halfspace_le by blast
  then have c ∈ h
    using face_ofD [OF abface xos] ⟨c ∈ S⟩ ⟨h ∈ F⟩ ab pint ⟨x ∈ h⟩ by blast
  with c have h ∩ rel_interior S ≠ { } by blast
  then show False
    using ⟨h ∈ F⟩ F_def face_of_disjoint_rel_interior hface by auto
qed
qed
have S ⊆ affine_hull S ∩  $\bigcap \{\{x. a \cdot h \cdot x \leq b \cdot h\} \mid h. h \in F\}$ 
  using ab by (auto simp: hull_subset)
moreover have affine_hull S ∩  $\bigcap \{\{x. a \cdot h \cdot x \leq b \cdot h\} \mid h. h \in F\} \subseteq S$ 
  using * by blast
ultimately have S = affine_hull S ∩  $\bigcap \{\{x. a \cdot h \cdot x \leq b \cdot h\} \mid h. h \in F\}$  ..
then show ?thesis
  apply (rule ssubst)
  apply (force intro: polyhedron_affine_hull polyhedron_halfspace_le simp:
⟨finite F⟩)
done
qed
qed

corollary polyhedron_eq_finite_faces:
  fixes S :: 'a :: euclidean_space set
  shows polyhedron S  $\longleftrightarrow$  closed S ∧ convex S ∧ finite {F. F face_of S}
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (simp add: finite_polyhedron_faces polyhedron_imp_closed polyhedron_imp_convex)
next

```

```

  assume ?rhs
  then show ?lhs
    by (force simp: polyhedron_eq_finite_exposed_faces exposed_face_of intro:
finite_subset)
qed

```

```

lemma polyhedron_linear_image_eq:
  fixes h :: 'a :: euclidean_space  $\Rightarrow$  'b :: euclidean_space
  assumes linear h bij h
  shows polyhedron (h ` S)  $\longleftrightarrow$  polyhedron S
proof -
  have *: {f. P f} = (image h) ` {f. P (h ` f)} for P
  apply safe
  apply (rule_tac x=inv h ` x in image_eqI)
  apply (auto simp: <bij h> bij_is_surj image_f_inv_f)
  done
  have inj h using bij_is_inj assms by blast
  then have injim: inj_on ((`) h) A for A
  by (simp add: inj_on_def inj_image_eq_iff)
  show ?thesis
    using <linear h> <inj h>
    apply (simp add: polyhedron_eq_finite_faces closed_injective_linear_image_eq)
    apply (simp add: * face_of_linear_image [of h _ S, symmetric] finite_image_iff injim)
    done
qed

```

```

lemma polyhedron_negations:
  fixes S :: 'a :: euclidean_space set
  shows polyhedron S  $\implies$  polyhedron (image uminus S)
  by (subst polyhedron_linear_image_eq) (auto simp: bij_uminus intro!: linear_uminus)

```

6.39.13 Relation between polytopes and polyhedra

```

proposition polytope_eq_bounded_polyhedron:
  fixes S :: 'a :: euclidean_space set
  shows polytope S  $\longleftrightarrow$  polyhedron S  $\wedge$  bounded S
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (simp add: finite_polytope_faces polyhedron_eq_finite_faces
polytope_imp_closed polytope_imp_convex polytope_imp_bounded)
next
  assume R: ?rhs
  then have finite {v. v extreme_point_of S}
    by (simp add: finite_polyhedron_extreme_points)
  moreover have S = convex hull {v. v extreme_point_of S}
  using R by (simp add: Krein_Milman_Minkowski compact_eq_bounded_closed)

```

```

polyhedron_imp_closed polyhedron_imp_convex)
  ultimately show ?lhs
    unfolding polytope_def by blast
qed

```

```

lemma polytope_Int:
  fixes  $S :: 'a :: euclidean\_space$  set
  shows  $\llbracket \text{polytope } S; \text{polytope } T \rrbracket \implies \text{polytope}(S \cap T)$ 
by (simp add: polytope_eq_bounded_polyhedron bounded_Int)

```

```

lemma polytope_Int_polyhedron:
  fixes  $S :: 'a :: euclidean\_space$  set
  shows  $\llbracket \text{polytope } S; \text{polyhedron } T \rrbracket \implies \text{polytope}(S \cap T)$ 
by (simp add: bounded_Int polytope_eq_bounded_polyhedron)

```

```

lemma polyhedron_Int_polytope:
  fixes  $S :: 'a :: euclidean\_space$  set
  shows  $\llbracket \text{polyhedron } S; \text{polytope } T \rrbracket \implies \text{polytope}(S \cap T)$ 
by (simp add: bounded_Int polytope_eq_bounded_polyhedron)

```

```

lemma polytope_imp_polyhedron:
  fixes  $S :: 'a :: euclidean\_space$  set
  shows  $\text{polytope } S \implies \text{polyhedron } S$ 
by (simp add: polytope_eq_bounded_polyhedron)

```

```

lemma polytope_facet_exists:
  fixes  $p :: 'a :: euclidean\_space$  set
  assumes  $\text{polytope } p$   $0 < \text{aff\_dim } p$ 
  obtains  $F$  where  $F \text{ facet\_of } p$ 
proof (cases  $p = \{\}$ )
  case True with assms show ?thesis by auto
next
  case False
  then obtain  $v$  where  $v \text{ extreme\_point\_of } p$ 
    using extreme_point_exists_convex
    by (blast intro:  $\langle \text{polytope } p \rangle \text{ polytope\_imp\_compact polytope\_imp\_convex}$ )
  then
    show ?thesis
    by (metis face_of_polyhedron_subset_facet polytope_imp_polyhedron aff_dim_singleton_not_in_conv assms face_of_singleton less_irrefl singletonI that)
qed

```

```

lemma polyhedron_interval [iff]:  $\text{polyhedron}(\text{cbox } a \ b)$ 
by (metis polytope_imp_polyhedron polytope_interval)

```

```

lemma polyhedron_convex_hull:
  fixes  $S :: 'a :: euclidean\_space$  set
  shows  $\text{finite } S \implies \text{polyhedron}(\text{convex\_hull } S)$ 

```


by (simp add: polytope_convex_hull polytope_imp_polyhedron)

6.39.14 Relative and absolute frontier of a polytope

lemma *rel_boundary_of_convex_hull*:

fixes $S :: 'a::euclidean\_space\ set$
assumes $\neg\ affine\_dependent\ S$
shows $(convex\ hull\ S) - rel\_interior(convex\ hull\ S) = (\bigcup_{a \in S} convex\ hull\ (S - \{a\}))$

proof –

have *finite S* **by** (metis *assms* *aff_independent_finite*)

then consider $card\ S = 0 \mid card\ S = 1 \mid 2 \leq card\ S$ **by** *arith*

then show *?thesis*

proof *cases*

case 1 **then have** $S = \{\}$ **by** (simp add: $\langle finite\ S \rangle$)

then show *?thesis* **by** *simp*

next

case 2 **show** *?thesis*

by (auto intro: *card_1_singletonE* [*OF* $\langle card\ S = 1 \rangle$])

next

case 3

with *assms* **show** *?thesis*

by (auto simp: *polyhedron_convex_hull* *rel_boundary_of_polyhedron* *facet_of_convex_hull* *affine_independent_finite*)

qed

qed

proposition *frontier_of_convex_hull*:

fixes $S :: 'a::euclidean\_space\ set$

assumes $card\ S = Suc\ (DIM('a))$

shows $frontier(convex\ hull\ S) = \bigcup \{convex\ hull\ (S - \{a\}) \mid a. a \in S\}$

proof (*cases* *affine_dependent S*)

case *True*

have [*iff*]: *finite S*

using *assms* **using** *card.infinite* **by** *force*

then have *ccs*: *closed* (*convex hull S*)

by (simp add: *compact_imp_closed* *finite_imp_compact_convex_hull*)

{ fix $x\ T$

assume $int\ (card\ T) \leq aff\_dim\ S + 1$ *finite T* $T \subseteq S$ $x \in convex\ hull\ T$

then have $S \neq T$

using *True* $\langle finite\ S \rangle$ *aff_dim_le_card* *affine_independent_iff_card* **by** *fastforce*

then obtain a **where** $a \in S$ $a \notin T$

using $\langle T \subseteq S \rangle$ **by** *blast*

then have $\exists y \in S. x \in convex\ hull\ (S - \{y\})$

using *True* *affine_independent_iff_card* [*of S*]

by (metis (*no_types*, *opaque_lifting*) *Diff_eq_empty_iff* *Diff_insert0* $\langle a \notin T \rangle$ $\langle T \subseteq S \rangle$ $\langle x \in convex\ hull\ T \rangle$ *hull_mono* *insert_Diff_single* *subsetCE*)

} note $*$ = *this*

```

have 1: convex hull  $S \subseteq (\bigcup_{a \in S} \text{convex hull } (S - \{a\}))$ 
  by (subst caratheodory_aff_dim) (blast dest: *)
have 2:  $\bigcup ((\lambda a. \text{convex hull } (S - \{a\})) \text{ ` } S) \subseteq \text{convex hull } S$ 
  by (rule Union_least) (metis (no_types, lifting) Diff_subset hull_mono
imageE)
show ?thesis using True
  apply (simp add: segment_convex_hull_frontier_def)
  using interior_convex_hull_eq_empty [OF assms]
  apply (simp add: closure_closed [OF ccs])
  using 1 2 by auto
next
  case False
  then have frontier (convex hull S) = closure (convex hull S) - interior (convex
hull S)
    by (simp add: rel_boundary_of_convex_hull frontier_def)
  also have  $\dots = (\text{convex hull } S) - \text{rel\_interior}(\text{convex hull } S)$ 
    by (metis False aff_independent_finite assms closure_convex_hull finite_imp_compact_convex_hull
hull_hull interior_convex_hull_eq_empty rel_interior_nonempty_interior)
  also have  $\dots = \bigcup \{\text{convex hull } (S - \{a\}) \mid a. a \in S\}$ 
    proof -
      have convex hull S - rel_interior (convex hull S) = rel_frontier (convex hull
S)
        by (simp add: False aff_independent_finite polyhedron_convex_hull rel_boundary_of_polyhedron
rel_frontier_of_polyhedron)
      then show ?thesis
        by (simp add: False rel_frontier_convex_hull_cases)
    qed
  finally show ?thesis .
qed

```

6.39.15 Special case of a triangle

proposition *frontier_of_triangle*:

fixes $a :: 'a::\text{euclidean\_space}$

assumes $\text{DIM}(a) = 2$

shows $\text{frontier}(\text{convex hull } \{a, b, c\}) = \text{closed\_segment } a \ b \cup \text{closed\_segment } b$
 $c \cup \text{closed\_segment } c \ a$

(**is** ?lhs = ?rhs)

proof (*cases b = a $\vee$ c = a $\vee$ c = b*)

case True **then show** ?thesis

by (*auto simp: assms segment_convex_hull frontier_def empty_interior_convex_hull*
insert_commute card_insert_le_m1 hull_inc insert_absorb)

next

case False **then have** [*simp*]: $\text{card } \{a, b, c\} = \text{Suc } (\text{DIM}(a))$

by (*simp add: card.insert_remove Set.insert_Diff_if assms*)

show ?thesis

proof

show ?lhs $\subseteq$?rhs

using False

```

    by (force simp: segment_convex_hull frontier_of_convex_hull insert_Diff_if
insert_commute split: if_split_asm)
  show ?rhs  $\subseteq$  ?lhs
    using False
    apply (simp add: frontier_of_convex_hull segment_convex_hull)
    apply (intro conjI subsetI)
    apply (rule_tac X=convex hull {a,b} in UnionI; force simp: Set.insert_Diff_if)
    apply (rule_tac X=convex hull {b,c} in UnionI; force)
    apply (rule_tac X=convex hull {a,c} in UnionI; force simp: insert_commute
Set.insert_Diff_if)
  done
qed
qed

corollary inside_of_triangle:
  fixes a :: 'a::euclidean_space
  assumes DIM('a) = 2
  shows inside (closed_segment a b  $\cup$  closed_segment b c  $\cup$  closed_segment c
a) = interior(convex hull {a,b,c})
by (metis assms frontier_of_triangle bounded_empty bounded_insert convex_convex_hull
inside_frontier_eq_interior bounded_convex_hull)

corollary interior_of_triangle:
  fixes a :: 'a::euclidean_space
  assumes DIM('a) = 2
  shows interior(convex hull {a,b,c}) =
    convex hull {a,b,c} - (closed_segment a b  $\cup$  closed_segment b c  $\cup$ 
closed_segment c a)
  using interior_subset
  by (force simp: frontier_of_triangle [OF assms, symmetric] frontier_def Diff_Diff_Int)

```

6.39.16 Subdividing a cell complex

```

lemma subdivide_interval:
  fixes x::real
  assumes a < |x - y| 0 < a
  obtains n where n  $\in \mathbb{Z}$  x < n * a  $\wedge$  n * a < y  $\vee$  y < n * a  $\wedge$  n * a < x
proof -
  consider a + x < y | a + y < x
  using assms by linarith
  then show ?thesis
  proof cases
    case 1
    let ?n = of_int (floor (x/a)) + 1
    have x: x < ?n * a
      by (meson 0 < a divide_less_eq floor_eq_iff)
    have ?n * a  $\leq$  a + x
      apply (simp add: algebra_simps)
      by (metis assms(2) floor_divide_lower mult.commute)
  
```

```

    also have ... < y
    by (rule 1)
  finally have ?n * a < y .
  with x show ?thesis
    using Ints_1 Ints_add Ints_of_int that by blast
next
case 2
let ?n = of_int (floor (y/a)) + 1
have y: y < ?n * a
  by (meson ‹0 < a› divide_less_eq floor_eq_iff)
have ?n * a ≤ a + y
  apply (simp add: algebra_simps)
  by (metis assms(2) floor_divide_lower mult.commute)
also have ... < x
  by (rule 2)
finally have ?n * a < x .
then show ?thesis
  using Ints_1 Ints_add Ints_of_int that y by blast
qed
qed

lemma cell_subdivision_lemma:
  assumes finite F
    and  $\bigwedge X. X \in \mathcal{F} \implies \text{polytope } X$ 
    and  $\bigwedge X. X \in \mathcal{F} \implies \text{aff\_dim } X \leq d$ 
    and  $\bigwedge X Y. \llbracket X \in \mathcal{F}; Y \in \mathcal{F} \rrbracket \implies (X \cap Y) \text{ face\_of } X$ 
    and finite I
  shows  $\exists \mathcal{G}. \bigcup \mathcal{G} = \bigcup \mathcal{F} \wedge$ 
     $\text{finite } \mathcal{G} \wedge$ 
     $(\forall C \in \mathcal{G}. \exists D. D \in \mathcal{F} \wedge C \subseteq D) \wedge$ 
     $(\forall C \in \mathcal{F}. \forall x \in C. \exists D. D \in \mathcal{G} \wedge x \in D \wedge D \subseteq C) \wedge$ 
     $(\forall X \in \mathcal{G}. \text{polytope } X) \wedge$ 
     $(\forall X \in \mathcal{G}. \text{aff\_dim } X \leq d) \wedge$ 
     $(\forall X \in \mathcal{G}. \forall Y \in \mathcal{G}. X \cap Y \text{ face\_of } X) \wedge$ 
     $(\forall X \in \mathcal{G}. \forall x \in X. \forall y \in X. \forall a b.$ 
       $(a, b) \in I \longrightarrow a \cdot x \leq b \wedge a \cdot y \leq b \vee$ 
       $a \cdot x \geq b \wedge a \cdot y \geq b)$ 

  using ‹finite I›
proof induction
case empty
then show ?case
  by (rule_tac x=F in exI) (auto simp: assms)
next
case (insert ab I)
then obtain G where eq:  $\bigcup \mathcal{G} = \bigcup \mathcal{F}$  and finite G
  and sub1:  $\bigwedge C. C \in \mathcal{G} \implies \exists D. D \in \mathcal{F} \wedge C \subseteq D$ 
  and sub2:  $\bigwedge C x. C \in \mathcal{F} \wedge x \in C \implies \exists D. D \in \mathcal{G} \wedge x \in D \wedge D$ 
 $\subseteq C$ 
  and poly:  $\bigwedge X. X \in \mathcal{G} \implies \text{polytope } X$ 

```

```

and aff:  $\bigwedge X. X \in \mathcal{G} \implies \text{aff\_dim } X \leq d$ 
and face:  $\bigwedge X Y. \llbracket X \in \mathcal{G}; Y \in \mathcal{G} \rrbracket \implies X \cap Y \text{ face\_of } X$ 
and I:  $\bigwedge X x y a b. \llbracket X \in \mathcal{G}; x \in X; y \in X; (a,b) \in I \rrbracket \implies$ 
 $a \cdot x \leq b \wedge a \cdot y \leq b \vee a \cdot x \geq b \wedge a \cdot y \geq b$ 

by (auto simp: that)
obtain a b where  $ab = (a,b)$ 
by fastforce
let  $?G = (\lambda X. X \cap \{x. a \cdot x \leq b\}) \text{ ' } \mathcal{G} \cup (\lambda X. X \cap \{x. a \cdot x \geq b\}) \text{ ' } \mathcal{G}$ 
have eqInt:  $(S \cap \text{Collect } P) \cap (T \cap \text{Collect } Q) = (S \cap T) \cap (\text{Collect } P \cap \text{Collect } Q)$ 
for S T :: 'a set and P Q
by blast
show ?case
proof (intro conjI exI)
show  $\bigcup ?G = \bigcup \mathcal{F}$ 
by (force simp: eq [symmetric])
show finite ?G
using  $\langle \text{finite } \mathcal{G} \rangle$  by force
show  $\forall X \in ?G. \text{polytope } X$ 
by (force simp: poly polytope_Int polyhedron polyhedron_halfspace_le polyhedron_halfspace_ge)
show  $\forall X \in ?G. \text{aff\_dim } X \leq d$ 
by (auto;metis order_trans aff aff_dim_subset inf_le1)
show  $\forall X \in ?G. \forall x \in X. \forall y \in X. \forall a b.$ 
 $(a,b) \in \text{insert } ab \text{ } I \longrightarrow a \cdot x \leq b \wedge a \cdot y \leq b \vee$ 
 $a \cdot x \geq b \wedge a \cdot y \geq b$ 

using  $\langle ab = (a, b) \rangle$  I by fastforce
show  $\forall X \in ?G. \forall Y \in ?G. X \cap Y \text{ face\_of } X$ 
by (auto simp: eqInt halfspace_Int_eq face_of_Int_Int face face_of_halfspace_le face_of_halfspace_ge)
show  $\forall C \in ?G. \exists D. D \in \mathcal{F} \wedge C \subseteq D$ 
using sub1 by force
show  $\forall C \in \mathcal{F}. \forall x \in C. \exists D. D \in ?G \wedge x \in D \wedge D \subseteq C$ 
proof (intro ballI)
fix C z
assume  $C \in \mathcal{F} z \in C$ 
with sub2 obtain D where  $D \in \mathcal{G} z \in D D \subseteq C$  by blast
have  $D \in \mathcal{G} \wedge z \in D \cap \{x. a \cdot x \leq b\} \wedge D \cap \{x. a \cdot x \leq b\} \subseteq C \vee$ 
 $D \in \mathcal{G} \wedge z \in D \cap \{x. a \cdot x \geq b\} \wedge D \cap \{x. a \cdot x \geq b\} \subseteq C$ 
using linorder_class.linear [of  $a \cdot z b$ ] D by blast
then show  $\exists D. D \in ?G \wedge z \in D \wedge D \subseteq C$ 
by blast
qed
qed
qed

```

proposition *cell_complex_subdivision_exists*:

fixes $\mathcal{F} :: \text{'a::euclidean\_space set set}$

assumes $0 < e$ *finite* $\mathcal{F}$

```

and poly:  $\bigwedge X. X \in \mathcal{F} \implies \text{polytope } X$ 
and aff:  $\bigwedge X. X \in \mathcal{F} \implies \text{aff\_dim } X \leq d$ 
and face:  $\bigwedge X Y. \llbracket X \in \mathcal{F}; Y \in \mathcal{F} \rrbracket \implies X \cap Y \text{ face\_of } X$ 
obtains  $\mathcal{F}'$  where finite  $\mathcal{F}' \cup \mathcal{F} = \bigcup \mathcal{F} \wedge X. X \in \mathcal{F}' \implies \text{diameter } X < e$ 
 $\bigwedge X. X \in \mathcal{F}' \implies \text{polytope } X \wedge X. X \in \mathcal{F}' \implies \text{aff\_dim } X \leq d$ 
 $\bigwedge X Y. \llbracket X \in \mathcal{F}'; Y \in \mathcal{F}' \rrbracket \implies X \cap Y \text{ face\_of } X$ 
 $\bigwedge C. C \in \mathcal{F}' \implies \exists D. D \in \mathcal{F} \wedge C \subseteq D$ 
 $\bigwedge C x. C \in \mathcal{F} \wedge x \in C \implies \exists D. D \in \mathcal{F}' \wedge x \in D \wedge D \subseteq C$ 
proof -
  have bounded( $\bigcup \mathcal{F}$ )
  by (simp add:  $\langle \text{finite } \mathcal{F} \rangle \text{ poly bounded\_Union polytope\_imp\_bounded}$ )
  then obtain B where  $B > 0$  and B:  $\bigwedge x. x \in \bigcup \mathcal{F} \implies \text{norm } x < B$ 
  by (meson bounded_pos_less)
  define C where  $C \equiv \{z \in \mathbb{Z}. |z * e / 2 / \text{real DIM}('a)| \leq B\}$ 
  define I where  $I \equiv \bigcup i \in \text{Basis}. \bigcup j \in C. \{ (i::'a, j * e / 2 / \text{DIM}('a)) \}$ 
  have  $C \subseteq \{x \in \mathbb{Z}. -B / (e / 2 / \text{real DIM}('a)) \leq x \wedge x \leq B / (e / 2 / \text{real DIM}('a))\}$ 
  using  $\langle 0 < e \rangle$  by (auto simp: field_split_simps C_def)
  then have finite C
  using finite_int_segment finite_subset by blast
  then have finite I
  by (simp add: I_def)
  obtain  $\mathcal{F}'$  where eq:  $\bigcup \mathcal{F}' = \bigcup \mathcal{F}$  and finite  $\mathcal{F}'$ 
  and poly:  $\bigwedge X. X \in \mathcal{F}' \implies \text{polytope } X$ 
  and aff:  $\bigwedge X. X \in \mathcal{F}' \implies \text{aff\_dim } X \leq d$ 
  and face:  $\bigwedge X Y. \llbracket X \in \mathcal{F}'; Y \in \mathcal{F}' \rrbracket \implies X \cap Y \text{ face\_of } X$ 
  and I:  $\bigwedge X x y a b. \llbracket X \in \mathcal{F}'; x \in X; y \in X; (a,b) \in I \rrbracket \implies$ 
 $a \cdot x \leq b \wedge a \cdot y \leq b \vee a \cdot x \geq b \wedge a \cdot y \geq b$ 
  and sub1:  $\bigwedge C. C \in \mathcal{F}' \implies \exists D. D \in \mathcal{F} \wedge C \subseteq D$ 
  and sub2:  $\bigwedge C x. C \in \mathcal{F} \wedge x \in C \implies \exists D. D \in \mathcal{F}' \wedge x \in D \wedge D \subseteq C$ 
  apply (rule exE [OF cell_subdivision_lemma])
  using assms  $\langle \text{finite } I \rangle$  by auto
  show ?thesis
  proof (rule_tac  $\mathcal{F}' = \mathcal{F}'$  in that)
  show diameter  $X < e$  if  $X \in \mathcal{F}'$  for X
  proof -
    have diameter  $X \leq e/2$ 
    proof (rule diameter_le)
    show norm  $(x - y) \leq e / 2$  if  $x \in X y \in X$  for x y
    proof -
      have norm  $x < B$  norm  $y < B$ 
      using B  $\langle X \in \mathcal{F}' \rangle$  eq that by blast+
      have norm  $(x - y) \leq (\sum b \in \text{Basis}. |(x-y) \cdot b|)$ 
      by (rule norm_le_l1)
      also have ...  $\leq \text{of\_nat } (\text{DIM}('a)) * (e / 2 / \text{DIM}('a))$ 
      proof (rule sum_bounded_above)
      fix i::'a
      assume i  $\in \text{Basis}$ 
      then have I':  $\bigwedge z b. \llbracket z \in C; b = z * e / (2 * \text{real DIM}('a)) \rrbracket \implies i \cdot x$ 

```

```

≤ b ∧ i · y ≤ b ∨ i · x ≥ b ∧ i · y ≥ b
  using I[of X x y] ⟨X ∈ F'⟩ that unfolding I_def by auto
  show |(x - y) · i| ≤ e / 2 / real DIM('a)
  proof (rule ccontr)
    assume ¬ |(x - y) · i| ≤ e / 2 / real DIM('a)
    then have xyi: |i · x - i · y| > e / 2 / real DIM('a)
      by (simp add: inner_commute inner_diff_right)
    obtain n where n ∈ ℤ and n: i · x < n * (e / 2 / real DIM('a)) ∧
      n * (e / 2 / real DIM('a)) < i · y ∨ i · y < n * (e / 2 / real DIM('a)) ∧ n * (e / 2 / real DIM('a)) < i · x
      using subdivide_interval [OF xyi] DIM_positive ⟨0 < e⟩
      by (auto simp: zero_less_divide_iff)
    have |i · x| < B
  by (metis ⟨i ∈ Basis⟩ ⟨norm x < B⟩ inner_commute norm_bound_Basis_lt)
    have |i · y| < B
  by (metis ⟨i ∈ Basis⟩ ⟨norm y < B⟩ inner_commute norm_bound_Basis_lt)
    have *: |n * e| ≤ B * (2 * real DIM('a))
      if |ix| < B |iy| < B
      and ix: ix * (2 * real DIM('a)) < n * e
      and iy: n * e < iy * (2 * real DIM('a)) for ix iy
    proof (rule abs_leI)
      have iy * (2 * real DIM('a)) ≤ B * (2 * real DIM('a))
        by (rule mult_right_mono) (use ⟨|iy| < B⟩ in linarith)+
      then show n * e ≤ B * (2 * real DIM('a))
        using iy by linarith
    next
      have - ix * (2 * real DIM('a)) ≤ B * (2 * real DIM('a))
        by (rule mult_right_mono) (use ⟨|ix| < B⟩ in linarith)+
      then show - (n * e) ≤ B * (2 * real DIM('a))
        using ix by linarith
    qed
    have n ∈ C
      using ⟨n ∈ ℤ⟩ n by (auto simp: C_def divide_simps intro: * ⟨|i · x|
< B⟩ ⟨|i · y| < B⟩)
    show False
      using I' [OF ⟨n ∈ C⟩ refl] n by auto
    qed
  qed
  also have ... = e / 2
    by simp
  finally show ?thesis .
  qed
  qed (use ⟨0 < e⟩ in force)
  also have ... < e
    by (simp add: ⟨0 < e⟩)
  finally show ?thesis .
  qed
  qed (auto simp: eq_poly aff face sub1 sub2 ⟨finite F'⟩)
  qed

```

6.39.17 Simplexes

The notion of n -simplex for integer $- (1::'a) \leq n$

definition *simplex* :: *int* $\Rightarrow$ *'a::euclidean_space set* $\Rightarrow$ *bool* (**infix** *simplex* 50)
where n *simplex* $S \equiv \exists C. \neg \text{affine\_dependent } C \wedge \text{int}(\text{card } C) = n + 1 \wedge S = \text{convex hull } C$

lemma *simplex*:

n *simplex* $S \longleftrightarrow (\exists C. \text{finite } C \wedge$
 $\neg \text{affine\_dependent } C \wedge$
 $\text{int}(\text{card } C) = n + 1 \wedge$
 $S = \text{convex hull } C)$
by (*auto simp add: simplex_def intro: aff_independent_finite*)

lemma *simplex_convex_hull*:

$\neg \text{affine\_dependent } C \wedge \text{int}(\text{card } C) = n + 1 \implies n$ *simplex* (*convex hull* C)
by (*auto simp add: simplex_def*)

lemma *convex_simplex*: n *simplex* $S \implies \text{convex } S$

by (*metis convex_convex_hull simplex_def*)

lemma *compact_simplex*: n *simplex* $S \implies \text{compact } S$

unfolding *simplex*

using *finite_imp_compact_convex_hull* **by** *blast*

lemma *closed_simplex*: n *simplex* $S \implies \text{closed } S$

by (*simp add: compact_imp_closed compact_simplex*)

lemma *simplex_imp_polytope*:

n *simplex* $S \implies \text{polytope } S$
unfolding *simplex_def polytope_def*
using *aff_independent_finite* **by** *blast*

lemma *simplex_imp_polyhedron*:

n *simplex* $S \implies \text{polyhedron } S$
by (*simp add: polytope_imp_polyhedron simplex_imp_polytope*)

lemma *simplex_dim_ge*: n *simplex* $S \implies -1 \leq n$

by (*metis (no_types, opaque_lifting) aff_dim_geq affine_independent_iff_card_diff_add_cancel diff_diff_eq2 simplex_def*)

lemma *simplex_empty* [*simp*]: n *simplex* $\{\}$ $\longleftrightarrow n = -1$

proof

assume n *simplex* $\{\}$

then show $n = -1$

unfolding *simplex* **by** (*metis card.empty convex_hull_eq_empty diff_0 diff_eq_eq of_nat_0*)

next

assume $n = -1$ **then show** n *simplex* $\{\}$

by (fastforce simp: simplex)
qed

lemma simplex_minus_1 [simp]: -1 simplex $S \longleftrightarrow S = \{\}$
by (metis simplex cancel_comm_monoid_add_class.diff_cancel card_0_eq diff_minus_eq_add of_nat_eq_0_iff simplex_empty)

lemma aff_dim_simplex:
n simplex $S \implies \text{aff\_dim } S = n$
by (metis simplex add.commute add_diff_cancel_left' aff_dim_convex_hull affine_independent_iff_card)

lemma zero_simplex_sing: 0 simplex $\{a\}$
apply (simp add: simplex_def)
using affine_independent_1 card_1_singleton_iff convex_hull_singleton by blast

lemma simplex_sing [simp]: n simplex $\{a\} \longleftrightarrow n = 0$
using aff_dim_simplex aff_dim_sing zero_simplex_sing by blast

lemma simplex_zero: 0 simplex $S \longleftrightarrow (\exists a. S = \{a\})$
by (metis aff_dim_eq_0 aff_dim_simplex simplex_sing)

lemma one_simplex_segment: $a \neq b \implies 1$ simplex closed_segment $a b$
unfolding simplex_def
by (rule_tac $x = \{a, b\}$ in exI) (auto simp: segment_convex_hull)

lemma simplex_segment_cases:
(if $a = b$ then 0 else 1) simplex closed_segment $a b$
by (auto simp: one_simplex_segment)

lemma simplex_segment:
 $\exists n. n$ simplex closed_segment $a b$
using simplex_segment_cases by metis

lemma polytope_lowdim_imp_simplex:
assumes polytope P aff_dim $P \leq 1$
obtains n where n simplex P
proof (cases $P = \{\}$)
case True
then show ?thesis
by (simp add: that)
next
case False
then show ?thesis
by (metis assms compact_convex_collinear_segment collinear_aff_dim polytope_imp_compact polytope_imp_convex simplex_segment_cases that)
qed

lemma simplex_insert_dimplus1:

```

fixes n::int
assumes n simplex S and a: a ∉ affine hull S
shows (n+1) simplex (convex hull (insert a S))
proof -
  obtain C where C: finite C ∧ affine_dependent C ∧ int(card C) = n+1 and S:
  S = convex hull C
  using assms unfolding simplex by force
  show ?thesis
  unfolding simplex
  proof (intro exI conjI)
    have aff_dim S = n
      using aff_dim_simplex assms(1) by blast
    moreover have a ∉ affine hull C
      using S a affine_hull_convex_hull by blast
    moreover have a ∉ C
      using S a hull_inc by fastforce
    ultimately show ¬ affine_dependent (insert a C)
      by (simp add: C S aff_dim_convex_hull aff_dim_insert affine_independent_iff_card)
  next
    have a ∉ C
      using S a hull_inc by fastforce
    then show int (card (insert a C)) = n + 1 + 1
      by (simp add: C)
  next
    show convex hull insert a S = convex hull (insert a C)
      by (simp add: S convex_hull_insert_segments)
  qed (use C in auto)
qed

```

6.39.18 Simplicial complexes and triangulations

definition *simplicial_complex* where

$$\begin{aligned}
 \text{simplicial\_complex } \mathcal{C} \equiv & \\
 & \text{finite } \mathcal{C} \wedge \\
 & (\forall S \in \mathcal{C}. \exists n. n \text{ simplex } S) \wedge \\
 & (\forall F S. S \in \mathcal{C} \wedge F \text{ face\_of } S \longrightarrow F \in \mathcal{C}) \wedge \\
 & (\forall S S'. S \in \mathcal{C} \wedge S' \in \mathcal{C} \longrightarrow (S \cap S') \text{ face\_of } S)
 \end{aligned}$$

definition *triangulation* where

$$\begin{aligned}
 \text{triangulation } \mathcal{T} \equiv & \\
 & \text{finite } \mathcal{T} \wedge \\
 & (\forall T \in \mathcal{T}. \exists n. n \text{ simplex } T) \wedge \\
 & (\forall T T'. T \in \mathcal{T} \wedge T' \in \mathcal{T} \longrightarrow (T \cap T') \text{ face\_of } T)
 \end{aligned}$$

6.39.19 Refining a cell complex to a simplicial complex

proposition *convex_hull_insert_Int_eq*:

fixes z :: 'a :: euclidean_space
 assumes z: z ∈ rel_interior S
 and T: T ⊆ rel_frontier S

```

    and U: U ⊆ rel_frontier S
    and convex S convex T convex U
  shows convex hull (insert z T) ∩ convex hull (insert z U) = convex hull (insert
z (T ∩ U))
    (is ?lhs = ?rhs)
proof
  show ?lhs ⊆ ?rhs
  proof (cases T={ } ∨ U={ })
    case True then show ?thesis by auto
  next
    case False
    then have T ≠ { } U ≠ { } by auto
    have TU: convex (T ∩ U)
      by (simp add: ⟨convex T⟩ ⟨convex U⟩ convex_Int)
    have (⋃ x∈T. closed_segment z x) ∩ (⋃ x∈U. closed_segment z x)
      ⊆ (if T ∩ U = { } then {z} else ⋃ ((closed_segment z) ‘ (T ∩ U))) (is _
⊆ ?IF)
    proof clarify
      fix x t u
      assume xt: x ∈ closed_segment z t
      and xu: x ∈ closed_segment z u
      and t ∈ T u ∈ U
      then have ne: t ≠ z u ≠ z
        using T U z unfolding rel_frontier_def by blast+
      show x ∈ ?IF
      proof (cases x = z)
        case True then show ?thesis by auto
      next
        case False
        have t: t ∈ closure S
          using T ⟨t ∈ T⟩ rel_frontier_def by auto
        have u: u ∈ closure S
          using U ⟨u ∈ U⟩ rel_frontier_def by auto
        show ?thesis
        proof (cases t = u)
          case True
          then show ?thesis
            using ⟨t ∈ T⟩ ⟨u ∈ U⟩ xt by auto
        next
          case False
          have tnot: t ∉ closed_segment u z
            proof -
              have t ∈ closure S - rel_interior S
                using T ⟨t ∈ T⟩ rel_frontier_def by blast
              then have t ∉ open_segment z u
                by (meson DiffD2 rel_interior_closure_convex_segment [OF ⟨convex
S⟩ z u] subsetD)
              then show ?thesis
                by (simp add: ⟨t ≠ u⟩ ⟨t ≠ z⟩ open_segment_commute open_segment_def)
            qed
          qed
        qed
      qed
    qed
  qed

```

```

qed
moreover have  $u \notin \text{closed\_segment } z \ t$ 
  using  $\text{rel\_interior\_closure\_convex\_segment } [OF \langle \text{convex } S \rangle z \ t] \langle u \in U \rangle \langle u \neq z \rangle$ 
   $U$  [unfolded  $\text{rel\_frontier\_def}$ ] tnot
  by (auto simp:  $\text{closed\_segment\_eq\_open}$ )
ultimately
have  $\neg(\text{between } (t,u) \ z \mid \text{between } (u,z) \ t \mid \text{between } (z,t) \ u)$  if  $x \neq z$ 
  using that  $xt \ xu$ 
  by (meson  $\text{between\_antisym between\_mem\_segment between\_trans\_2 ends\_in\_segment}(2)$ )
then have  $\neg \text{collinear } \{t, z, u\}$  if  $x \neq z$ 
  by (auto simp: that  $\text{collinear\_between\_cases between\_commute}$ )
moreover have  $\text{collinear } \{t, z, x\}$ 
  by (metis  $\text{closed\_segment\_commute collinear\_2 collinear\_closed\_segment collinear\_triples ends\_in\_segment}(1) \text{ insert\_absorb insert\_absorb2 } xt$ )
moreover have  $\text{collinear } \{z, x, u\}$ 
  by (metis  $\text{closed\_segment\_commute collinear\_2 collinear\_closed\_segment collinear\_triples ends\_in\_segment}(1) \text{ insert\_absorb insert\_absorb2 } xu$ )
ultimately have False
  using  $\text{collinear\_3\_trans } [of \ t \ z \ x \ u] \langle x \neq z \rangle$  by blast
then show ?thesis by metis
qed
qed
qed
then show ?thesis
  using False  $\langle \text{convex } T \rangle \langle \text{convex } U \rangle TU$ 
  by (simp add:  $\text{convex\_hull\_insert\_segments hull\_same split: if\_split\_asm}$ )
qed
show ?rhs  $\subseteq$  ?lhs
  by (metis  $\text{inf\_greatest hull\_mono inf.cobounded1 inf.cobounded2 insert\_mono}$ )
qed

```

lemma *simplicial_subdivision_aux*:

```

assumes finite  $\mathcal{M}$ 
  and  $\bigwedge C. C \in \mathcal{M} \implies \text{polytope } C$ 
  and  $\bigwedge C. C \in \mathcal{M} \implies \text{aff\_dim } C \leq \text{of\_nat } n$ 
  and  $\bigwedge C \ F. \llbracket C \in \mathcal{M}; F \text{ face\_of } C \rrbracket \implies F \in \mathcal{M}$ 
  and  $\bigwedge C1 \ C2. \llbracket C1 \in \mathcal{M}; C2 \in \mathcal{M} \rrbracket \implies C1 \cap C2 \text{ face\_of } C1$ 
shows  $\exists \mathcal{T}. \text{simplicial\_complex } \mathcal{T} \wedge$ 
   $(\forall K \in \mathcal{T}. \text{aff\_dim } K \leq \text{of\_nat } n) \wedge$ 
   $\bigcup \mathcal{T} = \bigcup \mathcal{M} \wedge$ 
   $(\forall C \in \mathcal{M}. \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F) \wedge$ 
   $(\forall K \in \mathcal{T}. \exists C. C \in \mathcal{M} \wedge K \subseteq C)$ 

```

using *assms*

proof (induction n arbitrary: $\mathcal{M}$ rule: *less_induct*)

case (*less* n)

then have $\text{poly}\mathcal{M}: \bigwedge C. C \in \mathcal{M} \implies \text{polytope } C$

and $\text{aff}\mathcal{M}: \bigwedge C. C \in \mathcal{M} \implies \text{aff\_dim } C \leq \text{of\_nat } n$

```

and faceM:  $\bigwedge C F. \llbracket C \in \mathcal{M}; F \text{ face\_of } C \rrbracket \implies F \in \mathcal{M}$ 
and intfaceM:  $\bigwedge C1 C2. \llbracket C1 \in \mathcal{M}; C2 \in \mathcal{M} \rrbracket \implies C1 \cap C2 \text{ face\_of } C1$ 
by metis+
show ?case
proof (cases  $n \leq 1$ )
  case True
    have  $\bigwedge s. \llbracket n \leq 1; s \in \mathcal{M} \rrbracket \implies \exists m. m \text{ simplex } s$ 
      using polyM affM by (force intro: polytope_lowdim_imp_simplex)
    then show ?thesis
      unfolding simplicial_complex_def using True
      by (rule_tac x=M in exI) (auto simp: less.prems)
  next
    case False
      define S where  $S \equiv \{C \in \mathcal{M}. \text{aff\_dim } C < n\}$ 
      have finite S  $\bigwedge C. C \in S \implies \text{polytope } C \bigwedge C. C \in S \implies \text{aff\_dim } C \leq \text{int}$ 
        ( $n - 1$ )
         $\bigwedge C1 C2. \llbracket C1 \in S; C2 \in S \rrbracket \implies C1 \cap C2 \text{ face\_of } C1$ 
        using less.prems by (auto simp: S_def)
      moreover have §:  $\bigwedge C F. \llbracket C \in S; F \text{ face\_of } C \rrbracket \implies F \in S$ 
        using less.prems unfolding S_def
      by (metis (no_types, lifting) mem_Collect_eq aff_dim_subset face_of_imp_subset
        less_le not_le)
      ultimately obtain U where simplicial_complex U
      and aff_dimU:  $\bigwedge K. K \in U \implies \text{aff\_dim } K \leq \text{int } (n - 1)$ 
      and  $\bigcup U = \bigcup S$ 
      and finU:  $\bigwedge C. C \in S \implies \exists F. \text{finite } F \wedge F \subseteq U \wedge C = \bigcup F$ 
      and CU:  $\bigwedge K. K \in U \implies \exists C. C \in S \wedge K \subseteq C$ 
      using less.IH [of  $n-1$  S] False by auto
      then have finite U
      and simplU:  $\bigwedge S. S \in U \implies \exists n. n \text{ simplex } S$ 
      and faceU:  $\bigwedge F S. \llbracket S \in U; F \text{ face\_of } S \rrbracket \implies F \in U$ 
      and faceIU:  $\bigwedge S S'. \llbracket S \in U; S' \in U \rrbracket \implies (S \cap S') \text{ face\_of } S$ 
      by (auto simp: simplicial_complex_def)
      define N where  $N \equiv \{C \in \mathcal{M}. \text{aff\_dim } C = n\}$ 
      have finite N
      by (simp add: N_def less.prems(1))
      have polyN:  $\bigwedge C. C \in N \implies \text{polytope } C$ 
      and convexN:  $\bigwedge C. C \in N \implies \text{convex } C$ 
      and closedN:  $\bigwedge C. C \in N \implies \text{closed } C$ 
      by (auto simp: N_def polyM polytope_imp_convex polytope_imp_closed)
      have in_rel_interior: (SOME z. z  $\in$  rel_interior C)  $\in$  rel_interior C if C  $\in$ 
        N for C
      using that polyM polytope_imp_convex rel_interior_aff_dim some_in_eq
      by (fastforce simp: N_def)
      have *:  $\exists T. \neg \text{affine\_dependent } T \wedge \text{card } T \leq n \wedge \text{aff\_dim } K < n \wedge K =$ 
        convex hull T
      if K  $\in$  U for K
      proof –
      obtain r where r: r simplex K

```

```

    using  $\langle K \in \mathcal{U} \rangle$  simpl $\mathcal{U}$  by blast
  have  $r = \text{aff\_dim } K$ 
    using  $\langle r \text{ simplex } K \rangle$  aff\_dim\_simplex by blast
  with  $r$ 
  show ?thesis
    unfolding simplex_def
    using False  $\langle \bigwedge K. K \in \mathcal{U} \implies \text{aff\_dim } K \leq \text{int } (n - 1) \rangle$  that by fastforce
qed
have ahK_C_disjoint: affine hull  $K \cap \text{rel\_interior } C = \{\}$ 
  if  $C \in \mathcal{N}$   $K \in \mathcal{U}$   $K \subseteq \text{rel\_frontier } C$  for  $C$   $K$ 
proof -
  have convex C closed C
    by (auto simp: convexN closedN  $\langle C \in \mathcal{N} \rangle$ )
  obtain  $F$  where  $F$ :  $F \text{ face\_of } C$  and  $F \neq C$   $K \subseteq F$ 
  proof -
    obtain  $L$  where  $L \in \mathcal{S}$   $K \subseteq L$ 
      using  $\langle K \in \mathcal{U} \rangle$  CU by blast
    have  $K \subseteq \text{rel\_frontier } C$ 
      by (simp add:  $\langle K \subseteq \text{rel\_frontier } C \rangle$ )
    also have  $\dots \subseteq C$ 
      by (simp add:  $\langle \text{closed } C \rangle$  rel\_frontier_def subset_iff)
    finally have  $K \subseteq C$  .
    have  $L \cap C \text{ face\_of } C$ 
      using N_def S_def  $\langle C \in \mathcal{N} \rangle$   $\langle L \in \mathcal{S} \rangle$  intfaceM by (simp add:
inf_commute)
    moreover have  $L \cap C \neq C$ 
      using  $\langle C \in \mathcal{N} \rangle$   $\langle L \in \mathcal{S} \rangle$ 
      by (metis (mono_tags, lifting) N_def S_def intfaceM mem_Collect_eq
not_le order_refl §)
    moreover have  $K \subseteq L \cap C$ 
      using  $\langle C \in \mathcal{N} \rangle$   $\langle L \in \mathcal{S} \rangle$   $\langle K \subseteq C \rangle$   $\langle K \subseteq L \rangle$  by (auto simp: N_def S_def)
    ultimately show ?thesis using that by metis
  qed
  have affine hull  $F \cap \text{rel\_interior } C = \{\}$ 
    by (rule affine_hull_face_of_disjoint_rel_interior [OF  $\langle \text{convex } C \rangle$   $F$   $\langle F \neq C \rangle$ ])
  with hull_mono [OF  $\langle K \subseteq F \rangle$ ]
  show affine hull  $K \cap \text{rel\_interior } C = \{\}$ 
    by fastforce
qed
let ? $\mathcal{T}$  =  $(\bigcup C \in \mathcal{N}. \bigcup K \in \mathcal{U} \cap \text{Pow } (\text{rel\_frontier } C). \{ \text{convex hull } (\text{insert } (\text{SOME } z. z \in \text{rel\_interior } C) K) \})$ 
have  $\exists \mathcal{T}. \text{simplicial\_complex } \mathcal{T} \wedge$ 
   $(\forall K \in \mathcal{T}. \text{aff\_dim } K \leq \text{of\_nat } n) \wedge$ 
   $(\forall C \in \mathcal{M}. \exists F. F \subseteq \mathcal{T} \wedge C = \bigcup F) \wedge$ 
   $(\forall K \in \mathcal{T}. \exists C. C \in \mathcal{M} \wedge K \subseteq C)$ 
proof (rule exI, intro conjI ballI)
  show simplicial_complex  $(\mathcal{U} \cup ?\mathcal{T})$ 
    unfolding simplicial_complex_def

```

```

proof (intro conjI impI ballI allI)
  show finite ( $\mathcal{U} \cup ?\mathcal{T}$ )
    using ⟨finite  $\mathcal{U}$ ⟩ ⟨finite  $\mathcal{N}$ ⟩ by simp
  show  $\exists n. n \text{ simplex } S \text{ if } S \in \mathcal{U} \cup ?\mathcal{T} \text{ for } S$ 
    using that ahK_C_disjoint in_rel_interior simpl $\mathcal{U}$  simplex_insert_dimplus1
by fastforce
  show  $F \in \mathcal{U} \cup ?\mathcal{T} \text{ if } S: S \in \mathcal{U} \cup ?\mathcal{T} \wedge F \text{ face\_of } S \text{ for } F S$ 
  proof –
    have  $F \in \mathcal{U} \text{ if } S \in \mathcal{U}$ 
      using S_face $\mathcal{U}$  that by blast
    moreover have  $F \in \mathcal{U} \cup ?\mathcal{T}$ 
      if  $F \text{ face\_of } S \ C \in \mathcal{N} \ K \in \mathcal{U} \text{ and } K \subseteq \text{rel\_frontier } C$ 
      and  $S: S = \text{convex hull insert } (\text{SOME } z. z \in \text{rel\_interior } C) \ K \text{ for } C$ 
    proof –
      let  $?z = \text{SOME } z. z \in \text{rel\_interior } C$ 
      have  $?z \in \text{rel\_interior } C$ 
        by (simp add: in_rel_interior ⟨ $C \in \mathcal{N}$ ⟩)
      moreover
        obtain  $I$  where  $\neg \text{affine\_dependent } I \ \text{card } I \leq n \ \text{aff\_dim } K < \text{int } n \ K$ 
        = convex hull  $I$ 
        using * [OF ⟨ $K \in \mathcal{U}$ ⟩] by auto
        ultimately have  $?z \notin \text{affine hull } I$ 
          using ahK_C_disjoint affine_hull_convex_hull that by blast
        have compact  $I$  finite  $I$ 
          by (auto simp: ⟨ $\neg \text{affine\_dependent } I$ ⟩ aff_independent_finite finite_imp_compact)
        moreover have  $F \text{ face\_of } \text{convex hull insert } ?z \ I$ 
          by (metis S ⟨ $F \text{ face\_of } S$ ⟩ ⟨ $K = \text{convex hull } I$ ⟩ convex_hull_eq_empty convex_hull_insert_segments hull_hull)
        ultimately obtain  $J$  where  $J \subseteq \text{insert } ?z \ I \ F = \text{convex hull } J$ 
          using face_of_convex_hull_subset [of insert  $?z \ I \ F$ ] by auto
        show ?thesis
        proof (cases  $?z \in J$ )
          case True
            have  $F \in (\bigcup K \in \mathcal{U} \cap \text{Pow } (\text{rel\_frontier } C). \{\text{convex hull insert } ?z \ K\})$ 
            proof
              have convex hull  $(J - \{?z\}) \text{ face\_of } K$ 
                by (metis True ⟨ $J \subseteq \text{insert } ?z \ I$ ⟩ ⟨ $K = \text{convex hull } I$ ⟩ ⟨ $\neg \text{affine\_dependent } I$ ⟩ face_of_convex_hull_affine_independent_subset_insert_iff)
              then have convex hull  $(J - \{?z\}) \in \mathcal{U}$ 
                by (rule fac $\mathcal{U}$  [OF ⟨ $K \in \mathcal{U}$ ⟩])
              moreover
                have  $\bigwedge x. x \in \text{convex hull } (J - \{?z\}) \implies x \in \text{rel\_frontier } C$ 
                  by (metis True ⟨ $J \subseteq \text{insert } ?z \ I$ ⟩ ⟨ $K = \text{convex hull } I$ ⟩ subsetD hull_mono subset_insert_iff that(4))
                ultimately show convex hull  $(J - \{?z\}) \in \mathcal{U} \cap \text{Pow } (\text{rel\_frontier } C)$  by auto
              let  $?F = \text{convex hull insert } ?z \ (\text{convex hull } (J - \{?z\}))$ 

```

```

have  $F \subseteq ?F$ 
  apply (clarsimp simp:  $\langle F = \text{convex hull } J \rangle$ )
  by (metis True subsetD hull_mono hull_subset subset_insert_iff)
moreover have  $?F \subseteq F$ 
  apply (clarsimp simp:  $\langle F = \text{convex hull } J \rangle$ )
  by (metis (no_types, lifting) True convex_hull_eq_empty convex_hull_insert_segments hull_hull insert_Diff)
ultimately
  show  $F \in \{?F\}$  by auto
qed
with  $\langle C \in \mathcal{N} \rangle$  show ?thesis by auto
next
  case False
  then have  $F \in \mathcal{U}$ 
  using face_of_convex_hull_affine_independent [ $OF \hookrightarrow \neg \text{affine\_dependent } I \rangle$ ]
    by (metis Int_absorb2 Int_insert_right_if0  $\langle F = \text{convex hull } J \rangle \langle J \subseteq \text{insert } ?z \ I \rangle \langle K = \text{convex hull } I \rangle \text{faceU inf\_le2 } \langle K \in \mathcal{U} \rangle$ )
  then show  $F \in \mathcal{U} \cup ?\mathcal{T}$ 
    by blast
qed
qed
ultimately show ?thesis
  using that by auto
qed
have  $\S$ :  $X \cap Y \text{ face\_of } X \wedge X \cap Y \text{ face\_of } Y$ 
  if  $XY$ :  $X \in \mathcal{U} \ Y \in ?\mathcal{T}$  for  $X \ Y$ 
proof -
  obtain  $C \ K$ 
  where  $C \in \mathcal{N} \ K \in \mathcal{U} \ K \subseteq \text{rel\_frontier } C$ 
  and  $Y$ :  $Y = \text{convex hull insert } (\text{SOME } z. z \in \text{rel\_interior } C) \ K$ 
  using  $XY$  by blast
  have convex  $C$ 
  by (simp add:  $\langle C \in \mathcal{N} \rangle \text{convexN}$ )
  have  $K \subseteq C$ 
  by (metis DiffE  $\langle C \in \mathcal{N} \rangle \langle K \subseteq \text{rel\_frontier } C \rangle \text{closedN closure\_closed rel\_frontier\_def subset\_iff}$ )
  let  $?z = (\text{SOME } z. z \in \text{rel\_interior } C)$ 
  have  $z$ :  $?z \in \text{rel\_interior } C$ 
  using  $\langle C \in \mathcal{N} \rangle \text{in\_rel\_interior}$  by blast
  obtain  $D$  where  $D \in \mathcal{S} \ X \subseteq D$ 
  using  $CU \ \langle X \in \mathcal{U} \rangle$  by blast
  have  $D \cap \text{rel\_interior } C = (C \cap D) \cap \text{rel\_interior } C$ 
  using rel_interior_subset by blast
  also have  $(C \cap D) \cap \text{rel\_interior } C = \{\}$ 
  proof (rule face_of_disjoint_rel_interior)
    show  $C \cap D \text{ face\_of } C$ 
    using  $\mathcal{N\_def} \ \mathcal{S\_def} \ \langle C \in \mathcal{N} \rangle \langle D \in \mathcal{S} \rangle \text{intfaceM}$  by blast
    show  $C \cap D \neq C$ 

```



```

    by (metis (mono_tags, lifting) Int_lower2  $\mathcal{N}$ _def  $\mathcal{S}$ _def  $\langle C \in \mathcal{N} \rangle \langle D \in \mathcal{S} \rangle$ 
    aff_dim_subset mem_Collect_eq not_le)
  qed
  finally have DC:  $D \cap \text{rel\_interior } C = \{\}$  .
  have eq:  $X \cap \text{convex hull } (\text{insert } ?z K) = X \cap \text{convex hull } K$ 
  proof (rule Int_convex_hull_insert_rel_exterior [OF  $\langle \text{convex } C \rangle \langle K \subseteq$ 
 $C \rangle z]$ )
    show disjnt  $X$  ( $\text{rel\_interior } C$ )
    using DC by (meson  $\langle X \subseteq D \rangle$  disjnt_def disjnt_subset1)
  qed
  obtain  $I$  where  $I: \neg \text{affine\_dependent } I$ 
    and  $Keq: K = \text{convex hull } I$  and [simp]:  $\text{convex hull } K = K$ 
    using *  $\langle K \in \mathcal{U} \rangle$  by force
  then have  $?z \notin \text{affine hull } I$ 
    using ahK_C_disjoint  $\langle C \in \mathcal{N} \rangle \langle K \in \mathcal{U} \rangle \langle K \subseteq \text{rel\_frontier } C \rangle$ 
    affine_hull_convex_hull  $z$  by blast
  have  $X \cap K \text{ face\_of } K$ 
    by (simp add: XY(1)  $\langle K \in \mathcal{U} \rangle$  faceIU inf_commute)
  also have ...  $\text{face\_of convex hull insert } ?z K$ 
    by (metis I Keq  $\langle ?z \notin \text{affine hull } I \rangle$  aff_independent_finite convex_convex_hull
    face_of_convex_hull_insert face_of_refl_hull_insert)
  finally have  $X \cap K \text{ face\_of convex hull insert } ?z K$  .
  then show ?thesis
    by (simp add: XY(1)  $Y \langle K \in \mathcal{U} \rangle$  eq faceIU)
  qed

show  $S \cap S' \text{ face\_of } S$ 
  if  $S \in \mathcal{U} \cup ?\mathcal{T} \wedge S' \in \mathcal{U} \cup ?\mathcal{T}$  for  $S S'$ 
  using that
  proof (elim conjE UnE)
    fix  $X Y$ 
    assume  $X \in \mathcal{U}$  and  $Y \in \mathcal{U}$ 
    then show  $X \cap Y \text{ face\_of } X$ 
      by (simp add: faceIU)
  next
    fix  $X Y$ 
    assume XY:  $X \in \mathcal{U} Y \in ?\mathcal{T}$ 
    then show  $X \cap Y \text{ face\_of } X Y \cap X \text{ face\_of } Y$ 
      using § [OF XY] by (auto simp: Int_commute)
  next
    fix  $X Y$ 
    assume XY:  $X \in ?\mathcal{T} Y \in ?\mathcal{T}$ 
    show  $X \cap Y \text{ face\_of } X$ 
    proof -
      obtain  $C K D L$ 
      where  $C \in \mathcal{N} K \in \mathcal{U} K \subseteq \text{rel\_frontier } C$ 
        and  $X: X = \text{convex hull insert } (\text{SOME } z. z \in \text{rel\_interior } C) K$ 
        and  $D \in \mathcal{N} L \in \mathcal{U} L \subseteq \text{rel\_frontier } D$ 
        and  $Y: Y = \text{convex hull insert } (\text{SOME } z. z \in \text{rel\_interior } D) L$ 

```

```

    using XY by blast
  let ?z = (SOME z. z ∈ rel_interior C)
  have z: ?z ∈ rel_interior C
  using ⟨C ∈ N⟩ in_rel_interior by blast
  have convex C
  by (simp add: ⟨C ∈ N⟩ convexN)
  have convex K
  using * ⟨K ∈ U⟩ by blast
  have convex L
  by (meson ⟨L ∈ U⟩ convex_simplex simplU)
  show ?thesis
  proof (cases D=C)
    case True
    then have L ⊆ rel_frontier C
    using ⟨L ⊆ rel_frontier D⟩ by auto
    have convex hull insert (SOME z. z ∈ rel_interior C) (K ∩ L) face_of
      convex hull insert (SOME z. z ∈ rel_interior C) K
    by (metis face_of_polytope_insert2 * IntI ⟨C ∈ N⟩ aff_independent_finite
      ahK_C_disjoint_empty_iff faceIU polytope_def z ⟨K ∈ U⟩ ⟨L ∈ U⟩ ⟨K ⊆ rel_frontier
      C⟩)
    then show ?thesis
    using True X Y ⟨K ⊆ rel_frontier C⟩ ⟨L ⊆ rel_frontier C⟩ ⟨convex
      C⟩ ⟨convex K⟩ ⟨convex L⟩ convex_hull_insert_Int_eq z by force
  next
    case False
    have convex D
    by (simp add: ⟨D ∈ N⟩ convexN)
    have K ⊆ C
    by (metis DiffE ⟨C ∈ N⟩ ⟨K ⊆ rel_frontier C⟩ closedN closure_closed
      rel_frontier_def subset_eq)
    have L ⊆ D
    by (metis DiffE ⟨D ∈ N⟩ ⟨L ⊆ rel_frontier D⟩ closedN closure_closed
      rel_frontier_def subset_eq)
    let ?w = (SOME w. w ∈ rel_interior D)
    have w: ?w ∈ rel_interior D
    using ⟨D ∈ N⟩ in_rel_interior by blast
    have C ∩ rel_interior D = (D ∩ C) ∩ rel_interior D
    using rel_interior_subset by blast
    also have (D ∩ C) ∩ rel_interior D = {}
    proof (rule face_of_disjoint_rel_interior)
      show D ∩ C face_of D
      using N_def ⟨C ∈ N⟩ ⟨D ∈ N⟩ intfaceM by blast
      have D ∈ M ∧ aff_dim D = int n
      using N_def ⟨D ∈ N⟩ by blast
      moreover have C ∈ M ∧ aff_dim C = int n
      using N_def ⟨C ∈ N⟩ by blast
      ultimately show D ∩ C ≠ D
      by (metis Int_commute False face_of_aff_dim_lt inf.idem inf_le1
        intfaceM not_le polyM polytope_imp_convex)
    qed
  qed

```

```

qed
finally have CD:  $C \cap (\text{rel\_interior } D) = \{\}$  .
have zKC:  $(\text{convex hull insert } ?z K) \subseteq C$ 
by (metis DiffE  $\langle C \in \mathcal{N} \rangle \langle K \subseteq \text{rel\_frontier } C \rangle \text{closed}\mathcal{N} \text{closure\_closed}$ 
 $\text{convex}\mathcal{N} \text{hull\_minimal insert\_subset rel\_frontier\_def rel\_interior\_subset subset\_iff}$ 
 $z$ )
have disjnt  $(\text{convex hull insert } (\text{SOME } z. z \in \text{rel\_interior } C) K)$ 
 $(\text{rel\_interior } D)$ 
using zKC CD by (force simp: disjnt_def)
then have eq:  $\text{convex hull } (\text{insert } ?z K) \cap \text{convex hull } (\text{insert } ?w L) =$ 
 $\text{convex hull } (\text{insert } ?z K) \cap \text{convex hull } L$ 
by (rule Int_convex_hull_insert_rel_exterior [OF  $\langle \text{convex } D \rangle \langle L \subseteq$ 
 $D \rangle w]$ )
have ch_id:  $\text{convex hull } K = K \text{ convex hull } L = L$ 
using *  $\langle K \in \mathcal{U} \rangle \langle L \in \mathcal{U} \rangle \text{hull\_same}$  by auto
have convex C
by (simp add:  $\langle C \in \mathcal{N} \rangle \text{convex}\mathcal{N}$ )
have  $\text{convex hull } (\text{insert } ?z K) \cap L = L \cap \text{convex hull } (\text{insert } ?z K)$ 
by blast
also have  $\dots = \text{convex hull } K \cap L$ 
proof (subst Int_convex_hull_insert_rel_exterior [OF  $\langle \text{convex } C \rangle \langle K$ 
 $\subseteq C \rangle z]$ )
have  $(C \cap D) \cap \text{rel\_interior } C = \{\}$ 
proof (rule face_of_disjoint_rel_interior)
show  $C \cap D \text{ face\_of } C$ 
using  $\mathcal{N\_def} \langle C \in \mathcal{N} \rangle \langle D \in \mathcal{N} \rangle \text{intface}\mathcal{M}$  by blast
have  $D \in \mathcal{M} \text{aff\_dim } D = \text{int } n$ 
using  $\mathcal{N\_def} \langle D \in \mathcal{N} \rangle$  by fastforce+
moreover have  $C \in \mathcal{M} \text{aff\_dim } C = \text{int } n$ 
using  $\mathcal{N\_def} \langle C \in \mathcal{N} \rangle$  by fastforce+
ultimately have  $\text{aff\_dim } D + - 1 * \text{aff\_dim } C \leq 0$ 
by fastforce
then have  $\neg C \text{ face\_of } D$ 
using False  $\langle \text{convex } D \rangle \text{face\_of\_aff\_dim\_lt}$  by fastforce
show  $C \cap D \neq C$ 
by (metis inf_commute  $\langle C \in \mathcal{M} \rangle \langle D \in \mathcal{M} \rangle \langle \neg C \text{ face\_of } D \rangle$ 
 $\text{intface}\mathcal{M}$ )
qed
then have  $D \cap \text{rel\_interior } C = \{\}$ 
by (metis inf.absorb_iff2 inf_assoc inf_sup_aci(1) rel_interior_subset)
then show disjnt L  $(\text{rel\_interior } C)$ 
by (meson  $\langle L \subseteq D \rangle \text{disjnt\_def disjnt\_subset1}$ )
next
show  $L \cap \text{convex hull } K = \text{convex hull } K \cap L$ 
by force
qed
finally have chKL:  $\text{convex hull } (\text{insert } ?z K) \cap L = \text{convex hull } K \cap$ 
 $L$  .
have  $\text{convex hull insert } ?z K \cap \text{convex hull } L \text{ face\_of } K$ 

```

```

    by (simp add:  $\langle K \in \mathcal{U} \rangle \langle L \in \mathcal{U} \rangle \text{ch\_id chKL faceIU}$ )
  also have ... face_of convex hull insert ?z K
  proof -
    obtain I where I:  $\neg \text{affine\_dependent } I \ K = \text{convex hull } I$ 
    using * [OF  $\langle K \in \mathcal{U} \rangle$ ] by auto
    then have  $\bigwedge a. a \notin \text{rel\_interior } C \vee a \notin \text{affine hull } I$ 
    using ahK_C_disjoint  $\langle C \in \mathcal{N} \rangle \langle K \in \mathcal{U} \rangle \langle K \subseteq \text{rel\_frontier } C \rangle$ 
  affine_hull_convex_hull by blast
    then show ?thesis
    by (metis I affine_independent_insert face_of_convex_hull_affine_independent
  hull_insert subset_insertI z)
    qed
    finally have 1:  $\text{convex hull insert ?z K} \cap \text{convex hull } L \text{ face\_of convex}$ 
  hull insert ?z K .
    have  $\text{convex hull insert ?z K} \cap \text{convex hull } L \text{ face\_of } L$ 
    by (metis  $\langle K \in \mathcal{U} \rangle \langle L \in \mathcal{U} \rangle \text{chKL ch\_id faceIU inf\_commute}$ )
    also have ... face_of convex hull insert ?w L
    proof -
      obtain I where I:  $\neg \text{affine\_dependent } I \ L = \text{convex hull } I$ 
      using * [OF  $\langle L \in \mathcal{U} \rangle$ ] by auto
      then have  $\bigwedge a. a \notin \text{rel\_interior } D \vee a \notin \text{affine hull } I$ 
      using  $\langle D \in \mathcal{N} \rangle \langle L \in \mathcal{U} \rangle \langle L \subseteq \text{rel\_frontier } D \rangle \text{affine\_hull\_convex\_hull}$ 
    ahK_C_disjoint by blast
      then show ?thesis
      by (metis I aff_independent_finite convex_convex_hull face_of_convex_hull_insert
    face_of_refl hull_insert w)
      qed
      finally have 2:  $\text{convex hull insert ?z K} \cap \text{convex hull } L \text{ face\_of convex}$ 
    hull insert ?w L .
      show ?thesis
      by (simp add: X Y eq 1 2)
    qed
  qed
  qed
  qed
  show  $\exists F \subseteq \mathcal{U} \cup ?\mathcal{T}. C = \bigcup F \text{ if } C \in \mathcal{M} \text{ for } C$ 
  proof (cases  $C \in \mathcal{S}$ )
    case True
    then show ?thesis
    by (meson UnCI finU subsetD subsetI)
  next
    case False
    then have  $C \in \mathcal{N}$ 
    by (simp add:  $\mathcal{N\_def } \mathcal{S\_def } \text{affM less\_le that}$ )
    let ?z = SOME z.  $z \in \text{rel\_interior } C$ 
    have z:  $?z \in \text{rel\_interior } C$ 
    using  $\langle C \in \mathcal{N} \rangle \text{in\_rel\_interior}$  by blast
    let ?F =  $\bigcup K \in \mathcal{U} \cap \text{Pow } (\text{rel\_frontier } C). \{\text{convex hull } (\text{insert ?z K})\}$ 
    have  $?F \subseteq ?\mathcal{T}$ 

```

```

    using  $\langle C \in \mathcal{N} \rangle$  by blast
  moreover have  $C \subseteq \bigcup ?F$ 
proof
  fix x
  assume  $x \in C$ 
  have convex C
  using  $\langle C \in \mathcal{N} \rangle$  convexN by blast
  have bounded C
  using  $\langle C \in \mathcal{N} \rangle$  by (simp add: polyM polytope_imp_bounded that)
  have polytope C
  using  $\langle C \in \mathcal{N} \rangle$  polyN by auto
  have  $\neg (?z = x \wedge C = \{?z\})$ 
  using  $\langle C \in \mathcal{N} \rangle$  aff_dim_sing [of ?z]  $\langle \neg n \leq 1 \rangle$  by (force simp: N_def)
  then obtain y where  $y: y \in \text{rel\_frontier } C$  and  $xzy: x \in \text{closed\_segment}$ 
    ?z y
    and sub:  $\text{open\_segment } ?z y \subseteq \text{rel\_interior } C$ 
  by (blast intro: segment_to_rel_frontier [OF  $\langle \text{convex } C \rangle \langle \text{bounded } C \rangle z$ 
 $\langle x \in C \rangle$ ])
  then obtain F where  $y \in F$   $F \text{ face\_of } C$   $F \neq C$ 
  by (auto simp: rel_frontier_of_polyhedron_alt [OF polytope_imp_polyhedron
[OF  $\langle \text{polytope } C \rangle$ ]])
  then obtain  $\mathcal{G}$  where  $\text{finite } \mathcal{G}$   $\mathcal{G} \subseteq \mathcal{U}$   $F = \bigcup \mathcal{G}$ 
  by (metis (mono_tags, lifting) S_def  $\langle C \in \mathcal{M} \rangle \langle \text{convex } C \rangle$  affM faceM
face_of_aff_dim_lt finU le_less_trans mem_Collect_eq not_less)
  then obtain K where  $y \in K$   $K \in \mathcal{G}$ 
  using  $\langle y \in F \rangle$  by blast
  moreover have  $x: x \in \text{convex hull } \{?z, y\}$ 
  using segment_convex_hull xzy by auto
  moreover have  $\text{convex hull } \{?z, y\} \subseteq \text{convex hull insert } ?z K$ 
  by (metis (full_types)  $\langle y \in K \rangle$  hull_mono empty_subsetI insertCI
insert_subset)
  moreover have  $K \in \mathcal{U}$ 
  using  $\langle K \in \mathcal{G} \rangle \langle \mathcal{G} \subseteq \mathcal{U} \rangle$  by blast
  moreover have  $K \subseteq \text{rel\_frontier } C$ 
  using  $\langle F = \bigcup \mathcal{G} \rangle \langle F \neq C \rangle \langle F \text{ face\_of } C \rangle \langle K \in \mathcal{G} \rangle$  face_of_subset_rel_frontier
by fastforce
  ultimately show  $x \in \bigcup ?F$ 
  by force
qed
moreover
have  $\text{convex hull insert } (\text{SOME } z. z \in \text{rel\_interior } C) K \subseteq C$ 
if  $K \in \mathcal{U}$   $K \subseteq \text{rel\_frontier } C$  for K
proof (rule hull_minimal)
  show  $\text{insert } (\text{SOME } z. z \in \text{rel\_interior } C) K \subseteq C$ 
  using that  $\langle C \in \mathcal{N} \rangle$  in_rel_interior rel_interior_subset
  by (force simp: closure_eq rel_frontier_def closedN)
  show convex C
  by (simp add:  $\langle C \in \mathcal{N} \rangle$  convexN)
qed

```

```

    then have  $\bigcup ?F \subseteq C$ 
      by auto
    ultimately show ?thesis
      by blast
  qed
  have  $(\exists C. C \in \mathcal{M} \wedge L \subseteq C) \wedge \text{aff\_dim } L \leq \text{int } n$  if  $L \in \mathcal{U} \cup ?\mathcal{T}$  for  $L$ 
    using that
  proof
    assume  $L \in \mathcal{U}$ 
    then show ?thesis
      using  $\text{CU } S\_def$  * by fastforce
  next
    assume  $L \in ?\mathcal{T}$ 
    then obtain  $C \ K$  where  $C \in \mathcal{N}$ 
      and  $L: L = \text{convex\_hull\_insert } (SOME\ z. z \in \text{rel\_interior } C) \ K$ 
      and  $K: K \in \mathcal{U} \ K \subseteq \text{rel\_frontier } C$ 
      by auto
    then have  $\text{convex\_hull } C = C$ 
      by (meson convex $\mathcal{N}$  convex_hull_eq)
    then have convex  $C$ 
      by (metis (no_types) convex_convex_hull)
    have  $\text{rel\_frontier } C \subseteq C$ 
      by (metis DiffE closed $\mathcal{N}$   $\langle C \in \mathcal{N} \rangle$  closure_closed rel_frontier_def subsetI)
    have  $K \subseteq C$ 
      using  $K \langle \text{rel\_frontier } C \subseteq C \rangle$  by blast
    have  $C \in \mathcal{M}$ 
      using  $\mathcal{N\_def} \langle C \in \mathcal{N} \rangle$  by auto
    moreover have  $L \subseteq C$ 
      using  $K \ L \langle C \in \mathcal{N} \rangle$ 
      by (metis  $\langle K \subseteq C \rangle \langle \text{convex\_hull } C = C \rangle$  contra_subsetD hull_mono
in_rel_interior_insert_subset rel_interior_subset)
    ultimately show ?thesis
      using  $\langle \text{rel\_frontier } C \subseteq C \rangle \langle L \subseteq C \rangle \text{aff}\mathcal{M} \text{aff\_dim\_subset } \langle C \in \mathcal{M} \rangle$ 
      dual_order.trans by blast
  qed
  then show  $\exists C. C \in \mathcal{M} \wedge L \subseteq C \text{aff\_dim } L \leq \text{int } n$  if  $L \in \mathcal{U} \cup ?\mathcal{T}$  for  $L$ 
    using that by auto
  qed
  then show ?thesis
    apply (rule ex_forward, safe)
    apply (meson Union_iff subsetCE, fastforce)
    by (meson infinite_super simplicial_complex_def)
  qed
  qed

```

lemma *simplicial_subdivision_of_cell_complex_lowdim:*
assumes *finite* $\mathcal{M}$
and poly: $\bigwedge C. C \in \mathcal{M} \implies \text{polytope } C$

```

    and face:  $\bigwedge C1\ C2. \llbracket C1 \in \mathcal{M};\ C2 \in \mathcal{M} \rrbracket \implies C1 \cap C2 \text{ face\_of } C1$ 
    and aff:  $\bigwedge C. C \in \mathcal{M} \implies \text{aff\_dim } C \leq d$ 
  obtains  $\mathcal{T}$  where simplicial_complex  $\mathcal{T} \bigwedge K. K \in \mathcal{T} \implies \text{aff\_dim } K \leq d$ 
     $\bigcup \mathcal{T} = \bigcup \mathcal{M}$ 
     $\bigwedge C. C \in \mathcal{M} \implies \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$ 
     $\bigwedge K. K \in \mathcal{T} \implies \exists C. C \in \mathcal{M} \wedge K \subseteq C$ 
proof (cases  $d \geq 0$ )
  case True
    then obtain n where n:  $d = \text{of\_nat } n$ 
    using zero_le_imp_eq_int by blast
    have  $\exists \mathcal{T}. \text{simplicial\_complex } \mathcal{T} \wedge$ 
       $(\forall K \in \mathcal{T}. \text{aff\_dim } K \leq \text{int } n) \wedge$ 
       $\bigcup \mathcal{T} = \bigcup (\bigcup C \in \mathcal{M}. \{F. F \text{ face\_of } C\}) \wedge$ 
       $(\forall C \in \bigcup C \in \mathcal{M}. \{F. F \text{ face\_of } C\}. \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F) \wedge$ 
       $(\forall K \in \mathcal{T}. \exists C. C \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face\_of } C\}) \wedge K \subseteq C)$ 
    proof (rule simplicial_subdivision_aux)
      show finite  $(\bigcup C \in \mathcal{M}. \{F. F \text{ face\_of } C\})$ 
        using  $\langle \text{finite } \mathcal{M} \rangle \text{ poly polyhedron\_eq\_finite\_faces polytope\_imp\_polyhedron}$ 
      by fastforce
      show polytope F if  $F \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face\_of } C\})$  for F
        using poly that face_of_polytope_polytope by blast
      show aff_dim F  $\leq \text{int } n$  if  $F \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face\_of } C\})$  for F
        using that
        by clarify (metis n aff_dim_subset aff_face_of_imp_subset order_trans)
      show  $F \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face\_of } C\})$ 
        if  $G \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face\_of } C\})$  and  $F \text{ face\_of } G$  for F G
        using that face_of_trans by blast
    next
      fix F1 F2
      assume  $F1 \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face\_of } C\})$  and  $F2 \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face\_of } C\})$ 
      then obtain C1 C2 where  $C1 \in \mathcal{M}\ C2 \in \mathcal{M}$  and  $F: F1 \text{ face\_of } C1\ F2 \text{ face\_of } C2$ 
      by auto
      show  $F1 \cap F2 \text{ face\_of } F1$ 
        using face_of_Int_subface [OF _ _ F]
        by (metis  $\langle C1 \in \mathcal{M} \rangle \langle C2 \in \mathcal{M} \rangle \text{face inf\_commute}$ )
    qed
  moreover
    have  $\bigcup (\bigcup C \in \mathcal{M}. \{F. F \text{ face\_of } C\}) = \bigcup \mathcal{M}$ 
    using face_of_imp_subset face by blast
    ultimately show ?thesis
      using face_of_imp_subset n
      by (fastforce intro!: that simp add: poly_face_of_refl polytope_imp_convex)
  next
    case False
    then have m1:  $\bigwedge C. C \in \mathcal{M} \implies \text{aff\_dim } C = -1$ 
    by (metis aff aff_dim_empty_eq aff_dim_negative_iff dual_order.trans not_less)

```

```

then have faceM:  $\bigwedge F S. \llbracket S \in \mathcal{M}; F \text{ face\_of } S \rrbracket \implies F \in \mathcal{M}$ 
  by (metis aff_dim_empty face_of_empty)
show ?thesis
proof
  have  $\bigwedge S. S \in \mathcal{M} \implies \exists n. n \text{ simplex } S$ 
    by (metis (no_types) m1 aff_dim_empty simplex_minus_1)
  then show simplicial_complex M
    by (auto simp: simplicial_complex_def  $\langle \text{finite } \mathcal{M} \rangle$  face_intro: faceM)
  show  $\text{aff\_dim } K \leq d$  if  $K \in \mathcal{M}$  for  $K$ 
    by (simp add: that aff)
  show  $\exists F. \text{finite } F \wedge F \subseteq \mathcal{M} \wedge C = \bigcup F$  if  $C \in \mathcal{M}$  for  $C$ 
    using  $\langle C \in \mathcal{M} \rangle \text{ equals0I}$  by auto
  show  $\exists C. C \in \mathcal{M} \wedge K \subseteq C$  if  $K \in \mathcal{M}$  for  $K$ 
    using  $\langle K \in \mathcal{M} \rangle$  by blast
qed auto
qed

proposition simplicial_subdivision_of_cell_complex:
  assumes finite M
    and poly:  $\bigwedge C. C \in \mathcal{M} \implies \text{polytope } C$ 
    and face:  $\bigwedge C1 C2. \llbracket C1 \in \mathcal{M}; C2 \in \mathcal{M} \rrbracket \implies C1 \cap C2 \text{ face\_of } C1$ 
  obtains  $\mathcal{T}$  where simplicial_complex T
     $\bigcup \mathcal{T} = \bigcup \mathcal{M}$ 
     $\bigwedge C. C \in \mathcal{M} \implies \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$ 
     $\bigwedge K. K \in \mathcal{T} \implies \exists C. C \in \mathcal{M} \wedge K \subseteq C$ 
  by (blast intro: simplicial_subdivision_of_cell_complex_lowdim [OF assms aff_dim_le_DIM])

corollary fine_simplicial_subdivision_of_cell_complex:
  assumes  $0 < e$  finite M
    and poly:  $\bigwedge C. C \in \mathcal{M} \implies \text{polytope } C$ 
    and face:  $\bigwedge C1 C2. \llbracket C1 \in \mathcal{M}; C2 \in \mathcal{M} \rrbracket \implies C1 \cap C2 \text{ face\_of } C1$ 
  obtains  $\mathcal{T}$  where simplicial_complex T
     $\bigwedge K. K \in \mathcal{T} \implies \text{diameter } K < e$ 
     $\bigcup \mathcal{T} = \bigcup \mathcal{M}$ 
     $\bigwedge C. C \in \mathcal{M} \implies \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$ 
     $\bigwedge K. K \in \mathcal{T} \implies \exists C. C \in \mathcal{M} \wedge K \subseteq C$ 
proof –
  obtain  $\mathcal{N}$  where  $\mathcal{N}$ : finite  $\mathcal{N} \bigcup \mathcal{N} = \bigcup \mathcal{M}$ 
    and diapoly:  $\bigwedge X. X \in \mathcal{N} \implies \text{diameter } X < e \wedge X. X \in \mathcal{N} \implies \text{polytope } X$ 
  and  $\bigwedge X Y. \llbracket X \in \mathcal{N}; Y \in \mathcal{N} \rrbracket \implies X \cap Y \text{ face\_of } X$ 
  and  $\mathcal{N} \text{ covers}$ :  $\bigwedge C x. C \in \mathcal{M} \wedge x \in C \implies \exists D. D \in \mathcal{N} \wedge x \in D \wedge D \subseteq C$ 
  and  $\mathcal{N} \text{ covered}$ :  $\bigwedge C. C \in \mathcal{N} \implies \exists D. D \in \mathcal{M} \wedge C \subseteq D$ 
  by (blast intro: cell_complex_subdivision_exists [OF  $\langle 0 < e \rangle \langle \text{finite } \mathcal{M} \rangle$  poly aff_dim_le_DIM face])
  then obtain  $\mathcal{T}$  where  $\mathcal{T}$ : simplicial_complex  $\mathcal{T} \bigcup \mathcal{T} = \bigcup \mathcal{N}$ 
    and  $\mathcal{T} \text{ covers}$ :  $\bigwedge C. C \in \mathcal{N} \implies \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$ 
    and  $\mathcal{T} \text{ covered}$ :  $\bigwedge K. K \in \mathcal{T} \implies \exists C. C \in \mathcal{N} \wedge K \subseteq C$ 

```



```

    using simplicial_subdivision_of_cell_complex [OF ‹finite  $\mathcal{N}$ ›] by metis
  show ?thesis
proof
  show simplicial_complex  $\mathcal{T}$ 
    by (rule  $\mathcal{T}$ )
  show diameter  $K < e$  if  $K \in \mathcal{T}$  for  $K$ 
    by (metis le_less_trans diapoly  $\mathcal{T}$  covered diameter_subset polytope_imp_bounded
that)
  show  $\bigcup \mathcal{T} = \bigcup \mathcal{M}$ 
    by (simp add:  $\mathcal{N}(2)$  ‹ $\bigcup \mathcal{T} = \bigcup \mathcal{N}$ ›)
  show  $\exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$  if  $C \in \mathcal{M}$  for  $C$ 
proof -
  { fix  $x$ 
    assume  $x \in C$ 
    then obtain  $D$  where  $D \in \mathcal{T} \ x \in D \ D \subseteq C$ 
      using  $\mathcal{N}$  covers ‹ $C \in \mathcal{M}$ ›  $\mathcal{T}$  covers by force
    then have  $\exists X \in \mathcal{T} \cap \text{Pow } C. x \in X$ 
      using ‹ $D \in \mathcal{T}$ › ‹ $D \subseteq C$ › ‹ $x \in D$ › by blast
  }
  moreover
  have finite  $(\mathcal{T} \cap \text{Pow } C)$ 
    using ‹simplicial_complex  $\mathcal{T}$ › simplicial_complex_def by auto
  ultimately show ?thesis
    by (rule_tac  $x = (\mathcal{T} \cap \text{Pow } C)$  in exI) auto
qed
show  $\exists C. C \in \mathcal{M} \wedge K \subseteq C$  if  $K \in \mathcal{T}$  for  $K$ 
  by (meson  $\mathcal{N}$  covered  $\mathcal{T}$  covered order_trans that)
qed
qed

```

6.39.20 Some results on cell division with full-dimensional cells only

```

lemma convex_Union_fulldim_cells:
  assumes finite  $\mathcal{S}$  and clo:  $\bigwedge C. C \in \mathcal{S} \implies \text{closed } C$  and con:  $\bigwedge C. C \in \mathcal{S} \implies$ 
    convex  $C$ 
  and eq:  $\bigcup \mathcal{S} = U$  and convex  $U$ 
  shows  $\bigcup \{C \in \mathcal{S}. \text{aff\_dim } C = \text{aff\_dim } U\} = U$  (is ?lhs =  $U$ )
proof -
  have closed  $U$ 
    using ‹finite  $\mathcal{S}$ › clo eq by blast
  have ?lhs  $\subseteq U$ 
    using eq by blast
  moreover have  $U \subseteq ?lhs$ 
  proof (cases  $\forall C \in \mathcal{S}. \text{aff\_dim } C = \text{aff\_dim } U$ )
  case True
    then show ?thesis
      using eq by blast
  next

```

```

case False
have closed ?lhs
  by (simp add: ⟨finite S⟩ clo closed_Union)
moreover have  $U \subseteq \text{closure } ?lhs$ 
proof -
  have  $U \subseteq \text{closure}(\bigcap \{U - C \mid C. C \in S \wedge \text{aff\_dim } C < \text{aff\_dim } U\})$ 
  proof (rule Baire [OF ⟨closed U⟩])
    show countable  $\{U - C \mid C. C \in S \wedge \text{aff\_dim } C < \text{aff\_dim } U\}$ 
      using ⟨finite S⟩ uncountable_infinite by fastforce
    have  $\bigwedge C. C \in S \implies \text{openin } (\text{top\_of\_set } U) (U - C)$ 
      by (metis Sup_upper clo closed_limpt closedin_limpt eq openin_diff
openin_subtopology_self)
    then show  $\text{openin } (\text{top\_of\_set } U) T \wedge U \subseteq \text{closure } T$ 
      if  $T \in \{U - C \mid C. C \in S \wedge \text{aff\_dim } C < \text{aff\_dim } U\}$  for T
      using that dense_complement_convex_closed ⟨closed U⟩ ⟨convex U⟩ by
auto
  qed
  also have  $\dots \subseteq \text{closure } ?lhs$ 
  proof -
    obtain C where  $C \in S$   $\text{aff\_dim } C < \text{aff\_dim } U$ 
      by (metis False Sup_upper aff_dim_subset eq eq_iff not_le)
    have  $\exists X. X \in S \wedge \text{aff\_dim } X = \text{aff\_dim } U \wedge x \in X$ 
      if  $\bigwedge V. (\exists C. V = U - C \wedge C \in S \wedge \text{aff\_dim } C < \text{aff\_dim } U) \implies x \in V$ 
      for x
    proof -
      have  $x \in U \wedge x \in \bigcup S$ 
        using ⟨C ∈ S⟩ ⟨aff_dim C < aff_dim U⟩ eq that by blast
      then show ?thesis
        by (metis Diff_iff Sup_upper Union_iff aff_dim_subset dual_order.order_iff_strict
eq that)
    qed
    then show ?thesis
      by (auto intro!: closure_mono)
  qed
  finally show ?thesis .
qed
ultimately show ?thesis
  using closure_subset_eq by blast
qed
ultimately show ?thesis by blast
qed

proposition fine_triangular_subdivision_of_cell_complex:
  assumes  $0 < e$  finite M
  and poly:  $\bigwedge C. C \in M \implies \text{polytope } C$ 
  and aff:  $\bigwedge C. C \in M \implies \text{aff\_dim } C = d$ 
  and face:  $\bigwedge C1\ C2. \llbracket C1 \in M; C2 \in M \rrbracket \implies C1 \cap C2 \text{ face\_of } C1$ 
  obtains T where triangulation T  $\bigwedge k. k \in T \implies \text{diameter } k < e$ 
     $\bigwedge k. k \in T \implies \text{aff\_dim } k = d \cup T = \bigcup M$ 

```

$$\bigwedge C. C \in \mathcal{M} \implies \exists f. \text{finite } f \wedge f \subseteq \mathcal{T} \wedge C = \bigcup f$$

$$\bigwedge k. k \in \mathcal{T} \implies \exists C. C \in \mathcal{M} \wedge k \subseteq C$$

proof –

obtain $\mathcal{T}$ **where** *simplicial_complex* $\mathcal{T}$

and *diaT*: $\bigwedge K. K \in \mathcal{T} \implies \text{diameter } K < e$

and $\bigcup \mathcal{T} = \bigcup \mathcal{M}$

and *inM*: $\bigwedge C. C \in \mathcal{M} \implies \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$

and *inT*: $\bigwedge K. K \in \mathcal{T} \implies \exists C. C \in \mathcal{M} \wedge K \subseteq C$

by (*blast intro: fine_simplicial_subdivision_of_cell_complex* [*OF* $\langle e > 0 \rangle$ $\langle \text{finite } \mathcal{M} \rangle$ *poly face*])

let $?T = \{K \in \mathcal{T}. \text{aff\_dim } K = d\}$

show *thesis*

proof

show *triangulation* $?T$

using $\langle \text{simplicial\_complex } \mathcal{T} \rangle$ **by** (*auto simp: triangulation_def simplicial_complex_def*)

show *diameter* $L < e$ **if** $L \in \{K \in \mathcal{T}. \text{aff\_dim } K = d\}$ **for** L

using *that* **by** (*auto simp: diaT*)

show *aff_dim* $L = d$ **if** $L \in \{K \in \mathcal{T}. \text{aff\_dim } K = d\}$ **for** L

using *that* **by** *auto*

show $\exists F. \text{finite } F \wedge F \subseteq \{K \in \mathcal{T}. \text{aff\_dim } K = d\} \wedge C = \bigcup F$ **if** $C \in \mathcal{M}$

for C

proof –

obtain F **where** *finite* F $F \subseteq \mathcal{T}$ $C = \bigcup F$

using *inM* [*OF* $\langle C \in \mathcal{M} \rangle$] **by** *auto*

show *?thesis*

proof (*intro exI conjI*)

show *finite* $\{K \in F. \text{aff\_dim } K = d\}$

by (*simp add: finite F*)

show $\{K \in F. \text{aff\_dim } K = d\} \subseteq \{K \in \mathcal{T}. \text{aff\_dim } K = d\}$

using $\langle F \subseteq \mathcal{T} \rangle$ **by** *blast*

have $d = \text{aff\_dim } C$

by (*simp add: aff that*)

moreover **have** $\bigwedge K. K \in F \implies \text{closed } K \wedge \text{convex } K$

using $\langle \text{simplicial\_complex } \mathcal{T} \rangle \langle F \subseteq \mathcal{T} \rangle$

unfolding *simplicial_complex_def* **by** (*metis subsetCE* $\langle F \subseteq \mathcal{T} \rangle$ *closed_simplex convex_simplex*)

moreover **have** *convex* $(\bigcup F)$

using $\langle C = \bigcup F \rangle$ *poly polytope_imp_convex that* **by** *blast*

ultimately **show** $C = \bigcup \{K \in F. \text{aff\_dim } K = d\}$

by (*simp add: convex_Union_fulldim_cells* $\langle C = \bigcup F \rangle \langle \text{finite } F \rangle$)

qed

qed

then **show** $\bigcup \{K \in \mathcal{T}. \text{aff\_dim } K = d\} = \bigcup \mathcal{M}$

by *auto* (*meson inT subsetCE*)

show $\exists C. C \in \mathcal{M} \wedge L \subseteq C$

if $L \in \{K \in \mathcal{T}. \text{aff\_dim } K = d\}$ **for** L

using *that* **by** (*auto simp: inT*)

qed

qed

end

6.40 Arcwise-Connected Sets

theory *Arcwise_Connected*

imports *Path_Connected Ordered_Euclidean_Space HOL-Computational_Algebra.Primes*
begin

lemma *path_connected_interval* [simp]:
 fixes $a b :: 'a :: \text{ordered\_euclidean\_space}$
 shows *path_connected* $\{a..b\}$
 using *is_interval_cc is_interval_path_connected* by blast

lemma *segment_to_closest_point*:
 fixes $S :: 'a :: \text{euclidean\_space}$ set
 shows $\llbracket \text{closed } S; S \neq \{\} \rrbracket \implies \text{open\_segment } a (\text{closest\_point } S a) \cap S = \{\}$
 unfolding *disjoint_iff*
 by (metis *closest_point_le dist_commute dist_in_open_segment not_le*)

lemma *segment_to_point_exists*:
 fixes $S :: 'a :: \text{euclidean\_space}$ set
 assumes *closed* S $S \neq \{\}$
 obtains b where $b \in S$ *open_segment* $a b \cap S = \{\}$
 by (metis *assms segment_to_closest_point closest_point_exists that*)

6.40.1 The Brouwer reduction theorem

theorem *Brouwer_reduction_theorem_gen*:
 fixes $S :: 'a :: \text{euclidean\_space}$ set
 assumes *closed* S φS
 and $\varphi: \bigwedge F. \llbracket \bigwedge n. \text{closed}(F n); \bigwedge n. \varphi(F n); \bigwedge n. F(\text{Suc } n) \subseteq F n \rrbracket \implies \varphi(\bigcap (\text{range } F))$
 obtains T where $T \subseteq S$ *closed* T $\varphi T \bigwedge U. \llbracket U \subseteq S; \text{closed } U; \varphi U \rrbracket \implies \neg (U \subset T)$
proof –
 obtain $B :: \text{nat} \Rightarrow 'a$ set
 where *inj* $B \bigwedge n. \text{open}(B n)$ and *open_cov*: $\bigwedge S. \text{open } S \implies \exists K. S = \bigcup (B \restriction K)$
 by (metis *Setcompr_eq_image that univ_second_countable_sequence*)
 define A where $A \equiv \text{rec\_nat } S (\lambda n a. \text{if } \exists U. U \subseteq a \wedge \text{closed } U \wedge \varphi U \wedge U \cap (B n) = \{\}$
 $\qquad \qquad \qquad \text{then SOME } U. U \subseteq a \wedge \text{closed } U \wedge \varphi U \wedge U \cap (B n) = \{\}$
 $\qquad \qquad \qquad \text{else } a)$
 have [simp]: $A 0 = S$
 by (simp add: *A_def*)
 have *ASuc*: $A(\text{Suc } n) = (\text{if } \exists U. U \subseteq A n \wedge \text{closed } U \wedge \varphi U \wedge U \cap (B n) = \{\}$

```

      then SOME U. U ⊆ A n ∧ closed U ∧ φ U ∧ U ∩ (B n) = {}
    else A n) for n
  by (auto simp: A_def)
have sub: ⋀n. A (Suc n) ⊆ A n
  by (auto simp: ASuc dest!: someI_ex)
have subS: A n ⊆ S for n
  by (induction n) (use sub in auto)
have clo: closed (A n) ∧ φ (A n) for n
  by (induction n) (auto simp: assms ASuc dest!: someI_ex)
show ?thesis
proof
  show ⋂ (range A) ⊆ S
    using ⋀n. A n ⊆ S by blast
  show closed (⋂ (A ` UNIV))
    using clo by blast
  show φ (⋂ (A ` UNIV))
    by (simp add: clo φ sub)
  show ¬ U ⊂ ⋂ (A ` UNIV) if U ⊆ S closed U φ U for U
  proof -
    have ∃ y. x ∉ A y if x ∉ U and Usub: U ⊆ (⋂ x. A x) for x
    proof -
      obtain e where e > 0 and e: ball x e ⊆ -U
        using ⋀closed U ⋀x ∉ U openE [of -U] by blast
      moreover obtain K where K: ball x e = ⋃ (B ` K)
        using open_cov [of ball x e] by auto
      ultimately have ⋃ (B ` K) ⊆ -U
        by blast
      have K ≠ {}
        using ⋀0 < e ⋀ball x e = ⋃ (B ` K) by auto
      then obtain n where n ∈ K x ∈ B n
        by (metis K UN_E ⋀0 < e centre_in_ball)
      then have U ∩ B n = {}
        using K e by auto
      show ?thesis
    proof (cases ∃ U ⊆ A n. closed U ∧ φ U ∧ U ∩ B n = {})
      case True
      then show ?thesis
        apply (rule_tac x = Suc n in exI)
        apply (simp add: ASuc)
        apply (erule someI2_ex)
        using ⋀x ∈ B n by blast
      next
      case False
      then show ?thesis
        by (meson Inf_lower Usub ⋀U ∩ B n = {} ⋀φ U ⋀closed U range_eqI
subset_trans)
    qed
  qed
  with that show ?thesis

```

```

      by (meson Inter_iff psubsetE rangeI subsetI)
    qed
  qed
qed

corollary Brouwer_reduction_theorem:
  fixes S :: 'a::euclidean_space set
  assumes compact S  $\varphi$  S  $S \neq \{\}$ 
  and  $\varphi: \bigwedge F. \llbracket \bigwedge n. \text{compact}(F\ n); \bigwedge n. F\ n \neq \{\}; \bigwedge n. \varphi(F\ n); \bigwedge n. F(\text{Suc}\ n) \subseteq F\ n \rrbracket \implies \varphi(\bigcap(\text{range}\ F))$ 
  obtains T where  $T \subseteq S$  compact T  $T \neq \{\}$   $\varphi$  T
     $\bigwedge U. \llbracket U \subseteq S; \text{closed}\ U; U \neq \{\}; \varphi\ U \rrbracket \implies \neg (U \subset T)$ 
proof (rule Brouwer_reduction_theorem_gen [of S  $\lambda T. T \neq \{\} \wedge T \subseteq S \wedge \varphi$  T])
  fix F
  assume cloF:  $\bigwedge n. \text{closed}\ (F\ n)$ 
  and F:  $\bigwedge n. F\ n \neq \{\} \wedge F\ n \subseteq S \wedge \varphi\ (F\ n)$  and Fsub:  $\bigwedge n. F\ (\text{Suc}\ n) \subseteq F\ n$ 
  show  $\bigcap(F' \text{ UNIV}) \neq \{\} \wedge \bigcap(F' \text{ UNIV}) \subseteq S \wedge \varphi(\bigcap(F' \text{ UNIV}))$ 
  proof (intro conjI)
    show  $\bigcap(F' \text{ UNIV}) \neq \{\}$ 
    by (metis F Fsub <compact S> cloF closed_Int_compact compact_nest inf.orderE lift_Suc_antimono_le)
    show  $\bigcap(F' \text{ UNIV}) \subseteq S$ 
    using F by blast
    show  $\varphi(\bigcap(F' \text{ UNIV}))$ 
    by (metis F Fsub  $\varphi$  <compact S> cloF closed_Int_compact inf.orderE)
  qed
next
  show  $S \neq \{\} \wedge S \subseteq S \wedge \varphi\ S$ 
  by (simp add: assms)
qed (meson assms compact_imp_closed seq_compact_closed_subset seq_compact_eq_compact)+

```

6.40.2 Arcwise Connections

6.40.3 Density of points with dyadic rational coordinates

proposition *closure_dyadic_rationals:*

closure $(\bigcup k. \bigcup f \in \text{Basis} \rightarrow \mathbb{Z}. \{ \sum i :: 'a :: \text{euclidean\_space} \in \text{Basis}. (f\ i / 2^k) *_R i \}) = \text{UNIV}$

proof –

have $x \in \text{closure}(\bigcup k. \bigcup f \in \text{Basis} \rightarrow \mathbb{Z}. \{ \sum i \in \text{Basis}. (f\ i / 2^k) *_R i \})$ **for** $x :: 'a$

proof (clarsimp simp: closure_approachable)

fix $e :: \text{real}$

assume $e > 0$

then obtain k **where** $k: (1/2)^k < e / \text{DIM}('a)$

by (meson DIM_positive divide_less_eq_1_pos of_nat_0_less_iff one_less_numeral_iff real_arch_pow_inv semiring_norm(76) zero_less_divide_iff zero_less_numeral)

have $\text{dist}(\sum i \in \text{Basis}. (\text{real\_of\_int}\ \lfloor 2^k * (x \cdot i) \rfloor / 2^k) *_R i) \ x =$

$\text{dist}(\sum i \in \text{Basis}. (\text{real\_of\_int}\ \lfloor 2^k * (x \cdot i) \rfloor / 2^k) *_R i) (\sum i \in \text{Basis}. (x \cdot$

```

i) *R i)
  by (simp add: euclidean_representation)
  also have ... = norm (( $\sum_{i \in \text{Basis.}} (\text{real\_of\_int } \lfloor 2^k * (x \cdot i) \rfloor / 2^k) *_{\text{R}} i -$ 
( $x \cdot i) *_{\text{R}} i$ ))
  by (simp add: dist_norm sum_subtractf)
  also have ...  $\leq \text{DIM}('a) * ((1/2)^k)$ 
  proof (rule sum_norm_bound, simp add: algebra_simps)
    fix i::'a
    assume i  $\in \text{Basis}$ 
    then have norm (( $\text{real\_of\_int } \lfloor x \cdot i * 2^k \rfloor / 2^k) *_{\text{R}} i - (x \cdot i) *_{\text{R}} i =$ 
 $|\text{real\_of\_int } \lfloor x \cdot i * 2^k \rfloor / 2^k - x \cdot i|$ )
      by (simp add: scaleR_left_diff_distrib [symmetric])
    also have ...  $\leq (1/2)^k$ 
    by (simp add: divide_simps) linarith
    finally show norm (( $\text{real\_of\_int } \lfloor x \cdot i * 2^k \rfloor / 2^k) *_{\text{R}} i - (x \cdot i) *_{\text{R}} i) \leq$ 
 $(1/2)^k$  .
  qed
  also have ...  $< \text{DIM}('a) * (e / \text{DIM}('a))$ 
  using DIM_positive k linordered_comm_semiring_strict_class.comm_mult_strict_left_mono
of_nat_0_less_iff by blast
  also have ... = e
  by simp
  finally have dist ( $\sum_{i \in \text{Basis.}} (\lfloor 2^k * (x \cdot i) \rfloor / 2^k) *_{\text{R}} i$ ) x < e .
  with Ints_of_int
  show  $\exists k. \exists f \in \text{Basis} \rightarrow \mathbb{Z}. \text{dist } (\sum_{b \in \text{Basis.}} (f b / 2^k) *_{\text{R}} b)$  x < e
  by fastforce
  qed
  then show ?thesis by auto
qed

corollary closure_rational_coordinates:
  closure ( $\bigcup f \in \text{Basis} \rightarrow \mathbb{Q}. \{ \sum i :: 'a :: \text{euclidean\_space} \in \text{Basis. } f i *_{\text{R}} i \}$ ) =
UNIV
proof -
  have *: ( $\bigcup k. \bigcup f \in \text{Basis} \rightarrow \mathbb{Z}. \{ \sum i :: 'a \in \text{Basis. } (f i / 2^k) *_{\text{R}} i \}$ )
 $\subseteq (\bigcup f \in \text{Basis} \rightarrow \mathbb{Q}. \{ \sum i \in \text{Basis. } f i *_{\text{R}} i \})$ 
  proof clarsimp
    fix k and f :: 'a  $\Rightarrow \text{real}$ 
    assume f: f  $\in \text{Basis} \rightarrow \mathbb{Z}$ 
    show  $\exists x \in \text{Basis} \rightarrow \mathbb{Q}. (\sum i \in \text{Basis. } (f i / 2^k) *_{\text{R}} i) = (\sum i \in \text{Basis. } x i *_{\text{R}}$ 
i)
    apply (rule_tac x= $\lambda i. f i / 2^k$  in bexI)
    using Ints_subset_Rats f by auto
  qed
  show ?thesis
  using closure_dyadic_rationals closure_mono [OF *] by blast
qed

lemma closure_dyadic_rationals_in_convex_set:

```

$\llbracket \text{convex } S; \text{interior } S \neq \{\} \rrbracket$
 $\implies \text{closure}(S \cap$
 $\quad (\bigcup k. \bigcup f \in \text{Basis} \rightarrow \mathbb{Z}.$
 $\quad \{ \sum i :: 'a :: \text{euclidean\_space} \in \text{Basis}. (f \ i / 2^k) *_R i \})) =$
 $\quad \text{closure } S$
by (simp add: closure_dyadic_rationals closure_convex_Int_superset)

lemma closure_rationals_in_convex_set:

$\llbracket \text{convex } S; \text{interior } S \neq \{\} \rrbracket$
 $\implies \text{closure}(S \cap (\bigcup f \in \text{Basis} \rightarrow \mathbb{Q}. \{ \sum i :: 'a :: \text{euclidean\_space} \in \text{Basis}. f \ i$
 $*_R i \})) =$
 $\quad \text{closure } S$
by (simp add: closure_rational_coordinates closure_convex_Int_superset)

Every path between distinct points contains an arc, and hence path connection is equivalent to arcwise connection for distinct points. The proof is based on Whyburn's "Topological Analysis".

lemma closure_dyadic_rationals_in_convex_set_pos_1:

fixes $S :: \text{real set}$
assumes $\text{convex } S$ **and** $\text{intnz: interior } S \neq \{\}$ **and** $\text{pos: } \bigwedge x. x \in S \implies 0 \leq x$
shows $\text{closure}(S \cap (\bigcup k \ m. \{ \text{of\_nat } m / 2^k \})) = \text{closure } S$
proof –
have $\exists m. f \ 1 / 2^k = \text{real } m / 2^k$ **if** $(f \ 1) / 2^k \in S$ **for** k **and** $f :: \text{real}$
 $\implies \text{real}$
using **that** **by** (force simp: Ints_def zero_le_divide_iff power_le_zero_eq dest:
 $\text{pos\_zero\_le\_imp\_eq\_int}$)
then have $S \cap (\bigcup k \ m. \{ \text{real } m / 2^k \}) = S \cap$
 $(\bigcup k. \bigcup f \in \text{Basis} \rightarrow \mathbb{Z}. \{ \sum i \in \text{Basis}. (f \ i / 2^k) *_R i \})$
by force
then show ?thesis
using closure_dyadic_rationals_in_convex_set [OF <convex S> intnz] **by** simp
qed

definition dyadics :: $'a :: \text{field\_char\_0 set}$ **where** $\text{dyadics} \equiv \bigcup k \ m. \{ \text{of\_nat } m / 2^k \}$

lemma real_in_dyadics [simp]: $\text{real } m \in \text{dyadics}$

by (simp add: dyadics_def) (metis divide_numeral_1 numeral_One power_0)

lemma nat_neq_4k1: $\text{of\_nat } m \neq (4 * \text{of\_nat } k + 1) / (2 * 2^n :: 'a :: \text{field\_char\_0})$

proof

assume $\text{of\_nat } m = (4 * \text{of\_nat } k + 1) / (2 * 2^n :: 'a)$
then have $\text{of\_nat } (m * (2 * 2^n)) = (\text{of\_nat } (\text{Suc } (4 * k)) :: 'a)$
by (simp add: field_split_simps)
then have $m * (2 * 2^n) = \text{Suc } (4 * k)$
using of_nat_eq_iff **by** blast
then have $\text{odd } (m * (2 * 2^n))$
by simp


```

    then show False
      by simp
  qed

```

```

lemma nat_neq_4k3: of_nat m ≠ (4 * of_nat k + 3) / (2 * 2^n :: 'a::field_char_0)
proof
  assume of_nat m = (4 * of_nat k + 3) / (2 * 2^n :: 'a)
  then have of_nat (m * (2 * 2^n)) = (of_nat (4 * k + 3) :: 'a)
    by (simp add: field_split_simps)
  then have m * (2 * 2^n) = (4 * k) + 3
    using of_nat_eq_iff by blast
  then have odd (m * (2 * 2^n))
    by simp
  then show False
    by simp
  qed

```

```

lemma iff_4k:
  assumes r = real k odd k
  shows (4 * real m + r) / (2 * 2^n) = (4 * real m' + r) / (2 * 2^n') ⟷
  m=m' ∧ n=n'
proof -
  { assume (4 * real m + r) / (2 * 2^n) = (4 * real m' + r) / (2 * 2^n')
    then have real ((4 * m + k) * (2 * 2^n')) = real ((4 * m' + k) * (2 * 2^n))
      using assms by (auto simp: field_simps)
    then have (4 * m + k) * (2 * 2^n') = (4 * m' + k) * (2 * 2^n)
      using of_nat_eq_iff by blast
    then have (4 * m + k) * (2^n') = (4 * m' + k) * (2^n)
      by linarith
    then obtain 4*m + k = 4*m' + k n=n'
      using prime_power_cancel2 [OF two_is_prime_nat] assms
      by (metis even_mult_iff even_numeral odd_add)
    then have m=m' n=n'
      by auto
  }
  then show ?thesis by blast
  qed

```

```

lemma neq_4k1_k43: (4 * real m + 1) / (2 * 2^n) ≠ (4 * real m' + 3) / (2 *
2^n')
proof
  assume (4 * real m + 1) / (2 * 2^n) = (4 * real m' + 3) / (2 * 2^n')
  then have real (Suc (4 * m) * (2 * 2^n')) = real ((4 * m' + 3) * (2 * 2^n))
    by (auto simp: field_simps)
  then have Suc (4 * m) * (2 * 2^n') = (4 * m' + 3) * (2 * 2^n)
    using of_nat_eq_iff by blast
  then have Suc (4 * m) * (2^n') = (4 * m' + 3) * (2^n)
    by linarith
  then have Suc (4 * m) = (4 * m' + 3)

```

```

    by (rule prime_power_cancel2 [OF two_is_prime_nat]) auto
  then have  $1 + 2 * m' = 2 * m$ 
    using  $\langle \text{Suc } (4 * m) = 4 * m' + 3 \rangle$  by linarith
  then show False
    using even_Suc by presburger
qed

lemma dyadic_413_cases:
  obtains (of_nat m::'a::field_char_0) /  $2^k \in \text{Nats}$ 
    |  $m' k'$  where  $k' < k$  (of_nat m::'a) /  $2^k = \text{of\_nat } (4 * m' + 1) / 2^{\text{Suc } k'}$ 
    |  $m' k'$  where  $k' < k$  (of_nat m::'a) /  $2^k = \text{of\_nat } (4 * m' + 3) / 2^{\text{Suc } k'}$ 
proof (cases m > 0)
  case False
    then have m=0 by simp
    with that show ?thesis by auto
  next
    case True
      obtain  $k' m'$  where  $m': \text{odd } m'$  and  $k': m = m' * 2^{k'}$ 
        using prime_power_canonical [OF two_is_prime_nat True] by blast
      then obtain  $q r$  where  $q: m' = 4 * q + r$  and  $r: r < 4$ 
        by (metis not_add_less2 split_div zero_neq_numeral)
      show ?thesis
      proof (cases  $k \leq k'$ )
        case True
          have (of_nat m::'a) /  $2^k = \text{of\_nat } m' * (2^{k'} / 2^k)$ 
            using  $k'$  by (simp add: field_simps)
          also have  $\dots = (\text{of\_nat } m': 'a) * 2^{(k'-k)}$ 
            using  $k'$  True by (simp add: power_diff)
          also have  $\dots \in \mathbb{N}$ 
            by (metis Nats_mult of_nat_in_Nats of_nat_numeral of_nat_power)
          finally show ?thesis by (auto simp: that)
        case False
          then obtain  $kd$  where  $kd: \text{Suc } kd = k - k'$ 
            using Suc_diff_Suc not_less by blast
          have (of_nat m::'a) /  $2^k = \text{of\_nat } m' * (2^{k'} / 2^k)$ 
            using  $k'$  by (simp add: field_simps)
          also have  $\dots = (\text{of\_nat } m': 'a) / 2^{(k-k')}$ 
            using  $k'$  False by (simp add: power_diff)
          also have  $\dots = ((\text{of\_nat } r + 4 * \text{of\_nat } q)::'a) / 2^{(k-k')}$ 
            using  $q$  by force
          finally have meq: (of_nat m::'a) /  $2^k = (\text{of\_nat } r + 4 * \text{of\_nat } q) / 2^{(k-k')}$  .
          have  $r \neq 0$   $r \neq 2$ 
            using  $q m'$  by presburger+
          with  $r$  consider  $r = 1 \mid r = 3$ 
            by linarith
          then show ?thesis
            proof cases

```

```

    assume r = 1
    with meq kd that(2) [of kd q] show ?thesis
    by simp
  next
    assume r = 3
    with meq kd that(3) [of kd q] show ?thesis
    by simp
  qed
qed
qed

```

lemma *dyadics_iff*:

```

  (dyadics :: 'a::field_char_0 set) =
    Nats ∪ (⋃ k m. {of_nat (4*m + 1) / 2^Suc k}) ∪ (⋃ k m. {of_nat (4*m +
3) / 2^Suc k})
  (is _ = ?rhs)

```

proof

```

  show dyadics ⊆ ?rhs
  unfolding dyadics_def
  apply clarify
  apply (rule dyadic_413_cases, force+)
  done
next
  have range of_nat ⊆ (⋃ k m. {(of_nat m::'a) / 2^k})
  by clarsimp (metis divide_numeral_1 numeral_One power_0)
  moreover have ⋀ k m. ∃ k' m'. ((1::'a) + 4 * of_nat m) / 2^Suc k = of_nat
m' / 2^k'
  by (metis (no_types) of_nat_Suc of_nat_mult of_nat_numeral)
  moreover have ⋀ k m. ∃ k' m'. (4 * of_nat m + (3::'a)) / 2^Suc k = of_nat
m' / 2^k'
  by (metis of_nat_add of_nat_mult of_nat_numeral)
  ultimately show ?rhs ⊆ dyadics
  by (auto simp: dyadics_def Nats_def)
qed

```

```

function (domintros) dyad_rec :: [nat ⇒ 'a, 'a ⇒ 'a, 'a ⇒ 'a, real] ⇒ 'a where
  dyad_rec b l r (real m) = b m
  | dyad_rec b l r ((4 * real m + 1) / 2^(Suc n)) = l (dyad_rec b l r ((2*m +
1) / 2^n))
  | dyad_rec b l r ((4 * real m + 3) / 2^(Suc n)) = r (dyad_rec b l r ((2*m +
1) / 2^n))
  | x ∉ dyadics ⇒ dyad_rec b l r x = undefined
using iff_4k [of _ 1] iff_4k [of _ 3]
  apply (simp_all add: nat_neq_4k1 nat_neq_4k3 neq_4k1_k43 dyadics_iff
Nats_def)
  by (fastforce simp: field_simps)+

```

```

lemma dyadics_levels: dyadics = ( $\bigcup K. \bigcup k < K. \bigcup m. \{of\_nat\ m / 2^k\}$ )
  unfolding dyadics_def by auto

lemma dyad_rec_level_termination:
  assumes  $k < K$ 
  shows dyad_rec_dom( $b, l, r, real\ m / 2^k$ )
  using assms
proof (induction  $K$  arbitrary:  $k\ m$ )
  case 0
  then show ?case by auto
next
  case (Suc  $K$ )
  then consider  $k = K \mid k < K$ 
  using less_antisym by blast
  then show ?case
  proof cases
    assume  $k = K$ 
    show ?case
  proof (rule dyadic_413_cases [of  $m\ k$ , where 'a=real'])
    show  $real\ m / 2^k \in \mathbb{N} \implies dyad\_rec\_dom\ (b, l, r, real\ m / 2^k)$ 
    by (force simp: Nats_def nat_neq_4k1 nat_neq_4k3 intro: dyad_rec.domintros)
    show ?case if  $k' < k$  and  $eq: real\ m / 2^k = real\ (4 * m' + 1) / 2^{Suc\ k'}$ 
  for  $m'\ k'$ 
  proof -
    have dyad_rec_dom ( $b, l, r, (4 * real\ m' + 1) / 2^{Suc\ k'}$ )
    proof (rule dyad_rec.domintros)
      fix  $m\ n$ 
      assume  $(4 * real\ m' + 1) / (2 * 2^{k'}) = (4 * real\ m + 1) / (2 * 2^n)$ 
      then have  $m' = m\ k' = n$  using iff_4k [of _ 1]
      by auto
      have dyad_rec_dom ( $b, l, r, real\ (2 * m + 1) / 2^{k'}$ )
      using Suc.IH  $\langle k = K \rangle \langle k' < k \rangle$  by blast
      then show dyad_rec_dom ( $b, l, r, (2 * real\ m + 1) / 2^n$ )
      using  $\langle k' = n \rangle$  by (auto simp: algebra_simps)
    next
      fix  $m\ n$ 
      assume  $(4 * real\ m' + 1) / (2 * 2^{k'}) = (4 * real\ m + 3) / (2 * 2^n)$ 
      then have False
      by (metis neq_4k1_k43)
      then show dyad_rec_dom ( $b, l, r, (2 * real\ m + 1) / 2^n$ ) ..
    qed
    then show ?case by (simp add: eq add_ac)
  qed
  show ?case if  $k' < k$  and  $eq: real\ m / 2^k = real\ (4 * m' + 3) / 2^{Suc\ k'}$ 
  for  $m'\ k'$ 
  proof -
    have dyad_rec_dom ( $b, l, r, (4 * real\ m' + 3) / 2^{Suc\ k'}$ )
    proof (rule dyad_rec.domintros)
      fix  $m\ n$ 

```

```

    assume  $(4 * \text{real } m' + 3) / (2 * 2^{\wedge} k') = (4 * \text{real } m + 1) / (2 * 2^{\wedge} n)$ 
    then have False
      by (metis neq_4k1_k43)
    then show  $\text{dyad\_rec\_dom } (b, l, r, (2 * \text{real } m + 1) / 2^{\wedge} n) ..$ 
  next
    fix  $m\ n$ 
    assume  $(4 * \text{real } m' + 3) / (2 * 2^{\wedge} k') = (4 * \text{real } m + 3) / (2 * 2^{\wedge} n)$ 
    then have  $m' = m\ k' = n$  using  $\text{iff\_4k } [of\_ 3]$ 
      by auto
    have  $\text{dyad\_rec\_dom } (b, l, r, \text{real } (2 * m + 1) / 2^{\wedge} k')$ 
      using  $\text{Suc.IH } \langle k = K \rangle \langle k' < k \rangle$  by blast
    then show  $\text{dyad\_rec\_dom } (b, l, r, (2 * \text{real } m + 1) / 2^{\wedge} n)$ 
      using  $\langle k' = n \rangle$  by (auto simp: algebra_simps)
    qed
    then show ?case by (simp add: eq_add_ac)
  qed
qed
next
  assume  $k < K$ 
  then show ?case
    using  $\text{Suc.IH}$  by blast
qed
qed

```

lemma $\text{dyad\_rec\_termination}: x \in \text{dyadics} \implies \text{dyad\_rec\_dom}(b, l, r, x)$
 by (auto simp: dyadics_levels intro: dyad_rec_level_termination)

lemma $\text{dyad\_rec\_of\_nat } [simp]: \text{dyad\_rec } b\ l\ r\ (\text{real } m) = b\ m$
 by (simp add: dyad_rec.psimps dyad_rec_termination)

lemma $\text{dyad\_rec\_41 } [simp]: \text{dyad\_rec } b\ l\ r\ ((4 * \text{real } m + 1) / 2^{\wedge} (\text{Suc } n)) = l$
 $(\text{dyad\_rec } b\ l\ r\ ((2 * m + 1) / 2^{\wedge} n))$

proof (rule dyad_rec.psimps)

show $\text{dyad\_rec\_dom } (b, l, r, (4 * \text{real } m + 1) / 2^{\wedge} \text{Suc } n)$

by (metis add.commute dyad_rec_level_termination lessI of_nat_Suc of_nat_mult of_nat_numeral)

qed

lemma $\text{dyad\_rec\_43 } [simp]: \text{dyad\_rec } b\ l\ r\ ((4 * \text{real } m + 3) / 2^{\wedge} (\text{Suc } n)) =$
 $r\ (\text{dyad\_rec } b\ l\ r\ ((2 * m + 1) / 2^{\wedge} n))$

proof (rule dyad_rec.psimps)

show $\text{dyad\_rec\_dom } (b, l, r, (4 * \text{real } m + 3) / 2^{\wedge} \text{Suc } n)$

by (metis dyad_rec_level_termination lessI of_nat_add of_nat_mult of_nat_numeral)

qed

lemma $\text{dyad\_rec\_41\_times2}:$

assumes $n > 0$

shows $\text{dyad_rec } b\ l\ r\ (2 * ((4 * \text{real } m + 1) / 2^{\wedge} \text{Suc } n)) = l\ (\text{dyad_rec } b\ l\ r\$

$(2 * (2 * \text{real } m + 1) / 2^n)$
proof –
obtain n' **where** $n': n = \text{Suc } n'$
using *assms not0_implies_Suc* **by** *blast*
have $\text{dyad\_rec } b \ l \ r \ (2 * ((4 * \text{real } m + 1) / 2^{\text{Suc } n})) = \text{dyad\_rec } b \ l \ r \ ((2 * (4 * \text{real } m + 1)) / (2 * 2^n))$
by *auto*
also have $\dots = \text{dyad\_rec } b \ l \ r \ ((4 * \text{real } m + 1) / 2^n)$
by (*subst mult_divide_mult_cancel_left*) *auto*
also have $\dots = l \ (\text{dyad\_rec } b \ l \ r \ ((2 * \text{real } m + 1) / 2^{n'}))$
by (*simp add: add.commute [of 1] n' del: power_Suc*)
also have $\dots = l \ (\text{dyad\_rec } b \ l \ r \ ((2 * (2 * \text{real } m + 1)) / (2 * 2^{n'})))$
by (*subst mult_divide_mult_cancel_left*) *auto*
also have $\dots = l \ (\text{dyad\_rec } b \ l \ r \ (2 * (2 * \text{real } m + 1) / 2^n))$
by (*simp add: add.commute n'*)
finally show *?thesis* .
qed

lemma *dyad_rec_43_times2*:
assumes $n > 0$
shows $\text{dyad\_rec } b \ l \ r \ (2 * ((4 * \text{real } m + 3) / 2^{\text{Suc } n})) = r \ (\text{dyad\_rec } b \ l \ r \ (2 * (2 * \text{real } m + 1) / 2^n))$
proof –
obtain n' **where** $n': n = \text{Suc } n'$
using *assms not0_implies_Suc* **by** *blast*
have $\text{dyad\_rec } b \ l \ r \ (2 * ((4 * \text{real } m + 3) / 2^{\text{Suc } n})) = \text{dyad\_rec } b \ l \ r \ ((2 * (4 * \text{real } m + 3)) / (2 * 2^n))$
by *auto*
also have $\dots = \text{dyad\_rec } b \ l \ r \ ((4 * \text{real } m + 3) / 2^n)$
by (*subst mult_divide_mult_cancel_left*) *auto*
also have $\dots = r \ (\text{dyad\_rec } b \ l \ r \ ((2 * \text{real } m + 1) / 2^{n'}))$
by (*simp add: n' del: power_Suc*)
also have $\dots = r \ (\text{dyad\_rec } b \ l \ r \ ((2 * (2 * \text{real } m + 1)) / (2 * 2^{n'})))$
by (*subst mult_divide_mult_cancel_left*) *auto*
also have $\dots = r \ (\text{dyad\_rec } b \ l \ r \ (2 * (2 * \text{real } m + 1) / 2^n))$
by (*simp add: n'*)
finally show *?thesis* .
qed

definition *dyad_rec2*
where $\text{dyad\_rec2 } u \ v \ lc \ rc \ x =$
 $\text{dyad\_rec } (\lambda z. (u, v)) \ (\lambda(a, b). (a, lc \ a \ b \ (\text{midpoint } a \ b))) \ (\lambda(a, b). (rc \ a \ b \ (\text{midpoint } a \ b), b)) \ (2 * x)$

abbreviation *leftrec* **where** $\text{leftrec } u \ v \ lc \ rc \ x \equiv \text{fst } (\text{dyad\_rec2 } u \ v \ lc \ rc \ x)$
abbreviation *rightrec* **where** $\text{rightrec } u \ v \ lc \ rc \ x \equiv \text{snd } (\text{dyad\_rec2 } u \ v \ lc \ rc \ x)$

lemma *leftrec_base*: $\text{leftrec } u \ v \ lc \ rc \ (\text{real } m / 2) = u$
by (*simp add: dyad_rec2_def*)

lemma *leftrec_41*: $n > 0 \implies \text{leftrec } u \ v \ \text{lc } \text{rc } ((4 * \text{real } m + 1) / 2^{\wedge}(\text{Suc } n))$
 $= \text{leftrec } u \ v \ \text{lc } \text{rc } ((2 * \text{real } m + 1) / 2^{\wedge}n)$
unfolding *dyad_rec2_def dyad_rec_41_times2*
by (*simp add: case_prod_beta*)

lemma *leftrec_43*: $n > 0 \implies$
 $\text{leftrec } u \ v \ \text{lc } \text{rc } ((4 * \text{real } m + 3) / 2^{\wedge}(\text{Suc } n)) =$
 $\text{rc } (\text{leftrec } u \ v \ \text{lc } \text{rc } ((2 * \text{real } m + 1) / 2^{\wedge}n)) (\text{rightrec } u \ v \ \text{lc } \text{rc } ((2$
 $* \text{real } m + 1) / 2^{\wedge}n))$
 $(\text{midpoint } (\text{leftrec } u \ v \ \text{lc } \text{rc } ((2 * \text{real } m + 1) / 2^{\wedge}n)) (\text{rightrec } u \ v \ \text{lc}$
 $\text{rc } ((2 * \text{real } m + 1) / 2^{\wedge}n)))$
unfolding *dyad_rec2_def dyad_rec_43_times2*
by (*simp add: case_prod_beta*)

lemma *rightrec_base*: $\text{rightrec } u \ v \ \text{lc } \text{rc } (\text{real } m / 2) = v$
by (*simp add: dyad_rec2_def*)

lemma *rightrec_41*: $n > 0 \implies$
 $\text{rightrec } u \ v \ \text{lc } \text{rc } ((4 * \text{real } m + 1) / 2^{\wedge}(\text{Suc } n)) =$
 $\text{lc } (\text{leftrec } u \ v \ \text{lc } \text{rc } ((2 * \text{real } m + 1) / 2^{\wedge}n)) (\text{rightrec } u \ v \ \text{lc } \text{rc } ((2$
 $* \text{real } m + 1) / 2^{\wedge}n))$
 $(\text{midpoint } (\text{leftrec } u \ v \ \text{lc } \text{rc } ((2 * \text{real } m + 1) / 2^{\wedge}n)) (\text{rightrec } u \ v \ \text{lc}$
 $\text{rc } ((2 * \text{real } m + 1) / 2^{\wedge}n)))$
unfolding *dyad_rec2_def dyad_rec_41_times2*
by (*simp add: case_prod_beta*)

lemma *rightrec_43*: $n > 0 \implies \text{rightrec } u \ v \ \text{lc } \text{rc } ((4 * \text{real } m + 3) / 2^{\wedge}(\text{Suc } n))$
 $= \text{rightrec } u \ v \ \text{lc } \text{rc } ((2 * \text{real } m + 1) / 2^{\wedge}n)$
unfolding *dyad_rec2_def dyad_rec_43_times2*
by (*simp add: case_prod_beta*)

lemma *dyadics_in_open_unit_interval*:
 $\{0 <..< 1\} \cap (\bigcup k \ m. \{\text{real } m / 2^{\wedge}k\}) = (\bigcup k. \bigcup m \in \{0 <..< 2^{\wedge}k\}. \{\text{real } m /$
 $2^{\wedge}k\})$
by (*auto simp: field_split_simps*)

theorem *homeomorphic_monotone_image_interval*:
fixes $f :: \text{real} \Rightarrow 'a :: \{\text{real\_normed\_vector}, \text{complete\_space}\}$
assumes *cont_f*: *continuous_on* $\{0..1\}$ f
and *conn*: $\bigwedge y. \text{connected } (\{0..1\} \cap f^{-1} \{y\})$
and *f_1not0*: $f \ 1 \neq f \ 0$
shows $(f^{-1} \{0..1\}) \text{ homeomorphic } \{0..1 :: \text{real}\}$
proof –
have $\exists c \ d. a \leq c \wedge c \leq m \wedge m \leq d \wedge d \leq b \wedge$
 $(\forall x \in \{c..d\}. f \ x = f \ m) \wedge$
 $(\forall x \in \{a..<c\}. (f \ x \neq f \ m)) \wedge$

```

      (∀ x ∈ {d<..b}. (f x ≠ f m)) ∧
      (∀ x ∈ {a..<c}. ∀ y ∈ {d<..b}. f x ≠ f y)
    if m: m ∈ {a..b} and ab01: {a..b} ⊆ {0..1} for a b m
  proof -
    have comp: compact (f -' {f m} ∩ {0..1})
      by (simp add: compact_eq_bounded_closed bounded_Int closed_vimage_Int
        cont_f)
    obtain c0 d0 where cd0: {0..1} ∩ f -' {f m} = {c0..d0}
      using connected_compact_interval_1 [of {0..1} ∩ f -' {f m}] conn comp
      by (metis Int_commute)
    with that have m ∈ cbox c0 d0
      by auto
    obtain c d where cd: {a..b} ∩ f -' {f m} = {c..d}
      using ab01 cd0
      by (rule_tac c=max a c0 and d=min b d0 in that) auto
    then have cdab: {c..d} ⊆ {a..b}
      by blast
    show ?thesis
  proof (intro exI conjI ballI)
    show a ≤ c d ≤ b
      using cdab cd m by auto
    show c ≤ m m ≤ d
      using cd m by auto
    show ∧x. x ∈ {c..d} ⇒ f x = f m
      using cd by blast
    show f x ≠ f m if x ∈ {a..<c} for x
      using that m cd [THEN equalityD1, THEN subsetD] ⟨c ≤ m⟩ by force
    show f x ≠ f m if x ∈ {d<..b} for x
      using that m cd [THEN equalityD1, THEN subsetD, of x] ⟨m ≤ d⟩ by force
    show f x ≠ f y if x ∈ {a..<c} y ∈ {d<..b} for x y
      proof (cases f x = f m ∨ f y = f m)
        case True
        then show ?thesis
          using ⟨∧x. x ∈ {a..<c} ⇒ f x ≠ f m⟩ that by auto
      next
        case False
        have False if f x = f y
          proof -
            have x ≤ m m ≤ y
              using ⟨c ≤ m⟩ ⟨x ∈ {a..<c}⟩ ⟨m ≤ d⟩ ⟨y ∈ {d<..b}⟩ by auto
            then have x ∈ ({0..1} ∩ f -' {f y}) y ∈ ({0..1} ∩ f -' {f y})
              using ⟨x ∈ {a..<c}⟩ ⟨y ∈ {d<..b}⟩ ab01 by (auto simp: that)
            then have m ∈ ({0..1} ∩ f -' {f y})
              by (meson ⟨m ≤ y⟩ ⟨x ≤ m⟩ is_interval_connected_1 conn [of f y]
                is_interval_1)
            with False show False by auto
          qed
        then show ?thesis by auto
      qed
  qed

```



```

qed
qed
then obtain leftcut rightcut where LR:
   $\bigwedge a\ b\ m. \llbracket m \in \{a..b\}; \{a..b\} \subseteq \{0..1\} \rrbracket \implies$ 
     $(a \leq \text{leftcut } a\ b\ m \wedge \text{leftcut } a\ b\ m \leq m \wedge m \leq \text{rightcut } a\ b\ m \wedge \text{rightcut}$ 
 $a\ b\ m \leq b \wedge$ 
     $(\forall x \in \{\text{leftcut } a\ b\ m.. \text{rightcut } a\ b\ m\}. f\ x = f\ m) \wedge$ 
     $(\forall x \in \{a..<\text{leftcut } a\ b\ m\}. f\ x \neq f\ m) \wedge$ 
     $(\forall x \in \{\text{rightcut } a\ b\ m<..b\}. f\ x \neq f\ m) \wedge$ 
     $(\forall x \in \{a..<\text{leftcut } a\ b\ m\}. \forall y \in \{\text{rightcut } a\ b\ m<..b\}. f\ x \neq f\ y))$ 
  apply atomize
  apply (clarsimp simp only: imp_conjL [symmetric] choice_iff choice_iff')
  apply (rule that, blast)
done
then have left_right:  $\bigwedge a\ b\ m. \llbracket m \in \{a..b\}; \{a..b\} \subseteq \{0..1\} \rrbracket \implies a \leq \text{leftcut } a$ 
 $b\ m \wedge \text{rightcut } a\ b\ m \leq b$ 
  and left_right_m:  $\bigwedge a\ b\ m. \llbracket m \in \{a..b\}; \{a..b\} \subseteq \{0..1\} \rrbracket \implies \text{leftcut } a\ b\ m$ 
 $\leq m \wedge m \leq \text{rightcut } a\ b\ m$ 
  by auto
  have left_neg:  $\llbracket a \leq x; x < \text{leftcut } a\ b\ m; a \leq m; m \leq b; \{a..b\} \subseteq \{0..1\} \rrbracket \implies$ 
 $f\ x \neq f\ m$ 
  and right_neg:  $\llbracket \text{rightcut } a\ b\ m < x; x \leq b; a \leq m; m \leq b; \{a..b\} \subseteq \{0..1\} \rrbracket$ 
 $\implies f\ x \neq f\ m$ 
  and left_right_neg:  $\llbracket a \leq x; x < \text{leftcut } a\ b\ m; \text{rightcut } a\ b\ m < y; y \leq b; a \leq$ 
 $m; m \leq b; \{a..b\} \subseteq \{0..1\} \rrbracket \implies f\ x \neq f\ m$ 
  and feqm:  $\llbracket \text{leftcut } a\ b\ m \leq x; x \leq \text{rightcut } a\ b\ m; a \leq m; m \leq b; \{a..b\} \subseteq$ 
 $\{0..1\} \rrbracket$ 
 $\implies f\ x = f\ m$  for  $a\ b\ m\ x\ y$ 
  by (meson atLeastAtMost_iff greaterThanAtMost_iff atLeastLessThan_iff LR)+
  have f_eqI:  $\bigwedge a\ b\ m\ x\ y. \llbracket \text{leftcut } a\ b\ m \leq x; x \leq \text{rightcut } a\ b\ m; \text{leftcut } a\ b\ m \leq$ 
 $y; y \leq \text{rightcut } a\ b\ m;$ 
 $a \leq m; m \leq b; \{a..b\} \subseteq \{0..1\} \rrbracket \implies f\ x = f\ y$ 
  by (metis feqm)
define u where  $u \equiv \text{rightcut } 0\ 1\ 0$ 
have lc[simp]:  $\text{leftcut } 0\ 1\ 0 = 0$  and u01:  $0 \leq u\ u \leq 1$ 
  using LR [of 0 0 1] by (auto simp: u_def)
have f0u:  $\bigwedge x. x \in \{0..u\} \implies f\ x = f\ 0$ 
  using LR [of 0 0 1] unfolding u_def [symmetric]
  by (metis <leftcut 0 1 0 = 0> atLeastAtMost_iff order_refl zero_le_one)
have fu1:  $\bigwedge x. x \in \{u<..1\} \implies f\ x \neq f\ 0$ 
  using LR [of 0 0 1] unfolding u_def [symmetric] by fastforce
define v where  $v \equiv \text{leftcut } u\ 1\ 1$ 
have rc[simp]:  $\text{rightcut } u\ 1\ 1 = 1$  and v01:  $u \leq v\ v \leq 1$ 
  using LR [of 1 u 1] u01 by (auto simp: v_def)
have fuv:  $\bigwedge x. x \in \{u..<v\} \implies f\ x \neq f\ 1$ 
  using LR [of 1 u 1] u01 v_def by fastforce
have f0v:  $\bigwedge x. x \in \{0..<v\} \implies f\ x \neq f\ 1$ 
  by (metis f_1not0 atLeastAtMost_iff atLeastLessThan_iff f0u fuv linear)
have fv1:  $\bigwedge x. x \in \{v..1\} \implies f\ x = f\ 1$ 

```

```

    using LR [of 1 u 1] u01 v_def by (metis atLeastAtMost_iff atLeastatMost_subset_iff
order_refl rc)
  define a where a  $\equiv$  leftrec u v leftcut rightcut
  define b where b  $\equiv$  rightrec u v leftcut rightcut
  define c where c  $\equiv$   $\lambda x$ . midpoint (a x) (b x)
  have a_real [simp]: a (real j) = u for j
    using a_def leftrec_base
  by (metis nonzero_mult_div_cancel_right of_nat_mult of_nat_numeral zero_neq_numeral)
  have b_real [simp]: b (real j) = v for j
    using b_def rightrec_base
  by (metis nonzero_mult_div_cancel_right of_nat_mult of_nat_numeral zero_neq_numeral)
  have a41: a ((4 * real m + 1) / 2Suc n) = a ((2 * real m + 1) / 2n) if n >
0 for m n
    using that a_def leftrec_41 by blast
  have b41: b ((4 * real m + 1) / 2Suc n) =
    leftcut (a ((2 * real m + 1) / 2n))
      (b ((2 * real m + 1) / 2n))
      (c ((2 * real m + 1) / 2n)) if n > 0 for m n
    using that a_def b_def c_def rightrec_41 by blast
  have a43: a ((4 * real m + 3) / 2Suc n) =
    rightcut (a ((2 * real m + 1) / 2n))
      (b ((2 * real m + 1) / 2n))
      (c ((2 * real m + 1) / 2n)) if n > 0 for m n
    using that a_def b_def c_def leftrec_43 by blast
  have b43: b ((4 * real m + 3) / 2Suc n) = b ((2 * real m + 1) / 2n) if n >
0 for m n
    using that b_def rightrec_43 by blast
  have uabv: u  $\leq$  a (real m / 2n)  $\wedge$  a (real m / 2n)  $\leq$  b (real m / 2n)  $\wedge$ 
b (real m / 2n)  $\leq$  v for m n
  proof (induction n arbitrary: m)
    case 0
    then show ?case by (simp add: v01)
  next
    case (Suc n p)
    show ?case
    proof (cases even p)
      case True
      then obtain m where p = 2*m by (metis evenE)
      then show ?thesis
        by (simp add: Suc.IH)
    next
      case False
      then obtain m where m: p = 2*m + 1 by (metis oddE)
      show ?thesis
      proof (cases n)
        case 0
        then show ?thesis
          by (simp add: a_def b_def leftrec_base rightrec_base v01)
      next

```

```

case (Suc n')
then have n > 0 by simp
have a_le_c:  $a \text{ (real } m / 2^n) \leq c \text{ (real } m / 2^n)$  for m
  unfolding c_def by (metis Suc.IH ge_midpoint_1)
have c_le_b:  $c \text{ (real } m / 2^n) \leq b \text{ (real } m / 2^n)$  for m
  unfolding c_def by (metis Suc.IH le_midpoint_1)
have c_ge_u:  $c \text{ (real } m / 2^n) \geq u$  for m
  using Suc.IH a_le_c order_trans by blast
have c_le_v:  $c \text{ (real } m / 2^n) \leq v$  for m
  using Suc.IH c_le_b order_trans by blast
have a_ge_0:  $0 \leq a \text{ (real } m / 2^n)$  for m
  using Suc.IH order_trans u01(1) by blast
have b_le_1:  $b \text{ (real } m / 2^n) \leq 1$  for m
  using Suc.IH order_trans v01(2) by blast
have left_le: leftcut ( $a \text{ (real } m / 2^n)$ ) ( $b \text{ (real } m / 2^n)$ ) ( $c \text{ (real } m / 2^n)$ )  $\leq c \text{ (real } m / 2^n)$  for m
  by (simp add: LR a_ge_0 a_le_c b_le_1 c_le_b)
have right_ge: rightcut ( $a \text{ (real } m / 2^n)$ ) ( $b \text{ (real } m / 2^n)$ ) ( $c \text{ (real } m / 2^n)$ )  $\geq c \text{ (real } m / 2^n)$  for m
  by (simp add: LR a_ge_0 a_le_c b_le_1 c_le_b)
show ?thesis
proof (cases even m)
case True
  then obtain r where r:  $m = 2*r$  by (metis evenE)
  show ?thesis
    using order_trans [OF left_le c_le_v, of 1+2*r] Suc.IH [of m+1]
    using a_le_c [of m+1] c_le_b [of m+1] a_ge_0 [of m+1] b_le_1 [of
m+1] left_right <n > 0>
    by (simp_all add: r m add.commute [of 1] a41 b41 del: power_Suc)
next
case False
  then obtain r where r:  $m = 2*r + 1$  by (metis oddE)
  show ?thesis
    using order_trans [OF c_ge_u right_ge, of 1+2*r] Suc.IH [of m]
    using a_le_c [of m] c_le_b [of m] a_ge_0 [of m] b_le_1 [of m]
left_right <n > 0>
    apply (simp_all add: r m add.commute [of 3] a43 b43 del: power_Suc)
    by (simp add: add.commute)
qed
qed
qed
qed
have a_ge_0 [simp]:  $0 \leq a(m / 2^n)$  and b_le_1 [simp]:  $b(m / 2^n) \leq 1$  for
m::nat and n
  using uabv order_trans u01 v01 by blast+
then have b_ge_0 [simp]:  $0 \leq b(m / 2^n)$  and a_le_1 [simp]:  $a(m / 2^n) \leq 1$  for
m::nat and n
  using uabv order_trans by blast+
have alec [simp]:  $a(m / 2^n) \leq c(m / 2^n)$  and cleb [simp]:  $c(m / 2^n) \leq b(m$ 

```

```

/ 2^n) for m::nat and n
  by (auto simp: c_def ge_midpoint_1 le_midpoint_1 uabv)
  have c_ge_0 [simp]: 0 ≤ c(m / 2^n) and c_le_1 [simp]: c(m / 2^n) ≤ 1 for
m::nat and n
  using a_ge_0 alec b_le_1 cleb order_trans by blast+
  have [|d = m-n; odd j; |real i / 2^m - real j / 2^n| < 1/2^n|]
    ⇒ (a(j / 2^n)) ≤ (c(i / 2^m)) ∧ (c(i / 2^m)) ≤ (b(j / 2^n)) for d i j m
n
  proof (induction d arbitrary: j n rule: less_induct)
    case (less d j n)
    show ?case
    proof (cases m ≤ n)
      case True
      have |2^n| * |real i / 2^m - real j / 2^n| = 0
      proof (rule Ints_nonzero_abs_less1)
        have (real i * 2^n - real j * 2^m) / 2^m = (real i * 2^n) / 2^m - (real j
* 2^m) / 2^m
          using diff_divide_distrib by blast
        also have ... = (real i * 2^(n-m)) - (real j)
          using True by (auto simp: power_diff field_simps)
        also have ... ∈ ℤ
          by simp
        finally have (real i * 2^n - real j * 2^m) / 2^m ∈ ℤ .
        with True Ints_abs show |2^n| * |real i / 2^m - real j / 2^n| ∈ ℤ
          by (fastforce simp: field_split_simps)
        show ||2^n| * |real i / 2^m - real j / 2^n|| < 1
          using less.premis by (auto simp: field_split_simps)
      qed
    then have real i / 2^m = real j / 2^n
      by auto
    then show ?thesis
      by auto
  next
  case False
  then have n < m by auto
  obtain k where k: j = Suc (2*k)
    using ⟨odd j⟩ oddE by fastforce
  show ?thesis
  proof (cases n > 0)
    case False
    then have a (real j / 2^n) = u
      by simp
    also have ... ≤ c (real i / 2^m)
      using alec uabv by (blast intro: order_trans)
    finally have ac: a (real j / 2^n) ≤ c (real i / 2^m) .
    have c (real i / 2^m) ≤ v
      using cleb uabv by (blast intro: order_trans)
    also have ... = b (real j / 2^n)
      using False by simp
  qed

```

```

finally show ?thesis
  by (auto simp: ac)
next
case True show ?thesis
proof (cases  $i / 2^m j / 2^n$  rule: linorder_cases)
  case less
    moreover have  $\text{real } (4 * k + 1) / 2^{\text{Suc } n} + 1 / (2^{\text{Suc } n}) = \text{real } j / 2^n$ 
      using k by (force simp: field_split_simps)
    moreover have  $|\text{real } i / 2^m - j / 2^n| < 2 / (2^{\text{Suc } n})$ 
      using less.prem1 by simp
    ultimately have closer:  $|\text{real } i / 2^m - \text{real } (4 * k + 1) / 2^{\text{Suc } n}| < 1 / (2^{\text{Suc } n})$ 
      using less.prem1 by linarith
    have a ( $\text{real } (4 * k + 1) / 2^{\text{Suc } n} \leq c (i / 2^m) \wedge c (i / 2^m) \leq b (\text{real } (4 * k + 1) / 2^{\text{Suc } n})$ )
      proof (rule less.IH [OF _ refl])
        show  $m - \text{Suc } n < d$ 
          using  $\langle n < m \rangle$  diff_less_mono2 less.prem1(1) lessI by presburger
        show  $|\text{real } i / 2^m - \text{real } (4 * k + 1) / 2^{\text{Suc } n}| < 1 / 2^{\text{Suc } n}$ 
          using closer  $\langle n < m \rangle$   $\langle d = m - n \rangle$  by (auto simp: field_split_simps)
      qed auto
    then show ?thesis
      using LR [of  $c((2*k + 1) / 2^n) a((2*k + 1) / 2^n) b((2*k + 1) / 2^n)$ ]
      using alec [of  $2*k+1$ ] cleb [of  $2*k+1$ ] a_ge_0 [of  $2*k+1$ ] b_le_1 [of  $2*k+1$ ]
      using k a41 b41  $\langle 0 < n \rangle$ 
      by (simp add: add.commute)
  next
case equal then show ?thesis by simp
next
case greater
    moreover have  $\text{real } (4 * k + 3) / 2^{\text{Suc } n} - 1 / (2^{\text{Suc } n}) = \text{real } j / 2^n$ 
      using k by (force simp: field_split_simps)
    moreover have  $|\text{real } i / 2^m - \text{real } j / 2^n| < 2 * 1 / (2^{\text{Suc } n})$ 
      using less.prem1 by simp
    ultimately have closer:  $|\text{real } i / 2^m - \text{real } (4 * k + 3) / 2^{\text{Suc } n}| < 1 / (2^{\text{Suc } n})$ 
      using less.prem1 by linarith
    have a ( $\text{real } (4 * k + 3) / 2^{\text{Suc } n} \leq c (i / 2^m) \wedge c (i / 2^m) \leq b (\text{real } (4 * k + 3) / 2^{\text{Suc } n})$ )
      proof (rule less.IH [OF _ refl])
        show  $m - \text{Suc } n < d$ 
          using  $\langle n < m \rangle$  diff_less_mono2 less.prem1(1) by blast
        show  $|\text{real } i / 2^m - \text{real } (4 * k + 3) / 2^{\text{Suc } n}| < 1 / 2^{\text{Suc } n}$ 
          using closer  $\langle n < m \rangle$   $\langle d = m - n \rangle$  by (auto simp: field_split_simps)
      qed

```

```

⟨n < m⟩ diff_less_mono2)
  qed auto
  then show ?thesis
    using LR [of c((2*k + 1) / 2^n) a((2*k + 1) / 2^n) b((2*k + 1) /
2^n)]
    using alec [of 2*k+1] cleb [of 2*k+1] a_ge_0 [of 2*k+1] b_le_1 [of
2*k+1]
    using k a43 b43 ⟨0 < n⟩
    by (simp add: add.commute)
  qed
qed
qed
qed
then have aj_le_ci: a (real j / 2^n) ≤ c (real i / 2^m)
  and ci_le_bj: c (real i / 2^m) ≤ b (real j / 2^n) if odd j | real i / 2^m -
real j / 2^n| < 1/2^n for i j m n
  using that by blast+
  have close_ab: odd m ⟹ |a (real m / 2^n) - b (real m / 2^n)| ≤ 2 / 2^n
for m n
  proof (induction n arbitrary: m)
    case 0
    with u01 v01 show ?case by auto
  next
    case (Suc n m)
    with oddE obtain k where k: m = Suc (2*k) by fastforce
    show ?case
    proof (cases n > 0)
      case False
      with u01 v01 show ?thesis
        by (simp add: a_def b_def leftrec_base rightrec_base)
    next
      case True
      show ?thesis
      proof (cases even k)
        case True
        then obtain j where j: k = 2*j by (metis evenE)
        have |a ((2 * real j + 1) / 2^n) - (b ((2 * real j + 1) / 2^n))| ≤ 2/2
^ n
        proof -
          have odd (Suc k)
            using True by auto
          then show ?thesis
            by (metis (no_types) Groups.add_ac(2) Suc.IH j of_nat_Suc of_nat_mult
of_nat_numeral)
        qed
        moreover have a ((2 * real j + 1) / 2^n) ≤
          leftcut (a ((2 * real j + 1) / 2^n)) (b ((2 * real j + 1) / 2^n)
n)) (c ((2 * real j + 1) / 2^n))
          using alec [of 2*j+1] cleb [of 2*j+1] a_ge_0 [of 2*j+1] b_le_1 [of

```

```

2*j+1]
  by (auto simp: add commute left_right)
  moreover have leftcut (a ((2 * real j + 1) / 2 ^ n)) (b ((2 * real j + 1)
/ 2 ^ n)) (c ((2 * real j + 1) / 2 ^ n)) ≤
    c ((2 * real j + 1) / 2 ^ n)
  using alec [of 2*j+1] cleb [of 2*j+1] a_ge_0 [of 2*j+1] b_le_1 [of
2*j+1]
  by (auto simp: add commute left_right_m)
  ultimately have |a ((2 * real j + 1) / 2 ^ n) -
    leftcut (a ((2 * real j + 1) / 2 ^ n)) (b ((2 * real j + 1) / 2
^ n)) (c ((2 * real j + 1) / 2 ^ n))|
    ≤ 2/2 ^ Suc n
  by (simp add: c_def midpoint_def)
  with j k <n > 0 show ?thesis
  by (simp add: add commute [of 1] a41 b41 del: power_Suc)
next
case False
then obtain j where j: k = 2*j + 1 by (metis oddE)
have |a ((2 * real j + 1) / 2 ^ n) - (b ((2 * real j + 1) / 2 ^ n))| ≤ 2/2
^ n
  using Suc.IH [OF False] j by (auto simp: algebra_simps)
  moreover have c ((2 * real j + 1) / 2 ^ n) ≤
    rightcut (a ((2 * real j + 1) / 2 ^ n)) (b ((2 * real j + 1) / 2
^ n)) (c ((2 * real j + 1) / 2 ^ n))
  using alec [of 2*j+1] cleb [of 2*j+1] a_ge_0 [of 2*j+1] b_le_1 [of
2*j+1]
  by (auto simp: add commute left_right_m)
  moreover have rightcut (a ((2 * real j + 1) / 2 ^ n)) (b ((2 * real j + 1)
/ 2 ^ n)) (c ((2 * real j + 1) / 2 ^ n)) ≤
    b ((2 * real j + 1) / 2 ^ n)
  using alec [of 2*j+1] cleb [of 2*j+1] a_ge_0 [of 2*j+1] b_le_1 [of
2*j+1]
  by (auto simp: add commute left_right)
  ultimately have |rightcut (a ((2 * real j + 1) / 2 ^ n)) (b ((2 * real j +
1) / 2 ^ n)) (c ((2 * real j + 1) / 2 ^ n)) -
    b ((2 * real j + 1) / 2 ^ n)| ≤ 2/2 ^ Suc n
  by (simp add: c_def midpoint_def)
  with j k <n > 0 show ?thesis
  by (simp add: add commute [of 3] a43 b43 del: power_Suc)
qed
qed
qed
have m1_to_3: 4 * real k - 1 = real (4 * (k-1)) + 3 if 0 < k for k
  using that by auto
have fb_eq_fa: [0 < j; 2*j < 2 ^ n] ⇒ f(b((2 * real j - 1) / 2 ^ n)) = f(a((2
* real j + 1) / 2 ^ n)) for n j
proof (induction n arbitrary: j)
case 0
then show ?case by auto

```

```

next
  case (Suc n j) show ?case
  proof (cases n > 0)
    case False
    with Suc.premis show ?thesis by auto
  next
    case True
    show ?thesis proof (cases even j)
      case True
      then obtain k where k: j = 2*k by (metis evenE)
      with ‹0 < j› have k > 0 2 * k < 2 ^ n
        using Suc.premis(2) k by auto
      with k ‹0 < n› Suc.IH [of k] show ?thesis
        by (simp add: m1_to_3 a41 b43 del: power_Suc) (auto simp: of_nat_diff)
    next
      case False
      then obtain k where k: j = 2*k + 1 by (metis oddE)
      have f (leftcut (a ((2 * k + 1) / 2 ^ n)) (b ((2 * k + 1) / 2 ^ n)) (c ((2 * k
+ 1) / 2 ^ n)))
        = f (c ((2 * k + 1) / 2 ^ n))
        f (c ((2 * k + 1) / 2 ^ n))
        = f (rightcut (a ((2 * k + 1) / 2 ^ n)) (b ((2 * k + 1) / 2 ^ n)) (c ((2
* k + 1) / 2 ^ n)))
        using Alec [of 2*k+1 n] cleb [of 2*k+1 n] a_ge_0 [of 2*k+1 n] b_le_1
[of 2*k+1 n] k
        using left_right_m [of c((2*k + 1) / 2 ^ n) a((2*k + 1) / 2 ^ n) b((2*k +
1) / 2 ^ n)]
        by (auto simp: add commute feqm [OF order_refl] feqm [OF __ order_refl,
symmetric])
      then
      show ?thesis
        by (simp add: k add commute [of 1] add commute [of 3] a43 b41 ‹0 < n›
del: power_Suc)
    qed
  qed
qed
have f_eq_fc: ‹‹0 < j; j < 2 ^ n››
  ⇒ f(b((2*j - 1) / 2 ^ (Suc n))) = f(c(j / 2 ^ n)) ∧
  f(a((2*j + 1) / 2 ^ (Suc n))) = f(c(j / 2 ^ n)) for n and j::nat
proof (induction n arbitrary: j)
  case 0
  then show ?case by auto
next
  case (Suc n)
  show ?case
  proof (cases even j)
    case True
    then obtain k where k: j = 2*k by (metis evenE)
    then have less2n: k < 2 ^ n

```



```

    using Suc.premis(2) by auto
  have  $0 < k$  using  $\langle 0 < j \rangle k$  by linarith
  then have  $m1\_to\_3$ :  $real (4 * k - Suc\ 0) = real (4 * (k-1)) + 3$ 
    by auto
  then show ?thesis
    using Suc.IH [of k]  $k < 0 < k$ 
    by (simp add: less2n add commute [of 1] m1_to_3 a41 b43 del: power_Suc)
(auto simp: of_nat_diff)
next
  case False
  then obtain k where  $k: j = 2*k + 1$  by (metis oddE)
  with Suc.premis have  $k < 2^n$  by auto
  show ?thesis
    using alec [of  $2*k+1$  Suc n] cleb [of  $2*k+1$  Suc n] a_ge_0 [of  $2*k+1$  Suc
n] b_le_1 [of  $2*k+1$  Suc n] k
    using left_right_m [of  $c((2*k + 1) / 2^{Suc\ n})$  a( $(2*k + 1) / 2^{Suc\ n}$ )
b( $(2*k + 1) / 2^{Suc\ n}$ )]
    apply (simp add: add commute [of 1] add commute [of 3] m1_to_3 b41 a43
del: power_Suc)
    apply (force intro: feqm)
    done
qed
qed
define D01 where  $D01 \equiv \{0 < .. < 1\} \cap (\bigcup k\ m. \{real\ m / 2^k\})$ 
have cloD01 [simp]:  $closure\ D01 = \{0..1\}$ 
  unfolding D01_def
  by (subst closure_dyadic_rationals_in_convex_set_pos_1) auto
have uniformly_continuous_on D01 (f  $\circ$  c)
proof (clarsimp simp: uniformly_continuous_on_def)
  fix e::real
  assume  $0 < e$ 
  have ucontf:  $uniformly\_continuous\_on\ \{0..1\}\ f$ 
    by (simp add: compact_uniformly_continuous [OF cont_f])
  then obtain d where  $0 < d$  and  $d: \bigwedge x\ x'. \llbracket x \in \{0..1\}; x' \in \{0..1\}; norm$ 
 $(x' - x) < d \rrbracket \implies norm\ (f\ x' - f\ x) < e/2$ 
    unfolding uniformly_continuous_on_def dist_norm
    by (metis  $\langle 0 < e \rangle$  less_divide_eq_numeral1(1) mult_zero_left)
  obtain n where  $n: 1/2^n < \min\ d\ 1$ 
  by (metis  $\langle 0 < d \rangle$  divide_less_eq_1 less_numeral_extra(1) min_def one_less_numeral_iff
power_one_over real_arch_pow_inv semiring_norm(76) zero_less_numeral)
  with gr0I have  $n > 0$ 
  by (force simp: field_split_simps)
  show  $\exists d > 0. \forall x \in D01. \forall x' \in D01. dist\ x'\ x < d \longrightarrow dist\ (f\ (c\ x'))\ (f\ (c\ x)) <$ 
 $e$ 
proof (intro exI ballI impI conjI)
  show  $(0::real) < 1/2^n$  by auto
next
  have dist_fc_close:  $dist\ (f(c(real\ i / 2^m)))\ (f(c(real\ j / 2^n))) < e/2$ 
    if  $i: 0 < i < 2^m$  and  $j: 0 < j < 2^n$  and clo:  $abs(i / 2^m - j /$ 

```

```

 $2^n) < 1/2^n$  for  $i j m$ 
proof -
  have  $abs3: |x - a| < e \implies x = a \vee |x - (a - e/2)| < e/2 \vee |x - (a + e/2)| < e/2$  for  $x a e::real$ 
  by linarith
  consider  $i / 2^m = j / 2^n$ 
   $|i / 2^m - (2 * j - 1) / 2^{Suc\ n}| < 1/2^{Suc\ n}$ 
   $|i / 2^m - (2 * j + 1) / 2^{Suc\ n}| < 1/2^{Suc\ n}$ 
  using  $abs3\ [OF\ clo]\ j$  by (auto simp: field_simps of_nat_diff)
  then show ?thesis
  proof cases
    case 1 with  $\langle 0 < e \rangle$  show ?thesis by auto
  next
    case 2
    have  $*$ :  $abs(a - b) \leq 1/2^n \wedge 1/2^n < d \wedge a \leq c \wedge c \leq b \implies b -$ 
 $c < d$  for  $a b c$ 
    by auto
    have  $norm\ (c\ (real\ i / 2^m) - b\ (real\ (2 * j - 1) / 2^{Suc\ n})) < d$ 
    using  $2\ j\ n\ close\_ab\ [of\ 2*j-1\ Suc\ n]$ 
    using  $b\_ge\_0\ [of\ 2*j-1\ Suc\ n]\ b\_le\_1\ [of\ 2*j-1\ Suc\ n]$ 
    using  $aj\_le\_ci\ [of\ 2*j-1\ i\ m\ Suc\ n]$ 
    using  $ci\_le\_bj\ [of\ 2*j-1\ i\ m\ Suc\ n]$ 
    apply (simp add: divide_simps of_nat_diff del: power_Suc)
    apply (auto simp: divide_simps intro!: *)
    done
    moreover have  $f(c(j / 2^n)) = f(b\ ((2*j - 1) / 2^{Suc\ n}))$ 
    using  $f\_eq\_fc\ [OF\ j]$  by metis
    ultimately show ?thesis
    by (metis dist_norm atLeastAtMost_iff b_ge_0 b_le_1 c_ge_0 c_le_1
 $d$ )
  next
    case 3
    have  $*$ :  $abs(a - b) \leq 1/2^n \wedge 1/2^n < d \wedge a \leq c \wedge c \leq b \implies c -$ 
 $a < d$  for  $a b c$ 
    by auto
    have  $norm\ (c\ (real\ i / 2^m) - a\ (real\ (2 * j + 1) / 2^{Suc\ n})) < d$ 
    using  $3\ j\ n\ close\_ab\ [of\ 2*j+1\ Suc\ n]$ 
    using  $b\_ge\_0\ [of\ 2*j+1\ Suc\ n]\ b\_le\_1\ [of\ 2*j+1\ Suc\ n]$ 
    using  $aj\_le\_ci\ [of\ 2*j+1\ i\ m\ Suc\ n]$ 
    using  $ci\_le\_bj\ [of\ 2*j+1\ i\ m\ Suc\ n]$ 
    apply (simp add: divide_simps of_nat_diff del: power_Suc)
    apply (auto simp: divide_simps intro!: *)
    done
    moreover have  $f(c(j / 2^n)) = f(a\ ((2*j + 1) / 2^{Suc\ n}))$ 
    using  $f\_eq\_fc\ [OF\ j]$  by metis
    ultimately show ?thesis
    by (metis dist_norm a_ge_0 atLeastAtMost_iff a_ge_0 a_le_1 c_ge_0
 $c\_le\_1\ d$ )
  qed

```

```

qed
show dist (f (c x')) (f (c x)) < e
  if  $x \in D01$   $x' \in D01$   $\text{dist } x' x < 1/2^n$  for  $x x'$ 
  using that unfolding D01_def dyadics_in_open_unit_interval
proof clarsimp
  fix  $i k::\text{nat}$  and  $m p$ 
  assume  $i: 0 < i$   $i < 2^m$  and  $k: 0 < k$   $k < 2^p$ 
  assume  $\text{clo}: \text{dist } (\text{real } k / 2^p) (\text{real } i / 2^m) < 1/2^n$ 
  obtain  $j::\text{nat}$  where  $0 < j$   $j < 2^n$ 
    and  $\text{clo\_ij}: \text{abs}(i / 2^m - j / 2^n) < 1/2^n$ 
    and  $\text{clo\_kj}: \text{abs}(k / 2^p - j / 2^n) < 1/2^n$ 
  proof -
    have  $\max(2^n * i / 2^m) (2^n * k / 2^p) \geq 0$ 
      by (auto simp: le_max_iff_disj)
    then obtain  $j$  where  $\text{floor } (\max(2^n * i / 2^m) (2^n * k / 2^p)) = \text{int } j$ 
      using zero_le_floor zero_le_imp_eq_int by blast
    then have  $j\_le: \text{real } j \leq \max(2^n * i / 2^m) (2^n * k / 2^p)$ 
      and  $\text{less\_j1}: \max(2^n * i / 2^m) (2^n * k / 2^p) < \text{real } j + 1$ 
      using floor_correct [of  $\max(2^n * i / 2^m) (2^n * k / 2^p)$ ] by
linarith+
    show thesis
  proof (cases  $j = 0$ )
    case True
      show thesis
    proof
      show  $(1::\text{nat}) < 2^n$ 
        by (metis Suc_1 ‹ $0 < n$ › lessI one_less_power)
      show  $|\text{real } i / 2^m - \text{real } 1/2^n| < 1/2^n$ 
        using  $i\_less\_j1$  by (simp add: dist_norm_field_simps True)
      show  $|\text{real } k / 2^p - \text{real } 1/2^n| < 1/2^n$ 
        using  $k\_less\_j1$  by (simp add: dist_norm_field_simps True)
    qed simp
  next
    case False
      have  $1: \text{real } j * 2^m < \text{real } i * 2^n$ 
        if  $j: \text{real } j * 2^p \leq \text{real } k * 2^n$  and  $k: \text{real } k * 2^m < \text{real } i * 2^p$ 
        for  $i k m p$ 
      proof -
        have  $\text{real } j * 2^p * 2^m \leq \text{real } k * 2^n * 2^m$ 
          using  $j$  by simp
        moreover have  $\text{real } k * 2^m * 2^n < \text{real } i * 2^p * 2^n$ 
          using  $k$  by simp
        ultimately have  $\text{real } j * 2^p * 2^m < \text{real } i * 2^p * 2^n$ 
          by (simp only: mult_ac)
        then show ?thesis
          by simp
      qed
    have  $2: \text{real } j * 2^m < 2^m + \text{real } i * 2^n$ 

```

```

    if j: real j * 2 ^ p ≤ real k * 2 ^ n and k: real k * (2 ^ m * 2 ^ n) <
2 ^ m * 2 ^ p + real i * (2 ^ n * 2 ^ p)
    for i k m p
    proof -
      have real j * 2 ^ p * 2 ^ m ≤ real k * (2 ^ m * 2 ^ n)
      using j by simp
      also have ... < 2 ^ m * 2 ^ p + real i * (2 ^ n * 2 ^ p)
      by (rule k)
      finally have (real j * 2 ^ m) * 2 ^ p < (2 ^ m + real i * 2 ^ n) * 2 ^ p
      by (simp add: algebra_simps)
      then show ?thesis
      by simp
    qed
    have 3: real j * 2 ^ p < 2 ^ p + real k * 2 ^ n
    if j: real j * 2 ^ m ≤ real i * 2 ^ n and i: real i * 2 ^ p ≤ real k * 2 ^
m
    proof -
      have real j * 2 ^ m * 2 ^ p ≤ real i * 2 ^ n * 2 ^ p
      using j by simp
      moreover have real i * 2 ^ p * 2 ^ n ≤ real k * 2 ^ m * 2 ^ n
      using i by simp
      ultimately have real j * 2 ^ m * 2 ^ p ≤ real k * 2 ^ m * 2 ^ n
      by (simp only: mult_ac)
      then have real j * 2 ^ p ≤ real k * 2 ^ n
      by simp
      also have ... < 2 ^ p + real k * 2 ^ n
      by auto
      finally show ?thesis by simp
    qed
    show ?thesis
    proof
      have 2 ^ n * real i / 2 ^ m < 2 ^ n 2 ^ n * real k / 2 ^ p < 2 ^ n
      using i k by (auto simp: field_simps)
      then have max (2 ^ n * i / 2 ^ m) (2 ^ n * k / 2 ^ p) < 2 ^ n
      by simp
      with j_le have real j < 2 ^ n by linarith
      then show j < 2 ^ n
      by auto
      have |real i * 2 ^ n - real j * 2 ^ m| < 2 ^ m
      using clo less_j1 j_le
      by (auto simp: le_max_iff_disj field_split_simps dist_norm abs_if
split: if_split_asm dest: 1 2)
      then show |real i / 2 ^ m - real j / 2 ^ n| < 1 / 2 ^ n
      by (auto simp: field_split_simps)
      have |real k * 2 ^ n - real j * 2 ^ p| < 2 ^ p
      using clo less_j1 j_le
      by (auto simp: le_max_iff_disj field_split_simps dist_norm abs_if
split: if_split_asm dest: 3 2)
      then show |real k / 2 ^ p - real j / 2 ^ n| < 1 / 2 ^ n

```

```

      by (auto simp: le_max_iff_disj field_split_simps dist_norm)
    qed (use False in simp)
  qed
qed
show dist (f (c (real k / 2 ^ p))) (f (c (real i / 2 ^ m))) < e
proof (rule dist_triangle_half_l)
  show dist (f (c (real k / 2 ^ p))) (f (c (j / 2 ^ n))) < e/2
    using ‹0 < j› ‹j < 2 ^ n› k clo_kj
    by (intro dist_fc_close) auto
  show dist (f (c (real i / 2 ^ m))) (f (c (real j / 2 ^ n))) < e/2
    using ‹0 < j› ‹j < 2 ^ n› i clo_ij
    by (intro dist_fc_close) auto
  qed
qed
qed
qed
then obtain h where ucont_h: uniformly_continuous_on {0..1} h
  and fc_eq:  $\bigwedge x. x \in D01 \implies (f \circ c) x = h x$ 
proof (rule uniformly_continuous_on_extension_on_closure [of D01 f o c])
  qed (use closure_subset [of D01] in ‹auto intro!: that›)
  then have cont_h: continuous_on {0..1} h
    using uniformly_continuous_imp_continuous by blast
  have h_eq:  $h (real k / 2 ^ m) = f (c (real k / 2 ^ m))$  if  $0 < k < 2^m$  for  $k$ 
  m
    using fc_eq that by (force simp: D01_def)
  have h ‹{0..1} = f ‹{0..1}
  proof
    have h ‹(closure D01)  $\subseteq$  f ‹{0..1}
    proof (rule image_closure_subset)
      show continuous_on (closure D01) h
        using cont_h by simp
      show closed (f ‹{0..1})
        using compact_continuous_image [OF cont_f] compact_imp_closed by
blast
      show h ‹D01  $\subseteq$  f ‹{0..1}
        by (force simp: dyadics_in_open_unit_interval D01_def h_eq)
    qed
  qed
  with cloD01 show h ‹{0..1}  $\subseteq$  f ‹{0..1} by simp
  have a12 [simp]:  $a (1/2) = u$ 
    by (metis a_def leftrec_base numeral_One of_nat_numeral)
  have b12 [simp]:  $b (1/2) = v$ 
    by (metis b_def rightrec_base numeral_One of_nat_numeral)
  have f ‹{0..1}  $\subseteq$  closure(h ‹D01)
  proof (clarsimp simp: closure_approachable dyadics_in_open_unit_interval
D01_def)
    fix x e::real
    assume  $0 \leq x \leq 1$   $0 < e$ 
    have ucont_f: uniformly_continuous_on {0..1} f
      using compact_uniformly_continuous cont_f by blast

```

```

then obtain  $\delta$  where  $\delta > 0$ 
  and  $\delta: \bigwedge x x'. \llbracket x \in \{0..1\}; x' \in \{0..1\}; \text{dist } x' x < \delta \rrbracket \implies \text{norm } (f x' - f x) < e$ 
  using  $\langle 0 < e \rangle$  by (auto simp: uniformly_continuous_on_def dist_norm)
  have *:  $\exists m::\text{nat}. \exists y. \text{odd } m \wedge 0 < m \wedge m < 2^n \wedge y \in \{a(m / 2^n) .. b(m / 2^n)\} \wedge f y = f x$ 
  if  $n \neq 0$  for  $n$ 
  using that
  proof (induction n)
    case 0 then show ?case by auto
  next
    case (Suc n)
    show ?case
    proof (cases n=0)
      case True
      consider  $x \in \{0..u\} \mid x \in \{u..v\} \mid x \in \{v..1\}$ 
      using  $\langle 0 \leq x \rangle \langle x \leq 1 \rangle$  by force
      then have  $\exists y \geq a(\text{real } 1/2). y \leq b(\text{real } 1/2) \wedge f y = f x$ 
      proof cases
        case 1
        then show ?thesis
          using uabv [of 1 1] f0u [of u] f0u [of x] by force
      next
        case 2
        then show ?thesis
          by (rule_tac x=x in exI) auto
      next
        case 3
        then show ?thesis
          using uabv [of 1 1] fv1 [of v] fv1 [of x] by force
      qed
    with  $\langle n=0 \rangle$  show ?thesis
      by (rule_tac x=1 in exI) auto
  next
    case False
    with Suc obtain m y
      where odd m  $0 < m$  and mless:  $m < 2^n$ 
      and y:  $y \in \{a(\text{real } m / 2^n) .. b(\text{real } m / 2^n)\}$  and feq:  $f y = f x$ 
      by metis
    then obtain j where j:  $m = 2*j + 1$  by (metis oddE)
    have j4:  $4*j + 1 < 2^{Suc n}$ 
      using mless j by (simp add: algebra_simps)

    consider  $y \in \{a((2*j + 1) / 2^n) .. b((4*j + 1) / 2^{Suc n})\}$ 
      |  $y \in \{b((4*j + 1) / 2^{Suc n}) .. a((4*j + 3) / 2^{Suc n})\}$ 
      |  $y \in \{a((4*j + 3) / 2^{Suc n}) .. b((2*j + 1) / 2^n)\}$ 
    using y j by force
    then show ?thesis
    proof cases

```

```

      case 1
      show ?thesis
      proof (intro exI conjI)
        show  $y \in \{a \text{ (real } (4 * j + 1) / 2^{\wedge} \text{Suc } n)..b \text{ (real } (4 * j + 1) / 2^{\wedge} \text{Suc } n)\}$ 
        using mless j <n ≠ 0> 1 by (simp add: a41 b41 add.commute [of 1]
del: power_Suc)
      qed (use feq j4 in auto)
    next
    case 2
    show ?thesis
    proof (intro exI conjI)
      show  $b \text{ (real } (4 * j + 1) / 2^{\wedge} \text{Suc } n) \in \{a \text{ (real } (4 * j + 1) / 2^{\wedge} \text{Suc } n)..b \text{ (real } (4 * j + 1) / 2^{\wedge} \text{Suc } n)\}$ 
      using <n ≠ 0> alec [of 2*j+1 n] cleb [of 2*j+1 n] a_ge_0 [of 2*j+1 n] b_le_1 [of 2*j+1 n]
      using left_right [of c((2*j + 1) / 2^ $\wedge$ n) a((2*j + 1) / 2^ $\wedge$ n) b((2*j + 1) / 2^ $\wedge$ n)]
      by (simp add: a41 b41 add.commute [of 1] del: power_Suc)
      show  $f \text{ (b (real } (4 * j + 1) / 2^{\wedge} \text{Suc } n)) = f x$ 
      using <n ≠ 0> 2
      using alec [of 2*j+1 n] cleb [of 2*j+1 n] a_ge_0 [of 2*j+1 n] b_le_1 [of 2*j+1 n]
      by (force simp add: b41 a43 add.commute [of 1] feq [symmetric] simp
del: power_Suc intro: f_eqI)
    qed (use j4 in auto)
  next
  case 3
  show ?thesis
  proof (intro exI conjI)
    show  $4 * j + 3 < 2^{\wedge} \text{Suc } n$ 
    using mless j by simp
    show  $f y = f x$ 
    by fact
    show  $y \in \{a \text{ (real } (4 * j + 3) / 2^{\wedge} \text{Suc } n)..b \text{ (real } (4 * j + 3) / 2^{\wedge} \text{Suc } n)\}$ 
    using 3 False b43 [of n j] by (simp add: add.commute)
  qed (use 3 in auto)
qed
qed
qed
obtain n where n:  $1/2^{\wedge} n < \min (\delta / 2) 1$ 
by (metis <0 < δ> divide_less_eq_1 less_numeral_extra(1) min_less_iff_conj
one_less_numeral_iff power_one_over real_arch_pow_inv semiring_norm(76)
zero_less_divide_iff zero_less_numeral)
with gr0I have n ≠ 0
by fastforce
with * obtain m::nat and y
where odd m 0 < m and mless:  $m < 2^{\wedge} n$ 

```

```

    and y:  $a(m / 2^n) \leq y \wedge y \leq b(m / 2^n)$  and feq:  $f x = f y$ 
  by (metis atLeastAtMost_iff)
then have  $0 \leq y \leq 1$ 
  by (meson a_ge_0 b_le_1 order.trans)+
moreover have  $y < \delta + c \text{ (real } m / 2^n) c \text{ (real } m / 2^n) < \delta + y$ 
  using y alec [of m n] cleb [of m n] n field_sum_of_halves close_ab [OF
<odd m>, of n]
  by linarith+
moreover note  $\langle 0 < m \rangle \text{ mless } \langle 0 \leq x \rangle \langle x \leq 1 \rangle$ 
ultimately have  $\text{dist } (h \text{ (real } m / 2^n)) (f x) < e$ 
  by (auto simp: dist_norm h_eq feq  $\delta$ )
then show  $\exists k. \exists m \in \{0 <..< 2^k\}. \text{dist } (h \text{ (real } m / 2^k)) (f x) < e$ 
  using  $\langle 0 < m \rangle \text{ greaterThanLessThan\_iff mless}$  by blast
qed
also have  $\dots \subseteq h \text{ ' } \{0..1\}$ 
proof (rule closure_minimal)
  show  $h \text{ ' } D01 \subseteq h \text{ ' } \{0..1\}$ 
    using cloD01 closure_subset by blast
  show closed  $(h \text{ ' } \{0..1\})$ 
    using compact_continuous_image [OF cont_h] compact_imp_closed by
auto
  qed
  finally show  $f \text{ ' } \{0..1\} \subseteq h \text{ ' } \{0..1\}$  .
qed
moreover have  $\text{inj\_on } h \text{ ' } \{0..1\}$ 
proof -
  have  $u < v$ 
  by (metis atLeastAtMost_iff f0u f_1not0 fv1 order.not_eq_order_implies_strict
u01(1) u01(2) v01(1))
  have  $f\_not\_fu: \bigwedge x. \llbracket u < x; x \leq v \rrbracket \implies f x \neq f u$ 
  by (metis atLeastAtMost_iff f0u fv1 greaterThanAtMost_iff order_refl or-
der_trans u01(1) v01(2))
  have  $f\_not\_fv: \bigwedge x. \llbracket u \leq x; x < v \rrbracket \implies f x \neq f v$ 
  by (metis atLeastAtMost_iff order_refl order_trans v01(2) atLeastLessThan_iff
fv fv1)
  have a_less_b:
     $a(j / 2^n) < b(j / 2^n) \wedge$ 
     $(\forall x. a(j / 2^n) < x \longrightarrow x \leq b(j / 2^n) \longrightarrow f x \neq f(a(j / 2^n))) \wedge$ 
     $(\forall x. a(j / 2^n) \leq x \longrightarrow x < b(j / 2^n) \longrightarrow f x \neq f(b(j / 2^n)))$  for n
and j::nat
  proof (induction n arbitrary: j)
  case 0 then show ?case
    by (simp add:  $\langle u < v \rangle f\_not\_fu f\_not\_fv$ )
  next
  case (Suc n j) show ?case
  proof (cases  $n > 0$ )
  case False then show ?thesis
    by (auto simp: a_def b_def leftrec_base rightrec_base  $\langle u < v \rangle f\_not\_fu$ 
f_not_fv)

```



```

next
  case True show ?thesis
  proof (cases even j)
    case True
    with ‹0 < n› Suc.IH show ?thesis
    by (auto elim!: evenE)
  next
    case False
    then obtain k where k: j = 2*k + 1 by (metis oddE)
    then show ?thesis
    proof (cases even k)
      case True
      then obtain m where m: k = 2*m by (metis evenE)
      have fleft: f (leftcut (a ((2*m + 1) / 2^n)) (b ((2*m + 1) / 2^n)) (c
        ((2*m + 1) / 2^n))) =
        f (c((2*m + 1) / 2^n))
        using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
        b_le_1 [of 2*m+1 n]
        using left_right_m [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
        b((2*m + 1) / 2^n)]
        by (auto intro: f_eqI)
      show ?thesis
      proof (intro conjI impI notI allI)
        have False if b (real j / 2 ^ Suc n) ≤ a (real j / 2 ^ Suc n)
        proof -
          have f (c ((1 + real m * 2) / 2 ^ n)) = f (a ((1 + real m * 2) / 2
            ^ n))
          using k m ‹0 < n› fleft that a41 [of n m] b41 [of n m]
          using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
          b_le_1 [of 2*m+1 n]
          using left_right [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
          b((2*m + 1) / 2^n)]
          by (auto simp: algebra_simps)
          moreover have a (real (1 + m * 2) / 2 ^ n) < c (real (1 + m *
            2) / 2 ^ n)
          using Suc.IH [of 1 + m * 2] by (simp add: c_def midpoint_def)
          moreover have c (real (1 + m * 2) / 2 ^ n) ≤ b (real (1 + m * 2)
            / 2 ^ n)
          using cleb by blast
          ultimately show ?thesis
          using Suc.IH [of 1 + m * 2] by force
        qed
      then show a (real j / 2 ^ Suc n) < b (real j / 2 ^ Suc n) by force
    next
      fix x
      assume a (real j / 2 ^ Suc n) < x x ≤ b (real j / 2 ^ Suc n) f x = f
        (a (real j / 2 ^ Suc n))
      then show False
      using Suc.IH [of 1 + m * 2, THEN conjunct2, THEN conjunct1]

```

```

      using k m ⟨0 < n⟩ a41 [of n m] b41 [of n m]
      using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
      using left_right_m [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
      by (auto simp: algebra_simps)
    next
      fix x
      assume a (real j / 2 ^ Suc n) ≤ x x < b (real j / 2 ^ Suc n) f x = f
(b (real j / 2 ^ Suc n))
      then show False
        using k m ⟨0 < n⟩ a41 [of n m] b41 [of n m] fleft left_neq
        using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
        by (auto simp: algebra_simps)
      qed
    next
      case False
      with oddE obtain m where m: k = Suc (2*m) by fastforce
      have fright: f (rightcut (a ((2*m + 1) / 2^n)) (b ((2*m + 1) / 2^n))
(c ((2*m + 1) / 2^n))) = f (c((2*m + 1) / 2^n))
        using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
        using left_right_m [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
        by (auto intro: f_eqI [OF __ order_refl])
      show ?thesis
      proof (intro conjI impI notI allI)
        have False if b (real j / 2 ^ Suc n) ≤ a (real j / 2 ^ Suc n)
        proof -
          have f (c ((1 + real m * 2) / 2 ^ n)) = f (b ((1 + real m * 2) / 2
^ n))
            using k m ⟨0 < n⟩ fright that a43 [of n m] b43 [of n m]
            using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
            using left_right [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
            by (auto simp: algebra_simps)
          moreover have a (real (1 + m * 2) / 2 ^ n) ≤ c (real (1 + m *
2) / 2 ^ n)
            using alec by blast
          moreover have c (real (1 + m * 2) / 2 ^ n) < b (real (1 + m * 2)
/ 2 ^ n)
            using Suc.IH [of 1 + m * 2] by (simp add: c_def midpoint_def)
          ultimately show ?thesis
            using Suc.IH [of 1 + m * 2] by force
        qed
      proof
        next show a (real j / 2 ^ Suc n) < b (real j / 2 ^ Suc n) by force
      next

```

```

    fix x
    assume a (real j / 2 ^ Suc n) < x x ≤ b (real j / 2 ^ Suc n) f x = f
  (a (real j / 2 ^ Suc n))
  then show False
    using k m ⟨0 < n⟩ a43 [of n m] b43 [of n m] fright right_neq
    using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
  b_le_1 [of 2*m+1 n]
  by (auto simp: algebra_simps)
next
fix x
assume a (real j / 2 ^ Suc n) ≤ x x < b (real j / 2 ^ Suc n) f x = f
(b (real j / 2 ^ Suc n))
then show False
  using Suc.IH [of 1 + m * 2, THEN conjunct2, THEN conjunct2]
  using k m ⟨0 < n⟩ a43 [of n m] b43 [of n m]
  using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
  b_le_1 [of 2*m+1 n]
  using left_right_m [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
  b((2*m + 1) / 2^n)]
  by (auto simp: algebra_simps fright simp del: power_Suc)
qed
qed
qed
qed
qed
have c_gt_0 [simp]: 0 < c(m / 2^n) and c_less_1 [simp]: c(m / 2^n) < 1
for m::nat and n
  using a_less_b [of m n] apply (simp_all add: c_def midpoint_def)
  using a_ge_0 [of m n] b_le_1 [of m n] by linarith+
have approx: ∃ j n. odd j ∧ n ≠ 0 ∧
  real i / 2^m ≤ real j / 2^n ∧
  real j / 2^n ≤ real k / 2^p ∧
  |real i / 2^m - real j / 2^n| < 1/2^n ∧
  |real k / 2^p - real j / 2^n| < 1/2^n
if 0 < i i < 2^m 0 < k k < 2^p i / 2^m < k / 2^p m + p = N for N m
p i k
  using that
proof (induction N arbitrary: m p i k rule: less_induct)
case (less N)
  then consider i / 2^m ≤ 1/2 1/2 ≤ k / 2^p | k / 2^p < 1/2 | k / 2^p ≥
  1/2 1/2 < i / 2^m
  by linarith
  then show ?case
proof cases
case 1
  with less.premis show ?thesis
  by (rule_tac x=1 in exI) + (fastforce simp: field_split_simps)
next
case 2 show ?thesis

```

```

proof (cases m)
  case 0 with less.prems show ?thesis
    by auto
next
  case (Suc m') show ?thesis
  proof (cases p)
    case 0 with less.prems show ?thesis by auto
  next
    case (Suc p')
    have §: False if real i * 2 ^ p' < real k * 2 ^ m' k < 2 ^ p' 2 ^ m' ≤ i
    proof -
      have real k * 2 ^ m' < 2 ^ p' * 2 ^ m'
      using that by simp
      then have real i * 2 ^ p' < 2 ^ p' * 2 ^ m'
      using that by linarith
      with that show ?thesis by simp
    qed
    moreover have *: real i / 2 ^ m' < real k / 2 ^ p' k < 2 ^ p'
    using less.prems ⟨m = Suc m'⟩ 2 Suc by (force simp: field_split_simps)+
    moreover have i < 2 ^ m'
      using § * by (clarsimp simp: divide_simps linorder_not_le) (meson
linorder_not_le)
    ultimately show ?thesis
      using less.IH [of m'+p' i m' k p'] less.prems ⟨m = Suc m'⟩ 2 Suc
      by (force simp: field_split_simps)
    qed
  qed
next
  case 3 show ?thesis
  proof (cases m)
    case 0 with less.prems show ?thesis
      by auto
  next
    case (Suc m') show ?thesis
    proof (cases p)
      case 0 with less.prems show ?thesis by auto
    next
      case (Suc p')
      have real (i - 2 ^ m') / 2 ^ m' < real (k - 2 ^ p') / 2 ^ p'
      using less.prems ⟨m = Suc m'⟩ Suc 3 by (auto simp: field_simps
of_nat_diff)
      moreover have k - 2 ^ p' < 2 ^ p' i - 2 ^ m' < 2 ^ m'
      using less.prems Suc ⟨m = Suc m'⟩ by auto
      moreover
      have 2 ^ p' ≤ k 2 ^ p' ≠ k
      using less.prems ⟨m = Suc m'⟩ Suc 3 by auto
      then have 2 ^ p' < k
      by linarith
      ultimately show ?thesis

```

```

      using less.IH [of  $m' + p'$   $i - 2^m m' k - 2^{p'} p'$ ] less.premis  $\langle m =$ 
    Suc  $m' \rangle$  Suc 3
      apply (clarsimp simp: field_simps of_nat_diff)
      apply (rule_tac  $x = 2^n + j$  in exI, simp)
      apply (rule_tac  $x = \text{Suc } n$  in exI)
      apply (auto simp: field_simps)
      done
    qed
  qed
  qed
  qed
  have clec:  $c(\text{real } i / 2^m) \leq c(\text{real } j / 2^n)$ 
  if  $i: 0 < i < 2^m$  and  $j: 0 < j < 2^n$  and  $ij: i / 2^m < j / 2^n$  for
   $m \ i \ n \ j$ 
  proof -
    obtain  $j' \ n'$  where  $\text{odd } j' \ n' \neq 0$ 
    and  $i\_le\_j: \text{real } i / 2^m \leq \text{real } j' / 2^{n'}$ 
    and  $j\_le\_j: \text{real } j' / 2^{n'} \leq \text{real } j / 2^n$ 
    and  $clo\_ij: |\text{real } i / 2^m - \text{real } j' / 2^{n'}| < 1/2^{n'}$ 
    and  $clo\_jj: |\text{real } j / 2^n - \text{real } j' / 2^{n'}| < 1/2^{n'}$ 
    using approx [of  $i \ m \ j \ n \ m+n$ ] that  $i \ j \ ij$  by auto
    with oddE obtain  $q$  where  $q: j' = \text{Suc } (2*q)$  by fastforce
    have  $c(\text{real } i / 2^m) \leq c((2*q + 1) / 2^{n'})$ 
    proof (cases  $i / 2^m = (2*q + 1) / 2^{n'}$ )
      case True then show ?thesis by simp
    next
      case False
      with  $i\_le\_j \ clo\_ij \ q$  have  $|\text{real } i / 2^m - \text{real } (4 * q + 1) / 2^{\text{Suc } n'}|$ 
      <  $1 / 2^{\text{Suc } n'}$ 
      by (auto simp: field_split_simps)
      then have  $c(i / 2^m) \leq b(\text{real}(4 * q + 1) / 2^{\text{Suc } n'})$ 
      by (meson ci_le_bj even_mult_iff even_numerical even_plus_one_iff)
      then show ?thesis
      using alec [of  $2*q+1 \ n'$ ] cleb [of  $2*q+1 \ n'$ ] a_ge_0 [of  $2*q+1 \ n'$ ] b_le_1
      [of  $2*q+1 \ n'$ ] b41 [of  $n' \ q$ ]  $\langle n' \neq 0 \rangle$ 
      using left_right_m [of  $c((2*q + 1) / 2^{n'}) \ a((2*q + 1) / 2^{n'}) \ b((2*q$ 
      +  $1) / 2^{n'})$ ]
      by (auto simp: algebra_simps)
    qed
    also have ...  $\leq c(\text{real } j / 2^n)$ 
    proof (cases  $j / 2^n = (2*q + 1) / 2^{n'}$ )
      case True
      then show ?thesis by simp
    next
      case False
      with  $j\_le\_j \ q$  have less:  $(2*q + 1) / 2^{n'} < j / 2^n$ 
      by auto
      have *:  $\llbracket q < i; \text{abs}(i - q) < s*2; r = q + s \rrbracket \implies \text{abs}(i - r) < s$  for  $i \ q \ s$ 
      r::real

```

```

    by auto
    have  $|real\ j / 2^n - real\ (4 * q + 3) / 2^{Suc\ n'}| < 1 / 2^{Suc\ n'}$ 
    by (rule * [OF less]) (use j_le_j clo_jj q in ‹auto simp: field_split_simps›)
    then have  $a(real(4*q + 3) / 2^{(Suc\ n')}) \leq c(j / 2^n)$ 
    by (metis Suc3_eq_add_3 add commute aj_le_ci even_Suc even_mult_iff
even_numeral)
    then show ?thesis
    using alec [of 2*q+1 n'] cleb [of 2*q+1 n'] a_ge_0 [of 2*q+1 n'] b_le_1
[of 2*q+1 n'] a43 [of n' q] ‹n' ≠ 0›
    using left_right_m [of c((2*q + 1) / 2^n') a((2*q + 1) / 2^n') b((2*q
+ 1) / 2^n')]
    by (auto simp: algebra_simps)
qed
finally show ?thesis .
qed
have  $x = y$  if  $0 \leq x$   $x \leq 1$   $0 \leq y$   $y \leq 1$   $h\ x = h\ y$  for  $x\ y$ 
using that
proof (induction x y rule: linorder_class.linorder_less_wlog)
case (less x1 x2)
obtain m n where m:  $0 < m$   $m < 2^n$ 
and x12:  $x1 < m / 2^n$   $m / 2^n < x2$ 
and neq:  $h\ x1 \neq h\ (real\ m / 2^n)$ 
proof -
have  $(x1 + x2) / 2 \in closure\ D01$ 
using cloD01 less.hyps less.premis by auto
with less obtain y where  $y \in D01$  and dist_y:  $dist\ y\ ((x1 + x2) / 2) < (x2 - x1) / 64$ 
unfolding closure_approachable
by (metis diff_gt_0_iff_gt less_divide_eq_numeral1(1) mult_zero_left)
obtain m n where m:  $0 < m$   $m < 2^n$ 
and clo:  $|real\ m / 2^n - (x1 + x2) / 2| < (x2 - x1) / 64$ 
and n:  $1/2^n < (x2 - x1) / 128$ 
proof -
have  $min\ 1\ ((x2 - x1) / 128) > 0$   $1/2 < (1::real)$ 
using less by auto
then obtain N where  $N: 1/2^N < min\ 1\ ((x2 - x1) / 128)$ 
by (metis power_one_over real_arch_pow_inv)
then have  $N > 0$ 
using less_divide_eq_1 by force
obtain p q where p:  $p < 2^q$   $p \neq 0$  and yeq:  $y = real\ p / 2^q$ 
using ‹y ∈ D01› by (auto simp: zero_less_divide_iff D01_def)
show ?thesis
proof
show  $0 < 2^N * p$ 
using p by auto
show  $2^N * p < 2^{(N+q)}$ 
by (simp add: p power_add)
have  $|real\ (2^N * p) / 2^{(N+q)} - (x1 + x2) / 2| = |real\ p / 2^q - (x1 + x2) / 2|$ 

```

```

      by (simp add: power_add)
    also have ... =  $|y - (x1 + x2) / 2|$ 
      by (simp add: yeq)
    also have ... <  $(x2 - x1) / 64$ 
      using dist_y by (simp add: dist_norm)
    finally show  $|real (2^N * p) / 2^{(N + q)} - (x1 + x2) / 2| < (x2$ 
 $- x1) / 64$  .
    have  $(1::real) / 2^{(N + q)} \leq 1/2^N$ 
      by (simp add: field_simps)
    also have ... <  $(x2 - x1) / 128$ 
      using N by force
    finally show  $1/2^{(N + q)} < (x2 - x1) / 128$  .
  qed
qed
obtain  $m' n' m'' n''$  where  $0 < m' m' < 2^{n'} x1 < m' / 2^{n'} m' / 2^{n'}$ 
 $< x2$ 
  and  $0 < m'' m'' < 2^{n''} x1 < m'' / 2^{n''} m'' / 2^{n''} < x2$ 
  and  $neg: h (real m'' / 2^{n''}) \neq h (real m' / 2^{n'})$ 
proof
  show  $0 < Suc (2*m)$ 
    by simp
  show  $m21: Suc (2*m) < 2^{Suc n}$ 
    using m by auto
  show  $x1 < real (Suc (2 * m)) / 2^{Suc n}$ 
    using clo by (simp add: field_simps abs_if_split: if_split_asm)
  show  $real (Suc (2 * m)) / 2^{Suc n} < x2$ 
    using n clo by (simp add: field_simps abs_if_split: if_split_asm)
  show  $0 < 4*m + 3$ 
    by simp
  have  $m+1 \leq 2^n$ 
    using m by simp
  then have  $4 * (m+1) \leq 4 * (2^n)$ 
    by simp
  then show  $m43: 4*m + 3 < 2^{(n+2)}$ 
    by (simp add: algebra_simps)
  show  $x1 < real (4 * m + 3) / 2^{(n + 2)}$ 
    using clo by (simp add: field_simps abs_if_split: if_split_asm)
  show  $real (4 * m + 3) / 2^{(n + 2)} < x2$ 
    using n clo by (simp add: field_simps abs_if_split: if_split_asm)
  have c_fold:  $midpoint (a ((2 * real m + 1) / 2^{Suc n})) (b ((2 * real m$ 
 $+ 1) / 2^{Suc n})) = c ((2 * real m + 1) / 2^{Suc n})$ 
    by (simp add: c_def)
  define R where  $R \equiv rightcut (a ((2 * real m + 1) / 2^{Suc n})) (b ((2$ 
 $* real m + 1) / 2^{Suc n})) (c ((2 * real m + 1) / 2^{Suc n}))$ 
  have  $R < b ((2 * real m + 1) / 2^{Suc n})$ 
    unfolding R_def using a_less_b [of  $4*m + 3$   $n+2$ ] a43 [of  $Suc n$   $m$ ]
    b43 [of  $Suc n$   $m$ ]
    by simp
  then have Rless:  $R < midpoint R (b ((2 * real m + 1) / 2^{Suc n}))$ 

```

```

      by (simp add: midpoint_def)
      have midR_le: midpoint R (b ((2 * real m + 1) / 2 ^ Suc n)) ≤ b ((2 *
real m + 1) / (2 * 2 ^ n))
      using ‹R < b ((2 * real m + 1) / 2 ^ Suc n)›
      by (simp add: midpoint_def)
      have (real (Suc (2 * m)) / 2 ^ Suc n) ∈ D01 real (4 * m + 3) / 2 ^ (n
+ 2) ∈ D01
      by (simp_all add: D01_def m21 m43 del: power_Suc of_nat_Suc
of_nat_add add_2_eq_Suc') blast+
      then show h (real (4 * m + 3) / 2 ^ (n + 2)) ≠ h (real (Suc (2 * m))
/ 2 ^ Suc n)
      using a_less_b [of 4*m + 3 n+2, THEN conjunct1]
      using a43 [of Suc n m] b43 [of Suc n m]
      using alec [of 2*m+1 Suc n] cleb [of 2*m+1 Suc n] a_ge_0 [of 2*m+1
Suc n] b_le_1 [of 2*m+1 Suc n]
      apply (simp add: fc_eq [symmetric] c_def del: power_Suc)
      apply (simp only: add commute [of 1] c_fold R_def [symmetric])
      apply (rule right_neq)
      using Rless apply (simp add: R_def)
      apply (rule midR_le, auto)
      done
    qed
    then show ?thesis by (metis that)
  qed
  have m_div: 0 < m / 2 ^ n m / 2 ^ n < 1
  using m by (auto simp: field_split_simps)
  have closure0m: {0..m / 2 ^ n} = closure ({0<..

```



```

proof clarsimp
  fix p q
  assume p:  $0 < \text{real } p / 2^q$   $\text{real } p / 2^q < \text{real } m / 2^n$ 
  then have [simp]:  $0 < p$ 
    by (simp add: field_split_simps)
  have [simp]:  $p < 2^q$ 
    by (blast intro: p less_2I m_div less_trans)
  have  $f (c (\text{real } p / 2^q)) \in f' \{0..c (\text{real } m / 2^n)\}$ 
    by (auto simp: clec p m)
  then show  $h (\text{real } p / 2^q) \in f' \{0..c (\text{real } m / 2^n)\}$ 
    by (simp add: h_eq)
qed
with m_div have  $h' \{0 .. m / 2^n\} \subseteq f' \{0 .. c(m / 2^n)\}$ 
  apply (subst closure0m)
  by (rule image_closure_subset [OF cont_h' closed_f]) auto
then have hx1:  $h x1 \in f' \{0 .. c(m / 2^n)\}$ 
  using x12 less.prem1 by auto
then obtain t1 where  $t1: h x1 = f t1$   $0 \leq t1$   $t1 \leq c (m / 2^n)$ 
  by auto
have  $h' (\{m / 2^n .. 1\} \cap (\bigcup q p. \{\text{real } p / 2^q\})) \subseteq f' \{c (m / 2^n) .. 1\}$ 
proof clarsimp
  fix p q
  assume p:  $\text{real } m / 2^n < \text{real } p / 2^q$  and [simp]:  $p < 2^q$ 
  then have [simp]:  $0 < p$ 
    using gr_zeroI m_div by fastforce
  have  $f (c (\text{real } p / 2^q)) \in f' \{c (m / 2^n) .. 1\}$ 
    by (auto simp: clec p m)
  then show  $h (\text{real } p / 2^q) \in f' \{c (m / 2^n) .. 1\}$ 
    by (simp add: h_eq)
qed
with m have  $h' \{m / 2^n .. 1\} \subseteq f' \{c(m / 2^n) .. 1\}$ 
  apply (subst closurem1)
  by (rule image_closure_subset [OF cont_h' closed_f]) auto
then have hx2:  $h x2 \in f' \{c(m / 2^n) .. 1\}$ 
  using x12 less.prem1 by auto
then obtain t2 where  $t2: h x2 = f t2$   $c (m / 2^n) \leq t2$   $t2 \leq 1$ 
  by auto
with t1 less neq have False
  using conn [of h x2, unfolded is_interval_connected_1 [symmetric] is_interval_1,
rule_format, of t1 t2 c(m / 2^n)]
    by (simp add: h_eq m)
  then show ?case by blast
qed auto
then show ?thesis
  by (auto simp: inj_on_def)
qed
ultimately have  $\{0..1::\text{real}\}$  homeomorphic  $f' \{0..1\}$ 
  using homeomorphic_compact [OF cont_h] by blast

```

```

then show ?thesis
  using homeomorphic_sym by blast
qed

```

```

theorem path_contains_arc:
  fixes p :: real  $\Rightarrow$  'a:: {complete_space, real_normed_vector}
  assumes path p and a: pathstart p = a and b: pathfinish p = b and a  $\neq$  b
  obtains q where arc q path_image q  $\subseteq$  path_image p pathstart q = a pathfinish
q = b
proof -
  have ucont_p: uniformly_continuous_on {0..1} p
    using path p unfolding path_def
  by (metis compact_Icc compact_uniformly_continuous)
  define  $\varphi$  where  $\varphi \equiv \lambda S. S \subseteq \{0..1\} \wedge 0 \in S \wedge 1 \in S \wedge$ 
     $(\forall x \in S. \forall y \in S. \text{open\_segment } x y \cap S = \{\}) \longrightarrow p x = p y$ 
  obtain T where closed T  $\varphi$  T and T:  $\bigwedge U. \llbracket \text{closed } U; \varphi U \rrbracket \Longrightarrow \neg (U \subset T)$ 
  proof (rule Brouwer_reduction_theorem_gen [of {0..1}  $\varphi$ ])
    have *:  $\{x <..< y\} \cap \{0..1\} = \{x <..< y\}$  if  $0 \leq x y \leq 1$   $x \leq y$  for  $x y :: \text{real}$ 
      using that by auto
    show  $\varphi \{0..1\}$ 
      by (auto simp:  $\varphi\_def$  open_segment_eq_real_ivl *)
    show  $\varphi (\bigcap (F \text{ ' UNIV}))$ 
      if  $\bigwedge n. \text{closed } (F n)$  and  $\varphi: \bigwedge n. \varphi (F n)$  and Fsub:  $\bigwedge n. F (Suc n) \subseteq F n$  for
F
  proof -
    have F01:  $\bigwedge n. F n \subseteq \{0..1\} \wedge 0 \in F n \wedge 1 \in F n$ 
      and pqF:  $\bigwedge n x y. \llbracket x \in F n; y \in F n; \text{open\_segment } x y \cap F n = \{\} \rrbracket \Longrightarrow p$ 
 $x = p y$ 
      by (metis  $\varphi \varphi\_def$ ) +
    have pqF: False if  $\forall u. x \in F u \forall x. y \in F x \text{open\_segment } x y \cap (\bigcap x. F x)$ 
    = {} and neg:  $p x \neq p y$ 
      for x y
      using that
    proof (induction x y rule: linorder_class.linorder_less_wlog)
      case (less x y)
      have xy:  $x \in \{0..1\} y \in \{0..1\}$ 
        by (metis less.prems subsetCE F01) +
      have norm(p x - p y) / 2 > 0
        using less by auto
      then obtain e where e > 0
        and e:  $\bigwedge u v. \llbracket u \in \{0..1\}; v \in \{0..1\}; \text{dist } v u < e \rrbracket \Longrightarrow \text{dist } (p v) (p u)$ 
< norm(p x - p y) / 2
      by (metis uniformly_continuous_onE [OF ucont_p])
      have minxy:  $\min e (y - x) < (y - x) * (3 / 2)$ 
        by (subst min_less_iff_disj) (simp add: less)
      define w where  $w \equiv x + (\min e (y - x) / 3)$ 
      define z where  $z \equiv y - (\min e (y - x) / 3)$ 
      have  $w < z$  and w:  $w \in \{x <..< y\}$  and z:  $z \in \{x <..< y\}$ 

```

```

    and wxe: norm(w - x) < e and zye: norm(z - y) < e
    using minxy ⟨0 < e⟩ less unfolding w_def z_def by auto
  have Fclo:  $\bigwedge T. T \in \text{range } F \implies \text{closed } T$ 
    by (metis ⟨ $\bigwedge n. \text{closed } (F\ n) \rangle$  image_iff)
  have eq:  $\{w..z\} \cap \bigcap (F \text{ ` } UNIV) = \{\}$ 
    using less w z by (simp add: open_segment_eq_real_ivl disjoint_iff)
  then obtain K where finite K and K:  $\{w..z\} \cap (\bigcap (F \text{ ` } K)) = \{\}$ 
    by (metis finite_subset_image compact_imp_fip [OF compact_interval
Fclo])
  then have K ≠ {}
    using ⟨w < z⟩ ⟨ $\{w..z\} \cap \bigcap (F \text{ ` } K) = \{\}$ ⟩ by auto
  define n where n ≡ Max K
  have n ∈ K unfolding n_def by (metis ⟨K ≠ {}⟩ ⟨finite K⟩ Max_in)
  have F n ⊆  $\bigcap (F \text{ ` } K)$ 
  unfolding n_def by (metis Fsub Max_ge ⟨K ≠ {}⟩ ⟨finite K⟩ cINF_greatest
lift_Suc_antimono_le)
  with K have wzF_null:  $\{w..z\} \cap F\ n = \{\}$ 
    by (metis disjoint_iff_not_equal subset_eq)
  obtain u where u: u ∈ F n u ∈ {x..w} ( $\{u..w\} - \{u\} \cap F\ n = \{\}$ )
  proof (cases w ∈ F n)
    case True
    then show ?thesis
      by (metis wzF_null ⟨w < z⟩ atLeastAtMost_iff disjoint_iff_not_equal
less_eq_real_def)
  next
    case False
    obtain u where u ∈ F n u ∈ {x..w} ( $\{u<..)
    proof (rule segment_to_point_exists [of F n ∩ {x..w} w])
      show closed (F n ∩ {x..w})
        by (metis ⟨ $\bigwedge n. \text{closed } (F\ n) \rangle$  closed_Int closed_real_atLeastAtMost)
      show F n ∩ {x..w} ≠ {}
    by (metis atLeastAtMost_iff disjoint_iff_not_equal greaterThanLessThan_iff
less.prem1) less_eq_real_def w
    qed (auto simp: open_segment_eq_real_ivl intro!: that)
    with False show thesis
      by (auto simp add: disjoint_iff less_eq_real_def intro!: that)
  qed
  obtain v where v: v ∈ F n v ∈ {z..y} ( $\{z..v\} - \{v\} \cap F\ n = \{\}$ )
  proof (cases z ∈ F n)
    case True
    have z ∈ {w..z}
      using ⟨w < z⟩ by auto
    then show ?thesis
      by (metis wzF_null Int_iff True empty_iff)
  next
    case False
    show ?thesis
    proof (rule segment_to_point_exists [of F n ∩ {z..y} z])
      show closed (F n ∩ {z..y})$ 
```

```

    by (metis ‹ $\bigwedge n. \text{closed } (F\ n) \rangle \text{closed\_Int closed\_atLeastAtMost}$ )
    show  $F\ n \cap \{z..y\} \neq \{\}$ 
    by (metis atLeastAtMost_iff disjoint_iff_not_equal greaterThanLessThan_iff
less.premis(2) less_eq_real_def z)
    show  $\bigwedge b. \llbracket b \in F\ n \cap \{z..y\}; \text{open\_segment } z\ b \cap (F\ n \cap \{z..y\}) = \{\} \rrbracket$ 
 $\implies$  thesis
    proof
    show  $\bigwedge b. \llbracket b \in F\ n \cap \{z..y\}; \text{open\_segment } z\ b \cap (F\ n \cap \{z..y\}) = \{\} \rrbracket$ 
 $\implies (\{z..b\} - \{b\}) \cap F\ n = \{\}$ 
    using False by (auto simp: open_segment_eq_real_ivl less_eq_real_def)
    qed auto
    qed
    qed
    obtain  $u\ v$  where  $u \in \{0..1\}\ v \in \{0..1\}\ \text{norm}(u - x) < e\ \text{norm}(v - y)$ 
 $< e\ p\ u = p\ v$ 
    proof
    show  $u \in \{0..1\}\ v \in \{0..1\}$ 
    by (metis F01 ‹ $u \in F\ n \rangle \langle v \in F\ n \rangle \text{subsetD}$ ) +
    show  $\text{norm}(u - x) < e\ \text{norm}(v - y) < e$ 
    using ‹ $u \in \{x..w\} \rangle \langle v \in \{z..y\} \rangle \text{atLeastAtMost\_iff real\_norm\_def } wx$ 
v(3)) by auto
    show  $p\ u = p\ v$ 
    proof (rule peg)
    show  $u \in F\ n\ v \in F\ n$ 
    by (auto simp: u v)
    have False if  $\xi \in F\ n\ u < \xi\ \xi < v$  for  $\xi$ 
    proof -
    have  $\xi \notin \{z..v\}$ 
    by (metis DiffI disjoint_iff_not_equal less_irrefl singletonD that(1,3))
    moreover have  $\xi \notin \{w..z\} \cap F\ n$ 
    by (metis equals0D wzF_null)
    ultimately have  $\xi \in \{u..w\}$ 
    using that by auto
    then show ?thesis
    by (metis DiffI disjoint_iff_not_equal less_eq_real_def not_le
singletonD that(1,2) u(3))
    qed
    moreover
    have  $\llbracket \xi \in F\ n; v < \xi; \xi < u \rrbracket \implies \text{False}$  for  $\xi$ 
    using ‹ $u \in \{x..w\} \rangle \langle v \in \{z..y\} \rangle \langle w < z \rangle$  by simp
    ultimately
    show  $\text{open\_segment } u\ v \cap F\ n = \{\}$ 
    by (force simp: open_segment_eq_real_ivl)
    qed
    qed
    then show ?case
    using e [of x u] e [of y v] xy
    by (metis dist_norm dist_triangle_half_r order_less_irrefl)

```

```

qed (auto simp: open_segment_commute)
show ?thesis
  unfolding  $\varphi\_def$  by (metis (no_types, opaque_lifting) INT_I Inf_lower2
rangeI that(3) F01 subsetCE pqF)
qed
show closed  $\{0..1::real\}$  by auto
qed (meson  $\varphi\_def$ )
then have  $T \subseteq \{0..1\}$   $0 \in T$   $1 \in T$ 
  and  $peq: \bigwedge x y. \llbracket x \in T; y \in T; open\_segment\ x\ y \cap T = \{\} \rrbracket \implies p\ x = p\ y$ 
  unfolding  $\varphi\_def$  by metis+
then have  $T \neq \{\}$  by auto
define  $h$  where  $h \equiv \lambda x. p(SOME\ y. y \in T \wedge open\_segment\ x\ y \cap T = \{\})$ 
have  $p\ y = p\ z$  if  $y \in T$   $z \in T$  and  $xyT: open\_segment\ x\ y \cap T = \{\}$  and  $xzT: open\_segment\ x\ z \cap T = \{\}$ 
  for  $x\ y\ z$ 
proof (cases  $x \in T$ )
case True
  with that show ?thesis by (metis  $\langle \varphi\ T \rangle \varphi\_def$ )
next
case False
  have  $insert\ x\ (open\_segment\ x\ y \cup open\_segment\ x\ z) \cap T = \{\}$ 
    by (metis False Int_Un_distrib2 Int_insert_left Un_empty_right xyT xzT)
  moreover have  $open\_segment\ y\ z \cap T \subseteq insert\ x\ (open\_segment\ x\ y \cup open\_segment\ x\ z) \cap T$ 
    by (auto simp: open_segment_eq_real_ivl)
  ultimately have  $open\_segment\ y\ z \cap T = \{\}$ 
    by blast
  with that  $peq$  show ?thesis by metis
qed
then have  $h\_eq\_p\_gen: h\ x = p\ y$  if  $y \in T$   $open\_segment\ x\ y \cap T = \{\}$  for  $x$ 
  using that unfolding  $h\_def$ 
  by (metis (mono_tags, lifting) some_eq_ex)
then have  $h\_eq\_p: \bigwedge x. x \in T \implies h\ x = p\ x$ 
  by simp
have disjoint:  $\bigwedge x. \exists y. y \in T \wedge open\_segment\ x\ y \cap T = \{\}$ 
  by (meson  $\langle T \neq \{\} \rangle \langle closed\ T \rangle segment\_to\_point\_exists$ )
have  $heq: h\ x = h\ x'$  if  $open\_segment\ x\ x' \cap T = \{\}$  for  $x\ x'$ 
proof (cases  $x \in T \vee x' \in T$ )
case True
  then show ?thesis
    by (metis  $h\_eq\_p\ h\_eq\_p\_gen\ open\_segment\_commute\ that$ )
next
case False
  obtain  $y\ y'$  where  $y \in T$   $open\_segment\ x\ y \cap T = \{\}$   $h\ x = p\ y$ 
     $y' \in T$   $open\_segment\ x'\ y' \cap T = \{\}$   $h\ x' = p\ y'$ 
    by (meson disjoint  $h\_eq\_p\_gen$ )
  moreover have  $open\_segment\ y\ y' \subseteq (insert\ x\ (insert\ x'\ (open\_segment\ x\ y \cup open\_segment\ x'\ y' \cup open\_segment\ x\ x')))$ 

```

```

    by (auto simp: open_segment_eq_real_ivl)
  ultimately show ?thesis
    using False that by (fastforce simp add: h_eq_p intro!: peg)
qed
have h ' {0..1} homeomorphic {0..1::real}
proof (rule homeomorphic_monotone_image_interval)
  show continuous_on {0..1} h
  proof (clarsimp simp add: continuous_on_iff)
    fix u ε::real
    assume 0 < ε 0 ≤ u u ≤ 1
    then obtain δ where δ > 0 and δ:  $\bigwedge v. v \in \{0..1\} \implies \text{dist } v \ u < \delta \longrightarrow$ 
dist (p v) (p u) < ε / 2
    using ucont_p [unfolded uniformly_continuous_on_def]
    by (metis atLeastAtMost_iff half_gt_zero_iff)
  then have dist (h v) (h u) < ε if v ∈ {0..1} dist v u < δ for v
  proof (cases open_segment u v ∩ T = {})
    case True
    then show ?thesis
      using ⟨0 < ε⟩ heq by auto
  next
    case False
    have uvT: closed (closed_segment u v ∩ T) closed_segment u v ∩ T ≠ {}
      using False open_closed_segment by (auto simp: ⟨closed T⟩ closed_Int)
    obtain w where w ∈ T and w: w ∈ closed_segment u v open_segment u
w ∩ T = {}
    proof (rule segment_to_point_exists [OF uvT])
      fix b
      assume b ∈ closed_segment u v ∩ T open_segment u b ∩ (closed_segment
u v ∩ T) = {}
      then show thesis
        by (metis IntD1 IntD2 ends_in_segment(1) inf.orderE inf_assoc
subset_oc_segment that)
    qed
    then have puw: dist (p u) (p w) < ε / 2
      by (metis (no_types) ⟨T ⊆ {0..1}⟩ ⟨dist v u < δ⟩ δ dist_commute
dist_in_closed_segment le_less_trans subsetCE)
    obtain z where z ∈ T and z: z ∈ closed_segment u v open_segment v z
∩ T = {}
    proof (rule segment_to_point_exists [OF uvT])
      fix b
      assume b ∈ closed_segment u v ∩ T open_segment v b ∩ (closed_segment
u v ∩ T) = {}
      then show thesis
        by (metis IntD1 IntD2 ends_in_segment(2) inf.orderE inf_assoc
subset_oc_segment that)
    qed
    then have dist (p u) (p z) < ε / 2
      by (metis ⟨T ⊆ {0..1}⟩ ⟨dist v u < δ⟩ δ dist_commute dist_in_closed_segment
le_less_trans subsetCE)

```

```

    then show ?thesis
      using puw by (metis (no_types) ‹w ∈ T› ‹z ∈ T› dist_commute
dist_triangle_half_l h_eq_p_gen w(2) z(2))
    qed
    with ‹0 < δ› show ∃δ>0. ∀v∈{0..1}. dist v u < δ ⟶ dist (h v) (h u) < ε
  by blast
  qed
  show connected ({0..1} ∩ h - ' {z}) for z
  proof (clarsimp simp add: connected_iff_connected_component)
    fix u v
    assume huv_eq: h v = h u and uv: 0 ≤ u u ≤ 1 0 ≤ v v ≤ 1
    have ∃ T. connected T ∧ T ⊆ {0..1} ∧ T ⊆ h - ' {h u} ∧ u ∈ T ∧ v ∈ T
    proof (intro exI conjI)
      show connected (closed_segment u v)
        by simp
      show closed_segment u v ⊆ {0..1}
        by (simp add: uv closed_segment_eq_real_ivl)
      have pxy: p x = p y
        if T ⊆ {0..1} 0 ∈ T 1 ∈ T x ∈ T y ∈ T
        and disjT: open_segment x y ∩ (T - open_segment u v) = {}
        and xynot: x ∉ open_segment u v y ∉ open_segment u v
        for x y
      proof (cases open_segment x y ∩ open_segment u v = {})
        case True
          then show ?thesis
            by (metis Diff_Int_distrib Diff_empty peq disjT ‹x ∈ T› ‹y ∈ T›)
        next
          case False
            then have open_segment x u ∪ open_segment y v ⊆ open_segment x y
            - open_segment u v ∨
              open_segment y u ∪ open_segment x v ⊆ open_segment x y -
              open_segment u v (is ?xuyv ∨ ?yuxv)
            using xynot by (fastforce simp add: open_segment_eq_real_ivl not_le
not_less split: if_split_asm)
            then show p x = p y
              proof
                assume ?xuyv
                then have open_segment x u ∩ T = {} open_segment y v ∩ T = {}
                  using disjT by auto
                then have h x = h y
                  using heq huv_eq by auto
                then show ?thesis
                  using h_eq_p ‹x ∈ T› ‹y ∈ T› by auto
              next
                assume ?yuxv
                then have open_segment y u ∩ T = {} open_segment x v ∩ T = {}
                  using disjT by auto
                then have h x = h y
                  using heq [of y u] heq [of x v] huv_eq by auto
              end
            end
          end
        end
      end
    end
  end

```

```

      then show ?thesis
        using h_eq_p ⟨x ∈ T⟩ ⟨y ∈ T⟩ by auto
    qed
  qed
  have ¬ T - open_segment u v ⊂ T
  proof (rule T)
    show closed (T - open_segment u v)
      by (simp add: closed_Diff [OF ⟨closed T⟩] open_segment_eq_real_ivl)
    have 0 ∉ open_segment u v 1 ∉ open_segment u v
      using open_segment_eq_real_ivl uv by auto
    then show φ (T - open_segment u v)
      using ⟨T ⊆ {0..1}⟩ ⟨0 ∈ T⟩ ⟨1 ∈ T⟩
      by (auto simp: φ_def) (meson peq prxy)
  qed
  then have open_segment u v ∩ T = {}
    by blast
  then show closed_segment u v ⊆ h - {h u}
    by (force intro: heq simp: open_segment_eq_real_ivl closed_segment_eq_real_ivl
split: if_split_asm)+
  qed auto
  then show connected_component ({0..1} ∩ h - {h u}) u v
    by (simp add: connected_component_def)
  qed
  show h 1 ≠ h 0
    by (metis ⟨φ T⟩ φ_def a ⟨a ≠ b⟩ b h_eq_p pathfinish_def pathstart_def)
  qed
  then obtain f and g :: real ⇒ 'a
    where gfeq: (∀ x ∈ h - {0..1}. (g(f x) = x)) and fhim: f - h - {0..1} = {0..1}
  and contf: continuous_on (h - {0..1}) f
    and fgeq: (∀ y ∈ {0..1}. (f(g y) = y)) and pag: path_image g = h - {0..1}
  and contg: continuous_on {0..1} g
    by (auto simp: homeomorphic_def homeomorphism_def path_image_def)
  then have arc g
    by (metis arc_def path_def inj_on_def)
  obtain u v where u ∈ {0..1} a = g u v ∈ {0..1} b = g v
    by (metis (mono_tags, opaque_lifting) ⟨φ T⟩ φ_def a b fhim gfeq h_eq_p im-
ageI path_image_def pathfinish_def pathfinish_in_path_image pathstart_def path-
start_in_path_image)
  then have a ∈ path_image g b ∈ path_image g
    using path_image_def by blast+
  have ph: path_image h ⊆ path_image p
    by (metis image_mono image_subset_iff path_image_def disjoint h_eq_p_gen
⟨T ⊆ {0..1}⟩)
  show ?thesis
  proof
    show pathstart (subpath u v g) = a pathfinish (subpath u v g) = b
      by (simp_all add: ⟨a = g u⟩ ⟨b = g v⟩)
    show path_image (subpath u v g) ⊆ path_image p
      by (metis ⟨u ∈ {0..1}⟩ ⟨v ∈ {0..1}⟩ order_trans pag path_image_def)
  end

```



```

path_image_subpath_subset ph)
  show arc (subpath u v g)
    using ⟨arc g⟩ ⟨a = g u⟩ ⟨b = g v⟩ ⟨u ∈ {0..1}⟩ ⟨v ∈ {0..1}⟩ arc_subpath_arc
  ⟨a ≠ b⟩ by blast
qed
qed

```

corollary *path_connected_arcwise:*

```

  fixes S :: 'a::{complete_space,real_normed_vector} set
  shows path_connected S ⟷
    (∀ x ∈ S. ∀ y ∈ S. x ≠ y ⟶ (∃ g. arc g ∧ path_image g ⊆ S ∧ pathstart g
= x ∧ pathfinish g = y))
    (is ?lhs = ?rhs)
proof (intro iffI impI ballI)
  fix x y
  assume path_connected S x ∈ S y ∈ S x ≠ y
  then obtain p where p: path p path_image p ⊆ S pathstart p = x pathfinish p
= y
  by (force simp: path_connected_def)
  then show ∃ g. arc g ∧ path_image g ⊆ S ∧ pathstart g = x ∧ pathfinish g = y
  by (metis ⟨x ≠ y⟩ order_trans path_contains_arc)
next
  assume R [rule_format]: ?rhs
  show ?lhs
    unfolding path_connected_def
  proof (intro ballI)
    fix x y
    assume x ∈ S y ∈ S
    show ∃ g. path g ∧ path_image g ⊆ S ∧ pathstart g = x ∧ pathfinish g = y
    proof (cases x = y)
      case True with ⟨x ∈ S⟩ path_component_def path_component_refl show
?thesis
        by blast
      next
        case False with R [OF ⟨x ∈ S⟩ ⟨y ∈ S⟩] show ?thesis
          by (auto intro: arc_imp_path)
    qed
  qed
qed

```

corollary *arc_connected_trans:*

```

  fixes g :: real ⟹ 'a::{complete_space,real_normed_vector}
  assumes arc g arc h pathfinish g = pathstart h pathstart g ≠ pathfinish h
  obtains i where arc i path_image i ⊆ path_image g ∪ path_image h
    pathstart i = pathstart g pathfinish i = pathfinish h
  by (metis (no_types, opaque_lifting) arc_imp_path assms path_contains_arc
path_image_join path_join pathfinish_join pathstart_join)

```

6.40.4 Accessibility of frontier points

lemma *dense_accessible_frontier_points*:

fixes $S :: 'a::\{complete\_space, real\_normed\_vector\}$ *set*

assumes *open S* **and** *opeSV: openin (top_of_set (frontier S)) V* **and** $V \neq \{\}$

obtains *g* **where** $arc\ g\ g\ ' \{0..<1\} \subseteq S$ *pathstart* $g \in S$ *pathfinish* $g \in V$

proof –

obtain z **where** $z \in V$

using $\langle V \neq \{\} \rangle$ **by** *auto*

then obtain r **where** $r > 0$ **and** $r: ball\ z\ r \cap frontier\ S \subseteq V$

by (*metis openin_contains_ball opeSV*)

then have $z \in frontier\ S$

using $\langle z \in V \rangle$ *opeSV openin_contains_ball* **by** *blast*

then have $z \in closure\ S$ $z \notin S$

by (*simp_all add: frontier_def assms interior_open*)

with $\langle r > 0 \rangle$ **have** *infinite* $(S \cap ball\ z\ r)$

by (*auto simp: closure_def islimpt_eq_infinite_ball*)

then obtain y **where** $y \in S$ **and** $y: y \in ball\ z\ r$

using *infinite_imp_nonempty* **by** *force*

then have $y \notin frontier\ S$

by (*meson* $\langle open\ S \rangle$ *disjoint_iff_not_equal frontier_disjoint_eq*)

have $y \neq z$

using $\langle y \in S \rangle \langle z \notin S \rangle$ **by** *blast*

have *path_connected* $(ball\ z\ r)$

by (*simp add: convex_imp_path_connected*)

with $y \langle r > 0 \rangle$ **obtain** g **where** *arc* g **and** *path_image* $g \subseteq ball\ z\ r$

and $g: pathstart\ g = y\ pathfinish\ g = z$

using $\langle y \neq z \rangle$ **by** (*force simp: path_connected_arcwise*)

have *continuous_on* $\{0..1\}$ g

using $\langle arc\ g \rangle$ *arc_imp_path path_def* **by** *blast*

then have *compact* $(g - ' frontier\ S \cap \{0..1\})$

by (*simp add: bounded_Int closed_Diff closed_vimage_Int compact_eq_bounded_closed*)

moreover have $g - ' frontier\ S \cap \{0..1\} \neq \{\}$

proof –

have $\exists r. r \in g - ' frontier\ S \wedge r \in \{0..1\}$

by (*metis* $\langle z \in frontier\ S \rangle$ $g(2)$ *imageE path_image_def pathfinish_in_path_image vimageI2*)

then show *?thesis*

by *blast*

qed

ultimately obtain t **where** $g\ t \in frontier\ S$ **and** $0 \leq t \leq 1$

and $t: \bigwedge u. \llbracket g\ u \in frontier\ S; 0 \leq u; u \leq 1 \rrbracket \implies t \leq u$

by (*force simp: dest!: compact_attains_inf*)

moreover have $t \neq 0$

by (*metis* $\langle y \notin frontier\ S \rangle$ $g(1)$ *gt pathstart_def*)

ultimately have $t01: 0 < t \leq 1$

by *auto*

have $V \subseteq frontier\ S$

using *opeSV openin_contains_ball* **by** *blast*

show *?thesis*

```

proof
  show arc (subpath 0 t g)
    by (simp add:  $\langle 0 \leq t \rangle \langle t \leq 1 \rangle \langle \text{arc } g \rangle \langle t \neq 0 \rangle \text{arc\_subpath\_arc}$ )
  have  $g \ 0 \in S$ 
    by (metis  $\langle y \in S \rangle g(1) \text{pathstart\_def}$ )
  then show pathstart (subpath 0 t g)  $\in S$ 
    by auto
  have  $g \ t \in V$ 
    by (metis IntI atLeastAtMost_iff gt image_eqI path_image_def pig r subsetCE
 $\langle 0 \leq t \rangle \langle t \leq 1 \rangle$ )
  then show pathfinish (subpath 0 t g)  $\in V$ 
    by auto
  then have inj_on (subpath 0 t g)  $\{0..1\}$ 
    using t01  $\langle \text{arc } (subpath \ 0 \ t \ g) \rangle \text{arc\_imp\_inj\_on}$  by blast
  then have subpath 0 t g  $' \{0..<1\} \subseteq \text{subpath } 0 \ t \ g \ ' \{0..1\} - \{\text{subpath } 0 \ t \ g$ 
1}
    by (force simp: dest: inj_onD)
  moreover have False if subpath 0 t g  $' (\{0..<1\}) - S \neq \{\}$ 
  proof -
    have contg: continuous_on  $\{0..1\}$  g
      using  $\langle \text{arc } g \rangle$  by (auto simp: arc_def path_def)
    have subpath 0 t g  $' \{0..<1\} \cap \text{frontier } S \neq \{\}$ 
    proof (rule connected_Int_frontier [OF _ _ that])
      show connected (subpath 0 t g  $' \{0..<1\}$ )
      proof (rule connected_continuous_image)
        show continuous_on  $\{0..<1\}$  (subpath 0 t g)
        by (meson  $\langle \text{arc } (subpath \ 0 \ t \ g) \rangle \text{arc\_def atLeastLessThan\_subseteq\_atLeastAtMost\_iff}$ 
continuous_on_subset order_refl path_def)
      qed auto
      show subpath 0 t g  $' \{0..<1\} \cap S \neq \{\}$ 
      using  $\langle y \in S \rangle g(1)$  by (force simp: subpath_def image_def pathstart_def)
    qed
    then obtain x where  $x \in \text{subpath } 0 \ t \ g \ ' \{0..<1\} \ x \in \text{frontier } S$ 
    by blast
    with t01  $\langle 0 \leq t \rangle \text{mult\_le\_one } t$  show False
      by (fastforce simp: subpath_def)
  qed
  then have subpath 0 t g  $' \{0..1\} - \{\text{subpath } 0 \ t \ g \ 1\} \subseteq S$ 
    using subsetD by fastforce
  ultimately show subpath 0 t g  $' \{0..<1\} \subseteq S$ 
    by auto
qed
qed
qed

```

```

lemma dense_accessible_frontier_points_connected:
  fixes  $S :: 'a :: \{\text{complete\_space}, \text{real\_normed\_vector}\}$  set
  assumes open S connected S  $x \in S \ V \neq \{\}$ 
  and ope: openin (top_of_set (frontier S)) V

```

obtains g **where** $\text{arc } g \text{ } g \text{ } ' \{0..<1\} \subseteq S \text{ pathstart } g = x \text{ pathfinish } g \in V$
proof –
have $V \subseteq \text{frontier } S$
using $\text{ope openin\_imp\_subset}$ **by** blast
with $\langle \text{open } S \rangle \langle x \in S \rangle$ **have** $x \notin V$
using interior_open **by** $(\text{auto simp: frontier\_def})$
obtain g **where** $\text{arc } g$ **and** $g: g \text{ } ' \{0..<1\} \subseteq S \text{ pathstart } g \in S \text{ pathfinish } g \in V$
by $(\text{metis dense\_accessible\_frontier\_points } [OF \langle \text{open } S \rangle \text{ ope } \langle V \neq \{\} \rangle])$
then have $\text{path\_connected } S$
by $(\text{simp add: assms connected\_open\_path\_connected})$
with $\langle \text{pathstart } g \in S \rangle \langle x \in S \rangle$ **have** $\text{path\_component } S \ x \ (\text{pathstart } g)$
by $(\text{simp add: path\_connected\_component})$
then obtain f **where** $\text{path } f$ **and** $f: \text{path\_image } f \subseteq S \text{ pathstart } f = x \text{ pathfinish } f = \text{pathstart } g$
by $(\text{auto simp: path\_component\_def})$
then have $\text{path } (f +++ g)$
by $(\text{simp add: } \langle \text{arc } g \rangle \text{ arc\_imp\_path})$
then obtain h **where** $\text{arc } h$
and $h: \text{path\_image } h \subseteq \text{path\_image } (f +++ g) \text{ pathstart } h = x \text{ pathfinish } h = \text{pathfinish } g$
using path_contains_arc $[of \ f \ +++ \ g \ x \ \text{pathfinish } g]$ $\langle x \notin V \rangle \langle \text{pathfinish } g \in V \rangle f$
by $(\text{metis pathfinish\_join pathstart\_join})$
have $\text{path\_image } h \subseteq \text{path\_image } f \cup \text{path\_image } g$
using $h(1) \text{ path\_image\_join\_subset}$ **by** auto
then have $h \text{ } ' \{0..1\} - \{h \ 1\} \subseteq S$
using $f \ g \ h$
apply $(\text{simp add: path\_image\_def pathfinish\_def subset\_iff image\_def Bex\_def})$
by (metis le_less)
then have $h \text{ } ' \{0..<1\} \subseteq S$
using $\langle \text{arc } h \rangle$ **by** $(\text{force simp: arc\_def dest: inj\_onD})$
then show thesis
using $\langle \text{arc } h \rangle \ g(3) \ h$ **that** **by** presburger
qed

lemma $\text{dense\_access\_fp\_aux}$:

fixes $S :: 'a :: \{\text{complete\_space, real\_normed\_vector}\} \text{ set}$
assumes $S: \text{open } S \text{ connected } S$
and $\text{opeSU: openin } (\text{top\_of\_set } (\text{frontier } S)) \ U$
and $\text{opeSV: openin } (\text{top\_of\_set } (\text{frontier } S)) \ V$
and $V \neq \{\} \neg U \subseteq V$
obtains g **where** $\text{arc } g \text{ pathstart } g \in U \text{ pathfinish } g \in V \text{ } g \text{ } ' \{0 < .. < 1\} \subseteq S$
proof –
have $S \neq \{\}$
using $\text{opeSV } \langle V \neq \{\} \rangle$ **by** $(\text{metis frontier\_empty openin\_subtopology\_empty})$
then obtain x **where** $x \in S$ **by** auto
obtain g **where** $\text{arc } g$ **and** $g: g \text{ } ' \{0..<1\} \subseteq S \text{ pathstart } g = x \text{ pathfinish } g \in V$
using $\text{dense\_accessible\_frontier\_points\_connected}$ $[OF \ S \ \langle x \in S \rangle \langle V \neq \{\} \rangle \text{ opeSV}]$ **by** blast

```

obtain  $h$  where  $\text{arc } h$  and  $h: h \in \{0..<1\} \subseteq S$   $\text{pathstart } h = x$   $\text{pathfinish } h \in U - \{\text{pathfinish } g\}$ 
proof (rule  $\text{dense\_accessible\_frontier\_points\_connected}$  [ $OF\ S\ \langle x \in S \rangle$ ])
  show  $U - \{\text{pathfinish } g\} \neq \{\}$ 
    using  $\langle \text{pathfinish } g \in V \rangle \langle \neg U \subseteq V \rangle$  by blast
  show  $\text{openin } (\text{top\_of\_set } (\text{frontier } S))\ (U - \{\text{pathfinish } g\})$ 
    by (simp add:  $\text{opeSU openin\_delete}$ )
qed auto
obtain  $\gamma$  where  $\text{arc } \gamma$ 
  and  $\gamma: \text{path\_image } \gamma \subseteq \text{path\_image } (\text{reversepath } h \text{ +++ } g)$ 
     $\text{pathstart } \gamma = \text{pathfinish } h$   $\text{pathfinish } \gamma = \text{pathfinish } g$ 
proof (rule  $\text{path\_contains\_arc}$  [ $of\ (\text{reversepath } h \text{ +++ } g)\ \text{pathfinish } h\ \text{pathfinish } g$ ])
  show  $\text{path } (\text{reversepath } h \text{ +++ } g)$ 
    by (simp add:  $\langle \text{arc } g \rangle \langle \text{arc } h \rangle \langle \text{pathstart } g = x \rangle \langle \text{pathstart } h = x \rangle \text{arc\_imp\_path}$ )
  show  $\text{pathstart } (\text{reversepath } h \text{ +++ } g) = \text{pathfinish } h$ 
     $\text{pathfinish } (\text{reversepath } h \text{ +++ } g) = \text{pathfinish } g$ 
    by auto
  show  $\text{pathfinish } h \neq \text{pathfinish } g$ 
    using  $\langle \text{pathfinish } h \in U - \{\text{pathfinish } g\} \rangle$  by auto
qed auto
show ?thesis
proof
  show  $\text{arc } \gamma$   $\text{pathstart } \gamma \in U$   $\text{pathfinish } \gamma \in V$ 
    using  $\gamma \langle \text{arc } \gamma \rangle \langle \text{pathfinish } h \in U - \{\text{pathfinish } g\} \rangle \langle \text{pathfinish } g \in V \rangle$  by
auto
  have  $\text{path\_image } \gamma \subseteq \text{path\_image } h \cup \text{path\_image } g$ 
    by (metis  $\gamma(1)\ g(2)\ h(2)\ \text{path\_image\_join}\ \text{path\_image\_reversepath}\ \text{pathfinish\_reversepath}$ )
  then have  $\gamma \in \{0..1\} - \{\gamma\ 0, \gamma\ 1\} \subseteq S$ 
    using  $\gamma\ g\ h$ 
    apply (simp add:  $\text{path\_image\_def}\ \text{pathstart\_def}\ \text{pathfinish\_def}\ \text{subset\_iff}\ \text{image\_def}\ \text{Bex\_def}$ )
    by (metis  $\text{linorder\_neqE}\ \text{linordered\_idom}\ \text{not\_less}$ )
  then show  $\gamma \in \{0..<1\} \subseteq S$ 
    using  $\langle \text{arc } h \rangle \langle \text{arc } \gamma \rangle$ 
    by (metis  $\text{arc\_imp\_simple\_path}\ \text{path\_image\_def}\ \text{pathfinish\_def}\ \text{pathstart\_def}\ \text{simple\_path\_endless}$ )
qed
qed

lemma  $\text{dense\_accessible\_frontier\_point\_pairs}$ :
fixes  $S :: 'a::\{\text{complete\_space}, \text{real\_normed\_vector}\}\ \text{set}$ 
assumes  $S$ :  $\text{open } S$   $\text{connected } S$ 
  and  $\text{opeSU}$ :  $\text{openin } (\text{top\_of\_set } (\text{frontier } S))\ U$ 
  and  $\text{opeSV}$ :  $\text{openin } (\text{top\_of\_set } (\text{frontier } S))\ V$ 
  and  $U \neq \{\}$   $V \neq \{\}$   $U \neq V$ 
obtains  $g$  where  $\text{arc } g$   $\text{pathstart } g \in U$   $\text{pathfinish } g \in V$   $g \in \{0..<1\} \subseteq S$ 
proof –

```

```

consider  $\neg U \subseteq V \mid \neg V \subseteq U$ 
  using  $\langle U \neq V \rangle$  by blast
then show ?thesis
proof cases
  case 1 then show ?thesis
    using assms dense_access_fp_aux [OF S opeSU opeSV] that by blast
  next
  case 2
  obtain g where arc g and g: pathstart g  $\in V$  pathfinish g  $\in U$  g '  $\{0 < .. < 1\}$ 
 $\subseteq S$ 
    using assms dense_access_fp_aux [OF S opeSV opeSU] 2 by blast
  show ?thesis
  proof
    show arc (reversepath g)
      by (simp add:  $\langle \text{arc } g \rangle$  arc_reversepath)
    show pathstart (reversepath g)  $\in U$  pathfinish (reversepath g)  $\in V$ 
      using g by auto
    show reversepath g '  $\{0 < .. < 1\} \subseteq S$ 
      using g by (auto simp: reversepath_def)
  qed
qed
qed
end

```

6.41 Absolute Retracts, Absolute Neighbourhood Retracts and Euclidean Neighbourhood Retracts

```

theory Retracts
imports
  Brouwer_Fixpoint
  Continuous_Extension
begin

```

Absolute retracts (AR), absolute neighbourhood retracts (ANR) and also Euclidean neighbourhood retracts (ENR). We define AR and ANR by specializing the standard definitions for a set to embedding in spaces of higher dimension.

John Harrison writes: "This turns out to be sufficient (since any set in $\mathbb{R}^n$ can be embedded as a closed subset of a convex subset of $\mathbb{R}^{n+1}$) to derive the usual definitions, but we need to split them into two implications because of the lack of type quantifiers. Then ENR turns out to be equivalent to ANR plus local compactness."

definition $AR :: 'a::topological\_space \text{ set} \Rightarrow \text{bool}$ **where**
 $AR\ S \equiv \forall U. \forall S': (a * \text{real}) \text{ set.}$

S homeomorphic $S' \wedge \text{closedin } (\text{top\_of\_set } U) S' \longrightarrow S' \text{ retract\_of } U$

definition $\text{ANR} :: 'a::\text{topological\_space} \text{ set} \Rightarrow \text{bool}$ **where**

$\text{ANR } S \equiv \forall U. \forall S'::('a * \text{real}) \text{ set.}$

$S \text{ homeomorphic } S' \wedge \text{closedin } (\text{top\_of\_set } U) S'$

$\longrightarrow (\exists T. \text{openin } (\text{top\_of\_set } U) T \wedge S' \text{ retract\_of } T)$

definition $\text{ENR} :: 'a::\text{topological\_space} \text{ set} \Rightarrow \text{bool}$ **where**

$\text{ENR } S \equiv \exists U. \text{open } U \wedge S \text{ retract\_of } U$

First, show that we do indeed get the "usual" properties of ARs and ANRs.

lemma $\text{AR\_imp\_absolute\_extensor}$:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$

assumes $\text{AR } S$ **and** $\text{contf: continuous\_on } T f$ **and** $f \text{ ' } T \subseteq S$

and $\text{cloUT: closedin } (\text{top\_of\_set } U) T$

obtains g **where** $\text{continuous\_on } U g$ $g \text{ ' } U \subseteq S \wedge x. x \in T \Longrightarrow g x = f x$

proof –

have $\text{aff\_dim } S < \text{int } (\text{DIM } ('b \times \text{real}))$

using $\text{aff\_dim\_le\_DIM [of } S]$ **by** simp

then obtain C **and** $S' :: ('b * \text{real}) \text{ set}$

where $C: \text{convex } C$ $C \neq \{\}$

and $\text{cloCS: closedin } (\text{top\_of\_set } C) S'$

and $\text{hom: } S \text{ homeomorphic } S'$

by $(\text{metis that homeomorphic\_closedin\_convex})$

then have $S' \text{ retract\_of } C$

using $\langle \text{AR } S \rangle$ **by** $(\text{simp add: AR\_def})$

then obtain r **where** $S' \subseteq C$ **and** $\text{contr: continuous\_on } C r$

and $r \text{ ' } C \subseteq S'$ **and** $\text{rid: } \wedge x. x \in S' \Longrightarrow r x = x$

by $(\text{auto simp: retraction\_def retract\_of\_def})$

obtain $g h$ **where** $\text{homeomorphism } S S' g h$

using hom **by** $(\text{force simp: homeomorphic\_def})$

then have $\text{continuous\_on } (f \text{ ' } T) g$

by $(\text{meson } \langle f \text{ ' } T \subseteq S \rangle \text{ continuous\_on\_subset homeomorphism\_def})$

then have $\text{contgf: continuous\_on } T (g \circ f)$

by $(\text{metis continuous\_on\_compose contf})$

have $\text{gfTC: } (g \circ f) \text{ ' } T \subseteq C$

proof –

have $g \text{ ' } S = S'$

by $(\text{metis (no\_types) } \langle \text{homeomorphism } S S' g h \rangle \text{ homeomorphism\_def})$

with $\langle S' \subseteq C \rangle \langle f \text{ ' } T \subseteq S \rangle$ **show** $?thesis$ **by** force

qed

obtain f' **where** $f': \text{continuous\_on } U f'$ $f' \text{ ' } U \subseteq C$

$\wedge x. x \in T \Longrightarrow f' x = (g \circ f) x$

by $(\text{metis Dugundji [OF } C \text{ cloUT contgf gfTC]})$

show $?thesis$

proof $(\text{rule\_tac } g = h \circ r \circ f' \text{ in that})$

show $\text{continuous\_on } U (h \circ r \circ f')$

proof $(\text{intro continuous\_on\_compose } f')$

show $\text{continuous\_on } (f' \text{ ' } U) r$

```

    using continuous_on_subset contr f' by blast
  show continuous_on (r ' f' ' U) h
    using ⟨homeomorphism S S' g h⟩ ⟨f' ' U ⊆ C⟩
    unfolding homeomorphism_def
    by (metis ⟨r ' C ⊆ S'⟩ continuous_on_subset image_mono)
qed
show (h ∘ r ∘ f') ' U ⊆ S
  using ⟨homeomorphism S S' g h⟩ ⟨r ' C ⊆ S'⟩ ⟨f' ' U ⊆ C⟩
  by (fastforce simp: homeomorphism_def)
show  $\bigwedge x. x \in T \implies (h \circ r \circ f') x = f x$ 
  using ⟨homeomorphism S S' g h⟩ ⟨f' ' T ⊆ S⟩ f'
  by (auto simp: rid homeomorphism_def)
qed
qed

lemma AR_imp_absolute_retract:
  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  assumes AR S S homeomorphic S'
    and clo: closedin (top_of_set U) S'
  shows S' retract_of U
proof -
  obtain g h where hom: homeomorphism S S' g h
    using assms by (force simp: homeomorphic_def)
  obtain h: continuous_on S' h h ' S' ⊆ S
    using hom homeomorphism_def by blast
  obtain h' where h': continuous_on U h' h' ' U ⊆ S
    and h'h:  $\bigwedge x. x \in S' \implies h' x = h x$ 
  by (blast intro: AR_imp_absolute_extensor [OF ⟨AR S⟩ h clo])
  have [simp]: S' ⊆ U using clo closedin_limp by blast
  show ?thesis
proof (simp add: retraction_def retract_of_def, intro exI conjI)
  show continuous_on U (g ∘ h')
    by (meson continuous_on_compose continuous_on_subset h' hom homeomorphism_cont1)
  show (g ∘ h') ' U ⊆ S'
    using h' by clarsimp (metis hom subsetD homeomorphism_def imageI)
  show  $\forall x \in S'. (g \circ h') x = x$ 
    by clarsimp (metis h'h hom homeomorphism_def)
qed
qed

lemma AR_imp_absolute_retract_UNIV:
  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  assumes AR S S homeomorphic S' closed S'
  shows S' retract_of UNIV
  using AR_imp_absolute_retract assms by fastforce

lemma absolute_extensor_imp_AR:
  fixes S :: 'a::euclidean_space set

```



```

    assumes  $\bigwedge f :: 'a * \text{real} \Rightarrow 'a.$ 
       $\bigwedge U T. \llbracket \text{continuous\_on } T f; f ' T \subseteq S;$ 
       $\text{closedin } (\text{top\_of\_set } U) T \rrbracket$ 
       $\Rightarrow \exists g. \text{continuous\_on } U g \wedge g ' U \subseteq S \wedge (\forall x \in T. g x = f x)$ 
    shows  $AR S$ 
  proof (clarsimp simp:  $AR\_def$ )
    fix  $U$  and  $T :: ('a * \text{real}) \text{ set}$ 
    assume  $S \text{ homeomorphic } T$  and  $clo: \text{closedin } (\text{top\_of\_set } U) T$ 
    then obtain  $g h$  where  $hom: \text{homeomorphism } S T g h$ 
      by (force simp:  $\text{homeomorphic\_def}$ )
    obtain  $h: \text{continuous\_on } T h \ h ' T \subseteq S$ 
      using  $hom \text{ homeomorphism\_def}$  by blast
    obtain  $h'$  where  $h': \text{continuous\_on } U h' h' ' U \subseteq S$ 
      and  $h'h: \forall x \in T. h' x = h x$ 
      using  $assms [OF h clo]$  by blast
    have  $[simp]: T \subseteq U$ 
      using  $clo \text{ closedin\_imp\_subset}$  by auto
    show  $T \text{ retract\_of } U$ 
  proof (simp add:  $\text{retraction\_def retract\_of\_def}$ , intro  $exI conjI$ )
    show  $\text{continuous\_on } U (g \circ h')$ 
      by (meson  $\text{continuous\_on\_compose continuous\_on\_subset } h' hom \text{ homeomorphism\_cont1}$ )
    show  $(g \circ h') ' U \subseteq T$ 
      using  $h'$  by clarsimp (metis  $hom \text{ subsetD homeomorphism\_def imageI}$ )
    show  $\forall x \in T. (g \circ h') x = x$ 
      by clarsimp (metis  $h'h hom \text{ homeomorphism\_def}$ )
  qed
qed

lemma  $AR\_eq\_absolute\_extensor$ :
  fixes  $S :: 'a::\text{euclidean\_space} \text{ set}$ 
  shows  $AR S \longleftrightarrow$ 
     $(\forall f :: 'a * \text{real} \Rightarrow 'a.$ 
       $\forall U T. \text{continuous\_on } T f \longrightarrow f ' T \subseteq S \longrightarrow$ 
       $\text{closedin } (\text{top\_of\_set } U) T \longrightarrow$ 
       $(\exists g. \text{continuous\_on } U g \wedge g ' U \subseteq S \wedge (\forall x \in T. g x = f x)))$ 
  by (metis ( $\text{mono\_tags}$ ,  $\text{opaque\_lifting}$ )  $AR\_imp\_absolute\_extensor \text{ absolute\_extensor\_imp\_AR}$ )

lemma  $AR\_imp\_retract$ :
  fixes  $S :: 'a::\text{euclidean\_space} \text{ set}$ 
  assumes  $AR S \wedge \text{closedin } (\text{top\_of\_set } U) S$ 
  shows  $S \text{ retract\_of } U$ 
  using  $AR\_imp\_absolute\_retract \text{ assms homeomorphic\_refl}$  by blast

lemma  $AR\_homeomorphic\_AR$ :
  fixes  $S :: 'a::\text{euclidean\_space} \text{ set}$  and  $T :: 'b::\text{euclidean\_space} \text{ set}$ 
  assumes  $AR T S \text{ homeomorphic } T$ 
  shows  $AR S$ 
  unfolding  $AR\_def$ 

```

by (metis assms AR_imp_absolute_retract homeomorphic_trans [of _ S] homeomorphic_sym)

lemma homeomorphic_AR_iff_AR:

fixes $S :: 'a::\text{euclidean\_space}$ set and $T :: 'b::\text{euclidean\_space}$ set

shows $S \text{ homeomorphic } T \implies AR\ S \longleftrightarrow AR\ T$

by (metis AR_homeomorphic_AR homeomorphic_sym)

lemma ANR_imp_absolute_neighbourhood_extensor:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$

assumes $ANR\ S$ and $contf: \text{continuous\_on } T\ f$ and $f\ ' T \subseteq S$

and $cloUT: \text{closedin } (top\_of\_set\ U)\ T$

obtains $V\ g$ where $T \subseteq V$ openin $(top\_of\_set\ U)\ V$

$\text{continuous\_on } V\ g$

$g\ ' V \subseteq S \wedge x. x \in T \implies g\ x = f\ x$

proof –

have $\text{aff\_dim } S < \text{int } (DIM('b \times \text{real}))$

using $\text{aff\_dim\_le\_DIM } [of\ S]$ by simp

then obtain C and $S' :: ('b * \text{real})$ set

where $C: \text{convex } C\ C \neq \{\}$

and $cloCS: \text{closedin } (top\_of\_set\ C)\ S'$

and $hom: S \text{ homeomorphic } S'$

by (metis that homeomorphic_closedin_convex)

then obtain D where $opD: \text{openin } (top\_of\_set\ C)\ D$ and S' retract_of D

using $\langle ANR\ S \rangle$ by (auto simp: ANR_def)

then obtain r where $S' \subseteq D$ and $contr: \text{continuous\_on } D\ r$

and $r\ ' D \subseteq S'$ and $rid: \wedge x. x \in S' \implies r\ x = x$

by (auto simp: retraction_def retract_of_def)

obtain $g\ h$ where $homgh: \text{homeomorphism } S\ S'\ g\ h$

using hom by (force simp: homeomorphic_def)

have $\text{continuous\_on } (f\ ' T)\ g$

by (meson $\langle f\ ' T \subseteq S \rangle \text{continuous\_on\_subset homeomorphism\_def homgh}$)

then have $contgf: \text{continuous\_on } T\ (g \circ f)$

by (intro continuous_on_compose contf)

have $gfTC: (g \circ f)\ ' T \subseteq C$

proof –

have $g\ ' S = S'$

by (metis (no_types) homeomorphism_def homgh)

then show ?thesis

by (metis (no_types) assms(3) cloCS closedin_def image_comp image_mono order.trans topspace_euclidean_subtopology)

qed

obtain f' where $contf': \text{continuous\_on } U\ f'$

and $f'\ ' U \subseteq C$

and $eq: \wedge x. x \in T \implies f'\ x = (g \circ f)\ x$

by (metis Dugundji [OF $C\ cloUT\ contgf\ gfTC$])

show ?thesis

proof (rule_tac $V = U \cap f'\ ' D$ and $g = h \circ r \circ f'$ in that)

```

show  $T \subseteq U \cap f' - ' D$ 
  using cloUT closedin_imp_subset  $\langle S' \subseteq D \rangle \langle f' - ' T \subseteq S \rangle$  eq homeomor-
phism_image1 homgh
  by fastforce
show ope: openin (top_of_set U) (U  $\cap$  f' - ' D)
  using  $\langle f' - ' U \subseteq C \rangle$  by (auto simp: opD contf' continuous_openin_preimage)
have conth: continuous_on (r - ' f' - ' (U  $\cap$  f' - ' D)) h
proof (rule continuous_on_subset [of S'])
  show continuous_on S' h
    using homeomorphism_def homgh by blast
qed (use  $\langle r - ' D \subseteq S' \rangle$  in blast)
show continuous_on (U  $\cap$  f' - ' D) (h  $\circ$  r  $\circ$  f')
  by (blast intro: continuous_on_compose conth continuous_on_subset [OF
contr] continuous_on_subset [OF contf'])
show  $(h \circ r \circ f') - ' (U \cap f' - ' D) \subseteq S$ 
  using  $\langle \text{homeomorphism } S \ S' \ g \ h \rangle \langle f' - ' U \subseteq C \rangle \langle r - ' D \subseteq S' \rangle$ 
  by (auto simp: homeomorphism_def)
show  $\bigwedge x. x \in T \implies (h \circ r \circ f') x = f x$ 
  using  $\langle \text{homeomorphism } S \ S' \ g \ h \rangle \langle f' - ' T \subseteq S \rangle$  eq
  by (auto simp: rid homeomorphism_def)
qed
qed

```

corollary *ANR_imp_absolute_neighbourhood_retract:*

```

fixes  $S :: 'a::\text{euclidean\_space set}$  and  $S' :: 'b::\text{euclidean\_space set}$ 
assumes ANR S S homeomorphic S'
and clo: closedin (top_of_set U) S'
obtains  $V$  where openin (top_of_set U) V S' retract_of V
proof -
obtain  $g \ h$  where hom: homeomorphism S S' g h
  using assms by (force simp: homeomorphic_def)
obtain  $h: \text{continuous\_on } S' \ h$   $h - ' S' \subseteq S$ 
  using hom homeomorphism_def by blast
from ANR_imp_absolute_neighbourhood_extensor [OF  $\langle \text{ANR } S \rangle h \text{ clo}$ ]
obtain  $V \ h'$  where  $S' \subseteq V$  and opUV: openin (top_of_set U) V
  and  $h': \text{continuous\_on } V \ h' \ h' - ' V \subseteq S$ 
  and  $h'h: \bigwedge x. x \in S' \implies h' x = h x$ 
by (blast intro: ANR_imp_absolute_neighbourhood_extensor [OF  $\langle \text{ANR } S \rangle h$ 
clo])
have  $S' \text{ retract\_of } V$ 
proof (simp add: retraction_def retract_of_def, intro exI conjI  $\langle S' \subseteq V \rangle$ )
  show continuous_on V (g  $\circ$  h')
    by (meson continuous_on_compose continuous_on_subset h'(1) h'(2) hom
homeomorphism_cont1)
  show  $(g \circ h') - ' V \subseteq S'$ 
    using  $h'$  by clarsimp (metis hom subsetD homeomorphism_def imageI)
  show  $\forall x \in S'. (g \circ h') x = x$ 
    by clarsimp (metis h'h hom homeomorphism_def)

```

3000

```

qed
then show ?thesis
  by (rule that [OF opUV])
qed

```

```

corollary ANR_imp_absolute_neighbourhood_retract_UNIV:
  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  assumes ANR S and hom: S homeomorphic S' and clo: closed S'
  obtains V where open V S' retract_of V
  using ANR_imp_absolute_neighbourhood_retract [OF ⟨ANR S⟩ hom]
  by (metis clo closed_closedin open_openin subtopology_UNIV)

```

```

corollary neighbourhood_extension_into_ANR:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes contf: continuous_on S f and fim: f ' S ⊆ T and ANR T closed S
  obtains V g where S ⊆ V open V continuous_on V g
    g ' V ⊆ T ∧ x. x ∈ S ⇒ g x = f x
  using ANR_imp_absolute_neighbourhood_extensor [OF ⟨ANR T⟩ contf fim]
  by (metis ⟨closed S⟩ closed_closedin open_openin subtopology_UNIV)

```

```

lemma absolute_neighbourhood_extensor_imp_ANR:
  fixes S :: 'a::euclidean_space set
  assumes ∧f :: 'a * real ⇒ 'a.
    ∧U T. [continuous_on T f; f ' T ⊆ S;
      closedin (top_of_set U) T]
    ⇒ ∃ V g. T ⊆ V ∧ openin (top_of_set U) V ∧
      continuous_on V g ∧ g ' V ⊆ S ∧ (∀ x ∈ T. g x = f x)

```

```

  shows ANR S
proof (clarsimp simp: ANR_def)
  fix U and T :: ('a * real) set
  assume S homeomorphic T and clo: closedin (top_of_set U) T
  then obtain g h where hom: homeomorphism S T g h
    by (force simp: homeomorphic_def)
  obtain h: continuous_on T h h ' T ⊆ S
    using hom homeomorphism_def by blast
  obtain V h' where T ⊆ V and opV: openin (top_of_set U) V
    and h': continuous_on V h' h' ' V ⊆ S
    and h'h: ∀ x ∈ T. h' x = h x
    using assms [OF h clo] by blast
  have [simp]: T ⊆ U
    using clo closedin_imp_subset by auto
  have T retract_of V
proof (simp add: retraction_def retract_of_def, intro exI conjI ⟨T ⊆ V⟩)
    show continuous_on V (g ∘ h')
      by (meson continuous_on_compose continuous_on_subset h' hom homeo-
        morphism_cont1)
    show (g ∘ h') ' V ⊆ T
      using h' by clarsimp (metis hom subsetD homeomorphism_def imageI)
    show ∀ x ∈ T. (g ∘ h') x = x

```

```

    by clarsimp (metis h'h hom homeomorphism_def)
  qed
  then show  $\exists V. \text{openin } (\text{top\_of\_set } U) \ V \wedge T \text{ retract\_of } V$ 
    using opV by blast
  qed

lemma ANR_eq_absolute_neighbourhood_extensor:
  fixes  $S :: 'a::\text{euclidean\_space set}$ 
  shows  $\text{ANR } S \longleftrightarrow$ 
    ( $\forall f :: 'a * \text{real} \Rightarrow 'a.$ 
       $\forall U \ T. \text{continuous\_on } T \ f \longrightarrow f' \ T \subseteq S \longrightarrow$ 
         $\text{closedin } (\text{top\_of\_set } U) \ T \longrightarrow$ 
        ( $\exists V \ g. T \subseteq V \wedge \text{openin } (\text{top\_of\_set } U) \ V \wedge$ 
           $\text{continuous\_on } V \ g \wedge g' \ V \subseteq S \wedge (\forall x \in T. g \ x = f \ x)))$ ) (is _
= ?rhs)
proof
  assume  $\text{ANR } S$  then show ?rhs
    by (metis ANR_imp_absolute_neighbourhood_extensor)
  qed (simp add: absolute_neighbourhood_extensor_imp_ANR)

lemma ANR_imp_neighbourhood_retract:
  fixes  $S :: 'a::\text{euclidean\_space set}$ 
  assumes  $\text{ANR } S$   $\text{closedin } (\text{top\_of\_set } U) \ S$ 
  obtains  $V$  where  $\text{openin } (\text{top\_of\_set } U) \ V \ S \text{ retract\_of } V$ 
using ANR_imp_absolute_neighbourhood_retract assms homeomorphic_refl by
blast

lemma ANR_imp_absolute_closed_neighbourhood_retract:
  fixes  $S :: 'a::\text{euclidean\_space set}$  and  $S' :: 'b::\text{euclidean\_space set}$ 
  assumes  $\text{ANR } S \ S \text{ homeomorphic } S'$  and  $US': \text{closedin } (\text{top\_of\_set } U) \ S'$ 
  obtains  $V \ W$ 
    where  $\text{openin } (\text{top\_of\_set } U) \ V$ 
       $\text{closedin } (\text{top\_of\_set } U) \ W$ 
       $S' \subseteq V \ V \subseteq W \ S' \text{ retract\_of } W$ 
proof -
  obtain  $Z$  where  $\text{openin } (\text{top\_of\_set } U) \ Z$  and  $S'Z: S' \text{ retract\_of } Z$ 
    by (blast intro: assms ANR_imp_absolute_neighbourhood_retract)
  then have  $UUZ: \text{closedin } (\text{top\_of\_set } U) \ (U - Z)$ 
    by auto
  have  $S' \cap (U - Z) = \{\}$ 
    using  $\langle S' \text{ retract\_of } Z \rangle \text{closedin\_retract closedin\_subtopology}$  by fastforce
  then obtain  $V \ W$ 
    where  $\text{openin } (\text{top\_of\_set } U) \ V$ 
      and  $\text{openin } (\text{top\_of\_set } U) \ W$ 
      and  $S' \subseteq V \ U - Z \subseteq W \ V \cap W = \{\}$ 
    using separation_normal_local [OF  $US' \ UUZ$ ] by auto
  moreover have  $S' \text{ retract\_of } U - W$ 
  proof (rule retract_of_subset [OF  $S'Z$ ])
    show  $S' \subseteq U - W$ 

```

```

    using  $US' \langle S' \subseteq V \rangle \langle V \cap W = \{\} \rangle$  closedin_subset by fastforce
  show  $U - W \subseteq Z$ 
    using Diff_subset_conv  $\langle U - Z \subseteq W \rangle$  by blast
qed
ultimately show ?thesis
  by (metis Diff_subset_conv Diff_triv Int_Diff_Un Int_absorb1 openin_closedin_eq
    that topspace_euclidean_subtopology)
qed

```

```

lemma ANR_imp_closed_neighbourhood_retract:
  fixes  $S :: 'a::euclidean\_space\ set$ 
  assumes ANR  $S$  closedin (top_of_set  $U$ )  $S$ 
  obtains  $V\ W$  where openin (top_of_set  $U$ )  $V$ 
    closedin (top_of_set  $U$ )  $W$ 
     $S \subseteq V\ V \subseteq W\ S$  retract_of  $W$ 
by (meson ANR_imp_absolute_closed_neighbourhood_retract assms homeomorphic_refl)

```

```

lemma ANR_homeomorphic_ANR:
  fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
  assumes ANR  $T$   $S$  homeomorphic  $T$ 
  shows ANR  $S$ 
unfolding ANR_def
by (metis assms ANR_imp_absolute_neighbourhood_retract homeomorphic_trans
  [of _  $S$ ] homeomorphic_sym)

```

```

lemma homeomorphic_ANR_iff_ANR:
  fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
  shows  $S$  homeomorphic  $T \implies$  ANR  $S \longleftrightarrow$  ANR  $T$ 
by (metis ANR_homeomorphic_ANR homeomorphic_sym)

```

6.41.1 Analogous properties of ENRs

```

lemma ENR_imp_absolute_neighbourhood_retract:
  fixes  $S :: 'a::euclidean\_space\ set$  and  $S' :: 'b::euclidean\_space\ set$ 
  assumes ENR  $S$  and hom:  $S$  homeomorphic  $S'$ 
    and  $S' \subseteq U$ 
  obtains  $V$  where openin (top_of_set  $U$ )  $V\ S'$  retract_of  $V$ 
proof -
  obtain  $X$  where open  $X\ S$  retract_of  $X$ 
    using  $\langle ENR\ S \rangle$  by (auto simp: ENR_def)
  then obtain  $r$  where retraction  $X\ S\ r$ 
    by (auto simp: retract_of_def)
  have locally_compact  $S'$ 
    using retract_of_locally_compact open_imp_locally_compact
      homeomorphic_local_compactness  $\langle S$  retract_of  $X \rangle \langle$  open  $X \rangle$  hom by blast
  then obtain  $W$  where openin (top_of_set  $U$ )  $W$ 
    and  $WS'$ : closedin (top_of_set  $W$ )  $S'$ 
  apply (rule locally_compact_closedin_open)

```

```

    by (meson Int_lower2 assms(3) closedin_imp_subset closedin_subset_trans
le_inf_iff openin_open)
  obtain f g where hom: homeomorphism S S' f g
    using assms by (force simp: homeomorphic_def)
  have contg: continuous_on S' g
    using hom homeomorphism_def by blast
  moreover have g ' S'  $\subseteq$  S by (metis hom equalityE homeomorphism_def)
  ultimately obtain h where conth: continuous_on W h and hg:  $\bigwedge x. x \in S' \Rightarrow h x = g x$ 
    using Tietze_unbounded [of S' g W] WS' by blast
  have W  $\subseteq$  U using UW openin_open by auto
  have S'  $\subseteq$  W using WS' closedin_closed by auto
  have him:  $\bigwedge x. x \in S' \Rightarrow h x \in X$ 
    by (metis (no_types)  $\langle S \text{ retract\_of } X \rangle$  hg hom homeomorphism_def image_insert
insert_absorb insert_iff retract_of_imp_subset subset_eq)
  have S' retract_of (W  $\cap$  h - ' X)
  proof (simp add: retraction_def retract_of_def, intro exI conjI)
    show S'  $\subseteq$  W S'  $\subseteq$  h - ' X
      using him WS' closedin_imp_subset by blast+
    show continuous_on (W  $\cap$  h - ' X) (f  $\circ$  r  $\circ$  h)
      proof (intro continuous_on_compose)
        show continuous_on (W  $\cap$  h - ' X) h
          by (meson conth continuous_on_subset inf_le1)
        show continuous_on (h ' (W  $\cap$  h - ' X)) r
          proof -
            have h ' (W  $\cap$  h - ' X)  $\subseteq$  X
              by blast
            then show continuous_on (h ' (W  $\cap$  h - ' X)) r
              by (meson  $\langle \text{retraction } X S r \rangle$  continuous_on_subset retraction)
          qed
        show continuous_on (r ' h ' (W  $\cap$  h - ' X)) f
          proof (rule continuous_on_subset [of S])
            show continuous_on S f
              using hom homeomorphism_def by blast
            show r ' h ' (W  $\cap$  h - ' X)  $\subseteq$  S
              by (metis  $\langle \text{retraction } X S r \rangle$  image_mono image_subset_iff_subset_vimage
inf_le2 retraction)
          qed
        qed
      show (f  $\circ$  r  $\circ$  h) ' (W  $\cap$  h - ' X)  $\subseteq$  S'
        using  $\langle \text{retraction } X S r \rangle$  hom
        by (auto simp: retraction_def homeomorphism_def)
      show  $\forall x \in S'. (f \circ r \circ h) x = x$ 
        using  $\langle \text{retraction } X S r \rangle$  hom by (auto simp: retraction_def homeomor-
phism_def hg)
    qed
  then show ?thesis
    using UW  $\langle \text{open } X \rangle$  conth continuous_openin_preimage_eq openin_trans that
    by blast

```

qed

corollary *ENR_imp_absolute_neighbourhood_retract_UNIV*:
fixes $S :: 'a::\text{euclidean\_space set}$ **and** $S' :: 'b::\text{euclidean\_space set}$
assumes $\text{ENR } S \ S \text{ homeomorphic } S'$
obtains T' **where** $\text{open } T' \ S' \text{ retract\_of } T'$
by (*metis ENR_imp_absolute_neighbourhood_retract_UNIV_I assms(1) assms(2)*
open_openin_subsetI subtopology_UNIV)

lemma *ENR_homeomorphic_ENR*:
fixes $S :: 'a::\text{euclidean\_space set}$ **and** $T :: 'b::\text{euclidean\_space set}$
assumes $\text{ENR } T \ S \text{ homeomorphic } T$
shows $\text{ENR } S$
unfolding *ENR_def*
by (*meson ENR_imp_absolute_neighbourhood_retract_UNIV assms homeomorphic_sym*)

lemma *homeomorphic_ENR_iff_ENR*:
fixes $S :: 'a::\text{euclidean\_space set}$ **and** $T :: 'b::\text{euclidean\_space set}$
assumes $S \text{ homeomorphic } T$
shows $\text{ENR } S \longleftrightarrow \text{ENR } T$
by (*meson ENR_homeomorphic_ENR assms homeomorphic_sym*)

lemma *ENR_translation*:
fixes $S :: 'a::\text{euclidean\_space set}$
shows $\text{ENR}(\text{image } (\lambda x. a + x) \ S) \longleftrightarrow \text{ENR } S$
by (*meson homeomorphic_sym homeomorphic_translation homeomorphic_ENR_iff_ENR*)

lemma *ENR_linear_image_eq*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes $\text{linear } f \text{ inj } f$
shows $\text{ENR } (\text{image } f \ S) \longleftrightarrow \text{ENR } S$
by (*meson assms homeomorphic_ENR_iff_ENR linear_homeomorphic_image*)

Some relations among the concepts. We also relate AR to being a retract of UNIV, which is often a more convenient proxy in the closed case.

lemma *AR_imp_ANR*: $\text{AR } S \Longrightarrow \text{ANR } S$
using *ANR_def AR_def* **by** *fastforce*

lemma *ENR_imp_ANR*:
fixes $S :: 'a::\text{euclidean\_space set}$
shows $\text{ENR } S \Longrightarrow \text{ANR } S$
by (*meson ANR_def ENR_imp_absolute_neighbourhood_retract closedin_imp_subset*)

lemma *ENR_ANR*:
fixes $S :: 'a::\text{euclidean\_space set}$
shows $\text{ENR } S \longleftrightarrow \text{ANR } S \wedge \text{locally compact } S$
proof
assume $\text{ENR } S$


```

then have locally compact S
  using ENR_def open_imp_locally_compact retract_of_locally_compact by
auto
then show ANR S ∧ locally compact S
  using ENR_imp_ANR  $\langle \text{ENR } S \rangle$  by blast
next
  assume ANR S ∧ locally compact S
  then have ANR S locally compact S by auto
  then obtain T :: ('a * real) set where closed T S homeomorphic T
    using locally_compact_homeomorphic_closed
    by (metis DIM_prod DIM_real Suc_eq_plus1 lessI)
  then show ENR S
    using  $\langle \text{ANR } S \rangle$ 
    by (meson ANR_imp_absolute_neighbourhood_retract_UNIV ENR_def ENR_homeomorphic_ENR)
qed

```

```

lemma AR_ANR:
  fixes S :: 'a::euclidean_space set
  shows AR S  $\longleftrightarrow$  ANR S ∧ contractible S ∧ S  $\neq$  {}
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  have aff_dim S < int DIM('a × real)
    using aff_dim_le_DIM [of S] by auto
  then obtain C and S' :: ('a * real) set
    where convex C C  $\neq$  {} closedin (top_of_set C) S' S homeomorphic S'
    using homeomorphic_closedin_convex by blast
  with  $\langle \text{AR } S \rangle$  have contractible S
    by (meson AR_def convex_imp_contractible homeomorphic_contractible_eq
retract_of_contractible)
  with  $\langle \text{AR } S \rangle$  show ?rhs
    using AR_imp_ANR AR_imp_retract by fastforce
next
  assume ?rhs
  then obtain a and h:: real × 'a  $\Rightarrow$  'a
    where conth: continuous_on ({0..1} × S) h
    and hS: h '({0..1} × S)  $\subseteq$  S
    and [simp]:  $\bigwedge x. h(0, x) = x$ 
    and [simp]:  $\bigwedge x. h(1, x) = a$ 
    and ANR S S  $\neq$  {}
    by (auto simp: contractible_def homotopic_with_def)
  then have a ∈ S
    by (metis all_not_in_conv atLeastAtMost_iff image_subset_iff mem_Sigma_iff
order_refl zero_le_one)
  have  $\exists g. \text{continuous\_on } W \ g \wedge g ' W \subseteq S \wedge (\forall x \in T. g \ x = f \ x)$ 
    if f: continuous_on T f f ' T  $\subseteq$  S
    and WT: closedin (top_of_set W) T
    for W T and f :: 'a × real  $\Rightarrow$  'a

```

```

proof –
  obtain  $U\ g$ 
    where  $T \subseteq U$  and  $WU$ :  $\text{openin } (\text{top\_of\_set } W) \ U$ 
      and  $\text{contg}$ :  $\text{continuous\_on } U\ g$ 
      and  $g \restriction U \subseteq S$  and  $gf$ :  $\bigwedge x. x \in T \implies g\ x = f\ x$ 
      using  $\text{iffD1 } [OF\ ANR\_eq\_absolute\_neighbourhood\_extensor\ \langle ANR\ S \rangle,$ 
 $\text{rule\_format},\ OF\ f\ WT]$ 
    by  $\text{auto}$ 
  have  $WWU$ :  $\text{closedin } (\text{top\_of\_set } W) \ (W - U)$ 
    using  $WU\ \text{closedin\_diff}$  by  $\text{fastforce}$ 
  moreover have  $(W - U) \cap T = \{\}$ 
    using  $\langle T \subseteq U \rangle$  by  $\text{auto}$ 
  ultimately obtain  $V\ V'$ 
    where  $WV'$ :  $\text{openin } (\text{top\_of\_set } W) \ V'$ 
      and  $WV$ :  $\text{openin } (\text{top\_of\_set } W) \ V$ 
      and  $W - U \subseteq V' \ T \subseteq V\ V' \cap V = \{\}$ 
      using  $\text{separation\_normal\_local } [of\ W\ W - U\ T]\ WT$  by  $\text{blast}$ 
  then have  $WVT$ :  $T \cap (W - V) = \{\}$ 
    by  $\text{auto}$ 
  have  $WWV$ :  $\text{closedin } (\text{top\_of\_set } W) \ (W - V)$ 
    using  $WV\ \text{closedin\_diff}$  by  $\text{fastforce}$ 
  obtain  $j :: 'a \times \text{real} \Rightarrow \text{real}$ 
    where  $\text{contj}$ :  $\text{continuous\_on } W\ j$ 
      and  $j$ :  $\bigwedge x. x \in W \implies j\ x \in \{0..1\}$ 
      and  $j0$ :  $\bigwedge x. x \in W - V \implies j\ x = 1$ 
      and  $j1$ :  $\bigwedge x. x \in T \implies j\ x = 0$ 
      by  $(\text{rule } \text{Urysohn\_local } [OF\ WT\ WWV\ WVT,\ of\ 0\ 1::\text{real}])\ (\text{auto } \text{simp}:$ 
 $\text{in\_segment})$ 
  have  $\text{Weq}$ :  $W = (W - V) \cup (W - V')$ 
    using  $\langle V' \cap V = \{\} \rangle$  by  $\text{force}$ 
  show  $?thesis$ 
    proof  $(\text{intro } \text{conjI } \text{exI})$ 
      have  $*$ :  $\text{continuous\_on } (W - V')\ (\lambda x. h\ (j\ x, g\ x))$ 
        proof  $(\text{rule } \text{continuous\_on\_compose2 } [OF\ \text{conth } \text{continuous\_on\_Pair}])$ 
          show  $\text{continuous\_on } (W - V')\ j$ 
            by  $(\text{rule } \text{continuous\_on\_subset } [OF\ \text{contj } \text{Diff\_subset}])$ 
          show  $\text{continuous\_on } (W - V')\ g$ 
            by  $(\text{metis } \text{Diff\_subset\_conv } \langle W - U \subseteq V' \rangle\ \text{contg } \text{continuous\_on\_subset } Un\_commute)$ 
          show  $(\lambda x. (j\ x, g\ x)) \restriction (W - V') \subseteq \{0..1\} \times S$ 
            using  $j\ \langle g \restriction U \subseteq S \rangle\ \langle W - U \subseteq V' \rangle$  by  $\text{fastforce}$ 
        qed
      show  $\text{continuous\_on } W\ (\lambda x. \text{if } x \in W - V \text{ then } a \text{ else } h\ (j\ x, g\ x))$ 
        proof  $(\text{subst } \text{Weq},\ \text{rule } \text{continuous\_on\_cases\_local})$ 
          show  $\text{continuous\_on } (W - V')\ (\lambda x. h\ (j\ x, g\ x))$ 
            using  $*$  by  $\text{blast}$ 
          qed  $(\text{use } WWV\ WV'\ \text{Weq } j0\ j1\ \text{in } \text{auto})$ 
    next
      have  $h\ (j\ (x, y), g\ (x, y)) \in S$  if  $(x, y) \in W\ (x, y) \in V$  for  $x\ y$ 

```

```

proof –
  have  $j(x, y) \in \{0..1\}$ 
    using  $j$  that by blast
  moreover have  $g(x, y) \in S$ 
    using  $\langle V' \cap V = \{ \} \rangle \langle W - U \subseteq V' \rangle \langle g \text{ ‘ } U \subseteq S \rangle$  that by fastforce
  ultimately show ?thesis
    using  $hS$  by blast
qed
with  $\langle a \in S \rangle \langle g \text{ ‘ } U \subseteq S \rangle$ 
show  $(\lambda x. \text{if } x \in W - V \text{ then } a \text{ else } h(j\ x, g\ x)) \text{ ‘ } W \subseteq S$ 
  by auto
next
  show  $\forall x \in T. (\text{if } x \in W - V \text{ then } a \text{ else } h(j\ x, g\ x)) = f\ x$ 
    using  $\langle T \subseteq V \rangle$  by (auto simp: j0 j1 gf)
qed
qed
then show ?lhs
  by (simp add: AR_eq_absolute_extensor)
qed

lemma ANR_retract_of_ANR:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes  $ANR\ T$  and  $ST: S\ \text{retract\_of}\ T$ 
  shows  $ANR\ S$ 
proof (clarsimp simp add: ANR_eq_absolute_neighbourhood_extensor)
  fix  $f::'a \times \text{real} \Rightarrow 'a$  and  $U\ W$ 
  assume  $W: \text{continuous\_on}\ W\ f\ \text{‘}\ W \subseteq S$  closedin (top_of_set  $U$ )  $W$ 
  then obtain  $r$  where  $S \subseteq T$  and  $r: \text{continuous\_on}\ T\ r\ \text{‘}\ T \subseteq S\ \forall x \in S. r\ x = x$ 
    continuous\_on  $W\ f\ \text{‘}\ W \subseteq S$ 
    closedin (top_of_set  $U$ )  $W$ 
  by (meson  $ST$  retract_of_def retraction_def)
  then have  $f \text{ ‘ } W \subseteq T$ 
  by blast
  with  $W$  obtain  $V\ g$  where  $V: W \subseteq V$  openin (top_of_set  $U$ )  $V$  continuous\_on
     $V\ g\ \text{‘}\ V \subseteq T\ \forall x \in W. g\ x = f\ x$ 
  by (metis ANR_imp_absolute_neighbourhood_extensor  $\langle ANR\ T \rangle$ )
  with  $r$  have continuous\_on  $V\ (r \circ g) \wedge (r \circ g) \text{ ‘ } V \subseteq S \wedge (\forall x \in W. (r \circ g)\ x = f\ x)$ 
  by (metis (no_types, lifting) comp_apply continuous\_on\_compose continuous\_on\_subset image_subset_iff)
  then show  $\exists V. W \subseteq V \wedge \text{openin}\ (\text{top\_of\_set}\ U)\ V \wedge (\exists g. \text{continuous\_on}\ V\ g \wedge g \text{ ‘ } V \subseteq S \wedge (\forall x \in W. g\ x = f\ x))$ 
  by (meson  $V$ )
qed

lemma AR_retract_of_AR:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  shows  $\llbracket AR\ T; S\ \text{retract\_of}\ T \rrbracket \Longrightarrow AR\ S$ 

```

using *ANR_retract_of_ANR* *AR_ANR_retract_of_contractible* **by** *fastforce*

lemma *ENR_retract_of_ENR*:
 $\llbracket \text{ENR } T; S \text{ retract\_of } T \rrbracket \implies \text{ENR } S$
by (*meson ENR_def retract_of_trans*)

lemma *retract_of_UNIV*:
fixes $S :: 'a::\text{euclidean\_space set}$
shows $S \text{ retract\_of } \text{UNIV} \longleftrightarrow \text{AR } S \wedge \text{closed } S$
by (*metis AR_ANR AR_imp_retract ENR_def ENR_imp_ANR closed_UNIV*
closed_closedin contractible_UNIV empty_not_UNIV open_UNIV retract_of_closed
retract_of_contractible retract_of_empty(1) subtopology_UNIV)

lemma *compact_AR*:
fixes $S :: 'a::\text{euclidean\_space set}$
shows $\text{compact } S \wedge \text{AR } S \longleftrightarrow \text{compact } S \wedge S \text{ retract\_of } \text{UNIV}$
using *compact_imp_closed retract_of_UNIV* **by** *blast*

More properties of ARs, ANRs and ENRs

lemma *not_AR_empty* [*simp*]: $\neg \text{AR}(\{\})$
by (*auto simp: AR_def*)

lemma *ENR_empty* [*simp*]: $\text{ENR } \{\}$
by (*simp add: ENR_def*)

lemma *ANR_empty* [*simp*]: $\text{ANR } (\{\} :: 'a::\text{euclidean\_space set})$
by (*simp add: ENR_imp_ANR*)

lemma *convex_imp_AR*:
fixes $S :: 'a::\text{euclidean\_space set}$
shows $\llbracket \text{convex } S; S \neq \{\} \rrbracket \implies \text{AR } S$
by (*metis (mono_tags, lifting) Dugundji absolute_extensor_imp_AR*)

lemma *convex_imp_ANR*:
fixes $S :: 'a::\text{euclidean\_space set}$
shows $\text{convex } S \implies \text{ANR } S$
using *ANR_empty AR_imp_ANR convex_imp_AR* **by** *blast*

lemma *ENR_convex_closed*:
fixes $S :: 'a::\text{euclidean\_space set}$
shows $\llbracket \text{closed } S; \text{convex } S \rrbracket \implies \text{ENR } S$
using *ENR_def ENR_empty convex_imp_AR retract_of_UNIV* **by** *blast*

lemma *AR_UNIV* [*simp*]: $\text{AR } (\text{UNIV} :: 'a::\text{euclidean\_space set})$
using *retract_of_UNIV* **by** *auto*

lemma *ANR_UNIV* [*simp*]: $\text{ANR } (\text{UNIV} :: 'a::\text{euclidean\_space set})$
by (*simp add: AR_imp_ANR*)

```
lemma ENR_UNIV [simp]: ENR UNIV
  using ENR_def by blast
```

```
lemma AR_singleton:
  fixes a :: 'a::euclidean_space
  shows AR {a}
  using retract_of_UNIV by blast
```

```
lemma ANR_singleton:
  fixes a :: 'a::euclidean_space
  shows ANR {a}
  by (simp add: AR_imp_ANR AR_singleton)
```

```
lemma ENR_singleton: ENR {a}
  using ENR_def by blast
```

ARs closed under union

```
lemma AR_closed_Un_local_aux:
  fixes U :: 'a::euclidean_space set
  assumes closedin (top_of_set U) S
           closedin (top_of_set U) T
           AR S AR T AR(S ∩ T)
  shows (S ∪ T) retract_of U
proof -
  have S ∩ T ≠ {}
    using assms AR_def by fastforce
  have S ⊆ U T ⊆ U
    using assms by (auto simp: closedin_imp_subset)
  define S' where S' ≡ {x ∈ U. setdist {x} S ≤ setdist {x} T}
  define T' where T' ≡ {x ∈ U. setdist {x} T ≤ setdist {x} S}
  define W where W ≡ {x ∈ U. setdist {x} S = setdist {x} T}
  have US': closedin (top_of_set U) S'
    using continuous_closedin_preimage [of U λx. setdist {x} S - setdist {x} T]
    {..0}]
  by (simp add: S'_def vimage_def Collect_conj_eq continuous_on_diff continuous_on_setdist)
  have UT': closedin (top_of_set U) T'
    using continuous_closedin_preimage [of U λx. setdist {x} T - setdist {x} S]
    {..0}]
  by (simp add: T'_def vimage_def Collect_conj_eq continuous_on_diff continuous_on_setdist)
  have S ⊆ S'
    using S'_def ⟨S ⊆ U⟩ setdist_sing_in_set by fastforce
  have T ⊆ T'
    using T'_def ⟨T ⊆ U⟩ setdist_sing_in_set by fastforce
  have S ∩ T ⊆ W W ⊆ U
    using ⟨S ⊆ U⟩ by (auto simp: W_def setdist_sing_in_set)
  have (S ∩ T) retract_of W
  proof (rule AR_imp_absolute_retract [OF ⟨AR(S ∩ T)⟩])
```



```

  by (force simp: S'_def T'_def)
  have cont: continuous_on U ( $\lambda x. \text{if } x \in S' \text{ then } g \ x \text{ else } h \ x$ )
    unfolding U
    apply (rule continuous_on_cases_local)
    using US' UT'  $\langle S' \cap T' = W \rangle \langle U = S' \cup T' \rangle$ 
      contg conth continuous_on_subset geqr heqr by auto
  have UST: ( $\lambda x. \text{if } x \in S' \text{ then } g \ x \text{ else } h \ x$ ) ' $U \subseteq S \cup T$ '
    using  $\langle g \text{ ' $U \subseteq S$ ' } \langle h \text{ ' $U \subseteq T$ ' } \rangle \text{ by auto}$ 
  show ?thesis
    apply (simp add: retract_of_def retraction_def  $\langle S \subseteq U \rangle \langle T \subseteq U \rangle$ )
    apply (rule_tac x= $\lambda x. \text{if } x \in S' \text{ then } g \ x \text{ else } h \ x$  in exI)
    using ST UST  $\langle S \subseteq S' \rangle \langle S' \cap T' = W \rangle \langle T \subseteq T' \rangle$  cont geqr heqr r_def by
auto
qed

lemma AR_closed_Un_local:
  fixes S :: 'a::euclidean_space set
  assumes STS: closedin (top_of_set (S  $\cup$  T)) S
    and STT: closedin (top_of_set (S  $\cup$  T)) T
    and AR S AR T AR(S  $\cap$  T)
  shows AR(S  $\cup$  T)
proof -
  have C retract_of U
    if hom: S  $\cup$  T homeomorphic C and UC: closedin (top_of_set U) C
    for U and C :: ('a * real) set
  proof -
    obtain f g where hom: homeomorphism (S  $\cup$  T) C f g
      using hom by (force simp: homeomorphic_def)
    have US: closedin (top_of_set U) (C  $\cap$  g - ' S)
      by (metis STS continuous_on_imp_closedin hom homeomorphism_def closedin_trans
[OF _ UC])
    have UT: closedin (top_of_set U) (C  $\cap$  g - ' T)
      by (metis STT continuous_on_closed hom homeomorphism_def closedin_trans
[OF _ UC])
    have homeomorphism (C  $\cap$  g - ' S) S g f
      using hom
      apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
      apply (rule_tac x=f x in image_eqI, auto)
      done
    then have ARS: AR (C  $\cap$  g - ' S)
      using  $\langle AR \ S \rangle$  homeomorphic_AR_iff_AR homeomorphic_def by blast
    have homeomorphism (C  $\cap$  g - ' T) T g f
      using hom
      apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
      apply (rule_tac x=f x in image_eqI, auto)
      done
    then have ART: AR (C  $\cap$  g - ' T)
      using  $\langle AR \ T \rangle$  homeomorphic_AR_iff_AR homeomorphic_def by blast

```

```

have homeomorphism  $(C \cap g - ' S \cap (C \cap g - ' T)) (S \cap T) g f$ 
  using hom
apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
apply (rule_tac x=f x in image_eqI, auto)
done
then have ARI:  $AR ((C \cap g - ' S) \cap (C \cap g - ' T))$ 
  using  $\langle AR (S \cap T) \rangle$  homeomorphic_AR_iff_AR homeomorphic_def by blast
have  $C = (C \cap g - ' S) \cup (C \cap g - ' T)$ 
  using hom by (auto simp: homeomorphism_def)
then show ?thesis
  by (metis AR_closed_Un_local_aux [OF US UT ARS ART ARI])
qed
then show ?thesis
  by (force simp: AR_def)
qed

```

```

corollary AR_closed_Un:
  fixes  $S :: 'a::euclidean\_space$  set
  shows  $\llbracket \text{closed } S; \text{closed } T; AR\ S; AR\ T; AR\ (S \cap T) \rrbracket \implies AR\ (S \cup T)$ 
by (metis AR_closed_Un_local_aux closed_closedin retract_of_UNIV subtopology_UNIV)

```

ANRs closed under union

```

lemma ANR_closed_Un_local_aux:
  fixes  $U :: 'a::euclidean\_space$  set
  assumes  $US: \text{closedin } (top\_of\_set\ U)\ S$ 
    and  $UT: \text{closedin } (top\_of\_set\ U)\ T$ 
    and  $ANR\ S\ ANR\ T\ ANR(S \cap T)$ 
  obtains  $V$  where  $\text{openin } (top\_of\_set\ U)\ V\ (S \cup T)\ \text{retract\_of } V$ 
proof (cases  $S = \{\} \vee T = \{\}$ )
case True with assms that show ?thesis
  by (metis ANR_imp_neighbourhood_retract Un_commute inf_bot_right sup_inf_absorb)
next
case False
then have [simp]:  $S \neq \{\} \ T \neq \{\}$  by auto
have  $S \subseteq U\ T \subseteq U$ 
  using assms by (auto simp: closedin_imp_subset)
define  $S'$  where  $S' \equiv \{x \in U. \text{setdist } \{x\}\ S \leq \text{setdist } \{x\}\ T\}$ 
define  $T'$  where  $T' \equiv \{x \in U. \text{setdist } \{x\}\ T \leq \text{setdist } \{x\}\ S\}$ 
define  $W$  where  $W \equiv \{x \in U. \text{setdist } \{x\}\ S = \text{setdist } \{x\}\ T\}$ 
have  $\text{clo } US': \text{closedin } (top\_of\_set\ U)\ S'$ 
  using continuous_closedin_preimage [of  $U\ \lambda x. \text{setdist } \{x\}\ S - \text{setdist } \{x\}\ T$ 
 $\{..0\}$ ]
  by (simp add:  $S'_\text{def}\ vimage\_def\ Collect\_conj\_eq\ continuous\_on\_diff\ continuous\_on\_setdist$ )
have  $\text{clo } UT': \text{closedin } (top\_of\_set\ U)\ T'$ 
  using continuous_closedin_preimage [of  $U\ \lambda x. \text{setdist } \{x\}\ T - \text{setdist } \{x\}\ S$ 
 $\{..0\}$ ]
  by (simp add:  $T'_\text{def}\ vimage\_def\ Collect\_conj\_eq\ continuous\_on\_diff\ con-$ 

```



```

tinuous_on_setdist)
  have  $S \subseteq S'$ 
    using  $S\_def \langle S \subseteq U \rangle$  setdist_sing_in_set by fastforce
  have  $T \subseteq T'$ 
    using  $T\_def \langle T \subseteq U \rangle$  setdist_sing_in_set by fastforce
  have  $S' \cup T' = U$ 
    by (auto simp:  $S\_def T\_def$ )
  have  $W \subseteq S'$ 
    by (simp add: Collect_mono  $S\_def W\_def$ )
  have  $W \subseteq T'$ 
    by (simp add: Collect_mono  $T\_def W\_def$ )
  have  $ST\_W: S \cap T \subseteq W$  and  $W \subseteq U$ 
    using  $\langle S \subseteq U \rangle$  by (force simp:  $W\_def$  setdist_sing_in_set)+
  have  $S' \cap T' = W$ 
    by (auto simp:  $S\_def T\_def W\_def$ )
  then have cloUW: closedin (top_of_set U) W
    using closedin_Int cloUS' cloUT' by blast
  obtain  $W' W0$  where openin (top_of_set W)  $W'$ 
    and cloWW0: closedin (top_of_set W)  $W0$ 
    and  $S \cap T \subseteq W' W' \subseteq W0$ 
    and ret:  $(S \cap T)$  retract_of  $W0$ 
    by (meson ANR_imp_closed_neighbourhood_retract  $ST\_W US UT \langle W \subseteq U \rangle$ 
 $\langle ANR(S \cap T) \rangle$  closedin_Int closedin_subset_trans)
  then obtain  $U0$  where opeUU0: openin (top_of_set U)  $U0$ 
    and  $U0: S \cap T \subseteq U0 U0 \cap W \subseteq W0$ 
    unfolding openin_open using  $\langle W \subseteq U \rangle$  by blast
  have  $W0 \subseteq U$ 
    using  $\langle W \subseteq U \rangle$  cloWW0 closedin_subset by fastforce
  obtain  $r0$ 
    where  $S \cap T \subseteq W0$  and contr0: continuous_on  $W0 r0$  and  $r0 \restriction W0 \subseteq S \cap T$ 
    and  $r0 [simp]: \bigwedge x. x \in S \cap T \implies r0 x = x$ 
    using ret by (force simp: retract_of_def retraction_def)
  have  $ST: x \in W \implies x \in S \longleftrightarrow x \in T$  for  $x$ 
    using assms by (auto simp:  $W\_def$  setdist_sing_in_set dest!: setdist_eq_0_closedin)
  define  $r$  where  $r \equiv \lambda x. \text{if } x \in W0 \text{ then } r0 x \text{ else } x$ 
  have  $r \restriction (W0 \cup S) \subseteq S$  and  $r \restriction (W0 \cup T) \subseteq T$ 
    using  $\langle r0 \restriction W0 \subseteq S \cap T \rangle$   $r\_def$  by auto
  have contr: continuous_on  $(W0 \cup (S \cup T)) r$ 
    unfolding  $r\_def$ 
  proof (rule continuous_on_cases_local [OF __ contr0 continuous_on_id])
    show closedin (top_of_set  $(W0 \cup (S \cup T))$ )  $W0$ 
      using closedin_subset_trans [of U]
      by (metis le_sup_iff order_refl cloWW0 cloUW closedin_trans  $\langle W0 \subseteq U \rangle$ 
 $\langle S \subseteq U \rangle \langle T \subseteq U \rangle$ )
    show closedin (top_of_set  $(W0 \cup (S \cup T))$ )  $(S \cup T)$ 
      by (meson  $\langle S \subseteq U \rangle \langle T \subseteq U \rangle \langle W0 \subseteq U \rangle$  assms closedin_Un closedin_subset_trans
sup.bounded_iff sup.cobounded2)
    show  $\bigwedge x. x \in W0 \wedge x \notin W0 \vee x \in S \cup T \wedge x \in W0 \implies r0 x = x$ 

```

```

    using ST cloWW0 closedin_subset by fastforce
  qed
  have cloS'WS: closedin (top_of_set S') (W0 ∪ S)
  by (meson closedin_subset_trans US cloUS' ⟨S ⊆ S'⟩ ⟨W ⊆ S'⟩ cloUW cloWW0

    closedin_Un closedin_imp_subset closedin_trans)
  obtain W1 g where W0 ∪ S ⊆ W1 and contg: continuous_on W1 g
    and opeSW1: openin (top_of_set S') W1
    and g ' W1 ⊆ S and geqr:  $\bigwedge x. x \in W0 \cup S \implies g\ x = r\ x$ 
  proof (rule ANR_imp_absolute_neighbourhood_extensor [OF ⟨ANR S⟩ _ ⟨r '
(W0 ∪ S) ⊆ S⟩ cloS'WS])
    show continuous_on (W0 ∪ S) r
    using continuous_on_subset contr sup_assoc by blast
  qed auto
  have cloT'WT: closedin (top_of_set T') (W0 ∪ T)
  by (meson closedin_subset_trans UT cloUT' ⟨T ⊆ T'⟩ ⟨W ⊆ T'⟩ cloUW
cloWW0

    closedin_Un closedin_imp_subset closedin_trans)
  obtain W2 h where W0 ∪ T ⊆ W2 and conth: continuous_on W2 h
    and opeSW2: openin (top_of_set T') W2
    and h ' W2 ⊆ T and heqr:  $\bigwedge x. x \in W0 \cup T \implies h\ x = r\ x$ 
  proof (rule ANR_imp_absolute_neighbourhood_extensor [OF ⟨ANR T⟩ _ ⟨r '
(W0 ∪ T) ⊆ T⟩ cloT'WT])
    show continuous_on (W0 ∪ T) r
    using continuous_on_subset contr sup_assoc by blast
  qed auto
  have S' ∩ T' = W
  by (force simp: S'_def T'_def W_def)
  obtain O1 O2 where O12: open O1 W1 = S' ∩ O1 open O2 W2 = T' ∩ O2
    using opeSW1 opeSW2 by (force simp: openin_open)
  show ?thesis
  proof
    have eq: W1 - (W - U0) ∪ (W2 - (W - U0))
      = ((U - T') ∩ O1 ∪ (U - S') ∩ O2 ∪ U ∩ O1 ∩ O2) - (W - U0)
    (is ?WW1 ∪ ?WW2 = ?rhs)
    using ⟨U0 ∩ W ⊆ W0⟩ ⟨W0 ∪ S ⊆ W1⟩ ⟨W0 ∪ T ⊆ W2⟩
    by (auto simp: ⟨S' ∪ T' = U⟩ [symmetric] ⟨S' ∩ T' = W⟩ [symmetric] ⟨W1
= S' ∩ O1⟩ ⟨W2 = T' ∩ O2⟩)
    show openin (top_of_set U) (?WW1 ∪ ?WW2)
    by (simp add: eq ⟨open O1⟩ ⟨open O2⟩ cloUS' cloUT' cloUW closedin_diff
opeUU0 openin_Int_open openin_Un openin_diff)
    obtain SU' where closed SU' S' = U ∩ SU'
    using cloUS' by (auto simp add: closedin_closed)
    moreover have ?WW1 = (?WW1 ∪ ?WW2) ∩ SU'
    using ⟨S' = U ∩ SU'⟩ ⟨W1 = S' ∩ O1⟩ ⟨S' ∪ T' = U⟩ ⟨W2 = T' ∩ O2⟩
    ⟨S' ∩ T' = W⟩ ⟨W0 ∪ S ⊆ W1⟩ U0
    by auto
    ultimately have cloW1: closedin (top_of_set (W1 - (W - U0) ∪ (W2 -
(W - U0)))) (W1 - (W - U0))

```

```

    by (metis closedin_closed_Int)
  obtain  $TU'$  where  $\text{closed } TU' \ T' = U \cap TU'$ 
    using  $\text{clo}UT'$  by (auto simp add: closedin_closed)
  moreover have  $?WW2 = (?WW1 \cup ?WW2) \cap TU'$ 
    using  $\langle T' = U \cap TU' \rangle \langle W1 = S' \cap O1 \rangle \langle S' \cup T' = U \rangle \langle W2 = T' \cap O2 \rangle$ 
 $\langle S' \cap T' = W \rangle \langle W0 \cup T \subseteq W2 \rangle \ U0$ 
    by auto
  ultimately have  $\text{clo}W2$ :  $\text{closedin } (\text{top\_of\_set } (?WW1 \cup ?WW2)) \ ?WW2$ 
    by (metis closedin_closed_Int)
  let  $?gh = \lambda x. \text{if } x \in S' \text{ then } g \ x \text{ else } h \ x$ 
  have  $\exists r. \text{continuous\_on } (?WW1 \cup ?WW2) \ r \wedge r' \ ( ?WW1 \cup ?WW2) \subseteq S$ 
 $\cup T \wedge (\forall x \in S \cup T. \ r \ x = x)$ 
    proof (intro exI conjI)
      show  $\forall x \in S \cup T. \ ?gh \ x = x$ 
        using  $ST \ \langle S' \cap T' = W \rangle \text{geqr heqr } O12$ 
        by (metis Int_iff Un_iff  $\langle W0 \cup S \subseteq W1 \rangle \langle W0 \cup T \subseteq W2 \rangle \ r0 \ r\_def$ 
 $\text{sup.order\_iff}$ )
      have  $\bigwedge x. \ x \in ?WW1 \wedge x \notin S' \vee x \in ?WW2 \wedge x \in S' \implies g \ x = h \ x$ 
        using  $O12$ 
        by (metis (full_types) DiffD1 DiffD2 DiffI IntE IntI  $U0(2) \ UnCI \ \langle S' \cap T' = W \rangle \text{geqr heqr in\_mono}$ )
      then show  $\text{continuous\_on } (?WW1 \cup ?WW2) \ ?gh$ 
        using  $\text{continuous\_on\_cases\_local } [OF \ \text{clo}W1 \ \text{clo}W2 \ \text{continuous\_on\_subset}$ 
 $[OF \ \text{contg}] \ \text{continuous\_on\_subset } [OF \ \text{conth}]$ 
        by simp
      show  $?gh' \ ( ?WW1 \cup ?WW2) \subseteq S \cup T$ 
        using  $\langle W1 = S' \cap O1 \rangle \langle W2 = T' \cap O2 \rangle \langle S' \cap T' = W \rangle \langle g' \ W1 \subseteq S \rangle$ 
 $\langle h' \ W2 \subseteq T \rangle \langle U0 \cap W \subseteq W0 \rangle \langle W0 \cup S \subseteq W1 \rangle$ 
        by (auto simp add: image_subset_iff)
    qed
  then show  $S \cup T \text{ retract\_of } ?WW1 \cup ?WW2$ 
    using  $\langle W0 \cup S \subseteq W1 \rangle \langle W0 \cup T \subseteq W2 \rangle \ ST \ \text{opeUU0 } U0$ 
    by (auto simp: retract_of_def retraction_def)
  qed
qed

```

lemma $ANR\_closed\_Un\_local$:

```

  fixes  $S :: 'a::euclidean\_space \text{ set}$ 
  assumes  $STS$ :  $\text{closedin } (\text{top\_of\_set } (S \cup T)) \ S$ 
    and  $STT$ :  $\text{closedin } (\text{top\_of\_set } (S \cup T)) \ T$ 
    and  $ANR \ S \ ANR \ T \ ANR(S \cap T)$ 
  shows  $ANR(S \cup T)$ 

```

proof –

```

  have  $\exists T. \text{openin } (\text{top\_of\_set } U) \ T \wedge C \text{ retract\_of } T$ 
    if  $\text{hom}: S \cup T \text{ homeomorphic } C$  and  $UC$ :  $\text{closedin } (\text{top\_of\_set } U) \ C$ 
    for  $U$  and  $C :: ('a * \text{real}) \text{ set}$ 

```

proof –

```

  obtain  $f \ g$  where  $\text{hom}: \text{homeomorphism } (S \cup T) \ C \ f \ g$ 

```

```

    using hom by (force simp: homeomorphic_def)
    have US: closedin (top_of_set U) (C ∩ g - ' S)
    by (metis STS UC closedin_trans continuous_on_imp_closedin hom homeo-
morphicism_def)
    have UT: closedin (top_of_set U) (C ∩ g - ' T)
    by (metis STT UC closedin_trans continuous_on_imp_closedin hom homeo-
morphicism_def)
    have homeomorphism (C ∩ g - ' S) S g f
    using hom
    apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
    by (rule_tac x=f x in image_eqI, auto)
    then have ANRS: ANR (C ∩ g - ' S)
    using ⟨ANR S⟩ homeomorphic_ANR_iff ANR homeomorphic_def by blast
    have homeomorphism (C ∩ g - ' T) T g f
    using hom apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
    by (rule_tac x=f x in image_eqI, auto)
    then have ANRT: ANR (C ∩ g - ' T)
    using ⟨ANR T⟩ homeomorphic_ANR_iff ANR homeomorphic_def by blast
    have homeomorphism (C ∩ g - ' S ∩ (C ∩ g - ' T)) (S ∩ T) g f
    using hom
    apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
    by (rule_tac x=f x in image_eqI, auto)
    then have ANRI: ANR ((C ∩ g - ' S) ∩ (C ∩ g - ' T))
    using ⟨ANR (S ∩ T)⟩ homeomorphic_ANR_iff ANR homeomorphic_def
by blast
    have C = (C ∩ g - ' S) ∪ (C ∩ g - ' T)
    using hom by (auto simp: homeomorphism_def)
    then show ?thesis
    by (metis ANR_closed_Un_local_aux [OF US UT ANRS ANRT ANRI])
qed
    then show ?thesis
    by (auto simp: ANR_def)
qed

corollary ANR_closed_Un:
  fixes S :: 'a::euclidean_space set
  shows ⟨closed S; closed T; ANR S; ANR T; ANR (S ∩ T)⟩ ⟹ ANR (S ∪ T)
by (simp add: ANR_closed_Un_local closedin_def diff_eq open_Compl openin_open_Int)

lemma ANR_openin:
  fixes S :: 'a::euclidean_space set
  assumes ANR T and opeTS: openin (top_of_set T) S
  shows ANR S
proof (clarsimp simp only: ANR_eq_absolute_neighbourhood_extensor)
  fix f :: 'a × real ⟹ 'a and U C
  assume contf: continuous_on C f and fim: f ' C ⊆ S
  and cloUC: closedin (top_of_set U) C
  have f ' C ⊆ T
  using fim opeTS openin_imp_subset by blast

```

```

obtain  $W\ g$  where  $C \subseteq W$ 
  and  $UW$ :  $\text{openin } (\text{top\_of\_set } U) \ W$ 
  and  $\text{contg}$ :  $\text{continuous\_on } W\ g$ 
  and  $\text{gim}$ :  $g \restriction W \subseteq T$ 
  and  $\text{geq}$ :  $\bigwedge x. x \in C \implies g\ x = f\ x$ 
  using  $\text{ANR\_imp\_absolute\_neighbourhood\_extensor}$  [ $OF \langle \text{ANR } T \rangle \text{ contf } \langle f \restriction C \subseteq T \rangle \text{ cloUC}$ ] fm by  $\text{auto}$ 
  show  $\exists V\ g. C \subseteq V \wedge \text{openin } (\text{top\_of\_set } U) \ V \wedge \text{continuous\_on } V\ g \wedge g \restriction V \subseteq S \wedge (\forall x \in C. g\ x = f\ x)$ 
  proof ( $\text{intro exI conjI}$ )
    show  $C \subseteq W \cap g \restriction S$ 
    using  $\langle C \subseteq W \rangle$  fm  $\text{geq}$  by  $\text{blast}$ 
    show  $\text{openin } (\text{top\_of\_set } U) \ (W \cap g \restriction S)$ 
    by ( $\text{metis } (\text{mono\_tags, lifting}) \ UW \ \text{contg} \ \text{continuous\_openin\_preimage} \ \text{gim} \ \text{opeTS} \ \text{openin\_trans}$ )
    show  $\text{continuous\_on } (W \cap g \restriction S) \ g$ 
    by ( $\text{blast intro: continuous\_on\_subset } [OF \ \text{contg}]$ )
    show  $g \restriction (W \cap g \restriction S) \subseteq S$ 
    using  $\text{gim}$  by  $\text{blast}$ 
    show  $\forall x \in C. g\ x = f\ x$ 
    using  $\text{geq}$  by  $\text{blast}$ 
  qed
qed

```

```

lemma  $\text{ENR\_openin}$ :
  fixes  $S :: 'a::\text{euclidean\_space} \ \text{set}$ 
  assumes  $\text{ENR } T \ \text{openin } (\text{top\_of\_set } T) \ S$ 
  shows  $\text{ENR } S$ 
by ( $\text{meson ANR\_openin ENR\_ANR assms locally\_open\_subset}$ )

```

```

lemma  $\text{ANR\_neighborhood\_retract}$ :
  fixes  $S :: 'a::\text{euclidean\_space} \ \text{set}$ 
  assumes  $\text{ANR } U \ S \ \text{retract\_of } T \ \text{openin } (\text{top\_of\_set } U) \ T$ 
  shows  $\text{ANR } S$ 
using  $\text{ANR\_openin ANR\_retract\_of\_ANR assms}$  by  $\text{blast}$ 

```

```

lemma  $\text{ENR\_neighborhood\_retract}$ :
  fixes  $S :: 'a::\text{euclidean\_space} \ \text{set}$ 
  assumes  $\text{ENR } U \ S \ \text{retract\_of } T \ \text{openin } (\text{top\_of\_set } U) \ T$ 
  shows  $\text{ENR } S$ 
using  $\text{ENR\_openin ENR\_retract\_of\_ENR assms}$  by  $\text{blast}$ 

```

```

lemma  $\text{ANR\_rel\_interior}$ :
  fixes  $S :: 'a::\text{euclidean\_space} \ \text{set}$ 
  shows  $\text{ANR } S \implies \text{ANR}(\text{rel\_interior } S)$ 
by ( $\text{blast intro: ANR\_openin openin\_set\_rel\_interior}$ )

```

```

lemma  $\text{ANR\_delete}$ :
  fixes  $S :: 'a::\text{euclidean\_space} \ \text{set}$ 

```

```

shows  $ANR\ S \implies ANR(S - \{a\})$ 
by (blast intro:  $ANR\_openin\ openin\_delete\ openin\_subtopology\_self$ )

```

```

lemma  $ENR\_rel\_interior$ :
  fixes  $S :: 'a::euclidean\_space\ set$ 
  shows  $ENR\ S \implies ENR(rel\_interior\ S)$ 
  by (blast intro:  $ENR\_openin\ openin\_set\_rel\_interior$ )

```

```

lemma  $ENR\_delete$ :
  fixes  $S :: 'a::euclidean\_space\ set$ 
  shows  $ENR\ S \implies ENR(S - \{a\})$ 
  by (blast intro:  $ENR\_openin\ openin\_delete\ openin\_subtopology\_self$ )

```

```

lemma  $open\_imp\_ENR$ :  $open\ S \implies ENR\ S$ 
  using  $ENR\_def$  by blast

```

```

lemma  $open\_imp\_ANR$ :
  fixes  $S :: 'a::euclidean\_space\ set$ 
  shows  $open\ S \implies ANR\ S$ 
  by (simp add:  $ENR\_imp\_ANR\ open\_imp\_ENR$ )

```

```

lemma  $ANR\_ball$  [iff]:
  fixes  $a :: 'a::euclidean\_space$ 
  shows  $ANR(ball\ a\ r)$ 
  by (simp add:  $convex\_imp\_ANR$ )

```

```

lemma  $ENR\_ball$  [iff]:  $ENR(ball\ a\ r)$ 
  by (simp add:  $open\_imp\_ENR$ )

```

```

lemma  $AR\_ball$  [simp]:
  fixes  $a :: 'a::euclidean\_space$ 
  shows  $AR(ball\ a\ r) \longleftrightarrow 0 < r$ 
  by (auto simp:  $AR\_ANR\ convex\_imp\_contractible$ )

```

```

lemma  $ANR\_cball$  [iff]:
  fixes  $a :: 'a::euclidean\_space$ 
  shows  $ANR(cball\ a\ r)$ 
  by (simp add:  $convex\_imp\_ANR$ )

```

```

lemma  $ENR\_cball$ :
  fixes  $a :: 'a::euclidean\_space$ 
  shows  $ENR(cball\ a\ r)$ 
  using  $ENR\_convex\_closed$  by blast

```

```

lemma  $AR\_cball$  [simp]:
  fixes  $a :: 'a::euclidean\_space$ 
  shows  $AR(cball\ a\ r) \longleftrightarrow 0 \leq r$ 
  by (auto simp:  $AR\_ANR\ convex\_imp\_contractible$ )

```

```

lemma ANR_box [iff]:
  fixes a :: 'a::euclidean_space
  shows ANR(cbox a b) ANR(box a b)
  by (auto simp: convex_imp_ANR open_imp_ANR)

lemma ENR_box [iff]:
  fixes a :: 'a::euclidean_space
  shows ENR(cbox a b) ENR(box a b)
  by (simp_all add: ENR_convex_closed closed_cbox open_box open_imp_ENR)

lemma AR_box [simp]:
  AR(cbox a b)  $\longleftrightarrow$  cbox a b  $\neq \{\}$  AR(box a b)  $\longleftrightarrow$  box a b  $\neq \{\}$ 
  by (auto simp: AR_ANR convex_imp_contractible)

lemma ANR_interior:
  fixes S :: 'a::euclidean_space set
  shows ANR(interior S)
  by (simp add: open_imp_ANR)

lemma ENR_interior:
  fixes S :: 'a::euclidean_space set
  shows ENR(interior S)
  by (simp add: open_imp_ENR)

lemma AR_imp_contractible:
  fixes S :: 'a::euclidean_space set
  shows AR S  $\implies$  contractible S
  by (simp add: AR_ANR)

lemma ENR_imp_locally_compact:
  fixes S :: 'a::euclidean_space set
  shows ENR S  $\implies$  locally_compact S
  by (simp add: ENR_ANR)

lemma ANR_imp_locally_path_connected:
  fixes S :: 'a::euclidean_space set
  assumes ANR S
  shows locally_path_connected S
proof -
  obtain U and T :: ('a  $\times$  real) set
    where convex U U  $\neq \{\}$ 
    and UT: closedin (top_of_set U) T and S homeomorphic T
  proof (rule homeomorphic_closedin_convex)
    show aff_dim S < int DIM('a  $\times$  real)
    using aff_dim_le_DIM [of S] by auto
  qed auto
  then have locally_path_connected T
  by (meson ANR_imp_absolute_neighbourhood_retract
    asms convex_imp_locally_path_connected locally_open_subset retract_of_locally_path_connected)

```

```

    then have  $S$ : locally path_connected  $S$ 
      if openin (top_of_set  $U$ )  $V$   $T$  retract_of  $V$   $U \neq \{\}$  for  $V$ 
      using  $\langle S$  homeomorphic  $T \rangle$  homeomorphic_locally homeomorphic_path_connectedness
by blast
    obtain  $Ta$  where (openin (top_of_set  $U$ )  $Ta \wedge T$  retract_of  $Ta$ )
      using ANR_def  $UT$   $\langle S$  homeomorphic  $T \rangle$  assms by moura
    then show ?thesis
      using  $S$   $\langle U \neq \{\} \rangle$  by blast
qed

```

```

lemma ANR_imp_locally_connected:
  fixes  $S :: 'a::euclidean\_space$  set
  assumes ANR  $S$ 
  shows locally connected  $S$ 
using locally_path_connected_imp_locally_connected ANR_imp_locally_path_connected
assms by auto

```

```

lemma AR_imp_locally_path_connected:
  fixes  $S :: 'a::euclidean\_space$  set
  assumes AR  $S$ 
  shows locally path_connected  $S$ 
by (simp add: ANR_imp_locally_path_connected AR_imp_ANR assms)

```

```

lemma AR_imp_locally_connected:
  fixes  $S :: 'a::euclidean\_space$  set
  assumes AR  $S$ 
  shows locally connected  $S$ 
using ANR_imp_locally_connected AR_ANR assms by blast

```

```

lemma ENR_imp_locally_path_connected:
  fixes  $S :: 'a::euclidean\_space$  set
  assumes ENR  $S$ 
  shows locally path_connected  $S$ 
by (simp add: ANR_imp_locally_path_connected ENR_imp_ANR assms)

```

```

lemma ENR_imp_locally_connected:
  fixes  $S :: 'a::euclidean\_space$  set
  assumes ENR  $S$ 
  shows locally connected  $S$ 
using ANR_imp_locally_connected ENR_ANR assms by blast

```

```

lemma ANR_Times:
  fixes  $S :: 'a::euclidean\_space$  set and  $T :: 'b::euclidean\_space$  set
  assumes ANR  $S$  ANR  $T$  shows ANR  $(S \times T)$ 
proof (clarsimp simp only: ANR_eq_absolute_neighbourhood_extensor)
  fix  $f :: ('a \times 'b) \times \text{real} \Rightarrow 'a \times 'b$  and  $U \subseteq C$ 
  assume continuous_on  $C$   $f$  and fin:  $f' C \subseteq S \times T$ 
  and cloUC: closedin (top_of_set  $U$ )  $C$ 
  have contf1: continuous_on  $C$  (fst  $\circ f$ )

```



```

    by (simp add: ‹continuous_on C f› continuous_on_fst)
  obtain W1 g where C ⊆ W1
    and UW1: openin (top_of_set U) W1
    and contg: continuous_on W1 g
    and gim: g ‹ W1 ⊆ S
    and geq:  $\bigwedge x. x \in C \implies g x = (fst \circ f) x$ 
  proof (rule ANR_imp_absolute_neighbourhood_extensor [OF ‹ANR S› contf1
  _ cloUC])
    show (fst ∘ f) ‹ C ⊆ S
      using fim by auto
    qed auto
    have contf2: continuous_on C (snd ∘ f)
      by (simp add: ‹continuous_on C f› continuous_on_snd)
    obtain W2 h where C ⊆ W2
      and UW2: openin (top_of_set U) W2
      and conth: continuous_on W2 h
      and him: h ‹ W2 ⊆ T
      and heq:  $\bigwedge x. x \in C \implies h x = (snd \circ f) x$ 
    proof (rule ANR_imp_absolute_neighbourhood_extensor [OF ‹ANR T› contf2
    _ cloUC])
      show (snd ∘ f) ‹ C ⊆ T
        using fim by auto
      qed auto
      show  $\exists V g. C \subseteq V \wedge$ 
        openin (top_of_set U) V  $\wedge$ 
        continuous_on V g  $\wedge g ‹ V \subseteq S \times T \wedge (\forall x \in C. g x = f x)$ 
      proof (intro exI conjI)
        show C ⊆ W1 ∩ W2
          by (simp add: ‹C ⊆ W1› ‹C ⊆ W2›)
        show openin (top_of_set U) (W1 ∩ W2)
          by (simp add: UW1 UW2 openin_Int)
        show continuous_on (W1 ∩ W2) ( $\lambda x. (g x, h x)$ )
          by (metis (no_types) contg conth continuous_on_Pair continuous_on_subset
          inf_commute inf_le1)
        show ( $\lambda x. (g x, h x)$ ) ‹ (W1 ∩ W2) ⊆ S × T
          using gim him by blast
        show ( $\forall x \in C. (g x, h x) = f x$ )
          using geq heq by auto
      qed
    qed

lemma AR_Times:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes AR S AR T shows AR(S × T)
  using assms by (simp add: AR_ANR ANR_Times contractible_Times)

```

6.41.2 More advanced properties of ANRs and ENRs

lemma ENR_rel_frontier_convex:

```

fixes  $S :: 'a::\text{euclidean\_space set}$ 
assumes  $\text{bounded } S \text{ convex } S$ 
shows  $\text{ENR}(\text{rel\_frontier } S)$ 
proof ( $\text{cases } S = \{\}$ )
  case  $\text{True}$  then show  $?thesis$ 
    by  $\text{simp}$ 
next
  case  $\text{False}$ 
    with  $\text{assms}$  have  $\text{rel\_interior } S \neq \{\}$ 
    by ( $\text{simp add: rel\_interior\_eq\_empty}$ )
    then obtain  $a$  where  $a: a \in \text{rel\_interior } S$ 
    by  $\text{auto}$ 
    have  $\text{ah}S: \text{affine hull } S - \{a\} \subseteq \{x. \text{closest\_point } (\text{affine hull } S) \ x \neq a\}$ 
    by ( $\text{auto simp: closest\_point\_self}$ )
    have  $\text{rel\_frontier } S \text{ retract\_of } \text{affine hull } S - \{a\}$ 
    by ( $\text{simp add: assms } a \text{ rel\_frontier\_retract\_of\_punctured\_affine\_hull}$ )
    also have  $\dots \text{ retract\_of } \{x. \text{closest\_point } (\text{affine hull } S) \ x \neq a\}$ 
    unfolding  $\text{retract\_of\_def retraction\_def ah}S$ 
    apply ( $\text{rule\_tac } x = \text{closest\_point } (\text{affine hull } S) \text{ in } exI$ )
    apply ( $\text{auto simp: False closest\_point\_self affine\_imp\_convex closest\_point\_in\_set}$ 
 $\text{continuous\_on\_closest\_point}$ )
    done
    finally have  $\text{rel\_frontier } S \text{ retract\_of } \{x. \text{closest\_point } (\text{affine hull } S) \ x \neq a\} .$ 
    moreover have  $\text{openin } (\text{top\_of\_set } UNIV) (UNIV \cap \text{closest\_point } (\text{affine hull } S) - \{a\})$ 
    by ( $\text{intro continuous\_openin\_preimage\_gen}$ ) ( $\text{auto simp: False affine\_imp\_convex}$ 
 $\text{continuous\_on\_closest\_point}$ )
    ultimately show  $?thesis$ 
    by ( $\text{meson ENR\_convex\_closed ENR\_delete ENR\_retract\_of\_ENR } \langle \text{rel\_frontier } S \text{ retract\_of } \text{affine hull } S - \{a\} \rangle$ 
 $\text{closed\_affine\_hull convex\_affine\_hull}$ )
qed

```

lemma $\text{ANR\_rel\_frontier\_convex}$:

```

fixes  $S :: 'a::\text{euclidean\_space set}$ 
assumes  $\text{bounded } S \text{ convex } S$ 
shows  $\text{ANR}(\text{rel\_frontier } S)$ 
by ( $\text{simp add: ENR\_imp\_ANR ENR\_rel\_frontier\_convex assms}$ )

```

lemma $\text{ENR\_closedin\_Un\_local}$:

```

fixes  $S :: 'a::\text{euclidean\_space set}$ 
shows  $\llbracket \text{ENR } S; \text{ENR } T; \text{ENR}(S \cap T);$ 
 $\text{closedin } (\text{top\_of\_set } (S \cup T)) \ S; \text{closedin } (\text{top\_of\_set } (S \cup T)) \ T \rrbracket$ 
 $\implies \text{ENR}(S \cup T)$ 
by ( $\text{simp add: ENR\_ANR ANR\_closed\_Un\_local locally\_compact\_closedin\_Un}$ )

```

lemma ENR_closed_Un :

```

fixes  $S :: 'a::\text{euclidean\_space set}$ 
shows  $\llbracket \text{closed } S; \text{closed } T; \text{ENR } S; \text{ENR } T; \text{ENR}(S \cap T) \rrbracket \implies \text{ENR}(S \cup T)$ 

```

by (auto simp: closed_subset ENR_closedin_Un_local)

lemma absolute_retract_Un:

fixes $S :: 'a::euclidean\_space\ set$

shows $\llbracket S\ retract\_of\ UNIV; T\ retract\_of\ UNIV; (S \cap T)\ retract\_of\ UNIV \rrbracket$

$\implies (S \cup T)\ retract\_of\ UNIV$

by (meson AR_closed_Un_local_aux closed_subset retract_of_UNIV retract_of_imp_subset)

lemma retract_from_Un_Int:

fixes $S :: 'a::euclidean\_space\ set$

assumes $clS: closedin\ (top\_of\_set\ (S \cup T))\ S$

and $clT: closedin\ (top\_of\_set\ (S \cup T))\ T$

and $Un: (S \cup T)\ retract\_of\ U$ and $Int: (S \cap T)\ retract\_of\ T$

shows $S\ retract\_of\ U$

proof -

obtain r where $r: continuous\_on\ T\ r\ r\ 'T \subseteq S \cap T\ \forall x \in S \cap T. r\ x = x$

using Int by (auto simp: retraction_def retract_of_def)

have $S\ retract\_of\ S \cup T$

unfolding retraction_def retract_of_def

proof (intro exI conjI)

show $continuous\_on\ (S \cup T)\ (\lambda x. if\ x \in S\ then\ x\ else\ r\ x)$

using r by (intro continuous_on_cases_local [OF $clS\ clT$]) auto

qed (use r in auto)

also have $\dots\ retract\_of\ U$

by (rule Un)

finally show ?thesis .

qed

lemma AR_from_Un_Int_local:

fixes $S :: 'a::euclidean\_space\ set$

assumes $clS: closedin\ (top\_of\_set\ (S \cup T))\ S$

and $clT: closedin\ (top\_of\_set\ (S \cup T))\ T$

and $Un: AR(S \cup T)$ and $Int: AR(S \cap T)$

shows $AR\ S$

by (meson AR_imp_retract AR_retract_of_AR Un assms closedin_closed_subset local.Int

$retract\_from\_Un\_Int\ retract\_of\_refl\ sup\_ge2$)

lemma AR_from_Un_Int_local':

fixes $S :: 'a::euclidean\_space\ set$

assumes $closedin\ (top\_of\_set\ (S \cup T))\ S$

and $closedin\ (top\_of\_set\ (S \cup T))\ T$

and $AR(S \cup T)\ AR(S \cap T)$

shows $AR\ T$

using AR_from_Un_Int_local [of $T\ S$] assms by (simp add: Un_commute Int_commute)

lemma AR_from_Un_Int:

fixes $S :: 'a::euclidean\_space\ set$

assumes clo : $closed\ S\ closed\ T$ and Un : $AR(S \cup T)$ and Int : $AR(S \cap T)$
 shows $AR\ S$
 by (metis $AR\_from\_Un\_Int\_local$ $[OF\_ \_ Un\ Int]$ $Un\_commute\ clo\ closed\_closedin$
 $closedin\_closed\_subset\ inf\_sup\_absorb\ subtopology\_UNIV\ top\_greatest$)

lemma $ANR\_from\_Un\_Int\_local$:

fixes $S :: 'a::euclidean\_space\ set$

assumes clS : $closedin\ (top\_of\_set\ (S \cup T))\ S$

and clT : $closedin\ (top\_of\_set\ (S \cup T))\ T$

and Un : $ANR(S \cup T)$ and Int : $ANR(S \cap T)$

shows $ANR\ S$

proof –

obtain V where clo : $closedin\ (top\_of\_set\ (S \cup T))\ (S \cap T)$

and ope : $openin\ (top\_of\_set\ (S \cup T))\ V$

and ret : $S \cap T\ retract\_of\ V$

using $ANR\_imp\_neighbourhood\_retract$ $[OF\ Int]$ by (metis $clS\ clT\ closedin\_Int$)

then obtain r where r : $continuous\_on\ V\ r$ and rim : $r\ 'V \subseteq S \cap T$ and req :

$\forall x \in S \cap T. r\ x = x$

by (auto simp: $retraction\_def\ retract\_of\_def$)

have V_{sub} : $V \subseteq S \cup T$

by (meson $ope\ openin\_contains\_cball$)

have V_{sup} : $S \cap T \subseteq V$

by (simp add: $retract\_of\_imp\_subset\ ret$)

then have eq : $S \cup V = ((S \cup T) - T) \cup V$

by auto

have eq' : $S \cup V = S \cup (V \cap T)$

using V_{sub} by blast

have $continuous\_on\ (S \cup V \cap T)$ $(\lambda x. if\ x \in S\ then\ x\ else\ r\ x)$

proof (rule $continuous\_on\_cases\_local$)

show $closedin\ (top\_of\_set\ (S \cup V \cap T))\ S$

using $clS\ closedin\_subset\_trans\ inf.boundedE$ by blast

show $closedin\ (top\_of\_set\ (S \cup V \cap T))\ (V \cap T)$

using $clT\ V_{sup}$ by (auto simp: $closedin\_closed$)

show $continuous\_on\ (V \cap T)\ r$

by (meson $Int\_lower1\ continuous\_on\_subset\ r$)

qed (use $req\ continuous\_on\_id$ in auto)

with rim have $S\ retract\_of\ S \cup V$

unfolding $retraction\_def\ retract\_of\_def$ using eq' by fastforce

then show ?thesis

using $ANR\_neighborhood\_retract$ $[OF\ Un]$

using $\langle S \cup V = S \cup T - T \cup V \rangle\ clT\ ope$ by fastforce

qed

lemma $ANR\_from\_Un\_Int$:

fixes $S :: 'a::euclidean\_space\ set$

assumes clo : $closed\ S\ closed\ T$ and Un : $ANR(S \cup T)$ and Int : $ANR(S \cap T)$

shows $ANR\ S$

by (metis $ANR\_from\_Un\_Int\_local$ $[OF\_ \_ Un\ Int]$ $Un\_commute\ clo\ closed\_closedin$
 $closedin\_closed\_subset\ inf\_sup\_absorb\ subtopology\_UNIV\ top\_greatest$)

```

lemma ANR_finite_Union_convex_closed:
  fixes  $\mathcal{T} :: 'a::euclidean\_space \text{ set set}$ 
  assumes  $\mathcal{T}$ : finite  $\mathcal{T}$  and clo:  $\bigwedge C. C \in \mathcal{T} \implies \text{closed } C$  and con:  $\bigwedge C. C \in \mathcal{T} \implies \text{convex } C$ 
  shows  $\text{ANR}(\bigcup \mathcal{T})$ 
proof -
  have  $\text{ANR}(\bigcup \mathcal{T})$  if card  $\mathcal{T} < n$  for  $n$ 
  using assms that
  proof (induction n arbitrary:  $\mathcal{T}$ )
    case 0 then show ?case by simp
  next
    case (Suc n)
    have  $\text{ANR}(\bigcup \mathcal{U})$  if finite  $\mathcal{U}$   $\mathcal{U} \subseteq \mathcal{T}$  for  $\mathcal{U}$ 
    using that
    proof (induction  $\mathcal{U}$ )
      case empty
      then show ?case by simp
    next
      case (insert C  $\mathcal{U}$ )
      have  $\text{ANR}(C \cup \bigcup \mathcal{U})$ 
      proof (rule ANR_closed_Un)
        show  $\text{ANR}(C \cap \bigcup \mathcal{U})$ 
        unfolding Int_Union
        proof (rule Suc)
          show finite  $((\cap) C \text{ ' } \mathcal{U})$ 
          by (simp add: insert.hyps(1))
          show  $\bigwedge Ca. Ca \in (\cap) C \text{ ' } \mathcal{U} \implies \text{closed } Ca$ 
          by (metis (no_types, opaque_lifting) Suc.prem(2) closed_Int subsetD
            imageE insert.prem(1) insertI1 insertI2)
          show  $\bigwedge Ca. Ca \in (\cap) C \text{ ' } \mathcal{U} \implies \text{convex } Ca$ 
          by (metis (mono_tags, lifting) Suc.prem(3) convex_Int imageE insert.prem(1) insert_subset subsetCE)
          show card  $((\cap) C \text{ ' } \mathcal{U}) < n$ 
          proof -
            have card  $\mathcal{T} \leq n$ 
            by (meson Suc.prem(4) not_less not_less_eq)
            then show ?thesis
            by (metis Suc.prem(1) card_image_le card_seteq insert.hyps insert.prem(1) insert_subset le_trans not_less)
          qed
        qed
      qed
    next
      case (insert C  $\mathcal{U}$ )
      show closed  $(\bigcup \mathcal{U})$ 
      using Suc.prem(2) insert.hyps(1) insert.prem by blast
    qed (use Suc.prem(1) convex_imp_ANR insert.prem(1) insert.IH in auto)
    then show ?case
    by simp
  qed
  then show ?case

```

```

      using Suc.prems(1) by blast
    qed
  then show ?thesis
    by blast
  qed

```

```

lemma finite_imp_ANR:
  fixes S :: 'a::euclidean_space set
  assumes finite S
  shows ANR S
proof -
  have ANR( $\bigcup x \in S. \{x\}$ )
    by (blast intro: ANR_finite_Union_convex_closed assms)
  then show ?thesis
    by simp
  qed

```

```

lemma ANR_insert:
  fixes S :: 'a::euclidean_space set
  assumes ANR S closed S
  shows ANR(insert a S)
  by (metis ANR_closed_Un ANR_empty ANR_singleton Diff_disjoint Diff_insert_absorb
assms closed_singleton insert_absorb insert_is_Un)

```

```

lemma ANR_path_component_ANR:
  fixes S :: 'a::euclidean_space set
  shows ANR S  $\implies$  ANR(path_component_set S x)
  using ANR_imp_locally_path_connected ANR_openin openin_path_component_locally_path_connected
  by blast

```

```

lemma ANR_connected_component_ANR:
  fixes S :: 'a::euclidean_space set
  shows ANR S  $\implies$  ANR(connected_component_set S x)
  by (metis ANR_openin openin_connected_component_locally_connected ANR_imp_locally_connected)

```

```

lemma ANR_component_ANR:
  fixes S :: 'a::euclidean_space set
  assumes ANR S c  $c \in \text{components } S$ 
  shows ANR c
  by (metis ANR_connected_component_ANR assms componentsE)

```

6.41.3 Original ANR material, now for ENRs

```

lemma ENR_bounded:
  fixes S :: 'a::euclidean_space set
  assumes bounded S
  shows ENR S  $\longleftrightarrow$  ( $\exists U. \text{open } U \wedge \text{bounded } U \wedge S \text{ retract\_of } U$ )
    (is ?lhs = ?rhs)

```

```

proof
  obtain r where 0 < r and r: S  $\subseteq$  ball 0 r
    using bounded_subset_ballD assms by blast
  assume ?lhs
  then show ?rhs
    by (meson ENR_def Elementary_Metric_Spaces.open_ball bounded_Int bounded_ball
inf_le2 le_inf_iff
open_Int r retract_of_imp_subset retract_of_subset)
next
  assume ?rhs
  then show ?lhs
    using ENR_def by blast
qed

```

```

lemma absolute_retract_imp_AR_gen:
  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  assumes S retract_of T convex T T  $\neq$  {} S homeomorphic S' closedin (top_of_set
U) S'
  shows S' retract_of U
proof -
  have AR T
    by (simp add: assms convex_imp_AR)
  then have AR S
    using AR_retract_of_AR assms by auto
  then show ?thesis
    using assms AR_imp_absolute_retract by metis
qed

```

```

lemma absolute_retract_imp_AR:
  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  assumes S retract_of UNIV S homeomorphic S' closed S'
  shows S' retract_of UNIV
  using AR_imp_absolute_retract_UNIV assms retract_of_UNIV by blast

```

```

lemma homeomorphic_compact_ariness:
  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  assumes S homeomorphic S'
  shows compact S  $\wedge$  S retract_of UNIV  $\longleftrightarrow$  compact S'  $\wedge$  S' retract_of UNIV
  using assms homeomorphic_compactness
  by (metis compact_AR homeomorphic_AR_iff_AR)

```

```

lemma absolute_retract_from_Un_Int:
  fixes S :: 'a::euclidean_space set
  assumes (S  $\cup$  T) retract_of UNIV (S  $\cap$  T) retract_of UNIV closed S closed T
  shows S retract_of UNIV
  using AR_from_Un_Int assms retract_of_UNIV by auto

```

```

lemma ENR_from_Un_Int_gen:
  fixes S :: 'a::euclidean_space set

```

```

assumes closedin (top_of_set ( $S \cup T$ )) S closedin (top_of_set ( $S \cup T$ )) T
ENR( $S \cup T$ ) ENR( $S \cap T$ )
shows ENR S
by (meson ANR_from_Un_Int_local ANR_imp_neighbourhood_retract ENR_ANR
ENR_neighborhood_retract assms)

```

```

lemma ENR_from_Un_Int:
fixes S :: 'a::euclidean_space set
assumes closed S closed T ENR( $S \cup T$ ) ENR( $S \cap T$ )
shows ENR S
by (meson ENR_from_Un_Int_gen assms closed_subset sup_ge1 sup_ge2)

```

```

lemma ENR_finite_Union_convex_closed:
fixes  $\mathcal{T}$  :: 'a::euclidean_space set set
assumes  $\mathcal{T}$ : finite  $\mathcal{T}$  and clo:  $\bigwedge C. C \in \mathcal{T} \implies \text{closed } C$  and con:  $\bigwedge C. C \in \mathcal{T} \implies \text{convex } C$ 
shows ENR( $\bigcup \mathcal{T}$ )
by (simp add: ENR_ANR ANR_finite_Union_convex_closed  $\mathcal{T}$  clo closed_Union
closed_imp_locally_compact con)

```

```

lemma finite_imp_ENR:
fixes S :: 'a::euclidean_space set
shows finite S  $\implies$  ENR S
by (simp add: ENR_ANR finite_imp_ANR finite_imp_closed closed_imp_locally_compact)

```

```

lemma ENR_insert:
fixes S :: 'a::euclidean_space set
assumes closed S ENR S
shows ENR(insert a S)
proof –
have ENR ( $\{a\} \cup S$ )
by (metis ANR_insert ENR_ANR Un_commute Un_insert_right assms closed_imp_locally_compact
closed_insert sup_bot_right)
then show ?thesis
by auto
qed

```

```

lemma ENR_path_component_ENR:
fixes S :: 'a::euclidean_space set
assumes ENR S
shows ENR(path_component_set S x)
by (metis ANR_imp_locally_path_connected ENR_empty ENR_imp_ANR ENR_openin
assms
locally_path_connected_2 openin_subtopology_self path_component_eq_empty)

```


6.41.4 Finally, spheres are ANRs and ENRs

lemma *absolute_retract_homeomorphic_convex_compact*:
fixes $S :: 'a::\text{euclidean\_space}$ *set* **and** $U :: 'b::\text{euclidean\_space}$ *set*
assumes S *homeomorphic* U $S \neq \{\}$ $S \subseteq T$ *convex* U *compact* U
shows S *retract_of* T
by (*metis* *UNIV_I* *assms* *compact_AR* *convex_imp_AR* *homeomorphic_AR_iff_AR* *homeomorphic_compactness* *homeomorphic_empty(1)* *retract_of_subset* *subsetI*)

lemma *frontier_retract_of_punctured_universe*:
fixes $S :: 'a::\text{euclidean\_space}$ *set*
assumes *convex* S *bounded* S $a \in \text{interior } S$
shows (*frontier* S) *retract_of* ($-\{a\}$)
using *rel_frontier_retract_of_punctured_affine_hull*
by (*metis* *Compl_eq_Diff_UNIV* *affine_hull_nonempty_interior* *assms* *empty_iff* *rel_frontier_frontier* *rel_interior_nonempty_interior*)

lemma *sphere_retract_of_punctured_universe_gen*:
fixes $a :: 'a::\text{euclidean\_space}$
assumes $b \in \text{ball } a \ r$
shows *sphere* $a \ r$ *retract_of* ($-\{b\}$)
proof –
have *frontier* (*cball* $a \ r$) *retract_of* ($-\{b\}$)
using *assms* *frontier_retract_of_punctured_universe* *interior_cball* **by** *blast*
then show *?thesis*
by *simp*
qed

lemma *sphere_retract_of_punctured_universe*:
fixes $a :: 'a::\text{euclidean\_space}$
assumes $0 < r$
shows *sphere* $a \ r$ *retract_of* ($-\{a\}$)
by (*simp* *add:* *assms* *sphere_retract_of_punctured_universe_gen*)

lemma *ENR_sphere*:
fixes $a :: 'a::\text{euclidean\_space}$
shows *ENR*(*sphere* $a \ r$)
proof (*cases* $0 < r$)
case *True*
then have *sphere* $a \ r$ *retract_of* $-\{a\}$
by (*simp* *add:* *sphere_retract_of_punctured_universe*)
with *open_delete* **show** *?thesis*
by (*auto* *simp:* *ENR_def*)
next
case *False*
then show *?thesis*
using *finite_imp_ENR*
by (*metis* *finite_insert* *infinite_imp_nonempty* *less_linear* *sphere_eq_empty* *sphere_trivial*)
qed

```

corollary ANR_sphere:
  fixes a :: 'a::euclidean_space
  shows ANR(sphere a r)
  by (simp add: ENR_imp_ANR ENR_sphere)

```

6.41.5 Spheres are connected, etc

```

lemma locally_path_connected_sphere_gen:
  fixes S :: 'a::euclidean_space set
  assumes bounded S and convex S
  shows locally_path_connected (rel_frontier S)
proof (cases rel_interior S = {})
  case True
    with assms show ?thesis
    by (simp add: rel_interior_eq_empty)
  next
    case False
    then obtain a where a: a ∈ rel_interior S
    by blast
    show ?thesis
    proof (rule retract_of_locally_path_connected)
      show locally_path_connected (affine hull S - {a})
      by (meson convex_affine_hull convex_imp_locally_path_connected locally_open_subset
openin_delete openin_subtopology_self)
      show rel_frontier S retract_of affine hull S - {a}
      using a assms rel_frontier_retract_of_punctured_affine_hull by blast
    qed
  qed

```

```

lemma locally_connected_sphere_gen:
  fixes S :: 'a::euclidean_space set
  assumes bounded S and convex S
  shows locally_connected (rel_frontier S)
  by (simp add: ANR_imp_locally_connected ANR_rel_frontier_convex assms)

```

```

lemma locally_path_connected_sphere:
  fixes a :: 'a::euclidean_space
  shows locally_path_connected (sphere a r)
  using ENR_imp_locally_path_connected ENR_sphere by blast

```

```

lemma locally_connected_sphere:
  fixes a :: 'a::euclidean_space
  shows locally_connected(sphere a r)
  using ANR_imp_locally_connected ANR_sphere by blast

```

6.41.6 Borsuk homotopy extension theorem

It's only this late so we can use the concept of retraction, saying that the domain sets or range set are ENRs.

```

theorem Borsuk_homotopy_extension_homotopic:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes cloTS: closedin (top_of_set T) S
    and anr: (ANR S  $\wedge$  ANR T)  $\vee$  ANR U
    and conf: continuous_on T f
    and f' : T  $\subseteq$  U
    and homotopic_with_canon ( $\lambda x.$  True) S U f g
  obtains g' where homotopic_with_canon ( $\lambda x.$  True) T U f g'
    continuous_on T g' image g' T  $\subseteq$  U
     $\bigwedge x. x \in S \implies g' x = g x$ 

proof –
  have S  $\subseteq$  T using assms closedin_imp_subset by blast
  obtain h where conth: continuous_on ( $\{0..1\} \times S$ ) h
    and him: h' : ( $\{0..1\} \times S$ )  $\subseteq$  U
    and [simp]:  $\bigwedge x. h(0, x) = f x \bigwedge x. h(1::real, x) = g x$ 
  using assms by (auto simp: homotopic_with_def)
  define h' where h'  $\equiv \lambda z. \text{if } \text{snd } z \in S \text{ then } h z \text{ else } (f \circ \text{snd}) z$ 
  define B where B  $\equiv \{0::real\} \times T \cup \{0..1\} \times S$ 
  have clo0T: closedin (top_of_set ( $\{0..1\} \times T$ )) ( $\{0::real\} \times T$ )
    by (simp add: Abstract_Topology.closedin_Times)
  moreover have cloT1S: closedin (top_of_set ( $\{0..1\} \times T$ )) ( $\{0..1\} \times S$ )
    by (simp add: Abstract_Topology.closedin_Times assms)
  ultimately have clo0TB: closedin (top_of_set ( $\{0..1\} \times T$ )) B
    by (auto simp: B_def)
  have cloBS: closedin (top_of_set B) ( $\{0..1\} \times S$ )
    by (metis (no_types) Un_subset_iff B_def closedin_subset_trans [OF cloT1S]
    clo0TB closedin_imp_subset closedin_self)
  moreover have cloBT: closedin (top_of_set B) ( $\{0\} \times T$ )
    using  $\langle S \subseteq T \rangle$  closedin_subset_trans [OF clo0T]
    by (metis B_def Un_upper1 clo0TB closedin_closed inf_le1)
  moreover have continuous_on ( $\{0\} \times T$ ) (f  $\circ$  snd)
  proof (rule continuous_intros)+
    show continuous_on (snd' : ( $\{0\} \times T$ )) f
      by (simp add: conf)
  qed
  ultimately have continuous_on ( $\{0..1\} \times S \cup \{0\} \times T$ ) ( $\lambda x. \text{if } \text{snd } x \in S \text{ then } h x \text{ else } (f \circ \text{snd}) x$ )
    by (auto intro!: continuous_on_cases_local conth simp: B_def Un_commute [of  $\{0\} \times T$ ])
  then have conth': continuous_on B h'
    by (simp add: h'_def B_def Un_commute [of  $\{0\} \times T$ ])
  have image h' B  $\subseteq$  U
    using  $\langle f' : T \subseteq U \rangle$  him by (auto simp: h'_def B_def)
  obtain V k where B  $\subseteq$  V and opeTV: openin (top_of_set ( $\{0..1\} \times T$ )) V
    and contk: continuous_on V k and kim: k' : V  $\subseteq$  U

```

```

    and keq:  $\bigwedge x. x \in B \implies k\ x = h'\ x$ 
using anr
proof
  assume ST: ANR S  $\wedge$  ANR T
  have eq:  $(\{0\} \times T \cap \{0..1\} \times S) = \{0::real\} \times S$ 
    using  $\langle S \subseteq T \rangle$  by auto
  have ANR B
    unfolding B_def
  proof (rule ANR_closed_Un_local)
    show closedin (top_of_set  $(\{0\} \times T \cup \{0..1\} \times S)$ )  $(\{0::real\} \times T)$ 
      by (metis cloBT B_def)
    show closedin (top_of_set  $(\{0\} \times T \cup \{0..1\} \times S)$ )  $(\{0..1::real\} \times S)$ 
      by (metis Un_commute cloBS B_def)
  qed (simp_all add: ANR_Times convex_imp_ANR ANR_singleton ST eq)
  note V $k$  = that
  have *: thesis if openin (top_of_set  $(\{0..1::real\} \times T)$ ) V
    retraction V B r for V r
  proof -
    have continuous_on V  $(h' \circ r)$ 
      using conth' continuous_on_compose retractionE that(2) by blast
    moreover have  $(h' \circ r) \text{ ' } V \subseteq U$ 
      by (metis  $\langle h' \text{ ' } B \subseteq U \rangle$  image_comp retractionE that(2))
    ultimately show ?thesis
      using V $k$  [of V  $h' \circ r$ ] by (metis comp_apply retraction that)
  qed
  show thesis
    by (meson * ANR_imp_neighbourhood_retract  $\langle$ ANR B $\rangle$  clo0TB retract_of_def)
next
  assume ANR U
  with ANR_imp_absolute_neighbourhood_extensor  $\langle h' \text{ ' } B \subseteq U \rangle$  clo0TB conth'
that
  show ?thesis by blast
qed
define S' where S'  $\equiv \{x. \exists u::real. u \in \{0..1\} \wedge (u, x::'a) \in \{0..1\} \times T - V\}$ 
have closedin (top_of_set T) S'
  unfolding S'_def using closedin_self opeTV
  by (blast intro: closedin_compact_projection)
have S'_def: S' =  $\{x. \exists u::real. (u, x::'a) \in \{0..1\} \times T - V\}$ 
  by (auto simp: S'_def)
have cloTS': closedin (top_of_set T) S'
  using S'_def  $\langle$ closedin (top_of_set T) S' $\rangle$  by blast
have S  $\cap$  S' = {}
  using S'_def B_def  $\langle B \subseteq V \rangle$  by force
obtain a :: 'a  $\Rightarrow$  real where conta: continuous_on T a
  and  $\bigwedge x. x \in T \implies a\ x \in \text{closed\_segment } 1\ 0$ 
  and a1:  $\bigwedge x. x \in S \implies a\ x = 1$ 
  and a0:  $\bigwedge x. x \in S' \implies a\ x = 0$ 
  by (rule Urysohn_local [OF cloTS cloTS'  $\langle S \cap S' = \{\} \rangle$ , of 1 0], blast)
then have ain:  $\bigwedge x. x \in T \implies a\ x \in \{0..1\}$ 

```

```

    using closed_segment_eq_real_ivl by auto
  have inV:  $(u * a \ t, t) \in V$  if  $t \in T$   $0 \leq u$   $u \leq 1$  for  $t \ u$ 
  proof (rule ccontr)
    assume  $(u * a \ t, t) \notin V$ 
    with ain [OF  $\langle t \in T \rangle$ ] have  $a \ t = 0$ 
    apply simp
    by (metis (no_types, lifting) a0 DiffI S'_def SigmaI atLeastAtMost_iff
    mem_Collect_eq mult_le_one mult_nonneg_nonneg that)
    show False
    using B_def  $\langle (u * a \ t, t) \notin V \rangle \langle B \subseteq V \rangle \langle a \ t = 0 \rangle$  that by auto
  qed
  show ?thesis
  proof
    show hom: homotopic_with_canon  $(\lambda x. \text{True}) \ T \ U \ f \ (\lambda x. k \ (a \ x, x))$ 
    proof (simp add: homotopic_with, intro exI conjI)
      show continuous_on  $(\{0..1\} \times T) \ (k \circ (\lambda z. (fst \ z *_{\mathbb{R}} (a \circ snd) \ z, snd \ z)))$ 
      apply (intro continuous_on_compose continuous_intros)
      apply (force intro: inV continuous_on_subset [OF contk] continuous_on_subset
      [OF conta])
      done
      show  $(k \circ (\lambda z. (fst \ z *_{\mathbb{R}} (a \circ snd) \ z, snd \ z))) \ ' \ (\{0..1\} \times T) \subseteq U$ 
      using inV kim by auto
      show  $\forall x \in T. (k \circ (\lambda z. (fst \ z *_{\mathbb{R}} (a \circ snd) \ z, snd \ z))) \ (0, x) = f \ x$ 
      by (simp add: B_def h'_def keq)
      show  $\forall x \in T. (k \circ (\lambda z. (fst \ z *_{\mathbb{R}} (a \circ snd) \ z, snd \ z))) \ (1, x) = k \ (a \ x, x)$ 
      by auto
    qed
    show continuous_on  $T \ (\lambda x. k \ (a \ x, x))$ 
    using homotopic_with_imp_continuous_maps [OF hom] by auto
    show  $(\lambda x. k \ (a \ x, x)) \ ' \ T \subseteq U$ 
    proof clarify
      fix t
      assume  $t \in T$ 
      show  $k \ (a \ t, t) \in U$ 
      by (metis  $\langle t \in T \rangle$  image_subset_iff inV kim not_one_le_zero linear_mult_cancel_right1)
    qed
    show  $\bigwedge x. x \in S \implies k \ (a \ x, x) = g \ x$ 
    by (simp add: B_def a1 h'_def keq)
  qed
qed

corollary nullhomotopic_into_ANR_extension:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes closed S
    and conf: continuous_on S f
    and ANR T
    and fim:  $f \ ' \ S \subseteq T$ 
    and S  $\neq \{\}$ 

```

shows $(\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S T f (\lambda x. c)) \longleftrightarrow$
 $(\exists g. \text{continuous\_on UNIV } g \wedge \text{range } g \subseteq T \wedge (\forall x \in S. g x = f x))$
(is ?lhs = ?rhs)

proof
assume ?lhs
then obtain c **where** $c: \text{homotopic\_with\_canon } (\lambda x. \text{True}) S T (\lambda x. c) f$
by (blast intro: homotopic_with_symD)
have closedin (top_of_set UNIV) S
using $\langle \text{closed } S \rangle$ closed_closedin **by** fastforce
then obtain g **where** continuous_on UNIV g range $g \subseteq T$
 $\bigwedge x. x \in S \implies g x = f x$
proof (rule Borsuk_homotopy_extension_homotopic)
show range $(\lambda x. c) \subseteq T$
using $\langle S \neq \{\} \rangle$ c homotopic_with_imp_subset1 **by** fastforce
qed (use assms c in auto)
then show ?rhs **by** blast

next
assume ?rhs
then obtain g **where** continuous_on UNIV g range $g \subseteq T \bigwedge x. x \in S \implies g x = f x$
by blast
then obtain c **where** homotopic_with_canon $(\lambda h. \text{True}) \text{UNIV } T g (\lambda x. c)$
using nullhomotopic_from_contractible [of UNIV $g T$] contractible_UNIV **by** blast
then have homotopic_with_canon $(\lambda x. \text{True}) S T g (\lambda x. c)$
by (simp add: homotopic_from_subtopology)
then show ?lhs
by (force elim: homotopic_with_eq [of _ _ _ $g \lambda x. c$] simp: $\langle \bigwedge x. x \in S \implies g x = f x \rangle$)
qed

corollary nullhomotopic_into_rel_frontier_extension:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes closed S
and conf: continuous_on $S f$
and convex T bounded T
and fim: $f \restriction S \subseteq \text{rel\_frontier } T$
and $S \neq \{\}$
shows $(\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S (\text{rel\_frontier } T) f (\lambda x. c)) \longleftrightarrow$
 $(\exists g. \text{continuous\_on UNIV } g \wedge \text{range } g \subseteq \text{rel\_frontier } T \wedge (\forall x \in S. g x = f x))$
by (simp add: nullhomotopic_into_ANR_extension assms ANR_rel_frontier_convex)

corollary nullhomotopic_into_sphere_extension:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes closed S **and** conf: continuous_on $S f$
and $S \neq \{\}$ **and** fim: $f \restriction S \subseteq \text{sphere } a r$
shows $(\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S (\text{sphere } a r) f (\lambda x. c)) \longleftrightarrow$

```

    (∃ g. continuous_on UNIV g ∧ range g ⊆ sphere a r ∧ (∀ x ∈ S. g x = f
x)))
  (is ?lhs = ?rhs)
proof (cases r = 0)
  case True with fim show ?thesis
    by (metis ANR_sphere ⟨closed S⟩ ⟨S ≠ {}⟩ contf nullhomotopic_into_ANR_extension)
next
  case False
  then have eq: sphere a r = rel_frontier (cball a r) by simp
  show ?thesis
    using fim nullhomotopic_into_rel_frontier_extension [OF ⟨closed S⟩ contf
convex_cball bounded_cball]
    by (simp add: ⟨S ≠ {}⟩ eq)
qed

proposition Borsuk_map_essential_bounded_component:
  fixes a :: 'a :: euclidean_space
  assumes compact S and a ∉ S
  shows bounded (connected_component_set (− S) a) ⟷
    ¬(∃ c. homotopic_with_canon (λx. True) S (sphere 0 1)
      (λx. inverse(norm(x − a)) *R (x − a)) (λx. c))
  (is ?lhs = ?rhs)
proof (cases S = {})
  case True then show ?thesis
    by simp
next
  case False
  have closed S bounded S
  using ⟨compact S⟩ compact_eq_bounded_closed by auto
  have s01: (λx. (x − a) /R norm (x − a)) ' S ⊆ sphere 0 1
  using ⟨a ∉ S⟩ by clarsimp (metis dist_eq_0_iff dist_norm mult.commute
right_inverse)
  have aincc: a ∈ connected_component_set (− S) a
  by (simp add: ⟨a ∉ S⟩)
  obtain r where r>0 and r: S ⊆ ball 0 r
  using bounded_subset_ballD ⟨bounded S⟩ by blast
  have ¬ ?rhs ⟷ ¬ ?lhs
  proof
    assume notr: ¬ ?rhs
    have nog: ∄ g. continuous_on (S ∪ connected_component_set (− S) a) g ∧
      g ' (S ∪ connected_component_set (− S) a) ⊆ sphere 0 1 ∧
      (∀ x ∈ S. g x = (x − a) /R norm (x − a))
    if bounded (connected_component_set (− S) a)
    using non_extensible_Borsuk_map [OF ⟨compact S⟩ componentsI _ aincc]
    ⟨a ∉ S⟩ that by auto
    obtain g where range g ⊆ sphere 0 1 continuous_on UNIV g
      ∧ x. x ∈ S ⟹ g x = (x − a) /R norm (x − a)
    using notr
    by (auto simp: nullhomotopic_into_sphere_extension)
  qed

```

```

      [OF ⟨closed S⟩ continuous_on Borsuk_map [OF ⟨a ∉ S⟩ False s01]]
    with ⟨a ∉ S⟩ show ¬ ?lhs
      by (metis UNIV_I continuous_on_subset image_subset_iff nog subsetI)
    next
      assume ¬ ?lhs
      then obtain b where b: b ∈ connected_component_set (− S) a and r ≤ norm
b
      using bounded_iff linear by blast
      then have bnot: b ∉ ball 0 r
      by simp
      have homotopic_with_canon (λx. True) S (sphere 0 1) (λx. (x − a) /R norm
(x − a))
      (λx. (x − b) /R norm (x − b))

    proof −
      have path_component (− S) a b
      by (metis (full_types) ⟨closed S⟩ b mem_Collect_eq open_Compl open_path_connected_component
      then show ?thesis
      using Borsuk_maps_homotopic_in_path_component by blast
    qed
  moreover
  obtain c where homotopic_with_canon (λx. True) (ball 0 r) (sphere 0 1)
      (λx. inverse (norm (x − b)) *R (x − b)) (λx. c)
  proof (rule nullhomotopic_from_contractible)
    show contractible (ball (0::'a) r)
      by (metis convex_imp_contractible convex_ball)
    show continuous_on (ball 0 r) (λx. inverse(norm (x − b)) *R (x − b))
      by (rule continuous_on_Borsuk_map [OF bnot])
    show (λx. (x − b) /R norm (x − b)) ‘ ball 0 r ⊆ sphere 0 1
      using bnot Borsuk_map_into_sphere by blast
    qed blast
  ultimately have homotopic_with_canon (λx. True) S (sphere 0 1) (λx. (x −
a) /R norm (x − a)) (λx. c)
    by (meson homotopic_with_subset_left homotopic_with_trans r)
  then show ¬ ?rhs
    by blast
  qed
  then show ?thesis by blast
qed

lemma homotopic_Borsuk_maps_in_bounded_component:
  fixes a :: 'a :: euclidean_space
  assumes compact S and a ∉ S and b ∉ S
    and boc: bounded (connected_component_set (− S) a)
    and hom: homotopic_with_canon (λx. True) S (sphere 0 1)
      (λx. (x − a) /R norm (x − a))
      (λx. (x − b) /R norm (x − b))
  shows connected_component (− S) a b
proof (rule ccontr)
  assume notcc: ¬ connected_component (− S) a b

```



```

let ?T = S  $\cup$  connected_component_set ( $-$  S) a
have  $\nexists g$ . continuous_on (S  $\cup$  connected_component_set ( $-$  S) a) g  $\wedge$ 
  g ' (S  $\cup$  connected_component_set ( $-$  S) a)  $\subseteq$  sphere 0 1  $\wedge$ 
  ( $\forall x \in S$ . g x = (x - a) /R norm (x - a))
by (simp add:  $\langle a \notin S \rangle$  componentsI non_extensible_Borsuk_map [OF  $\langle$ compact
S $\rangle$  _ boc])
moreover obtain g where continuous_on (S  $\cup$  connected_component_set ( $-$ 
S) a) g
  g ' (S  $\cup$  connected_component_set ( $-$  S) a)  $\subseteq$  sphere 0 1
   $\wedge x$ . x  $\in$  S  $\implies$  g x = (x - a) /R norm (x - a)
proof (rule Borsuk_homotopy_extension_homotopic)
show closedin (top_of_set ?T) S
  by (simp add:  $\langle$ compact S $\rangle$  closed_subset compact_imp_closed)
show continuous_on ?T ( $\lambda x$ . (x - b) /R norm (x - b))
  by (simp add:  $\langle b \notin S \rangle$  notcc_continuous_on_Borsuk_map)
show ( $\lambda x$ . (x - b) /R norm (x - b)) ' ?T  $\subseteq$  sphere 0 1
  by (simp add:  $\langle b \notin S \rangle$  notcc_Borsuk_map_into_sphere)
show homotopic_with_canon ( $\lambda x$ . True) S (sphere 0 1)
  ( $\lambda x$ . (x - b) /R norm (x - b)) ( $\lambda x$ . (x - a) /R norm (x - a))
  by (simp add: hom_homotopic_with_symD)
qed (auto simp: ANR_sphere intro: that)
ultimately show False by blast
qed

```

lemma Borsuk_maps_homotopic_in_connected_component_eq:

```

fixes a :: 'a :: euclidean_space
assumes S: compact S a  $\notin$  S b  $\notin$  S and 2: 2  $\leq$  DIM('a)
shows (homotopic_with_canon ( $\lambda x$ . True) S (sphere 0 1)
  ( $\lambda x$ . (x - a) /R norm (x - a))
  ( $\lambda x$ . (x - b) /R norm (x - b)))  $\longleftrightarrow$ 
  connected_component ( $-$  S) a b
(is ?lhs = ?rhs)
proof
assume L: ?lhs
show ?rhs
proof (cases bounded(connected_component_set ( $-$  S) a))
case True
show ?thesis
  by (rule homotopic_Borsuk_maps_in_bounded_component [OF S True L])
next
case not_bo_a: False
show ?thesis
proof (cases bounded(connected_component_set ( $-$  S) b))
case True
show ?thesis
  using homotopic_Borsuk_maps_in_bounded_component [OF S]
  by (simp add: L True assms connected_component_sym homotopic_Borsuk_maps_in_bounded_component
homotopic_with_sym)

```

```

    next
    case False
    then show ?thesis
    using cobounded_unique_unbounded_component [of  $-S$   $a$   $b$ ]  $\langle \text{compact } S \rangle$ 
not_bo_a
    by (auto simp: compact_eq_bounded_closed assms connected_component_eq_eq)
    qed
  qed
next
  assume R: ?rhs
  then have path_component ( $- S$ )  $a$   $b$ 
  using assms(1) compact_eq_bounded_closed open_Compl open_path_connected_component_set
by fastforce
  then show ?lhs
  by (simp add: Borsuk_maps_homotopic_in_path_component)
qed

```

6.41.7 More extension theorems

```

lemma extension_from_clopen:
  assumes ope: openin (top_of_set  $S$ )  $T$ 
    and clo: closedin (top_of_set  $S$ )  $T$ 
    and contf: continuous_on  $T$   $f$  and fim:  $f \restriction T \subseteq U$  and null:  $U = \{\} \implies S = \{\}$ 
  obtains g where continuous_on  $S$   $g$   $g \restriction S \subseteq U \wedge x. x \in T \implies g\ x = f\ x$ 
proof (cases  $U = \{\}$ )
  case True
  then show ?thesis
  by (simp add: null that)
next
  case False
  then obtain  $a$  where  $a \in U$ 
  by auto
  let ?g =  $\lambda x. \text{if } x \in T \text{ then } f\ x \text{ else } a$ 
  have Seq:  $S = T \cup (S - T)$ 
  using clo closedin_imp_subset by fastforce
  show ?thesis
proof
  have continuous_on ( $T \cup (S - T)$ ) ?g
  using Seq clo ope by (intro continuous_on_cases_local) (auto simp: contf)
  with Seq show continuous_on  $S$  ?g
  by metis
  show ?g  $\restriction S \subseteq U$ 
  using  $\langle a \in U \rangle$  fim by auto
  show  $\bigwedge x. x \in T \implies ?g\ x = f\ x$ 
  by auto
qed
qed

```

```

lemma extension_from_component:
  fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: euclidean_space
  assumes S: locally_connected S  $\vee$  compact S and ANR U
    and C: C  $\in$  components S and contf: continuous_on C f and fim: f ' $C \subseteq U$ 
  obtains g where continuous_on S g g ' $S \subseteq U \wedge x. x \in C \implies g x = f x$ 
proof -
  obtain T g where ope: openin (top_of_set S) T
    and clo: closedin (top_of_set S) T
    and C  $\subseteq$  T and contg: continuous_on T g and gim: g ' $T \subseteq U$ 
    and gf:  $\wedge x. x \in C \implies g x = f x$ 
  using S
proof
  assume locally_connected S
  show ?thesis
    by (metis C  $\langle$ locally_connected S $\rangle$  openin_components_locally_connected
        closedin_component contf fim order_refl that)
  next
  assume compact S
  then obtain W g where C  $\subseteq$  W and opeW: openin (top_of_set S) W
    and contg: continuous_on W g
    and gim: g ' $W \subseteq U$  and gf:  $\wedge x. x \in C \implies g x = f x$ 
  using ANR_imp_absolute_neighbourhood_extensor [of U C f S] C  $\langle$ ANR U $\rangle$ 
        closedin_component contf fim by blast
  then obtain V where open V and V: W = S  $\cap$  V
    by (auto simp: openin_open)
  moreover have locally_compact S
    by (simp add:  $\langle$ compact S $\rangle$  closed_imp_locally_compact compact_imp_closed)
  ultimately obtain K where opeK: openin (top_of_set S) K and compact K
    C  $\subseteq$  K K  $\subseteq$  V
    by (metis C Int_subset_iff  $\langle$ C  $\subseteq$  W $\rangle$   $\langle$ compact S $\rangle$  compact_components
        Sura_Bura_clopen_subset)
  show ?thesis
proof
  show closedin (top_of_set S) K
    by (meson  $\langle$ compact K $\rangle$   $\langle$ compact S $\rangle$  closedin_compact_eq opeK openin_imp_subset)
  show continuous_on K g
    by (metis Int_subset_iff V  $\langle$ K  $\subseteq$  V $\rangle$  contg continuous_on_subset opeK
        openin_subtopology_subset_eq)
  show g ' $K \subseteq U$ 
    using V  $\langle$ K  $\subseteq$  V $\rangle$  gim opeK openin_imp_subset by fastforce
  qed (use opeK gf  $\langle$ C  $\subseteq$  K $\rangle$  in auto)
qed
obtain h where continuous_on S h h ' $S \subseteq U \wedge x. x \in T \implies h x = g x$ 
  using extension_from_clopen
  by (metis C bot.extremum_uniqueI clo contg gim fim image_is_empty in_components_nonempty
      ope)
  then show ?thesis
    by (metis  $\langle$ C  $\subseteq$  T $\rangle$  gf subset_eq that)

```

3040

qed

lemma *tube_lemma*:

fixes $S :: 'a::euclidean\_space\ set$ **and** $T :: 'b::euclidean\_space\ set$

assumes *compact* S **and** $S: S \neq \{\}$ $(\lambda x. (x,a)) ' S \subseteq U$

and $ope: openin\ (top\_of\_set\ (S \times T))\ U$

obtains V **where** $openin\ (top\_of\_set\ T)\ V \wedge a \in V \wedge S \times V \subseteq U$

proof –

let $?W = \{y. \exists x. x \in S \wedge (x, y) \in (S \times T - U)\}$

have $U \subseteq S \times T$ *closedin* $(top\_of\_set\ (S \times T))\ (S \times T - U)$

using ope **by** $(auto\ simp: openin\_closedin\_eq)$

then have *closedin* $(top\_of\_set\ T)\ ?W$

using $\langle compact\ S \rangle$ *closedin_compact_projection* **by** *blast*

moreover have $a \in T - ?W$

using $\langle U \subseteq S \times T \rangle\ S$ **by** *auto*

moreover have $S \times (T - ?W) \subseteq U$

by *auto*

ultimately show $?thesis$

by $(metis\ (no\_types,\ lifting)\ Sigma\_cong\ closedin\_def\ that\ topspace\_euclidean\_subtopology)$

qed

lemma *tube_lemma_gen*:

fixes $S :: 'a::euclidean\_space\ set$ **and** $T :: 'b::euclidean\_space\ set$

assumes *compact* S $S \neq \{\}$ $T \subseteq T' \wedge S \times T \subseteq U$

and $ope: openin\ (top\_of\_set\ (S \times T'))\ U$

obtains V **where** $openin\ (top\_of\_set\ T')\ V \wedge T \subseteq V \wedge S \times V \subseteq U$

proof –

have $\bigwedge x. x \in T \implies \exists V. openin\ (top\_of\_set\ T')\ V \wedge x \in V \wedge S \times V \subseteq U$

using *assms* **by** $(auto\ intro: tube\_lemma\ [OF\ \langle compact\ S \rangle])$

then obtain F **where** $F: \bigwedge x. x \in T \implies openin\ (top\_of\_set\ T')\ (F\ x) \wedge x \in F\ x \wedge S \times F\ x \subseteq U$

by *metis*

show $?thesis$

proof

show $openin\ (top\_of\_set\ T')\ (\bigcup (F ' T))$

using F **by** *blast*

show $T \subseteq \bigcup (F ' T)$

using F **by** *blast*

show $S \times \bigcup (F ' T) \subseteq U$

using F **by** *auto*

qed

qed

proposition *homotopic_neighbourhood_extension*:

fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$

assumes *contf*: *continuous_on* $S\ f$ **and** *fm*: $f ' S \subseteq U$

and *contg*: *continuous_on* $S\ g$ **and** *gim*: $g ' S \subseteq U$

and *clo*: *closedin* $(top\_of\_set\ S)\ T$

```

    and ANR U and hom: homotopic_with_canon ( $\lambda x. \text{True}$ )  $T \ U \ f \ g$ 
  obtains V where  $T \subseteq V$  openin (top_of_set S) V
                    homotopic_with_canon ( $\lambda x. \text{True}$ )  $V \ U \ f \ g$ 
proof -
  have  $T \subseteq S$ 
  using clo closedin_imp_subset by blast
  obtain h where conth: continuous_on ( $\{0..1::\text{real}\} \times T$ ) h
    and him:  $h' \text{ ' } (\{0..1\} \times T) \subseteq U$ 
    and h0:  $\bigwedge x. h(0, x) = f \ x$  and h1:  $\bigwedge x. h(1, x) = g \ x$ 
  using hom by (auto simp: homotopic_with_def)
  define h' where  $h' \equiv \lambda z. \text{if fst } z \in \{0\} \text{ then } f(\text{snd } z)$ 
                     $\text{else if fst } z \in \{1\} \text{ then } g(\text{snd } z)$ 
                     $\text{else } h \ z$ 
  let ?S0 =  $\{0::\text{real}\} \times S$  and ?S1 =  $\{1::\text{real}\} \times S$ 
  have continuous_on(?S0  $\cup$  (?S1  $\cup$   $\{0..1\} \times T$ )) h'
    unfolding h'_def
  proof (intro continuous_on_cases_local)
    show closedin (top_of_set (?S0  $\cup$  (?S1  $\cup$   $\{0..1\} \times T$ ))) ?S0
      closedin (top_of_set (?S1  $\cup$   $\{0..1\} \times T$ )) ?S1
    using  $\langle T \subseteq S \rangle$  by (force intro: closedin_Times closedin_subset_trans [of
 $\{0..1\} \times S$ ]) +
    show closedin (top_of_set (?S0  $\cup$  (?S1  $\cup$   $\{0..1\} \times T$ ))) (?S1  $\cup$   $\{0..1\} \times T$ )
      closedin (top_of_set (?S1  $\cup$   $\{0..1\} \times T$ )) ( $\{0..1\} \times T$ )
    using  $\langle T \subseteq S \rangle$  by (force intro: clo closedin_Times closedin_subset_trans [of
 $\{0..1\} \times S$ ]) +
    show continuous_on (?S0) ( $\lambda x. f \ (\text{snd } x)$ )
      by (intro continuous_intros continuous_on_compose2 [OF contf]) auto
    show continuous_on (?S1) ( $\lambda x. g \ (\text{snd } x)$ )
      by (intro continuous_intros continuous_on_compose2 [OF contg]) auto
  qed (use h0 h1 conth in auto)
  then have continuous_on ( $\{0,1\} \times S \cup (\{0..1\} \times T)$ ) h'
    by (metis Sigma_Un_distrib1 Un_assoc insert_is_Un)
  moreover have  $h' \text{ ' } (\{0,1\} \times S \cup \{0..1\} \times T) \subseteq U$ 
    using fim gim him  $\langle T \subseteq S \rangle$  unfolding h'_def by force
  moreover have closedin (top_of_set ( $\{0..1::\text{real}\} \times S$ )) ( $\{0,1\} \times S \cup \{0..1::\text{real}\} \times T$ )
    by (intro closedin_Times closedin_Un clo) (simp_all add: closed_subset)
  ultimately
  obtain W k where  $W: (\{0,1\} \times S) \cup (\{0..1\} \times T) \subseteq W$ 
    and opeW: openin (top_of_set ( $\{0..1\} \times S$ )) W
    and contk: continuous_on W k
    and kim:  $k \text{ ' } W \subseteq U$ 
    and kh':  $\bigwedge x. x \in (\{0,1\} \times S) \cup (\{0..1\} \times T) \implies k \ x = h' \ x$ 
  by (metis ANR_imp_absolute_neighbourhood_extensor [OF  $\langle \text{ANR } U \rangle$ , of
 $(\{0,1\} \times S) \cup (\{0..1\} \times T)$ ] h'  $\{0..1\} \times S$ ])
  obtain T' where opeT': openin (top_of_set S) T'
    and  $T \subseteq T'$  and TW:  $\{0..1\} \times T' \subseteq W$ 
  using tube_lemma_gen [of  $\{0..1::\text{real}\} \ T \ S \ W$ ] W  $\langle T \subseteq S \rangle$  opeW by auto
  moreover have homotopic_with_canon ( $\lambda x. \text{True}$ )  $T' \ U \ f \ g$ 

```

```

proof (simp add: homotopic_with, intro exI conjI)
  show continuous_on ( $\{0..1\} \times T'$ )  $k$ 
    using TW continuous_on_subset contk by auto
  show  $k \restriction (\{0..1\} \times T') \subseteq U$ 
    using TW kim by fastforce
  have  $T' \subseteq S$ 
    by (meson opeT' subsetD openin_imp_subset)
  then show  $\forall x \in T'. k(0, x) = f x \ \forall x \in T'. k(1, x) = g x$ 
    by (auto simp: kh' h'_def)
qed
ultimately show ?thesis
  by (blast intro: that)
qed

```

Homotopy on a union of closed-open sets.

```

proposition homotopic_on_clopen_Union:
  fixes  $\mathcal{F} :: 'a::euclidean\_space \text{ set set}$ 
  assumes  $\bigwedge S. S \in \mathcal{F} \implies \text{closedin}(\text{top\_of\_set}(\bigcup \mathcal{F})) S$ 
    and  $\bigwedge S. S \in \mathcal{F} \implies \text{openin}(\text{top\_of\_set}(\bigcup \mathcal{F})) S$ 
    and  $\bigwedge S. S \in \mathcal{F} \implies \text{homotopic\_with\_canon}(\lambda x. \text{True}) S T f g$ 
  shows homotopic_with_canon ( $\lambda x. \text{True}$ ) ( $\bigcup \mathcal{F}$ ) T f g
proof -
  obtain  $\mathcal{V}$  where  $\mathcal{V} \subseteq \mathcal{F}$  countable  $\mathcal{V}$  and  $\text{eqU}: \bigcup \mathcal{V} = \bigcup \mathcal{F}$ 
    using Lindelof_openin_assms by blast
  show ?thesis
proof (cases  $\mathcal{V} = \{\}$ )
  case True
    then show ?thesis
      by (metis Union_empty eqU homotopic_with_canon_on_empty)
  next
    case False
    then obtain  $V :: \text{nat} \Rightarrow 'a \text{ set}$  where  $V: \text{range } V = \mathcal{V}$ 
      using range_from_nat_into_countable  $\mathcal{V}$  by metis
    with  $\langle \mathcal{V} \subseteq \mathcal{F} \rangle$  have clo:  $\bigwedge n. \text{closedin}(\text{top\_of\_set}(\bigcup \mathcal{F})) (V n)$ 
      and ope:  $\bigwedge n. \text{openin}(\text{top\_of\_set}(\bigcup \mathcal{F})) (V n)$ 
      and hom:  $\bigwedge n. \text{homotopic\_with\_canon}(\lambda x. \text{True}) (V n) T f g$ 
      using assms by auto
    then obtain  $h$  where conth:  $\bigwedge n. \text{continuous\_on}(\{0..1::\text{real}\} \times V n) (h n)$ 
      and him:  $\bigwedge n. h n \restriction (\{0..1\} \times V n) \subseteq T$ 
      and h0:  $\bigwedge n. \bigwedge x. x \in V n \implies h n(0, x) = f x$ 
      and h1:  $\bigwedge n. \bigwedge x. x \in V n \implies h n(1, x) = g x$ 
      by (simp add: homotopic_with) metis
    have wop:  $b \in V x \implies \exists k. b \in V k \wedge (\forall j < k. b \notin V j)$  for  $b x$ 
      using nat_less_induct [where  $P = \lambda i. b \notin V i$ ] by meson
    obtain  $\zeta$  where cont: continuous_on ( $\{0..1\} \times \bigcup (V \restriction \text{UNIV})$ )  $\zeta$ 
      and eq:  $\bigwedge x i. \llbracket x \in \{0..1\} \times \bigcup (V \restriction \text{UNIV}) \cap \{0..1\} \times (V i - (\bigcup_{m < i} V m)) \rrbracket \implies \zeta x = h i x$ 
proof (rule pasting_lemma_exists)
  let  $?X = \text{top\_of\_set}(\{0..1::\text{real}\} \times \bigcup (\text{range } V))$ 

```

```

show topspace ?X  $\subseteq (\bigcup i. \{0..1::\text{real}\} \times (V\ i - (\bigcup m<i. V\ m)))$ 
  by (force simp: Ball_def dest: wop)
show openin (top_of_set ( $\{0..1\} \times \bigcup (V\ ' UNIV)$ ))
  ( $\{0..1::\text{real}\} \times (V\ i - (\bigcup m<i. V\ m)))$  for i
proof (intro openin_Times openin_subtopology_self openin_diff)
  show openin (top_of_set ( $\bigcup (V\ ' UNIV)$ )) (V i)
    using ope V eqU by auto
  show closedin (top_of_set ( $\bigcup (V\ ' UNIV)$ )) ( $\bigcup m<i. V\ m$ )
    using V clo eqU by (force intro: closedin_Union)
qed
show continuous_map (subtopology ?X ( $\{0..1\} \times (V\ i - \bigcup (V\ ' \{..<i\}))$ ))
euclidean (h i) for i
  by (auto simp add: subtopology_subtopology intro!: continuous_on_subset
    [OF conth])
  show  $\bigwedge i\ j\ x. x \in \text{topspace } ?X \cap \{0..1\} \times (V\ i - (\bigcup m<i. V\ m)) \cap \{0..1\}$ 
     $\times (V\ j - (\bigcup m<j. V\ m))$ 
     $\implies h\ i\ x = h\ j\ x$ 
    by clarsimp (metis lessThan_iff linorder_neqE_nat)
qed auto
show ?thesis
proof (simp add: homotopic_with eqU [symmetric], intro exI conjI ballI)
  show continuous_on ( $\{0..1\} \times \bigcup \mathcal{V}$ )  $\zeta$ 
    using V eqU by (blast intro!: continuous_on_subset [OF cont])
  show  $\zeta\ ' (\{0..1\} \times \bigcup \mathcal{V}) \subseteq T$ 
proof clarsimp
  fix t :: real and y :: 'a and X :: 'a set
  assume  $y \in X\ X \in \mathcal{V}$  and  $t: 0 \leq t \leq 1$ 
  then obtain k where  $y \in V\ k$  and  $j: \forall j<k. y \notin V\ j$ 
    by (metis image_iff V wop)
  with him t show  $\zeta(t, y) \in T$ 
    by (subst eq) force+
qed
fix X y
assume  $X \in \mathcal{V}\ y \in X$ 
then obtain k where  $y \in V\ k$  and  $j: \forall j<k. y \notin V\ j$ 
    by (metis image_iff V wop)
then show  $\zeta(0, y) = f\ y$  and  $\zeta(1, y) = g\ y$ 
    by (subst eq [where i=k]; force simp: h0 h1) $+$ 
qed
qed
qed

```

lemma *homotopic_on_components_eq*:

```

fixes S :: 'a :: euclidean_space set and T :: 'b :: euclidean_space set
assumes S: locally connected S  $\vee$  compact S and ANR T
shows homotopic_with_canon ( $\lambda x. \text{True}$ ) S T  $f\ g \longleftrightarrow$ 
  (continuous_on S  $f \wedge f\ ' S \subseteq T \wedge \text{continuous\_on } S\ g \wedge g\ ' S \subseteq T$ )  $\wedge$ 
  ( $\forall C \in \text{components } S. \text{homotopic\_with\_canon } (\lambda x. \text{True})\ C\ T\ f\ g$ )
(is ?lhs  $\longleftrightarrow$  ?C  $\wedge$  ?rhs)

```

```

proof –
  have continuous_on S f f ' S ⊆ T continuous_on S g g ' S ⊆ T if ?lhs
    using homotopic_with_imp_continuous homotopic_with_imp_subset1 homotopic_with_imp_subset2 that by blast+
    moreover have ?lhs  $\longleftrightarrow$  ?rhs
    if contf: continuous_on S f and fim: f ' S ⊆ T and contg: continuous_on S g
and gim: g ' S ⊆ T
  proof
    assume ?lhs
    with that show ?rhs
      by (simp add: homotopic_with_subset_left in_components_subset)
  next
    assume R: ?rhs
    have  $\exists U. C \subseteq U \wedge \text{closedin } (\text{top\_of\_set } S) \ U \wedge$ 
       $\text{openin } (\text{top\_of\_set } S) \ U \wedge$ 
       $\text{homotopic\_with\_canon } (\lambda x. \text{True}) \ U \ T \ f \ g$  if C: C ∈ components S
for C
  proof –
    have  $C \subseteq S$ 
    by (simp add: in_components_subset that)
    show ?thesis
    using S
    proof
      assume locally connected S
      show ?thesis
      proof (intro exI conjI)
        show  $\text{closedin } (\text{top\_of\_set } S) \ C$ 
        by (simp add: closedin_component that)
        show  $\text{openin } (\text{top\_of\_set } S) \ C$ 
        by (simp add: ⟨locally connected S⟩ openin_components_locally_connected
          that)
        show  $\text{homotopic\_with\_canon } (\lambda x. \text{True}) \ C \ T \ f \ g$ 
        by (simp add: R that)
      qed auto
    next
      assume compact S
      have hom: homotopic_with_canon (λx. True) C T f g
      using R that by blast
      obtain U where  $C \subseteq U$  and opeU: openin (top_of_set S) U
      and hom: homotopic_with_canon (λx. True) U T f g
      using homotopic_neighbourhood_extension [OF contf fim contg gim _
        ⟨ANR T⟩ hom]
       $\langle C \in \text{components } S \rangle \text{closedin\_component}$  by blast
      then obtain V where open V and  $V: U = S \cap V$ 
      by (auto simp: openin_open)
      moreover have locally compact S
      by (simp add: ⟨compact S⟩ closed_imp_locally_compact compact_imp_closed)
      ultimately obtain K where opeK: openin (top_of_set S) K and compact
         $K \ C \subseteq K \ K \subseteq V$ 

```



```

      by (metis C Int_subset_iff Sura_Bura_clopen_subset  $\langle C \subseteq U \rangle$   $\langle compact$ 
 $S \rangle compact\_components$ )
    show ?thesis
    proof (intro exI conjI)
      show closedin (top_of_set S) K
        by (meson  $\langle compact K \rangle$   $\langle compact S \rangle$  closedin_compact_eq opeK
openin_imp_subset)
      show homotopic_with_canon ( $\lambda x. True$ ) K T f g
        using V  $\langle K \subseteq V \rangle$  hom homotopic_with_subset_left opeK openin_imp_subset
by fastforce
      qed (use opeK  $\langle C \subseteq K \rangle$  in auto)
    qed
  qed
  then obtain  $\varphi$  where  $\varphi: \bigwedge C. C \in components\ S \implies C \subseteq \varphi\ C$ 
    and  $clo\varphi: \bigwedge C. C \in components\ S \implies closedin\ (top\_of\_set\ S)\ (\varphi$ 
C)
    and  $ope\varphi: \bigwedge C. C \in components\ S \implies openin\ (top\_of\_set\ S)\ (\varphi$ 
C)
    and  $hom\varphi: \bigwedge C. C \in components\ S \implies homotopic\_with\_canon$ 
( $\lambda x. True$ ) ( $\varphi\ C$ ) T f g
    by metis
  have Seq:  $S = \bigcup (\varphi\ ' components\ S)$ 
  proof
    show  $S \subseteq \bigcup (\varphi\ ' components\ S)$ 
      by (metis Sup_mono Union_components  $\varphi$  imageI)
    show  $\bigcup (\varphi\ ' components\ S) \subseteq S$ 
      using ope $\varphi$  openin_imp_subset by fastforce
  qed
  show ?lhs
    apply (subst Seq)
    using Seq clo $\varphi$  ope $\varphi$  hom $\varphi$  by (intro homotopic_on_clopen_Union) auto
  qed
  ultimately show ?thesis by blast
qed

```

lemma *cohomotopically_trivial_on_components:*

```

  fixes S :: 'a :: euclidean_space set and T :: 'b :: euclidean_space set
  assumes S: locally_connected S  $\vee$  compact S and ANR T
  shows
    ( $\forall f\ g. continuous\_on\ S\ f \longrightarrow f\ ' S \subseteq T \longrightarrow continuous\_on\ S\ g \longrightarrow g\ ' S \subseteq T$ 
 $\longrightarrow$ 
      homotopic_with_canon ( $\lambda x. True$ ) S T f g)
 $\longleftrightarrow$ 
    ( $\forall C \in components\ S.$ 
       $\forall f\ g. continuous\_on\ C\ f \longrightarrow f\ ' C \subseteq T \longrightarrow continuous\_on\ C\ g \longrightarrow g\ ' C$ 
 $\subseteq T \longrightarrow$ 
      homotopic_with_canon ( $\lambda x. True$ ) C T f g)
    (is ?lhs = ?rhs)

```

```

proof
  assume  $L[\text{rule\_format}]$ : ?lhs
  show ?rhs
  proof clarify
    fix  $C\ f\ g$ 
    assume  $\text{contf}: \text{continuous\_on}\ C\ f$  and  $\text{fim}: f \text{ ' } C \subseteq T$ 
    and  $\text{contg}: \text{continuous\_on}\ C\ g$  and  $\text{gim}: g \text{ ' } C \subseteq T$  and  $C: C \in \text{components}\ S$ 
    obtain  $f'$  where  $\text{contf}': \text{continuous\_on}\ S\ f'$  and  $f'im: f' \text{ ' } S \subseteq T$  and  $f'f$ :
 $\bigwedge x. x \in C \implies f' x = f x$ 
    using  $\text{extension\_from\_component}\ [OF\ S\ \langle ANR\ T \rangle\ C\ \text{contf}\ \text{fim}]$  by metis
    obtain  $g'$  where  $\text{contg}': \text{continuous\_on}\ S\ g'$  and  $g'im: g' \text{ ' } S \subseteq T$  and  $g'g$ :
 $\bigwedge x. x \in C \implies g' x = g x$ 
    using  $\text{extension\_from\_component}\ [OF\ S\ \langle ANR\ T \rangle\ C\ \text{contg}\ \text{gim}]$  by metis
    have  $\text{homotopic\_with\_canon}\ (\lambda x. \text{True})\ C\ T\ f'\ g'$ 
    using  $L\ [OF\ \text{contf}'\ f'im\ \text{contg}'\ g'im]\ \text{homotopic\_with\_subset\_left}\ C\ \text{in\_components\_subset}$ 
by fastforce
    then show  $\text{homotopic\_with\_canon}\ (\lambda x. \text{True})\ C\ T\ f\ g$ 
    using  $f'f\ g'g\ \text{homotopic\_with\_eq}$  by force
  qed
next
  assume  $R[\text{rule\_format}]$ : ?rhs
  show ?lhs
  proof clarify
    fix  $f\ g$ 
    assume  $\text{contf}: \text{continuous\_on}\ S\ f$  and  $\text{fim}: f \text{ ' } S \subseteq T$ 
    and  $\text{contg}: \text{continuous\_on}\ S\ g$  and  $\text{gim}: g \text{ ' } S \subseteq T$ 
    moreover have  $\text{homotopic\_with\_canon}\ (\lambda x. \text{True})\ C\ T\ f\ g$  if  $C \in \text{components}\ S$ 
for  $C$ 
    using  $R\ [OF\ \text{that}]$ 
    by (meson  $\text{contf}\ \text{contg}\ \text{continuous\_on\_subset}\ \text{fim}\ \text{gim}\ \text{image\_mono}\ \text{in\_components\_subset}\ \text{order.trans}\ \text{that}$ )
    ultimately show  $\text{homotopic\_with\_canon}\ (\lambda x. \text{True})\ S\ T\ f\ g$ 
    by (subst  $\text{homotopic\_on\_components\_eq}\ [OF\ S\ \langle ANR\ T \rangle])\ \text{auto}$ 
  qed
qed

```

6.41.8 The complement of a set and path-connectedness

Complement in dimension $N > 1$ of set homeomorphic to any interval in any dimension is (path-)connected. This naively generalizes the argument in Ryuji Maehara's paper "The Jordan curve theorem via the Brouwer fixed point theorem", American Mathematical Monthly 1984.

```

lemma  $\text{unbounded\_components\_complement\_absolute\_retract}$ :
  fixes  $S :: 'a::\text{euclidean\_space}\ \text{set}$ 
  assumes  $C: C \in \text{components}(-\ S)$  and  $S: \text{compact}\ S\ AR\ S$ 
  shows  $\neg \text{bounded}\ C$ 
proof –

```

```

obtain  $y$  where  $y$ :  $C = \text{connected\_component\_set } (- S) y$  and  $y \notin S$ 
  using  $C$  by (auto simp: components_def)
have  $\text{open}(- S)$ 
  using  $S$  by (simp add: closed_open compact_eq_bounded_closed)
have  $S \text{ retract\_of } UNIV$ 
  using  $S \text{ compact\_AR}$  by blast
then obtain  $r$  where  $\text{contr}$ :  $\text{continuous\_on } UNIV r$  and  $\text{ontor}$ :  $\text{range } r \subseteq S$ 
  and  $r$ :  $\bigwedge x. x \in S \implies r x = x$ 
  by (auto simp: retract_of_def retraction_def)
show ?thesis
proof
  assume  $\text{bounded } C$ 
  have  $\text{connected\_component\_set } (- S) y \subseteq S$ 
  proof (rule frontier_subset_retraction)
    show  $\text{bounded } (\text{connected\_component\_set } (- S) y)$ 
    using  $\langle \text{bounded } C \rangle y$  by blast
    show  $\text{frontier } (\text{connected\_component\_set } (- S) y) \subseteq S$ 
    using  $C \langle \text{compact } S \rangle \text{ compact\_eq\_bounded\_closed frontier\_of\_components\_closed\_complement}$ 
  y by blast
    show  $\text{continuous\_on } (\text{closure } (\text{connected\_component\_set } (- S) y)) r$ 
    by (blast intro: continuous_on_subset [OF  $\text{contr}$ ])
  qed (use  $\text{ontor } r$  in auto)
  with  $\langle y \notin S \rangle$  show  $\text{False}$  by force
qed
qed

lemma  $\text{connected\_complement\_absolute\_retract}$ :
  fixes  $S :: 'a::\text{euclidean\_space set}$ 
  assumes  $S$ :  $\text{compact } S \text{ AR } S$  and  $2$ :  $2 \leq \text{DIM}('a)$ 
  shows  $\text{connected}(- S)$ 
proof -
  have  $S \text{ retract\_of } UNIV$ 
  using  $S \text{ compact\_AR}$  by blast
  show ?thesis
proof (clarsimp simp: connected_iff_connected_component_eq)
  have  $\neg \text{bounded } (\text{connected\_component\_set } (- S) x)$  if  $x \notin S$  for  $x$ 
  by (meson Compl_iff assms componentsI that unbounded_components_complement_absolute_retract)
  then show  $\text{connected\_component\_set } (- S) x = \text{connected\_component\_set}$ 
 $(- S) y$ 
    if  $x \notin S y \notin S$  for  $x y$ 
    using  $\text{cobounded\_unique\_unbounded\_component [OF } 2]$ 
    by (metis  $\langle \text{compact } S \rangle \text{ compact\_imp\_bounded double\_compl that}$ )
  qed
qed

lemma  $\text{path\_connected\_complement\_absolute\_retract}$ :
  fixes  $S :: 'a::\text{euclidean\_space set}$ 
  assumes  $\text{compact } S \text{ AR } S$   $2 \leq \text{DIM}('a)$ 
  shows  $\text{path\_connected}(- S)$ 

```

```

using connected_complement_absolute_retract [OF assms]
using  $\langle \text{compact } S \rangle$  compact_eq_bounded_closed connected_open_path_connected
by blast

theorem connected_complement_homeomorphic_convex_compact:
  fixes  $S :: 'a::\text{euclidean\_space}$  set and  $T :: 'b::\text{euclidean\_space}$  set
  assumes hom:  $S$  homeomorphic  $T$  and  $T$ : convex  $T$  compact  $T$  and  $2: 2 \leq$ 
     $\text{DIM}('a)$ 
  shows connected( $- S$ )
proof (cases  $S = \{\}$ )
  case True
  then show ?thesis
    by (simp add: connected_UNIV)
next
  case False
  show ?thesis
  proof (rule connected_complement_absolute_retract)
    show compact  $S$ 
    using  $\langle \text{compact } T \rangle$  hom homeomorphic_compactness by auto
    show  $AR\ S$ 
    by (meson  $AR\_ANR\ False \langle \text{convex } T \rangle \text{convex\_imp\_ANR}\ \text{convex\_imp\_contractible}$ 
       $\text{hom homeomorphic\_ANR\_iff\_ANR}\ \text{homeomorphic\_contractible\_eq}$ )
    qed (rule 2)
  qed

```

```

corollary path_connected_complement_homeomorphic_convex_compact:
  fixes  $S :: 'a::\text{euclidean\_space}$  set and  $T :: 'b::\text{euclidean\_space}$  set
  assumes hom:  $S$  homeomorphic  $T$  convex  $T$  compact  $T$   $2 \leq \text{DIM}('a)$ 
  shows path_connected( $- S$ )
  using connected_complement_homeomorphic_convex_compact [OF assms]
  using  $\langle \text{compact } T \rangle$  compact_eq_bounded_closed connected_open_path_connected
    hom homeomorphic_compactness by blast

```

```

lemma path_connected_complement_homeomorphic_interval:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes  $S$  homeomorphic cbox  $a\ b$   $2 \leq \text{DIM}('a)$ 
  shows path_connected( $-S$ )
  using assms compact_cbox convex_box(1) path_connected_complement_homeomorphic_convex_compact
by blast

```

```

lemma connected_complement_homeomorphic_interval:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes  $S$  homeomorphic cbox  $a\ b$   $2 \leq \text{DIM}('a)$ 
  shows connected( $-S$ )
  using assms path_connected_complement_homeomorphic_interval path_connected_imp_connected
by blast

```

end

6.42 Extending Continous Maps, Invariance of Domain, etc

Ported from HOL Light (moretop.ml) by L C Paulson

theory Further_Topology

imports Weierstrass_Theorems Polytope Complex_Transcendental Equivalence_Lebesgue_Henstock_Integration
Retracts

begin

6.42.1 A map from a sphere to a higher dimensional sphere is nullhomotopic

lemma spheremap_lemma1:

fixes $f :: 'a::euclidean\_space \Rightarrow 'a::euclidean\_space$

assumes $subspace\ S\ subspace\ T$ **and** $dimST: dim\ S < dim\ T$

and $S \subseteq T$

and $diff\_f: f\ differentiable\_on\ sphere\ 0\ 1 \cap S$

shows $f\ ' (sphere\ 0\ 1 \cap S) \neq sphere\ 0\ 1 \cap T$

proof

assume $fin: f\ ' (sphere\ 0\ 1 \cap S) = sphere\ 0\ 1 \cap T$

have $inS: \bigwedge x. [x \in S; x \neq 0] \implies (x /_R norm\ x) \in S$

using $subspace\_mul\ \langle subspace\ S \rangle$ **by** blast

have $subS01: (\lambda x. x /_R norm\ x) \ ' (S - \{0\}) \subseteq sphere\ 0\ 1 \cap S$

using $\langle subspace\ S \rangle\ subspace\_mul$ **by** fastforce

then have $diff\_f': f\ differentiable\_on\ (\lambda x. x /_R norm\ x) \ ' (S - \{0\})$

by (rule differentiable_on_subset [OF diff_f])

define g **where** $g \equiv \lambda x. norm\ x *_R f(inverse(norm\ x) *_R x)$

have $gdiff: g\ differentiable\_on\ S - \{0\}$

unfolding g_def

by (rule diff_f' derivative_intros differentiable_on_compose [where f=f] |
force)+

have $geq: g\ ' (S - \{0\}) = T - \{0\}$

proof

have $\bigwedge u. [u \in S; norm\ u *_R f(u /_R norm\ u) \notin T] \implies u = 0$

by (metis (mono_tags, lifting) DiffI subS01 subspace_mul [OF $\langle subspace\ T \rangle$])

$fin\ image\_subset\ iff\ inf\_le2\ singletonD$

then have $g\ ' (S - \{0\}) \subseteq T$

using g_def **by** blast

moreover have $g\ ' (S - \{0\}) \subseteq UNIV - \{0\}$

proof (clarsimp simp: g_def)

fix y

assume $y \in S$ **and** $f0: f(y /_R norm\ y) = 0$

then have $y \neq 0 \implies y /_R norm\ y \in sphere\ 0\ 1 \cap S$

by (auto simp: subspace_mul [OF $\langle subspace\ S \rangle$])

then show $y = 0$

by (metis fin f0 Int_iff image_iff mem_sphere_0 norm_eq_zero zero_neq_one)

qed

ultimately show $g\ ' (S - \{0\}) \subseteq T - \{0\}$

```

    by auto
  next
  have *:  $\text{sphere } 0 \ 1 \cap T \subseteq f \text{ ` } (\text{sphere } 0 \ 1 \cap S)$ 
    using  $\text{fim by (simp add: image\_subset\_iff)}$ 
  have  $x \in (\lambda x. \text{norm } x *_R f \ (x /_R \text{norm } x)) \text{ ` } (S - \{0\})$ 
    if  $x \in T \ x \neq 0$  for  $x$ 
  proof -
    have  $x /_R \text{norm } x \in T$ 
      using  $\langle \text{subspace } T \rangle \text{ subspace\_mul that by blast}$ 
    then obtain  $u$  where  $u: f \ u \in T \ x /_R \text{norm } x = f \ u \ \text{norm } u = 1 \ u \in S$ 
      using  $* \ [THEN \ \text{subsetD}, \ \text{of } x /_R \text{norm } x] \ \langle x \neq 0 \rangle$  by auto
    with that have  $[\text{simp}]: \text{norm } x *_R f \ u = x$ 
      by  $(\text{metis divideR\_right norm\_eq\_zero})$ 
    moreover have  $\text{norm } x *_R u \in S - \{0\}$ 
      using  $\langle \text{subspace } S \rangle \text{ subspace\_scale that(2) } u$  by auto
    with  $u$  show ?thesis
      by  $(\text{simp add: image\_eqI [where } x = \text{norm } x *_R u])$ 
  qed
  then have  $T - \{0\} \subseteq (\lambda x. \text{norm } x *_R f \ (x /_R \text{norm } x)) \text{ ` } (S - \{0\})$ 
    by force
  then show  $T - \{0\} \subseteq g \text{ ` } (S - \{0\})$ 
    by  $(\text{simp add: g\_def})$ 
  qed
  define  $T'$  where  $T' \equiv \{y. \forall x \in T. \text{orthogonal } x \ y\}$ 
  have  $\text{subspace } T'$ 
    by  $(\text{simp add: subspace\_orthogonal\_to\_vectors } T'\_\text{def})$ 
  have  $\text{dim\_eq: dim } T' + \text{dim } T = \text{DIM}('a)$ 
    using  $\text{dim\_subspace\_orthogonal\_to\_vectors [of } T \ \text{UNIV}] \ \langle \text{subspace } T \rangle$ 
    by  $(\text{simp add: } T'\_\text{def})$ 
  have  $\exists v1 \ v2. v1 \in \text{span } T \wedge (\forall w \in \text{span } T. \text{orthogonal } v2 \ w) \wedge x = v1 + v2$ 
  for  $x$ 
    by  $(\text{force intro: orthogonal\_subspace\_decomp\_exists [of } T \ x])$ 
  then obtain  $p1 \ p2$  where  $p1\text{span}: p1 \ x \in \text{span } T$ 
    and  $\bigwedge w. w \in \text{span } T \implies \text{orthogonal } (p2 \ x) \ w$ 
    and  $\text{eq: } p1 \ x + p2 \ x = x$  for  $x$ 
    by  $\text{metis}$ 
  then have  $p1: \bigwedge z. p1 \ z \in T$  and  $\text{ortho: } \bigwedge w. w \in T \implies \text{orthogonal } (p2 \ x) \ w$ 
  for  $x$ 
    using  $\text{span\_eq\_iff } \langle \text{subspace } T \rangle$  by  $\text{blast+}$ 
  then have  $p2: \bigwedge z. p2 \ z \in T'$ 
    by  $(\text{simp add: } T'\_\text{def orthogonal\_commute})$ 
  have  $p12\_eq: \bigwedge x \ y. \llbracket x \in T; y \in T \rrbracket \implies p1(x + y) = x \wedge p2(x + y) = y$ 
  proof  $(\text{rule orthogonal\_subspace\_decomp\_unique [OF eq } p1\text{span, where } T = T'])$ 
    show  $\bigwedge x \ y. \llbracket x \in T; y \in T \rrbracket \implies p2 \ (x + y) \in \text{span } T'$ 
      using  $\text{span\_eq\_iff } p2 \ \langle \text{subspace } T' \rangle$  by  $\text{blast}$ 
    show  $\bigwedge a \ b. \llbracket a \in T; b \in T \rrbracket \implies \text{orthogonal } a \ b$ 
      using  $T'\_\text{def}$  by  $\text{blast}$ 
  qed  $(\text{auto simp: span\_base})$ 
  then have  $\bigwedge c \ x. p1 \ (c *_R x) = c *_R p1 \ x \wedge p2 \ (c *_R x) = c *_R p2 \ x$ 

```

```

proof -
  fix c :: real and x :: 'a
  have f1: c *R x = c *R p1 x + c *R p2 x
    by (metis eq_pth_6)
  have f2: c *R p2 x ∈ T'
    by (simp add: ⟨subspace T'⟩ p2 subspace_scale)
  have c *R p1 x ∈ T
    by (metis (full_types) assms(2) p1span span_eq_iff subspace_scale)
  then show p1 (c *R x) = c *R p1 x ∧ p2 (c *R x) = c *R p2 x
    using f2 f1 p12_eq by presburger
qed
moreover have lin_add:  $\bigwedge x y. p1 (x + y) = p1 x + p1 y \wedge p2 (x + y) = p2 x + p2 y$ 
proof (rule orthogonal_subspace_decomp_unique [OF _ p1span, where T=T'])
  show  $\bigwedge x y. p1 (x + y) + p2 (x + y) = p1 x + p1 y + (p2 x + p2 y)$ 
    by (simp add: add.assoc add.left_commute eq)
  show  $\bigwedge a b. \llbracket a \in T; b \in T' \rrbracket \implies \text{orthogonal } a b$ 
    using T'_def by blast
qed (auto simp: p1span p2 span_base span_add)
ultimately have linear p1 linear p2
  by unfold_locales auto
have g differentiable_on p1 ' {x + y | x y. x ∈ S - {0} ∧ y ∈ T' }
  using p12_eq ⟨S ⊆ T'⟩ by (force intro: differentiable_on_subset [OF gdiff])
then have (λz. g (p1 z)) differentiable_on {x + y | x y. x ∈ S - {0} ∧ y ∈ T'}
  by (rule differentiable_on_compose [OF linear_imp_differentiable_on [OF
    ⟨linear p1⟩]])
then have diff: (λx. g (p1 x) + p2 x) differentiable_on {x + y | x y. x ∈ S - {0} ∧ y ∈ T'}
  by (intro derivative_intros linear_imp_differentiable_on [OF ⟨linear p2⟩])
have dim {x + y | x y. x ∈ S - {0} ∧ y ∈ T'} ≤ dim {x + y | x y. x ∈ S ∧ y ∈ T'}
  by (blast intro: dim_subset)
also have ... = dim S + dim T' - dim (S ∩ T')
  using dim_sums_Int [OF ⟨subspace S⟩ ⟨subspace T'⟩]
  by (simp add: algebra_simps)
also have ... < DIM('a)
  using dimST dim_eq by auto
finally have neg: negligible {x + y | x y. x ∈ S - {0} ∧ y ∈ T'}
  by (rule negligible_lowdim)
have negligible ((λx. g (p1 x) + p2 x) ' {x + y | x y. x ∈ S - {0} ∧ y ∈ T'})
  by (rule negligible_differentiable_image_negligible [OF order_refl neg diff])
then have negligible {x + y | x y. x ∈ g ' (S - {0}) ∧ y ∈ T'}
proof (rule negligible_subset)
  have  $\llbracket t' \in T'; s \in S; s \neq 0 \rrbracket \implies g s + t' \in (\lambda x. g (p1 x) + p2 x) ' \{x + t' | x t'. x \in S \wedge x \neq 0 \wedge t' \in T'\}$ 
    for t' s
    using ⟨S ⊆ T'⟩ p12_eq by (rule_tac x=s+t' in image_eqI) auto
  then show {x + y | x y. x ∈ g ' (S - {0}) ∧ y ∈ T'}
    ⊆ (λx. g (p1 x) + p2 x) ' {x + y | x y. x ∈ S - {0} ∧ y ∈ T'}

```

```

    by auto
  qed
  moreover have  $-T' \subseteq \{x + y \mid x \in g \text{ ` } (S - \{0\}) \wedge y \in T'\}$ 
  proof clarsimp
    fix z assume  $z \notin T'$ 
    show  $\exists x y. z = x + y \wedge x \in g \text{ ` } (S - \{0\}) \wedge y \in T'$ 
    by (metis Diff_iff  $\langle z \notin T' \rangle$  add.left_neutral eq geq p1 p2 singletonD)
  qed
  ultimately have negligible  $(-T')$ 
  using negligible_subset by blast
  moreover have negligible  $T'$ 
  using negligible_lowdim
  by (metis add.commute assms(3) diff_add_inverse2 diff_self_eq_0 dim_eq
  le_add1 le_antisym linordered_semidom_class.add_diff_inverse not_less0)
  ultimately have negligible  $(-T' \cup T')$ 
  by (metis negligible_Un_eq)
  then show False
  using negligible_Un_eq non_negligible_UNIV by simp
  qed

```

lemma *spheremap_lemma2*:

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  'a::euclidean_space
  assumes ST: subspace S subspace T  $\dim S < \dim T$ 
  and S  $\subseteq T$ 
  and conf: continuous_on (sphere 0 1  $\cap$  S) f
  and fim: f  $\text{ ` } (sphere\ 0\ 1 \cap S) \subseteq sphere\ 0\ 1 \cap T$ 
  shows  $\exists c. homotopic\_with\_canon\ (\lambda x. True)\ (sphere\ 0\ 1 \cap S)\ (sphere\ 0\ 1 \cap T)\ f\ (\lambda x. c)$ 
  proof -
    have [simp]:  $\bigwedge x. \llbracket norm\ x = 1; x \in S \rrbracket \Longrightarrow norm\ (f\ x) = 1$ 
    using fim by (simp add: image_subset_iff)
    have compact (sphere 0 1  $\cap$  S)
    by (simp add:  $\langle subspace\ S \rangle$  closed_subspace compact_Int_closed)
    then obtain g where pfg: polynomial_function g and gim: g  $\text{ ` } (sphere\ 0\ 1 \cap S) \subseteq T$ 
    and g12:  $\bigwedge x. x \in sphere\ 0\ 1 \cap S \Longrightarrow norm(f\ x - g\ x) < 1/2$ 
    apply (rule Stone_Weierstrass_polynomial_function_subspace [OF _ conf _
     $\langle subspace\ T \rangle$ , of 1/2])
    using fim by auto
    have gnz:  $g\ x \neq 0$  if  $x \in sphere\ 0\ 1 \cap S$  for x
    proof -
      have norm (f x) = 1
      using fim that by (simp add: image_subset_iff)
      then show ?thesis
      using g12 [OF that] by auto
    qed
    have diffg: g differentiable_on sphere 0 1  $\cap$  S
    by (metis pfg differentiable_on_polynomial_function)

```



```

define h where h  $\equiv \lambda x. \text{inverse}(\text{norm}(g\ x)) *_{\mathbb{R}} g\ x$ 
have h:  $x \in \text{sphere } 0\ 1 \cap S \implies h\ x \in \text{sphere } 0\ 1 \cap T$  for x
  unfolding h_def
  using gnz [of x]
  by (auto simp: subspace_mul [OF ‹subspace T›] subsetD [OF gim])
have diffh: h differentiable_on sphere 0 1  $\cap S$ 
  unfolding h_def using gnz
  by (fastforce intro: derivative_intros diffg differentiable_on_compose [OF diffg])
have homfg: homotopic_with_canon ( $\lambda z. \text{True}$ ) (sphere 0 1  $\cap S$ ) (T - {0}) f g
proof (rule homotopic_with_linear [OF contf])
  show continuous_on (sphere 0 1  $\cap S$ ) g
    using pfg by (simp add: differentiable_imp_continuous_on diffg)
next
have non0fg:  $0 \notin \text{closed\_segment } (f\ x) (g\ x)$  if norm x = 1  $x \in S$  for x
proof -
  have f x  $\in \text{sphere } 0\ 1$ 
    using fim that by (simp add: image_subset_iff)
  moreover have norm(f x - g x) < 1/2
    using g12 that by auto
  ultimately show ?thesis
    by (auto simp: norm_minus_commute dest: segment_bound)
qed
show closed_segment (f x) (g x)  $\subseteq T - \{0\}$  if  $x \in \text{sphere } 0\ 1 \cap S$  for x
proof -
  have convex T
    by (simp add: ‹subspace T› subspace_imp_convex)
  then have convex_hull {f x, g x}  $\subseteq T$ 
    by (metis IntD2 closed_segment_subset fim gim image_subset_iff segment_convex_hull that)
  then show ?thesis
    using that non0fg segment_convex_hull by fastforce
qed
qed
obtain d where d:  $d \in (\text{sphere } 0\ 1 \cap T) - h\ `(\text{sphere } 0\ 1 \cap S)$ 
  using h_spheremap_lemma1 [OF ST ‹S  $\subseteq T$ › diffh] by force
then have non0hd:  $0 \notin \text{closed\_segment } (h\ x) (-d)$  if norm x = 1  $x \in S$  for x
  using midpoint_between [of 0 h x -d] that h [of x]
  by (auto simp: between_mem_segment midpoint_def)
have conth: continuous_on (sphere 0 1  $\cap S$ ) h
  using differentiable_imp_continuous_on diffh by blast
have hom_hd: homotopic_with_canon ( $\lambda z. \text{True}$ ) (sphere 0 1  $\cap S$ ) (T - {0})
  h ( $\lambda x. -d$ )
proof (rule homotopic_with_linear [OF conth continuous_on_const])
  fix x
  assume x:  $x \in \text{sphere } 0\ 1 \cap S$ 
  have convex_hull {h x, -d}  $\subseteq T$ 
  proof (rule hull_minimal)
    show {h x, -d}  $\subseteq T$ 
      using h d x by (force simp: subspace_neg [OF ‹subspace T›])

```

```

    qed (simp add: subspace_imp_convex [OF ⟨subspace T⟩])
    with x segment_convex_hull show closed_segment (h x) (- d) ⊆ T - {0}
    by (auto simp add: subset_Diff_insert non0hd)
  qed
  have conT0: continuous_on (T - {0}) (λy. inverse(norm y) *R y)
  by (intro continuous_intros) auto
  have sub0T: (λy. y /R norm y) ‘ (T - {0}) ⊆ sphere 0 1 ∩ T
  by (fastforce simp: asms(2) subspace_mul)
  obtain c where homhc: homotopic_with_canon (λz. True) (sphere 0 1 ∩ S)
    (sphere 0 1 ∩ T) h (λx. c)
  proof
    show homotopic_with_canon (λz. True) (sphere 0 1 ∩ S) (sphere 0 1 ∩ T) h
      (λx. - d)
    using d
    by (force simp: h_def
      intro: homotopic_with_eq homotopic_with_compose_continuous_left [OF
        hom_hd conT0 sub0T])
  qed
  have homotopic_with_canon (λx. True) (sphere 0 1 ∩ S) (sphere 0 1 ∩ T) f h
  by (force simp: h_def
    intro: homotopic_with_eq homotopic_with_compose_continuous_left
      [OF homfg conT0 sub0T])
  then show ?thesis
  by (metis homotopic_with_trans [OF _ homhc])
  qed

```

lemma *spheremap_lemma3*:

assumes *bounded S convex S subspace U* **and** *affSU: aff_dim S ≤ dim U*
obtains *T* **where** *subspace T T ⊆ U S ≠ {} ⇒ aff_dim T = aff_dim S*
(rel_frontier S) homeomorphic (sphere 0 1 ∩ T)

proof (cases *S = {}*)

case *True*

with *⟨subspace U⟩ subspace_0* **show** *?thesis*

by (rule_tac *T = {}* **in** *that*) **auto**

next

case *False*

then obtain *a* **where** *a ∈ S*

by *auto*

then have *affS: aff_dim S = int (dim ((λx. -a+x) ‘ S))*

by (metis *hull_inc aff_dim_eq_dim*)

with *affSU* **have** *dim ((λx. -a+x) ‘ S) ≤ dim U*

by *linarith*

with *choose_subspace_of_subspace*

obtain *T* **where** *subspace T T ⊆ span U* **and** *dimT: dim T = dim ((λx. -a+x) ‘ S)* .

show *?thesis*

proof (rule *that* [OF *⟨subspace T⟩*])

show *T ⊆ U*

```

    using span_eq_iff ⟨subspace U⟩ ⟨T ⊆ span U⟩ by blast
  show aff_dim T = aff_dim S
    using dimT ⟨subspace T⟩ affS aff_dim_subspace by fastforce
  show rel_frontier S homeomorphic sphere 0 1 ∩ T
  proof -
    have aff_dim (ball 0 1 ∩ T) = aff_dim (T)
    by (metis IntI interior_ball ⟨subspace T⟩ aff_dim_convex_Int_nonempty_interior
centre_in_ball empty_iff inf_commute subspace_0 subspace_imp_convex zero_less_one)
    then have affS_eq: aff_dim S = aff_dim (ball 0 1 ∩ T)
      using ⟨aff_dim T = aff_dim S⟩ by simp
    have rel_frontier S homeomorphic rel_frontier (ball 0 1 ∩ T)
    proof (rule homeomorphic_rel_frontiers_convex_bounded_sets [OF ⟨convex
S⟩ ⟨bounded S⟩])
      show convex (ball 0 1 ∩ T)
        by (simp add: ⟨subspace T⟩ convex_Int subspace_imp_convex)
      show bounded (ball 0 1 ∩ T)
        by (simp add: bounded_Int)
      show aff_dim S = aff_dim (ball 0 1 ∩ T)
        by (rule affS_eq)
    qed
    also have ... = frontier (ball 0 1) ∩ T
    proof (rule convex_affine_rel_frontier_Int [OF convex_ball])
      show affine T
        by (simp add: ⟨subspace T⟩ subspace_imp_affine)
      show interior (ball 0 1) ∩ T ≠ {}
        using ⟨subspace T⟩ subspace_0 by force
    qed
    also have ... = sphere 0 1 ∩ T
      by auto
    finally show ?thesis .
  qed
qed
qed
qed

```

```

proposition inessential_spheremap_lowdim_gen:
  fixes f :: 'M::euclidean_space ⇒ 'a::euclidean_space
  assumes convex S bounded S convex T bounded T
    and affST: aff_dim S < aff_dim T
    and contf: continuous_on (rel_frontier S) f
    and fim: f ` (rel_frontier S) ⊆ rel_frontier T
  obtains c where homotopic_with_canon (λx. True) (rel_frontier S) (rel_frontier
T) f (λx. c)
proof (cases S = {})
  case True
    then show ?thesis
      by (simp add: that)
  next
    case False

```

```

then show ?thesis
proof (cases T = {})
  case True
    then show ?thesis
    using fim that by auto
  next
    case False
    obtain T':: 'a set
      where subspace T' and affT': aff_dim T' = aff_dim T
      and homT: rel_frontier T homeomorphic sphere 0 1  $\cap$  T'
    apply (rule spheremap_lemma3 [OF  $\langle$ bounded T $\rangle$   $\langle$ convex T $\rangle$  subspace_UNIV,
where 'b='a])
      using  $\langle$ T  $\neq$  {} $\rangle$  by (auto simp add: aff_dim_le_DIM)
    with homeomorphic_imp_homotopy_eqv
    have relT: sphere 0 1  $\cap$  T' homotopy_eqv rel_frontier T
      using homotopy_equivalent_space_sym by blast
    have aff_dim S  $\leq$  int (dim T')
      using affT'  $\langle$ subspace T' $\rangle$  affST aff_dim_subspace by force
    with spheremap_lemma3 [OF  $\langle$ bounded S $\rangle$   $\langle$ convex S $\rangle$   $\langle$ subspace T' $\rangle$ ]  $\langle$ S  $\neq$  {} $\rangle$ 
    obtain S':: 'a set where subspace S' S'  $\subseteq$  T'
      and affS': aff_dim S' = aff_dim S
      and homT: rel_frontier S homeomorphic sphere 0 1  $\cap$  S'
      by metis
    with homeomorphic_imp_homotopy_eqv
    have relS: sphere 0 1  $\cap$  S' homotopy_eqv rel_frontier S
      using homotopy_equivalent_space_sym by blast
    have dimST': dim S' < dim T'
      by (metis  $\langle$ S'  $\subseteq$  T' $\rangle$   $\langle$ subspace S' $\rangle$   $\langle$ subspace T' $\rangle$  affS' affST affT' less_irrefl
not_le subspace_dim_equal)
    have  $\exists$  c. homotopic_with_canon ( $\lambda$ z. True) (rel_frontier S) (rel_frontier T)
f ( $\lambda$ x. c)
      apply (rule homotopy_eqv_homotopic_triviality_null_imp [OF relT contf
fim])
      apply (rule homotopy_eqv_cohomotopic_triviality_null[OF relS, THEN
iffD1, rule_format])
      apply (metis dimST'  $\langle$ subspace S' $\rangle$   $\langle$ subspace T' $\rangle$   $\langle$ S'  $\subseteq$  T' $\rangle$  spheremap_lemma2,
blast)
    done
  with that show ?thesis by blast
qed
qed

lemma inessential_spheremap_lowdim:
  fixes f :: 'M::euclidean_space  $\Rightarrow$  'a::euclidean_space
  assumes
    DIM('M) < DIM('a) and f: continuous_on (sphere a r) f f ' (sphere a r)  $\subseteq$ 
(sphere b s)
  obtains c where homotopic_with_canon ( $\lambda$ z. True) (sphere a r) (sphere b s)
f ( $\lambda$ x. c)

```

```

proof (cases  $s \leq 0$ )
  case True then show ?thesis
    by (meson nullhomotopic_into_contractible f contractible_sphere that)
next
  case False
  show ?thesis
  proof (cases  $r \leq 0$ )
    case True then show ?thesis
      by (meson f nullhomotopic_from_contractible contractible_sphere that)
    next
    case False
    with  $\langle \neg s \leq 0 \rangle$  have  $r > 0 \ s > 0$  by auto
    show thesis
    apply (rule inessential_spheremap_lowdim_gen [of cball a r cball b s f])
    using  $\langle 0 < r \rangle \langle 0 < s \rangle$  assms(1) that by (simp_all add: f_aff_dim_cball)
  qed
qed

```

6.42.2 Some technical lemmas about extending maps from cell complexes

```

lemma extending_maps_Union_aux:
  assumes fin: finite  $\mathcal{F}$ 
    and  $\bigwedge S. S \in \mathcal{F} \implies \text{closed } S$ 
    and  $\bigwedge S T. \llbracket S \in \mathcal{F}; T \in \mathcal{F}; S \neq T \rrbracket \implies S \cap T \subseteq K$ 
    and  $\bigwedge S. S \in \mathcal{F} \implies \exists g. \text{continuous\_on } S \ g \wedge g \restriction S \subseteq T \wedge (\forall x \in S \cap K. g \ x = h \ x)$ 
  shows  $\exists g. \text{continuous\_on } (\bigcup \mathcal{F}) \ g \wedge g \restriction (\bigcup \mathcal{F}) \subseteq T \wedge (\forall x \in \bigcup \mathcal{F} \cap K. g \ x = h \ x)$ 
  using assms
  proof (induction  $\mathcal{F}$ )
    case empty show ?case by simp
  next
    case (insert  $S \ \mathcal{F}$ )
    then obtain  $f$  where contf: continuous_on  $(S) \ f$  and fim:  $f \restriction S \subseteq T$  and feq:  $\forall x \in S \cap K. f \ x = h \ x$ 
      by (meson insertI1)
    obtain  $g$  where contg: continuous_on  $(\bigcup \mathcal{F}) \ g$  and gim:  $g \restriction \bigcup \mathcal{F} \subseteq T$  and geq:  $\forall x \in \bigcup \mathcal{F} \cap K. g \ x = h \ x$ 
      using insert by auto
    have fg:  $f \ x = g \ x$  if  $x \in T \cap S$  for  $x \in S$  for  $x \in T$ 
    proof -
      have  $T \cap S \subseteq K \vee S = T$ 
      using that by (metis (no_types) insert.prem1(2) insertCI)
      then show ?thesis
      using UnionI feq geq  $\langle S \notin \mathcal{F} \rangle$  subsetD that by fastforce
    qed
  show ?case
    apply (rule_tac  $x = \lambda x. \text{if } x \in S \text{ then } f \ x \text{ else } g \ x$  in exI, simp)

```

```

    apply (intro conjI continuous_on_cases)
    using fim gim feq geq
    apply (force simp: insert closed_Union contf contg inf_commute intro: fg)+
    done
qed

```

lemma *extending_maps_Union*:

```

  assumes fin: finite  $\mathcal{F}$ 
    and  $\bigwedge S. S \in \mathcal{F} \implies \exists g. \text{continuous\_on } S \ g \wedge g \restriction S \subseteq T \wedge (\forall x \in S \cap K. g \ x = h \ x)$ 
    and  $\bigwedge S. S \in \mathcal{F} \implies \text{closed } S$ 
    and  $K: \bigwedge X \ Y. \llbracket X \in \mathcal{F}; Y \in \mathcal{F}; \neg X \subseteq Y; \neg Y \subseteq X \rrbracket \implies X \cap Y \subseteq K$ 
  shows  $\exists g. \text{continuous\_on } (\bigcup \mathcal{F}) \ g \wedge g \restriction (\bigcup \mathcal{F}) \subseteq T \wedge (\forall x \in \bigcup \mathcal{F} \cap K. g \ x = h \ x)$ 
  apply (simp flip: Union_maximal_sets [OF fin])
  apply (rule extending_maps_Union_aux)
  apply (simp_all add: Union_maximal_sets [OF fin] assms)
  by (metis K psubsetI)

```

lemma *extend_map_lemma*:

```

  assumes finite  $\mathcal{F}$   $\mathcal{G} \subseteq \mathcal{F}$  convex  $T$  bounded  $T$ 
    and poly:  $\bigwedge X. X \in \mathcal{F} \implies \text{polytope } X$ 
    and aff:  $\bigwedge X. X \in \mathcal{F} - \mathcal{G} \implies \text{aff\_dim } X < \text{aff\_dim } T$ 
    and face:  $\bigwedge S \ T. \llbracket S \in \mathcal{F}; T \in \mathcal{F} \rrbracket \implies (S \cap T) \text{ face\_of } S$ 
    and contf: continuous_on  $(\bigcup \mathcal{G}) \ f$  and fim:  $f \restriction (\bigcup \mathcal{G}) \subseteq \text{rel\_frontier } T$ 
  obtains  $g$  where continuous_on  $(\bigcup \mathcal{F}) \ g \wedge g \restriction (\bigcup \mathcal{F}) \subseteq \text{rel\_frontier } T \wedge x \in \bigcup \mathcal{G} \implies g \ x = f \ x$ 
  proof (cases  $\mathcal{F} - \mathcal{G} = \{\}$ )
    case True
      show ?thesis
      proof
        show continuous_on  $(\bigcup \mathcal{F}) \ f$ 
          using True  $\langle \mathcal{G} \subseteq \mathcal{F} \rangle$  contf by auto
        show  $f \restriction (\bigcup \mathcal{F}) \subseteq \text{rel\_frontier } T$ 
          using True fim by auto
      qed auto
    next
      case False
      then have  $0 \leq \text{aff\_dim } T$ 
        by (metis aff aff_dim_empty aff_dim_geq aff_dim_negative_iff all_not_in_conv not_less)
      then obtain  $i::\text{nat}$  where  $i: \text{int } i = \text{aff\_dim } T$ 
        by (metis nonneg_eq_int)
      have Union_empty_eq:  $\bigcup \{D. D = \{\} \wedge P \ D\} = \{\}$  for  $P :: 'a \text{ set} \Rightarrow \text{bool}$ 
        by auto
      have face':  $\bigwedge S \ T. \llbracket S \in \mathcal{F}; T \in \mathcal{F} \rrbracket \implies (S \cap T) \text{ face\_of } S \wedge (S \cap T) \text{ face\_of } T$ 
        by (metis face inf_commute)
      have extendf:  $\exists g. \text{continuous\_on } (\bigcup (\mathcal{G} \cup \{D. \exists C \in \mathcal{F}. D \text{ face\_of } C \wedge \text{aff\_dim } T \leq \text{aff\_dim } C\})) \ g \wedge g \restriction (\bigcup \mathcal{F}) \subseteq \text{rel\_frontier } T$ 

```

```

D < i})) g ∧
  rel_frontier T ∧
    g ' (⋃ (G ∪ {D. ∃ C ∈ F. D face_of C ∧ aff_dim D < i})) ⊆
      (∀ x ∈ ⋃ G. g x = f x)
  if i ≤ aff_dim T for i::nat
using that
proof (induction i)
  case 0
  show ?case
    using 0 contf fm by (auto simp add: Union_empty_eq)
  next
  case (Suc p)
  with ⟨bounded T⟩ have rel_frontier T ≠ {}
  by (auto simp: rel_frontier_eq_empty affine_bounded_eq_lowdim [of T])
  then obtain t where t: t ∈ rel_frontier T by auto
  have ple: int p ≤ aff_dim T using Suc.prem by force
  obtain h where conth: continuous_on (⋃ (G ∪ {D. ∃ C ∈ F. D face_of C ∧
    aff_dim D < p})) h
    and him: h ' (⋃ (G ∪ {D. ∃ C ∈ F. D face_of C ∧ aff_dim D < p}))
      ⊆ rel_frontier T
    and heq: ⋀ x. x ∈ ⋃ G ⇒ h x = f x
  using Suc.IH [OF ple] by auto
  let ?Faces = {D. ∃ C ∈ F. D face_of C ∧ aff_dim D ≤ p}
  have extendh: ∃ g. continuous_on D g ∧
    g ' D ⊆ rel_frontier T ∧
    (∀ x ∈ D ∩ ⋃ (G ∪ {D. ∃ C ∈ F. D face_of C ∧ aff_dim D <
    p}). g x = h x)
  if D: D ∈ G ∪ ?Faces for D
  proof (cases D ⊆ ⋃ (G ∪ {D. ∃ C ∈ F. D face_of C ∧ aff_dim D < p}))
    case True
    have continuous_on D h
    using True conth continuous_on_subset by blast
    moreover have h ' D ⊆ rel_frontier T
    using True him by blast
    ultimately show ?thesis
    by blast
  next
  case False
  note notDsub = False
  show ?thesis
  proof (cases ∃ a. D = {a})
    case True
    then obtain a where D = {a} by auto
    with notDsub t show ?thesis
    by (rule_tac x=λx. t in exI) simp
  next
  case False
  have D ≠ {} using notDsub by auto
  have Dnotin: D ∉ G ∪ {D. ∃ C ∈ F. D face_of C ∧ aff_dim D < p}

```

```

    using notDsub by auto
  then have  $D \notin \mathcal{G}$  by simp
  have  $D \in ?Faces - \{D. \exists C \in \mathcal{F}. D \text{ face\_of } C \wedge \text{aff\_dim } D < p\}$ 
    using Dnotin that by auto
  then obtain  $C$  where  $C \in \mathcal{F}$   $D \text{ face\_of } C$  and affD:  $\text{aff\_dim } D = \text{int } p$ 
    by auto
  then have bounded  $D$ 
    using face_of_polytope_polytope poly polytope_imp_bounded by blast
  then have [simp]:  $\neg \text{affine } D$ 
    using affine_bounded_eq_trivial False  $\langle D \neq \{\} \rangle \langle \text{bounded } D \rangle$  by blast
  have  $\{F. F \text{ facet\_of } D\} \subseteq \{E. E \text{ face\_of } C \wedge \text{aff\_dim } E < \text{int } p\}$ 
    by clarify (metis  $\langle D \text{ face\_of } C \rangle$  affD eq_iff face_of_trans facet_of_def
zle_diff1_eq)
  moreover have polyhedron  $D$ 
    using  $\langle C \in \mathcal{F} \rangle \langle D \text{ face\_of } C \rangle$  face_of_polytope_polytope poly poly-
tope_imp_polyhedron by auto
  ultimately have rel_sub:  $\text{rel\_frontier } D \subseteq \bigcup \{E. E \text{ face\_of } C \wedge \text{aff\_dim}$ 
 $E < p\}$ 
    by (simp add: rel_frontier_of_polyhedron Union_mono)
  then have him_rel:  $h \text{ ' rel\_frontier } D \subseteq \text{rel\_frontier } T$ 
    using  $\langle C \in \mathcal{F} \rangle$  him by blast
  have convex  $D$ 
    by (simp add:  $\langle \text{polyhedron } D \rangle$  polyhedron_imp_convex)
  have affD_lessT:  $\text{aff\_dim } D < \text{aff\_dim } T$ 
    using Suc.premis affD by linarith
  have contDh: continuous_on (rel_frontier  $D$ )  $h$ 
    using  $\langle C \in \mathcal{F} \rangle$  rel_sub by (blast intro: continuous_on_subset [OF conth])
    then have *:  $(\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) (\text{rel\_frontier } D)$ 
 $(\text{rel\_frontier } T) \text{ } h (\lambda x. c)) =$ 
 $(\exists g. \text{continuous\_on UNIV } g \wedge \text{range } g \subseteq \text{rel\_frontier } T \wedge$ 
 $(\forall x \in \text{rel\_frontier } D. g \text{ } x = h \text{ } x))$ 
    by (simp add: assms rel_frontier_eq_empty him_rel nullhomotopic_into_rel_frontier_extension
[OF closed_rel_frontier])
  have  $(\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) (\text{rel\_frontier } D) (\text{rel\_frontier}$ 
 $T) \text{ } h (\lambda x. c))$ 
    by (metis inessential_spheremap_lowdim_gen
[OF  $\langle \text{convex } D \rangle \langle \text{bounded } D \rangle \langle \text{convex } T \rangle \langle \text{bounded } T \rangle$  affD_lessT
contDh him_rel])
  then obtain  $g$  where contg: continuous_on UNIV  $g$ 
    and gim:  $\text{range } g \subseteq \text{rel\_frontier } T$ 
    and gh:  $\bigwedge x. x \in \text{rel\_frontier } D \implies g \text{ } x = h \text{ } x$ 
    by (metis *)
  have  $D \cap E \subseteq \text{rel\_frontier } D$ 
    if  $E \in \mathcal{G} \cup \{D. \exists x \in \mathcal{F} ((\text{face\_of}) \text{ } D) \wedge \text{aff\_dim } D < \text{int } p\}$  for  $E$ 
  proof (rule face_of_subset_rel_frontier)
    show  $D \cap E \text{ face\_of } D$ 
      using that
    proof safe
      assume  $E \in \mathcal{G}$ 

```



```

      then show  $D \cap E \text{ face\_of } D$ 
      by (meson  $\langle C \in \mathcal{F} \rangle \langle D \text{ face\_of } C \rangle \text{ assms}(2) \text{ face' face\_of\_Int\_subface}$ 
         $\text{face\_of\_refl\_eq poly polytope\_imp\_convex subsetD}$ )
    next
      fix  $x$ 
      assume  $\text{aff\_dim } E < \text{int } p \ x \in \mathcal{F} \ E \text{ face\_of } x$ 
      then show  $D \cap E \text{ face\_of } D$ 
      by (meson  $\langle C \in \mathcal{F} \rangle \langle D \text{ face\_of } C \rangle \text{ face' face\_of\_Int\_subface that}$ )
    qed
  show  $D \cap E \neq D$ 
  using that notDsub by auto
qed
moreover have continuous_on  $D \ g$ 
  using contg continuous_on_subset by blast
ultimately show ?thesis
  by (rule_tac  $x=g$  in exI) (use gh gim in fastforce)
qed
qed
have intle:  $i < 1 + \text{int } j \longleftrightarrow i \leq \text{int } j$  for  $i \ j$ 
  by auto
have finite  $\mathcal{G}$ 
  using  $\langle \text{finite } \mathcal{F} \rangle \langle \mathcal{G} \subseteq \mathcal{F} \rangle \text{ rev\_finite\_subset}$  by blast
moreover have finite ( $?Faces$ )
proof -
  have §: finite  $(\bigcup \{ \{ D. D \text{ face\_of } C \} \mid C. C \in \mathcal{F} \})$ 
  by (auto simp:  $\langle \text{finite } \mathcal{F} \rangle \text{ finite\_polytope\_faces poly}$ )
  show ?thesis
  by (auto intro: finite_subset [OF _ §])
qed
ultimately have fin: finite  $(\mathcal{G} \cup ?Faces)$ 
  by simp
have clo: closed  $S$  if  $S \in \mathcal{G} \cup ?Faces$  for  $S$ 
  using that  $\langle \mathcal{G} \subseteq \mathcal{F} \rangle \text{ face\_of\_polytope\_polytope poly polytope\_imp\_closed}$  by
blast
have  $K: X \cap Y \subseteq \bigcup (\mathcal{G} \cup \{ D. \exists C \in \mathcal{F}. D \text{ face\_of } C \wedge \text{aff\_dim } D < \text{int } p \})$ 
  if  $X \in \mathcal{G} \cup ?Faces \ Y \in \mathcal{G} \cup ?Faces \neg Y \subseteq X$  for  $X \ Y$ 
proof -
  have ff:  $X \cap Y \text{ face\_of } X \wedge X \cap Y \text{ face\_of } Y$ 
  if  $XY: X \text{ face\_of } D \ Y \text{ face\_of } E$  and  $DE: D \in \mathcal{F} \ E \in \mathcal{F}$  for  $D \ E$ 
  by (rule face\_of\_Int\_subface [OF _ _ XY]) (auto simp: face' DE)
  show ?thesis
  using that
  apply auto
  apply (drule_tac  $x=X \cap Y$  in spec, safe)
  using ff face\_of\_imp\_convex [of  $X$ ] face\_of\_imp\_convex [of  $Y$ ]
  apply (fastforce dest: face\_of\_aff\_dim_lt)
  by (meson face\_of\_trans ff)
qed
obtain  $g$  where continuous_on  $(\bigcup (\mathcal{G} \cup ?Faces)) \ g$ 

```

```

      g '  $\bigcup (\mathcal{G} \cup ?Faces) \subseteq rel\_frontier\ T$ 
      ( $\forall x \in \bigcup (\mathcal{G} \cup ?Faces) \cap$ 
         $\bigcup (\mathcal{G} \cup \{D. \exists C \in \mathcal{F}. D\ face\_of\ C \wedge aff\_dim\ D < p\})$ ).  $g\ x =$ 
      h x)
    by (rule exE [OF extending_maps_Union [OF fin_extend_h clo K]], blast+)
  then show ?case
    by (simp add: intle_local.heq [symmetric], blast)
qed
have eq:  $\bigcup (\mathcal{G} \cup \{D. \exists C \in \mathcal{F}. D\ face\_of\ C \wedge aff\_dim\ D < i\}) = \bigcup \mathcal{F}$ 
proof
  show  $\bigcup (\mathcal{G} \cup \{D. \exists C \in \mathcal{F}. D\ face\_of\ C \wedge aff\_dim\ D < int\ i\}) \subseteq \bigcup \mathcal{F}$ 
    using  $\langle \mathcal{G} \subseteq \mathcal{F} \rangle\ face\_of\_imp\_subset$  by fastforce
  show  $\bigcup \mathcal{F} \subseteq \bigcup (\mathcal{G} \cup \{D. \exists C \in \mathcal{F}. D\ face\_of\ C \wedge aff\_dim\ D < i\})$ 
    proof (rule Union_mono)
      show  $\mathcal{F} \subseteq \mathcal{G} \cup \{D. \exists C \in \mathcal{F}. D\ face\_of\ C \wedge aff\_dim\ D < int\ i\}$ 
        using face by (fastforce simp: aff i)
    qed
  qed
have int i  $\leq aff\_dim\ T$  by (simp add: i)
then show ?thesis
  using extendf [of i] unfolding eq by (metis that)
qed

lemma extend_map_lemma_cofinite0:
  assumes finite  $\mathcal{F}$ 
    and pairwise  $(\lambda S\ T. S \cap T \subseteq K)\ \mathcal{F}$ 
    and  $\bigwedge S. S \in \mathcal{F} \implies \exists a\ g. a \notin U \wedge continuous\_on\ (S - \{a\})\ g \wedge g\ ' (S - \{a\}) \subseteq T \wedge (\forall x \in S \cap K. g\ x = h\ x)$ 
    and  $\bigwedge S. S \in \mathcal{F} \implies closed\ S$ 
  shows  $\exists C\ g. finite\ C \wedge disjoint\ C\ U \wedge card\ C \leq card\ \mathcal{F} \wedge$ 
     $continuous\_on\ (\bigcup \mathcal{F} - C)\ g \wedge g\ ' (\bigcup \mathcal{F} - C) \subseteq T$ 
     $\wedge (\forall x \in (\bigcup \mathcal{F} - C) \cap K. g\ x = h\ x)$ 
  using assms
proof induction
  case empty then show ?case
    by force
next
  case (insert X  $\mathcal{F}$ )
  then have closed X and clo:  $\bigwedge X. X \in \mathcal{F} \implies closed\ X$ 
    and  $\mathcal{F}: \bigwedge S. S \in \mathcal{F} \implies \exists a\ g. a \notin U \wedge continuous\_on\ (S - \{a\})\ g \wedge g\ ' (S - \{a\}) \subseteq T \wedge (\forall x \in S \cap K. g\ x = h\ x)$ 
    and pwX:  $\bigwedge Y. Y \in \mathcal{F} \wedge Y \neq X \longrightarrow X \cap Y \subseteq K \wedge Y \cap X \subseteq K$ 
    and pwF: pairwise  $(\lambda S\ T. S \cap T \subseteq K)\ \mathcal{F}$ 
    by (simp_all add: pairwise_insert)
  obtain C g where C: finite C disjoint C U card C  $\leq card\ \mathcal{F}$ 
    and contg: continuous_on  $(\bigcup \mathcal{F} - C)\ g$ 
    and gim:  $g\ ' (\bigcup \mathcal{F} - C) \subseteq T$ 
    and gh:  $\bigwedge x. x \in (\bigcup \mathcal{F} - C) \cap K \implies g\ x = h\ x$ 
    using insert.IH [OF pwF  $\mathcal{F}$  clo] by auto

```

```

obtain  $a f$  where  $a \notin U$ 
  and  $\text{contf}$ :  $\text{continuous\_on } (X - \{a\}) f$ 
  and  $\text{fim}$ :  $f \text{ ' } (X - \{a\}) \subseteq T$ 
  and  $\text{fh}$ :  $(\forall x \in X \cap K. f x = h x)$ 
using  $\text{insert.premis}$  by ( $\text{meson insertI1}$ )
show  $?case$ 
proof ( $\text{intro exI conjI}$ )
  show  $\text{finite } (\text{insert } a C)$ 
    by ( $\text{simp add: } C$ )
  show  $\text{disjnt } (\text{insert } a C) U$ 
    using  $C \langle a \notin U \rangle$  by  $\text{simp}$ 
  show  $\text{card } (\text{insert } a C) \leq \text{card } (\text{insert } X \mathcal{F})$ 
    by ( $\text{simp add: } C \text{ card\_insert\_if insert.hyps le\_SucI}$ )
  have  $\text{closed } (\bigcup \mathcal{F})$ 
    using  $\text{clo insert.hyps}$  by  $\text{blast}$ 
  have  $\text{continuous\_on } (X - \text{insert } a C) f$ 
    using  $\text{contf}$  by ( $\text{force simp: elim: continuous\_on\_subset}$ )
  moreover have  $\text{continuous\_on } (\bigcup \mathcal{F} - \text{insert } a C) g$ 
    using  $\text{contg}$  by ( $\text{force simp: elim: continuous\_on\_subset}$ )
  ultimately
  have  $\text{continuous\_on } (X - \text{insert } a C \cup (\bigcup \mathcal{F} - \text{insert } a C)) (\lambda x. \text{if } x \in X$ 
then  $f x$  else  $g x$ )
    apply ( $\text{intro continuous\_on\_cases\_local; simp add: closedin\_closed}$ )
    using  $\langle \text{closed } X \rangle$  apply  $\text{blast}$ 
    using  $\langle \text{closed } (\bigcup \mathcal{F}) \rangle$  apply  $\text{blast}$ 
    using  $\text{fh gh insert.hyps pwX}$  by  $\text{fastforce}$ 
  then show  $\text{continuous\_on } (\bigcup (\text{insert } X \mathcal{F}) - \text{insert } a C) (\lambda a. \text{if } a \in X \text{ then } f$ 
 $a$  else  $g a$ )
    by ( $\text{blast intro: continuous\_on\_subset}$ )
  show  $\forall x \in (\bigcup (\text{insert } X \mathcal{F}) - \text{insert } a C) \cap K. (\text{if } x \in X \text{ then } f x \text{ else } g x) = h$ 
 $x$ 
    using  $\text{gh}$  by ( $\text{auto simp: fh}$ )
  show  $(\lambda a. \text{if } a \in X \text{ then } f a \text{ else } g a) \text{ ' } (\bigcup (\text{insert } X \mathcal{F}) - \text{insert } a C) \subseteq T$ 
    using  $\text{fim gim}$  by  $\text{auto force}$ 
qed
qed

```

lemma $\text{extend\_map\_lemma\_cofinite1}$:

assumes $\text{finite } \mathcal{F}$

and $\mathcal{F}$: $\bigwedge X. X \in \mathcal{F} \implies \exists a g. a \notin U \wedge \text{continuous\_on } (X - \{a\}) g \wedge g \text{ ' } (X - \{a\}) \subseteq T \wedge (\forall x \in X \cap K. g x = h x)$

and clo : $\bigwedge X. X \in \mathcal{F} \implies \text{closed } X$

and K : $\bigwedge X Y. [X \in \mathcal{F}; Y \in \mathcal{F}; \neg X \subseteq Y; \neg Y \subseteq X] \implies X \cap Y \subseteq K$

obtains $C g$ **where** $\text{finite } C \text{ disjnt } C U \text{ card } C \leq \text{card } \mathcal{F} \text{ continuous\_on } (\bigcup \mathcal{F} - C) g$

$g \text{ ' } (\bigcup \mathcal{F} - C) \subseteq T$

$\bigwedge x. x \in (\bigcup \mathcal{F} - C) \cap K \implies g x = h x$

proof –

```

let ?F = {X ∈ F. ∀ Y ∈ F. ¬ X ⊂ Y}
have [simp]: ⋃ ?F = ⋃ F
  by (simp add: Union_maximal_sets assms)
have fin: finite ?F
  by (force intro: finite_subset [OF_ ⟨finite F⟩])
have pw: pairwise (λ S T. S ∩ T ⊆ K) ?F
  by (simp add: pairwise_def) (metis K psubsetI)
have card {X ∈ F. ∀ Y ∈ F. ¬ X ⊂ Y} ≤ card F
  by (simp add: ⟨finite F⟩ card_mono)
moreover
obtain C g where finite C ∧ disjnt C U ∧ card C ≤ card ?F ∧
  continuous_on (⋃ ?F - C) g ∧ g ' (⋃ ?F - C) ⊆ T
  ∧ (∀ x ∈ (⋃ ?F - C) ∩ K. g x = h x)
  using extend_map_lemma_cofinite0 [OF fin pw, of U T h] by (fastforce intro!:
clo F)
ultimately show ?thesis
  by (rule_tac C=C and g=g in that) auto
qed

```

```

lemma extend_map_lemma_cofinite:
  assumes finite F G ⊆ F and T: convex T bounded T
  and poly: ⋀ X. X ∈ F ⇒ polytope X
  and conf: continuous_on (⋃ G) f and fim: f ' (⋃ G) ⊆ rel_frontier T
  and face: ⋀ X Y. [X ∈ F; Y ∈ F] ⇒ (X ∩ Y) face_of X
  and aff: ⋀ X. X ∈ F - G ⇒ aff_dim X ≤ aff_dim T
  obtains C g where
    finite C disjnt C (⋃ G) card C ≤ card F continuous_on (⋃ F - C) g
    g ' (⋃ F - C) ⊆ rel_frontier T ⋀ x. x ∈ ⋃ G ⇒ g x = f x
proof -
  define H where H ≡ G ∪ {D. ∃ C ∈ F - G. D face_of C ∧ aff_dim D <
aff_dim T}
  have finite G
  using assms finite_subset by blast
  have *: finite (⋃ {D. D face_of C} | C. C ∈ F)
  using finite_polytope_faces poly ⟨finite F⟩ by force
  then have finite H
  by (auto simp: H_def ⟨finite G⟩ intro: finite_subset [OF_*])
  have face': ⋀ S T. [S ∈ F; T ∈ F] ⇒ (S ∩ T) face_of S ∧ (S ∩ T) face_of T
  by (metis face inf_commute)
  have *: ⋀ X Y. [X ∈ H; Y ∈ H] ⇒ X ∩ Y face_of X
  unfolding H_def
  using subsetD [OF ⟨G ⊆ F⟩] apply (auto simp add: face)
  apply (meson face' face_of_Int_subface face_of_refl_eq poly polytope_imp_convex)+
  done
  obtain h where conth: continuous_on (⋃ H) h and him: h ' (⋃ H) ⊆ rel_frontier
T
  and hf: ⋀ x. x ∈ ⋃ G ⇒ h x = f x
  proof (rule extend_map_lemma [OF ⟨finite H⟩ [unfolded H_def] Un_upper1

```

```

T])
  show  $\bigwedge X. \llbracket X \in \mathcal{G} \cup \{D. \exists C \in \mathcal{F} - \mathcal{G}. D \text{ face\_of } C \wedge \text{aff\_dim } D < \text{aff\_dim } T\} \rrbracket \implies \text{polytope } X$ 
  using  $\langle \mathcal{G} \subseteq \mathcal{F} \rangle \text{ face\_of\_polytope\_polytope } \text{poly}$  by fastforce
qed (use *  $\mathcal{H\_def}$  contf fim in auto)
have bounded  $(\bigcup \mathcal{G})$ 
  using  $\langle \text{finite } \mathcal{G} \rangle \langle \mathcal{G} \subseteq \mathcal{F} \rangle \text{ poly\_polytope\_imp\_bounded}$  by blast
then have  $\bigcup \mathcal{G} \neq \text{UNIV}$ 
  by auto
then obtain a where  $a: a \notin \bigcup \mathcal{G}$ 
  by blast
have  $\mathcal{F}: \exists a \ g. a \notin \bigcup \mathcal{G} \wedge \text{continuous\_on } (D - \{a\}) \ g \wedge$ 
 $g \text{ ' } (D - \{a\}) \subseteq \text{rel\_frontier } T \wedge (\forall x \in D \cap \bigcup \mathcal{H}. g \ x = h \ x)$ 
  if  $D \in \mathcal{F}$  for D
proof (cases  $D \subseteq \bigcup \mathcal{H}$ )
  case True
  then have  $h \text{ ' } (D - \{a\}) \subseteq \text{rel\_frontier } T \text{ continuous\_on } (D - \{a\}) \ h$ 
    using him by (blast intro!:  $\langle a \notin \bigcup \mathcal{G} \rangle \text{ continuous\_on\_subset } [OF \text{ conth}]) +$ 
  then show ?thesis
    using a by blast
  next
  case False
  note  $D\_not\_subset = \text{False}$ 
  show ?thesis
  proof (cases  $D \in \mathcal{G}$ )
    case True
    with  $D\_not\_subset$  show ?thesis
      by (auto simp:  $\mathcal{H\_def}$ )
  next
  case False
  then have  $\text{affD}: \text{aff\_dim } D \leq \text{aff\_dim } T$ 
    by (simp add:  $\langle D \in \mathcal{F} \rangle \text{ aff}$ )
  show ?thesis
  proof (cases  $\text{rel\_interior } D = \{\}$ )
    case True
    with  $\langle D \in \mathcal{F} \rangle \text{ poly } a$  show ?thesis
      by (force simp:  $\text{rel\_interior\_eq\_empty polytope\_imp\_convex}$ )
  next
  case False
  then obtain b where  $\text{brelD}: b \in \text{rel\_interior } D$ 
    by blast
  have  $\text{polyhedron } D$ 
    by (simp add:  $\text{poly\_polytope\_imp\_polyhedron that}$ )
  have  $\text{rel\_frontier } D \text{ retract\_of affine hull } D - \{b\}$ 
    by (simp add:  $\text{rel\_frontier\_retract\_of\_punctured\_affine\_hull poly polytope\_imp\_bounded polytope\_imp\_convex that brelD}$ )
  then obtain r where  $\text{relfD}: \text{rel\_frontier } D \subseteq \text{affine hull } D - \{b\}$ 
    and  $\text{contr}: \text{continuous\_on } (\text{affine hull } D - \{b\}) \ r$ 
    and  $\text{rim}: r \text{ ' } (\text{affine hull } D - \{b\}) \subseteq \text{rel\_frontier } D$ 

```

```

      and rid:  $\bigwedge x. x \in \text{rel\_frontier } D \implies r \ x = x$ 
    by (auto simp: retract_of_def retraction_def)
  show ?thesis
proof (intro exI conjI ballI)
  show  $b \notin \bigcup \mathcal{G}$ 
  proof clarify
    fix E
    assume  $b \in E \ E \in \mathcal{G}$ 
    then have  $E \cap D \text{ face\_of } E \wedge E \cap D \text{ face\_of } D$ 
    using  $\langle \mathcal{G} \subseteq \mathcal{F} \rangle \text{ face' that}$  by auto
  with face_of_subset_rel_frontier  $\langle E \in \mathcal{G} \rangle \langle b \in E \rangle \text{ brelD rel\_interior\_subset}$ 
[ of D]
    D_not_subset rel_frontier_def  $\mathcal{H\_def}$ 
  show False
  by blast
qed
have  $r \ (D - \{b\}) \subseteq r \ (\text{affine hull } D - \{b\})$ 
  by (simp add: Diff_mono hull_subset image_mono)
also have  $\dots \subseteq \text{rel\_frontier } D$ 
  by (rule rim)
also have  $\dots \subseteq \bigcup \{E. E \text{ face\_of } D \wedge \text{aff\_dim } E < \text{aff\_dim } T\}$ 
  using affD
  by (force simp: rel_frontier_of_polyhedron [OF  $\langle \text{polyhedron } D \rangle$ 
facet_of_def])
also have  $\dots \subseteq \bigcup (\mathcal{H})$ 
  using D_not_subset  $\mathcal{H\_def}$  that by fastforce
finally have rsub:  $r \ (D - \{b\}) \subseteq \bigcup (\mathcal{H})$  .
show continuous_on  $(D - \{b\}) (h \circ r)$ 
proof (rule continuous_on_compose)
  show continuous_on  $(D - \{b\}) r$ 
  by (meson Diff_mono continuous_on_subset contr hull_subset or-
der_refl)
  show continuous_on  $(r \ (D - \{b\})) h$ 
  by (simp add: Diff_mono hull_subset continuous_on_subset [OF conth
rsub])
qed
show  $(h \circ r) \ (D - \{b\}) \subseteq \text{rel\_frontier } T$ 
  using brelD him rsub by fastforce
show  $(h \circ r) \ x = h \ x$  if  $x: x \in D \cap \bigcup \mathcal{H}$  for x
proof -
  consider A where  $x \in D \ A \in \mathcal{G} \ x \in A$ 
  | A B where  $x \in D \ A \text{ face\_of } B \ B \in \mathcal{F} \ B \notin \mathcal{G} \ \text{aff\_dim } A < \text{aff\_dim}$ 
T x  $\in A$ 
  using x by (auto simp:  $\mathcal{H\_def}$ )
  then have xrel:  $x \in \text{rel\_frontier } D$ 
  proof cases
    case 1 show ?thesis
    proof (rule face_of_subset_rel_frontier [THEN subsetD])
      show  $D \cap A \text{ face\_of } D$ 

```

```

      using ⟨A ∈ G⟩ ⟨G ⊆ F⟩ face ⟨D ∈ F⟩ by blast
    show D ∩ A ≠ D
      using ⟨A ∈ G⟩ D_not_subset H_def by blast
  qed (auto simp: 1)
next
  case 2 show ?thesis
  proof (rule face_of_subset_rel_frontier [THEN subsetD])
    have D face_of D
      by (simp add: ⟨polyhedron D⟩ polyhedron_imp_convex face_of_refl)
    then show D ∩ A face_of D
      by (meson 2(2) 2(3) ⟨D ∈ F⟩ face' face_of_Int_Int face_of_face)
    show D ∩ A ≠ D
      using 2 D_not_subset H_def by blast
  qed (auto simp: 2)
qed
show ?thesis
  by (simp add: rid xrel)
qed
qed
qed
qed
have clo:  $\bigwedge S. S \in \mathcal{F} \implies \text{closed } S$ 
  by (simp add: poly polytope_imp_closed)
obtain C g where finite C disjoint C (⋃ G) card C ≤ card F continuous_on (⋃ F - C) g
  g ' (⋃ F - C) ⊆ rel_frontier T
  and gh:  $\bigwedge x. x \in (\bigcup \mathcal{F} - C) \cap \bigcup \mathcal{H} \implies g\ x = h\ x$ 
proof (rule extend_map_lemma_cofinite1 [OF ⟨finite F⟩ F clo])
  show X ∩ Y ⊆ ⋃ H if XY: X ∈ F Y ∈ F and  $\neg X \subseteq Y \neg Y \subseteq X$  for X Y
  proof (cases X ∈ G)
    case True
    then show ?thesis
      by (auto simp: H_def)
  next
    case False
    have X ∩ Y ≠ X
      using ⟨ $\neg X \subseteq Y$ ⟩ by blast
    with XY
    show ?thesis
      by (clarsimp simp: H_def)
      (metis Diff_iff Int_iff aff antisym_conv face face_of_aff_dim_lt
face_of_refl
not_le poly polytope_imp_convex)
  qed
qed (blast)+
with ⟨G ⊆ F⟩ show ?thesis
  by (rule_tac C=C and g=g in that) (auto simp: disjoint_def hf [symmetric]
H_def intro!: gh)

```

qed

The next two proofs are similar

theorem *extend_map_cell_complex_to_sphere:*

assumes *finite* $\mathcal{F}$ **and** $S: S \subseteq \bigcup \mathcal{F}$ *closed* S **and** $T: \text{convex } T \text{ bounded } T$

and *poly*: $\bigwedge X. X \in \mathcal{F} \implies \text{polytope } X$

and *aff*: $\bigwedge X. X \in \mathcal{F} \implies \text{aff\_dim } X < \text{aff\_dim } T$

and *face*: $\bigwedge X Y. \llbracket X \in \mathcal{F}; Y \in \mathcal{F} \rrbracket \implies (X \cap Y) \text{ face\_of } X$

and *contf*: *continuous_on* $S f$ **and** *fim*: $f \restriction S \subseteq \text{rel\_frontier } T$

obtains g **where** *continuous_on* $(\bigcup \mathcal{F}) g$

$g \restriction (\bigcup \mathcal{F}) \subseteq \text{rel\_frontier } T \wedge x. x \in S \implies g x = f x$

proof –

obtain $V g$ **where** $S \subseteq V$ *open* V *continuous_on* $V g$ **and** *gim*: $g \restriction V \subseteq \text{rel\_frontier } T$ **and** *gf*: $\bigwedge x. x \in S \implies g x = f x$

using *neighbourhood_extension_into_ANR* [*OF* *contf* *fim* $\langle \text{closed } S \rangle$] *ANR_rel_frontier_convex* T **by** *blast*

have *compact* S

by (*meson* *assms* *compact_Union* *poly* *polytope_imp_compact* *seq_compact_closed_subset* *seq_compact_eq_compact*)

then obtain d **where** $d > 0$ **and** $d: \bigwedge x y. \llbracket x \in S; y \in - V \rrbracket \implies d \leq \text{dist } x y$

using *separate_compact_closed* [*of* $S - V$] $\langle \text{open } V \rangle \langle S \subseteq V \rangle$ **by** *force*

obtain $\mathcal{G}$ **where** *finite* $\mathcal{G}$ $\bigcup \mathcal{G} = \bigcup \mathcal{F}$

and *diaG*: $\bigwedge X. X \in \mathcal{G} \implies \text{diameter } X < d$

and *polyG*: $\bigwedge X. X \in \mathcal{G} \implies \text{polytope } X$

and *affG*: $\bigwedge X. X \in \mathcal{G} \implies \text{aff\_dim } X \leq \text{aff\_dim } T - 1$

and *faceG*: $\bigwedge X Y. \llbracket X \in \mathcal{G}; Y \in \mathcal{G} \rrbracket \implies X \cap Y \text{ face\_of } X$

proof (*rule* *cell_complex_subdivision_exists* [*OF* $\langle d > 0 \rangle \langle \text{finite } \mathcal{F} \rangle$ *poly* face])

show $\bigwedge X. X \in \mathcal{F} \implies \text{aff\_dim } X \leq \text{aff\_dim } T - 1$

by (*simp* *add*: *aff*)

qed *auto*

obtain h **where** *conth*: *continuous_on* $(\bigcup \mathcal{G}) h$ **and** *him*: $h \restriction \bigcup \mathcal{G} \subseteq \text{rel\_frontier } T$ **and** *hg*: $\bigwedge x. x \in \bigcup (\mathcal{G} \cap \text{Pow } V) \implies h x = g x$

proof (*rule* *extend_map_lemma* [*of* $\mathcal{G}$ $\mathcal{G} \cap \text{Pow } V$ $T g$])

show *continuous_on* $(\bigcup (\mathcal{G} \cap \text{Pow } V)) g$

by (*metis* *Union_Int_subset* *Union_Pow_eq* $\langle \text{continuous\_on } V g \rangle$ *continuous_on_subset* *le_inf_iff*)

qed (*use* $\langle \text{finite } \mathcal{G} \rangle$ T *polyG* *affG* *faceG* *gim* **in** *fastforce*) +

show *?thesis*

proof

show *continuous_on* $(\bigcup \mathcal{F}) h$

using $\langle \bigcup \mathcal{G} = \bigcup \mathcal{F} \rangle$ *conth* **by** *auto*

show $h \restriction \bigcup \mathcal{F} \subseteq \text{rel\_frontier } T$

using $\langle \bigcup \mathcal{G} = \bigcup \mathcal{F} \rangle$ *him* **by** *auto*

show $h x = f x$ **if** $x \in S$ **for** x

proof –

have $x \in \bigcup \mathcal{G}$

using $\langle \bigcup \mathcal{G} = \bigcup \mathcal{F} \rangle \langle S \subseteq \bigcup \mathcal{F} \rangle$ **that** **by** *auto*

then obtain X **where** $x \in X$ $X \in \mathcal{G}$ **by** *blast*

then have *diameter* $X < d$ *bounded* X


```

    by (auto simp: diaG  $\langle X \in \mathcal{G} \rangle$  polyG polytope_imp_bounded)
    then have  $X \subseteq V$  using d [OF  $\langle x \in S \rangle$ ] diameter_bounded_bound [OF
 $\langle \text{bounded } X \rangle \langle x \in X \rangle$ ]
    by fastforce
    have  $h\ x = g\ x$ 
    using  $\langle X \in \mathcal{G} \rangle \langle X \subseteq V \rangle \langle x \in X \rangle$  hg by auto
    also have  $\dots = f\ x$ 
    by (simp add: gf that)
    finally show  $h\ x = f\ x$  .
qed
qed
qed

```

theorem *extend_map_cell_complex_to_sphere_cofinite:*

```

    assumes finite  $\mathcal{F}$  and  $S$ :  $S \subseteq \bigcup \mathcal{F}$  closed  $S$  and  $T$ : convex  $T$  bounded  $T$ 
    and poly:  $\bigwedge X. X \in \mathcal{F} \implies \text{polytope } X$ 
    and aff:  $\bigwedge X. X \in \mathcal{F} \implies \text{aff\_dim } X \leq \text{aff\_dim } T$ 
    and face:  $\bigwedge X\ Y. \llbracket X \in \mathcal{F}; Y \in \mathcal{F} \rrbracket \implies (X \cap Y) \text{ face\_of } X$ 
    and contf: continuous_on  $S\ f$  and fim:  $f \restriction S \subseteq \text{rel\_frontier } T$ 
    obtains  $C\ g$  where finite  $C$  disjnt  $C\ S$  continuous_on  $(\bigcup \mathcal{F} - C)\ g$ 
     $g \restriction (\bigcup \mathcal{F} - C) \subseteq \text{rel\_frontier } T$   $\bigwedge x. x \in S \implies g\ x = f\ x$ 
proof -
    obtain  $V\ g$  where  $S \subseteq V$  open  $V$  continuous_on  $V\ g$  and gim:  $g \restriction V \subseteq$ 
    rel_frontier  $T$  and gf:  $\bigwedge x. x \in S \implies g\ x = f\ x$ 
    using neighbourhood_extension_into_ANR [OF contf fim  $\langle \text{closed } S \rangle$ ] ANR_rel_frontier_convex
     $T$  by blast
    have compact  $S$ 
    by (meson assms compact_Union poly polytope_imp_compact seq_compact_closed_subset
    seq_compact_eq_compact)
    then obtain  $d$  where  $d > 0$  and d:  $\bigwedge x\ y. \llbracket x \in S; y \in -\ V \rrbracket \implies d \leq \text{dist } x\ y$ 
    using separate_compact_closed [of  $S - V$ ]  $\langle \text{open } V \rangle \langle S \subseteq V \rangle$  by force
    obtain  $\mathcal{G}$  where finite  $\mathcal{G}$   $\bigcup \mathcal{G} = \bigcup \mathcal{F}$ 
    and diaG:  $\bigwedge X. X \in \mathcal{G} \implies \text{diameter } X < d$ 
    and polyG:  $\bigwedge X. X \in \mathcal{G} \implies \text{polytope } X$ 
    and affG:  $\bigwedge X. X \in \mathcal{G} \implies \text{aff\_dim } X \leq \text{aff\_dim } T$ 
    and faceG:  $\bigwedge X\ Y. \llbracket X \in \mathcal{G}; Y \in \mathcal{G} \rrbracket \implies X \cap Y \text{ face\_of } X$ 
    by (rule cell_complex_subdivision_exists [OF  $\langle d > 0 \rangle \langle \text{finite } \mathcal{F} \rangle$  poly aff face])
    auto
    obtain  $C\ h$  where finite  $C$  and dis: disjnt  $C\ (\bigcup (\mathcal{G} \cap \text{Pow } V))$ 
    and card:  $\text{card } C \leq \text{card } \mathcal{G}$  and conth: continuous_on  $(\bigcup \mathcal{G} - C)\ h$ 
    and him:  $h \restriction (\bigcup \mathcal{G} - C) \subseteq \text{rel\_frontier } T$ 
    and hg:  $\bigwedge x. x \in \bigcup (\mathcal{G} \cap \text{Pow } V) \implies h\ x = g\ x$ 
    proof (rule extend_map_lemma_cofinite [of  $\mathcal{G}\ \mathcal{G} \cap \text{Pow } V\ T\ g$ ])
    show continuous_on  $(\bigcup (\mathcal{G} \cap \text{Pow } V))\ g$ 
    by (metis Union_Int_subset Union_Pow_eq  $\langle \text{continuous\_on } V\ g \rangle$  continu-
    ous_on_subset le_inf_iff)
    show  $g \restriction (\bigcup (\mathcal{G} \cap \text{Pow } V)) \subseteq \text{rel\_frontier } T$ 
    using gim by force

```

```

qed (auto intro: ⟨finite  $\mathcal{G}$ ⟩  $T$  polyG affG dest: faceG)
have Ssub:  $S \subseteq \bigcup (\mathcal{G} \cap \text{Pow } V)$ 
proof
  fix x
  assume  $x \in S$ 
  then have  $x \in \bigcup \mathcal{G}$ 
  using ⟨ $\bigcup \mathcal{G} = \bigcup \mathcal{F}$ ⟩ ⟨ $S \subseteq \bigcup \mathcal{F}$ ⟩ by auto
  then obtain X where  $x \in X$   $X \in \mathcal{G}$  by blast
  then have diameter X < d bounded X
  by (auto simp: diaG ⟨ $X \in \mathcal{G}$ ⟩ polyG polytope_imp_bounded)
  then have  $X \subseteq V$  using d [OF ⟨ $x \in S$ ⟩] diameter_bounded_bound [OF
    ⟨bounded X⟩ ⟨ $x \in X$ ⟩]
  by fastforce
  then show  $x \in \bigcup (\mathcal{G} \cap \text{Pow } V)$ 
  using ⟨ $X \in \mathcal{G}$ ⟩ ⟨ $x \in X$ ⟩ by blast
qed
show ?thesis
proof
  show continuous_on ( $\bigcup \mathcal{F} - C$ ) h
  using ⟨ $\bigcup \mathcal{G} = \bigcup \mathcal{F}$ ⟩ conth by auto
  show h ' ( $\bigcup \mathcal{F} - C$ )  $\subseteq$  rel_frontier T
  using ⟨ $\bigcup \mathcal{G} = \bigcup \mathcal{F}$ ⟩ him by auto
  show h x = f x if  $x \in S$  for x
  proof -
    have h x = g x
    using Ssub hg that by blast
    also have ... = f x
    by (simp add: gf that)
    finally show h x = f x .
  qed
  show disjoint C S
  using dis Ssub by (meson disjoint_iff subset_eq)
qed (intro ⟨finite C⟩)
qed

```

6.42.3 Special cases and corollaries involving spheres

lemma *disjnt_Diff1*: $X \subseteq Y' \implies \text{disjnt } (X - Y) (X' - Y')$
 by (auto simp: disjnt_def)

proposition *extend_map_affine_to_sphere_cofinite_simple*:

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$

assumes compact S convex U bounded U

and aff: $\text{aff\_dim } T \leq \text{aff\_dim } U$

and $S \subseteq T$ and conf: continuous_on S f

and fim: $f ' S \subseteq \text{rel\_frontier } U$

obtains K g where finite K $K \subseteq T$ disjoint K S continuous_on (T - K) g

$g ' (T - K) \subseteq \text{rel\_frontier } U$

$\bigwedge x. x \in S \implies g x = f x$

```

proof -
  have  $\exists K. g. \text{finite } K \wedge \text{disjnt } K \ S \wedge \text{continuous\_on } (T - K) \ g \wedge$ 
     $g \text{ ` } (T - K) \subseteq \text{rel\_frontier } U \wedge (\forall x \in S. g \ x = f \ x)$ 
    if  $\text{affine } T \ S \subseteq T$  and  $\text{aff: } \text{aff\_dim } T \leq \text{aff\_dim } U$  for  $T$ 
  proof (cases  $S = \{\}$ )
    case True
      show ?thesis
      proof (cases  $\text{rel\_frontier } U = \{\}$ )
        case True
          with  $\langle \text{bounded } U \rangle$  have  $\text{aff\_dim } U \leq 0$ 
            using  $\text{affine\_bounded\_eq\_lowdim } \text{rel\_frontier\_eq\_empty}$  by auto
          with  $\text{aff}$  have  $\text{aff\_dim } T \leq 0$  by auto
          then obtain  $a$  where  $T \subseteq \{a\}$ 
            using  $\langle \text{affine } T \rangle \text{affine\_bounded\_eq\_lowdim } \text{affine\_bounded\_eq\_trivial}$  by
            auto
          then show ?thesis
            using  $\langle S = \{\} \rangle \text{fim}$ 
            by (metis  $\text{Diff\_cancel } \text{contf } \text{disjnt\_empty2 } \text{finite.emptyI } \text{finite.insert } \text{finite\_subset}$ )
        next
          case False
            then obtain  $a$  where  $a \in \text{rel\_frontier } U$ 
              by auto
            then show ?thesis
              using  $\text{continuous\_on\_const } [\text{of } a] \langle S = \{\} \rangle$  by force
        qed
      next
        case False
          have  $\text{bounded } S$ 
            by (simp add:  $\langle \text{compact } S \rangle \text{compact\_imp\_bounded}$ )
          then obtain  $b$  where  $b: S \subseteq \text{cbox } (-b) \ b$ 
            using  $\text{bounded\_subset\_cbox\_symmetric}$  by blast
          define  $\text{bbox}$  where  $\text{bbox} \equiv \text{cbox } (-(b+\text{One})) \ (b+\text{One})$ 
          have  $\text{cbox } (-b) \ b \subseteq \text{bbox}$ 
            by (auto simp:  $\text{bbox\_def } \text{algebra\_simps } \text{intro!} \text{subset\_box\_imp}$ )
          with  $b \langle S \subseteq T \rangle$  have  $S \subseteq \text{bbox} \cap T$ 
            by auto
          then have  $S_{\text{sub}}: S \subseteq \bigcup \{\text{bbox} \cap T\}$ 
            by auto
          then have  $\text{aff\_dim } (\text{bbox} \cap T) \leq \text{aff\_dim } U$ 
            by (metis  $\text{aff\_aff\_dim\_subset } \text{inf\_commute } \text{inf\_le1 } \text{order\_trans}$ )
          obtain  $K \ g$  where  $K: \text{finite } K \ \text{disjnt } K \ S$ 
            and  $\text{contg: } \text{continuous\_on } (\bigcup \{\text{bbox} \cap T\} - K) \ g$ 
            and  $\text{gim: } g \text{ ` } (\bigcup \{\text{bbox} \cap T\} - K) \subseteq \text{rel\_frontier } U$ 
            and  $\text{gf: } \bigwedge x. x \in S \implies g \ x = f \ x$ 
          proof (rule  $\text{extend\_map\_cell\_complex\_to\_sphere\_cofinite}$ 
             $[\text{OF } S_{\text{sub}} \text{ ` } \langle \text{convex } U \rangle \langle \text{bounded } U \rangle \text{ ` } \text{contf } \text{fim}]$ )
            show  $\text{closed } S$ 
              using  $\langle \text{compact } S \rangle \text{compact\_eq\_bounded\_closed}$  by auto
          end
        end
      end
    end
  end

```

```

    show poly:  $\bigwedge X. X \in \{bbox \cap T\} \implies \text{polytope } X$ 
    by (simp add: polytope_Int_polyhedron bbox_def polytope_interval affine_imp_polyhedron
    <affine T>)
    show  $\bigwedge X Y. \llbracket X \in \{bbox \cap T\}; Y \in \{bbox \cap T\} \rrbracket \implies X \cap Y \text{ face\_of } X$ 
    by (simp add: poly_face_of_refl polytope_imp_convex)
    show  $\bigwedge X. X \in \{bbox \cap T\} \implies \text{aff\_dim } X \leq \text{aff\_dim } U$ 
    by (simp add: <aff\_dim (bbox  $\cap$  T)  $\leq$  aff\_dim U>)
  qed auto
  define fro where fro  $\equiv \lambda d. \text{frontier}(cbox \text{ } (- (b + d *_{\mathbb{R}} \text{One})) (b + d *_{\mathbb{R}} \text{One}))$ 
  obtain d where d12:  $1/2 \leq d \leq 1$  and dd: disjoint K (fro d)
  proof (rule disjoint_family_elem_disjnt [OF _ <finite K>])
    show infinite {1/2..1::real}
    by (simp add: infinite_Icc)
    have dis1: disjoint (fro x) (fro y) if x<y for x y
    by (auto simp: algebra_simps that subset_box_imp disjoint_Diff1 frontier_def
    fro_def)
    then show disjoint_family_on fro {1/2..1}
    by (auto simp: disjoint_family_on_def disjoint_def neq_iff)
  qed auto
  define c where c  $\equiv b + d *_{\mathbb{R}} \text{One}$ 
  have cbsub: cbox (-b) b  $\subseteq$  box (-c) c cbox (-b) b  $\subseteq$  cbox (-c) c cbox (-c)
  c  $\subseteq$  bbox
    using d12 by (auto simp: algebra_simps subset_box_imp c_def bbox_def)
  have clo_cbT: closed (cbox (-c) c  $\cap$  T)
    by (simp add: affine_closed closed_Int closed_cbox <affine T>)
  have cpT_ne: cbox (-c) c  $\cap$  T  $\neq \{\}$ 
    using <S  $\neq \{\}$ > b cbsub(2) <S  $\subseteq$  T> by fastforce
  have closest_point (cbox (-c) c  $\cap$  T) x  $\notin$  K if x  $\in$  T x  $\notin$  K for x
  proof (cases x  $\in$  cbox (-c) c)
    case True with that show ?thesis
    by (simp add: closest_point_self)
  next
    case False
    have int_ne: interior (cbox (-c) c)  $\cap$  T  $\neq \{\}$ 
      using <S  $\neq \{\}$ > <S  $\subseteq$  T> b <cbox (-b) b  $\subseteq$  box (-c) c> by force
    have convex T
      by (meson <affine T> affine_imp_convex)
    then have x  $\in$  affine hull (cbox (-c) c  $\cap$  T)
      by (metis Int_commute Int_iff <S  $\neq \{\}$ > <S  $\subseteq$  T> cbsub(1) <x  $\in$  T>
      affine_hull_convex_Int_nonempty_interior all_not_in_conv b hull_inc inf.orderE
      interior_cbox)
    then have x  $\in$  affine hull (cbox (-c) c  $\cap$  T) - rel_interior (cbox (-c) c
     $\cap$  T)
      by (meson DiffI False Int_iff rel_interior_subset subsetCE)
    then have closest_point (cbox (-c) c  $\cap$  T) x  $\in$  rel_frontier (cbox (-c) c
     $\cap$  T)
      by (rule closest_point_in_rel_frontier [OF clo_cbT cpT_ne])
  moreover have (rel_frontier (cbox (-c) c  $\cap$  T))  $\subseteq$  fro d
    by (subst convex_affine_rel_frontier_Int [OF _ <affine T> int_ne]) (auto

```

```

simp: fro_def c_def)
  ultimately show ?thesis
    using dd by (force simp: disjnt_def)
qed
then have cpt_subset: closest_point (cbox (- c) c  $\cap$  T) ' (T - K)  $\subseteq \bigcup \{bbox$ 
 $\cap$  T} - K
  using closest_point_in_set [OF clo_cbT cpT_ne] csub(3) by force
show ?thesis
proof (intro conjI ballI exI)
  have continuous_on (T - K) (closest_point (cbox (- c) c  $\cap$  T))
  proof (rule continuous_on_closest_point)
    show convex (cbox (- c) c  $\cap$  T)
      by (simp add: affine_imp_convex convex_Int ⟨affine T⟩)
    show closed (cbox (- c) c  $\cap$  T)
      using clo_cbT by blast
    show cbox (- c) c  $\cap$  T  $\neq \{\}$ 
      using ⟨S  $\neq \{\}$ ⟩ csub(2) b that by auto
  qed
  then show continuous_on (T - K) (g  $\circ$  closest_point (cbox (- c) c  $\cap$  T))
  by (metis continuous_on_compose continuous_on_subset [OF contg cpt_subset])
  have (g  $\circ$  closest_point (cbox (- c) c  $\cap$  T)) ' (T - K)  $\subseteq$  g ' ( $\bigcup \{bbox$ 
- K)
    by (metis image_comp image_mono cpt_subset)
  also have ...  $\subseteq$  rel_frontier U
    by (rule gim)
  finally show (g  $\circ$  closest_point (cbox (- c) c  $\cap$  T)) ' (T - K)  $\subseteq$  rel_frontier
U .
  show (g  $\circ$  closest_point (cbox (- c) c  $\cap$  T)) x = f x if x  $\in$  S for x
  proof -
    have (g  $\circ$  closest_point (cbox (- c) c  $\cap$  T)) x = g x
      unfolding o_def
      by (metis IntI ⟨S  $\subseteq$  T⟩ b csub(2) closest_point_self subset_eq that)
    also have ... = f x
      by (simp add: that gf)
    finally show ?thesis .
  qed
qed
qed (auto simp: K)
qed
then obtain K g where finite K disjnt K S
  and contg: continuous_on (affine hull T - K) g
  and gim: g ' (affine hull T - K)  $\subseteq$  rel_frontier U
  and gf:  $\bigwedge x. x \in S \implies g x = f x$ 
by (metis aff_affine_affine_hull aff_dim_affine_hull
order_trans [OF ⟨S  $\subseteq$  T⟩ hull_subset [of T affine]])
then obtain K g where finite K disjnt K S
  and contg: continuous_on (T - K) g
  and gim: g ' (T - K)  $\subseteq$  rel_frontier U
  and gf:  $\bigwedge x. x \in S \implies g x = f x$ 
by (rule_tac K=K and g=g in that) (auto simp: hull_inc elim: continu-

```

```

ous_on_subset)
  then show ?thesis
    by (rule_tac K=K ∩ T and g=g in that) (auto simp: disjnt_iff Diff_Int
contg)
qed

```

6.42.4 Extending maps to spheres

```

lemma extend_map_affine_to_sphere1:
  fixes f :: 'a::euclidean_space ⇒ 'b::topological_space
  assumes finite K affine U and contf: continuous_on (U - K) f
    and fim: f ` (U - K) ⊆ T
    and comps: ⋀ C. [C ∈ components (U - S); C ∩ K ≠ {}] ⇒ C ∩ L ≠ {}
    and clo: closedin (top_of_set U) S and K: disjnt K S K ⊆ U
  obtains g where continuous_on (U - L) g g ` (U - L) ⊆ T ∧ x. x ∈ S ⇒ g
x = f x
proof (cases K = {})
  case True
  then show ?thesis
    by (metis Diff_empty Diff_subset contf fim continuous_on_subset image_subsetI
rev_image_eqI subset_iff that)
  next
  case False
  have S ⊆ U
    using clo closedin_limpt by blast
  then have (U - S) ∩ K ≠ {}
    by (metis Diff_triv False Int_Diff K disjnt_def inf.absorb_iff2 inf_commute)
  then have ⋃ (components (U - S)) ∩ K ≠ {}
    using Union_components by simp
  then obtain C0 where C0: C0 ∈ components (U - S) C0 ∩ K ≠ {}
    by blast
  have convex U
    by (simp add: affine_imp_convex ⟨affine U⟩)
  then have locally_connected U
    by (rule convex_imp_locally_connected)
  have ∃ a g. a ∈ C ∧ a ∈ L ∧ continuous_on (S ∪ (C - {a})) g ∧
    g ` (S ∪ (C - {a})) ⊆ T ∧ (∀ x ∈ S. g x = f x)
    if C: C ∈ components (U - S) and CK: C ∩ K ≠ {} for C
  proof -
    have C ⊆ U - S C ∩ L ≠ {}
      by (simp_all add: in_components_subset comps that)
    then obtain a where a: a ∈ C a ∈ L by auto
    have opeUC: openin (top_of_set U) C
      proof (rule openin_trans)
        show openin (top_of_set (U - S)) C
          by (simp add: ⟨locally_connected U⟩ clo locally_diff_closed openin_components_locally_connected
[OF _ C])
        show openin (top_of_set U) (U - S)
          by (simp add: clo openin_diff)
      qed
  end

```

```

qed
then obtain d where  $C \subseteq U$   $0 < d$  and  $d$ :  $\text{cball } a \ d \cap U \subseteq C$ 
  using openin_contains_cball by (metis  $\langle a \in C \rangle$ )
then have  $\text{ball } a \ d \cap U \subseteq C$ 
  by auto
obtain h k where homhk: homeomorphism  $(S \cup C) (S \cup C)$  h k
  and subC:  $\{x. (\neg (h \ x = x \wedge k \ x = x))\} \subseteq C$ 
  and bou: bounded  $\{x. (\neg (h \ x = x \wedge k \ x = x))\}$ 
  and hin:  $\bigwedge x. x \in C \cap K \implies h \ x \in \text{ball } a \ d \cap U$ 
proof (rule homeomorphism_grouping_points_exists_gen [of  $C \ \text{ball } a \ d \cap U$ 
 $C \cap K \ S \cup C$ ])
  show openin (top_of_set  $C$ ) ( $\text{ball } a \ d \cap U$ )
    by (metis open_ball  $\langle C \subseteq U \rangle \langle \text{ball } a \ d \cap U \subseteq C \rangle$  inf.absorb_iff2 inf.orderE
inf_assoc open_openin openin_subtopology)
  show openin (top_of_set (affine hull  $C$ ))  $C$ 
    by (metis  $\langle a \in C \rangle \langle \text{openin } (\text{top_of_set } U) \ C \rangle$  affine_hull_eq affine_hull_openin
all_not_in_conv  $\langle \text{affine } U \rangle$ )
  show  $\text{ball } a \ d \cap U \neq \{\}$ 
    using  $\langle 0 < d \rangle \langle C \subseteq U \rangle \langle a \in C \rangle$  by force
  show finite  $(C \cap K)$ 
    by (simp add:  $\langle \text{finite } K \rangle$ )
  show  $S \cup C \subseteq \text{affine hull } C$ 
    by (metis  $\langle C \subseteq U \rangle \langle S \subseteq U \rangle \langle a \in C \rangle$  opeUC affine_hull_eq affine_hull_openin
all_not_in_conv assms(2) sup.bounded_iff)
  show connected  $C$ 
    by (metis  $C$  in_components_connected)
qed auto
have  $a\_BU$ :  $a \in \text{ball } a \ d \cap U$ 
  using  $\langle 0 < d \rangle \langle C \subseteq U \rangle \langle a \in C \rangle$  by auto
have rel_frontier  $(\text{cball } a \ d \cap U)$  retract_of (affine hull  $(\text{cball } a \ d \cap U) - \{a\}$ )
proof (rule rel_frontier_retract_of_punctured_affine_hull)
  show bounded  $(\text{cball } a \ d \cap U)$  convex  $(\text{cball } a \ d \cap U)$ 
    by (auto simp:  $\langle \text{convex } U \rangle$  convex_Int)
  show  $a \in \text{rel\_interior } (\text{cball } a \ d \cap U)$ 
    by (metis  $\langle \text{affine } U \rangle$  convex_cball empty_iff_interior_cball  $a\_BU$  rel_interior_convex_Int_affine)
qed
moreover have rel_frontier  $(\text{cball } a \ d \cap U) = \text{frontier } (\text{cball } a \ d) \cap U$ 
  by (metis  $a\_BU$   $\langle \text{affine } U \rangle$  convex_affine_rel_frontier_Int convex_cball
equals0D interior_cball)
moreover have affine hull  $(\text{cball } a \ d \cap U) = U$ 
  by (metis  $\langle \text{convex } U \rangle$   $a\_BU$  affine_hull_convex_Int_nonempty_interior
affine_hull_eq  $\langle \text{affine } U \rangle$  equals0D inf commute interior_cball)
ultimately have frontier  $(\text{cball } a \ d) \cap U$  retract_of  $(U - \{a\})$ 
  by metis
then obtain r where contr: continuous_on  $(U - \{a\})$  r
  and rim:  $r \restriction (U - \{a\}) \subseteq \text{sphere } a \ d$   $r \restriction (U - \{a\}) \subseteq U$ 
  and req:  $\bigwedge x. x \in \text{sphere } a \ d \cap U \implies r \ x = x$ 
  using  $\langle \text{affine } U \rangle$  by (auto simp: retract_of_def retraction_def hull_same)

```

```

define j where j ≡ λx. if x ∈ ball a d then r x else x
have kj: ⋀x. x ∈ S ⇒ k (j x) = x
  using ⟨C ⊆ U - S⟩ ⟨S ⊆ U⟩ ⟨ball a d ∩ U ⊆ C⟩ j_def subC by auto
have Uaeq: U - {a} = (cball a d - {a}) ∩ U ∪ (U - ball a d)
  using ⟨0 < d⟩ by auto
have jim: j ‘ (S ∪ (C - {a})) ⊆ (S ∪ C) - ball a d
proof clarify
  fix y assume y ∈ S ∪ (C - {a})
  then have y ∈ U - {a}
    using ⟨C ⊆ U - S⟩ ⟨S ⊆ U⟩ ⟨a ∈ C⟩ by auto
  then have r y ∈ sphere a d
    using rim by auto
  then show j y ∈ S ∪ C - ball a d
    unfolding j_def
    using ⟨r y ∈ sphere a d⟩ ⟨y ∈ U - {a}⟩ ⟨y ∈ S ∪ (C - {a})⟩ d rim
  by (metis Diff_iff Int_iff Un_iff subsetD cball_diff_eq sphere image_subset_iff)
qed
have contj: continuous_on (U - {a}) j
  unfolding j_def Uaeq
proof (intro continuous_on_cases_local continuous_on_id, simp_all add: req
closedin_closed Uaeq [symmetric])
  show ∃ T. closed T ∧ (cball a d - {a}) ∩ U = (U - {a}) ∩ T
    using affine_closed ⟨affine U⟩ by (rule_tac x=(cball a d) ∩ U in exI) blast
  show ∃ T. closed T ∧ U - ball a d = (U - {a}) ∩ T
    using ⟨0 < d⟩ ⟨affine U⟩
    by (rule_tac x=U - ball a d in exI) (force simp: affine_closed)
  show continuous_on ((cball a d - {a}) ∩ U) r
    by (force intro: continuous_on_subset [OF contr])
qed
have fT: x ∈ U - K ⇒ f x ∈ T for x
  using fim by blast
show ?thesis
proof (intro conjI exI)
  show continuous_on (S ∪ (C - {a})) (f ∘ k ∘ j)
  proof (intro continuous_on_compose)
    have S ∪ (C - {a}) ⊆ U - {a}
      using ⟨C ⊆ U - S⟩ ⟨S ⊆ U⟩ ⟨a ∈ C⟩ by force
    then show continuous_on (S ∪ (C - {a})) j
      by (rule continuous_on_subset [OF contj])
    have j ‘ (S ∪ (C - {a})) ⊆ S ∪ C
      using jim ⟨C ⊆ U - S⟩ ⟨S ⊆ U⟩ ⟨ball a d ∩ U ⊆ C⟩ j_def by blast
    then show continuous_on (j ‘ (S ∪ (C - {a}))) k
      by (rule continuous_on_subset [OF homeomorphism_cont2 [OF homhk]])
    show continuous_on (k ‘ j ‘ (S ∪ (C - {a}))) f
  proof (clarify intro!: continuous_on_subset [OF contf])
    fix y assume y ∈ S ∪ (C - {a})
    have ky: k y ∈ S ∪ C
      using homeomorphism_image2 [OF homhk] ⟨y ∈ S ∪ (C - {a})⟩ by
blast

```



```

    have jy:  $j\ y \in S \cup C - \text{ball } a\ d$ 
      using  $Un\_iff\ \langle y \in S \cup (C - \{a\}) \rangle\ jim\ by\ auto$ 
    have k  $(j\ y) \in U$ 
      using  $\langle C \subseteq U \rangle\ \langle S \subseteq U \rangle\ homeomorphism\_image2\ [OF\ homhk]\ jy\ by$ 
blast
    moreover have  $k\ (j\ y) \notin K$ 
      using  $K\ unfolding\ disjnt\_iff$ 
    by  $(metis\ DiffE\ Int\_iff\ Un\_iff\ hin\ homeomorphism\_def\ homhk\ image\_eqI\ jy)$ 
    ultimately show  $k\ (j\ y) \in U - K$ 
      by blast
  qed
qed
have  $ST: \bigwedge x. x \in S \implies (f \circ k \circ j)\ x \in T$ 
proof  $(simp\ add: kj)$ 
  show  $\bigwedge x. x \in S \implies f\ x \in T$ 
    using  $K\ unfolding\ disjnt\_iff\ by\ (metis\ DiffI\ \langle S \subseteq U \rangle\ subsetD\ fim\ image\_subset\_iff)$ 
  qed
  moreover have  $(f \circ k \circ j)\ x \in T\ if\ x \in C\ x \neq a\ x \notin S\ for\ x$ 
  proof -
    have  $rx: r\ x \in sphere\ a\ d$ 
      using  $\langle C \subseteq U \rangle\ rim\ that\ by\ fastforce$ 
    have  $jj: j\ x \in S \cup C - \text{ball } a\ d$ 
      using  $jim\ that\ by\ blast$ 
    have  $k\ (j\ x) = j\ x \longrightarrow k\ (j\ x) \in C \vee j\ x \in C$ 
    by  $(metis\ Diff\_iff\ Int\_iff\ Un\_iff\ \langle S \subseteq U \rangle\ subsetD\ d\_j\_def\ jj\ rx\ sphere\_cbal\ that(1))$ 
    then have  $kj: k\ (j\ x) \in C$ 
      using  $homeomorphism\_apply2\ [OF\ homhk,\ of\ j\ x]\ \langle C \subseteq U \rangle\ \langle S \subseteq U \rangle\ a$ 
      rx
    by  $(metis\ (mono\_tags,\ lifting)\ Diff\_iff\ subsetD\ jj\ mem\_Collect\_eq\ subC)$ 
    then show  $?thesis$ 
      by  $(metis\ DiffE\ DiffI\ IntD1\ IntI\ \langle C \subseteq U \rangle\ comp\_apply\ fT\ hin\ homeomor- phism\_apply2\ homhk\ jj\ kj\ subset\_eq)$ 
  qed
  ultimately show  $(f \circ k \circ j)\ '(S \cup (C - \{a\})) \subseteq T$ 
    by force
  show  $\forall x \in S. (f \circ k \circ j)\ x = f\ x\ using\ kj\ by\ simp$ 
qed  $(auto\ simp: a)$ 
qed
then obtain  $a\ h\ where$ 

$$ah: \bigwedge C. \llbracket C \in components\ (U - S); C \cap K \neq \{\} \rrbracket$$


$$\implies a\ C \in C \wedge a\ C \in L \wedge continuous\_on\ (S \cup (C - \{a\}))\ (h\ C) \wedge$$


$$h\ C\ '(S \cup (C - \{a\})) \subseteq T \wedge (\forall x \in S. h\ C\ x = f\ x)$$

using  $that\ by\ metis$ 
define  $F\ where\ F \equiv \{C \in components\ (U - S). C \cap K \neq \{\}\}$ 
define  $G\ where\ G \equiv \{C \in components\ (U - S). C \cap K = \{\}\}$ 
define  $UF\ where\ UF \equiv (\bigcup C \in F. C - \{a\})$ 

```

```

have C0 ∈ F
  by (auto simp: F_def C0)
have finite F
proof (subst finite_image_iff [of λC. C ∩ K F, symmetric])
  show inj_on (λC. C ∩ K) F
    unfolding F_def inj_on_def
    using components_nonoverlap by blast
  show finite ((λC. C ∩ K) ' F)
    unfolding F_def
    by (rule finite_subset [of _ Pow K]) (auto simp: ⟨finite K⟩)
qed
obtain g where contg: continuous_on (S ∪ UF) g
  and gh: ∧x i. [i ∈ F; x ∈ (S ∪ UF) ∩ (S ∪ (i - {a i}))]
    ⇒ g x = h i x
proof (rule pasting_lemma_exists_closed [OF ⟨finite F⟩])
  let ?X = top_of_set (S ∪ UF)
  show topspace ?X ⊆ (⋃ C∈F. S ∪ (C - {a C}))
    using ⟨C0 ∈ F⟩ by (force simp: UF_def)
  show closedin (top_of_set (S ∪ UF)) (S ∪ (C - {a C}))
    if C ∈ F for C
  proof (rule closedin_closed_subset [of U S ∪ C])
    have C ∈ components (U - S)
      using F_def that by blast
    then show closedin (top_of_set U) (S ∪ C)
      by (rule closedin_Un_complement_component [OF ⟨locally connected U⟩
clo])
  next
    have x = a C' if C' ∈ F x ∈ C' x ∉ U for x C'
    proof -
      have ∀ A. x ∈ ⋃ A ∨ C' ∉ A
        using ⟨x ∈ C'⟩ by blast
      with that show x = a C'
        by (metis (lifting) DiffD1 F_def Union_components mem_Collect_eq)
    qed
    then show S ∪ UF ⊆ U
      using ⟨S ⊆ U⟩ by (force simp: UF_def)
  next
    show S ∪ (C - {a C}) = (S ∪ C) ∩ (S ∪ UF)
      using F_def UF_def components_nonoverlap that by auto
    qed
  show continuous_map (subtopology ?X (S ∪ (C' - {a C'}))) euclidean (h C')
if C' ∈ F for C'
proof -
  have C': C' ∈ components (U - S) C' ∩ K ≠ {}
    using F_def that by blast+
  show ?thesis
    using ah [OF C'] by (auto simp: F_def subtopology_subtopology intro:
continuous_on_subset)
qed

```

```

show  $\bigwedge i j x. \llbracket i \in F; j \in F;$ 
 $x \in \text{topspace } ?X \cap (S \cup (i - \{a\ i\})) \cap (S \cup (j - \{a\ j\})) \rrbracket$ 
 $\implies h\ i\ x = h\ j\ x$ 
using components_eq by (fastforce simp: components_eq F_def ah)
qed auto
have  $SU': S \cup \bigcup G \cup (S \cup UF) \subseteq U$ 
using  $\langle S \subseteq U \rangle$  in_components_subset by (auto simp: F_def G_def UF_def)
have clo1: closedin (top_of_set ( $S \cup \bigcup G \cup (S \cup UF)$ )) ( $S \cup \bigcup G$ )
proof (rule closedin_closed_subset [OF _ SU'])
have *:  $\bigwedge C. C \in F \implies \text{openin} (\text{top_of_set } U) C$ 
unfolding F_def
by clarify (metis (no_types, lifting)  $\langle \text{locally connected } U \rangle$  clo closedin_def locally_diff_closed openin_components_locally_connected openin_trans topspace_euclidean_subtopology)
show closedin (top_of_set U) ( $U - UF$ )
unfolding UF_def
by (force intro: openin_delete *)
show  $S \cup \bigcup G = (U - UF) \cap (S \cup \bigcup G \cup (S \cup UF))$ 
using  $\langle S \subseteq U \rangle$  apply (auto simp: F_def G_def UF_def)
apply (metis Diff_iff UnionI Union_components)
apply (metis DiffD1 UnionI Union_components)
by (metis (no_types, lifting) IntI components_nonoverlap empty_iff)
qed
have clo2: closedin (top_of_set ( $S \cup \bigcup G \cup (S \cup UF)$ )) ( $S \cup UF$ )
proof (rule closedin_closed_subset [OF _ SU'])
show closedin (top_of_set U) ( $\bigcup_{C \in F} S \cup C$ )
proof (rule closedin_Union)
show  $\bigwedge T. T \in (\bigcup) S \text{ ' } F \implies \text{closedin} (\text{top_of_set } U) T$ 
using F_def  $\langle \text{locally connected } U \rangle$  clo closedin_Un_complement_component
by blast
qed (simp add: finite F)
show  $S \cup UF = (\bigcup_{C \in F} S \cup C) \cap (S \cup \bigcup G \cup (S \cup UF))$ 
using  $\langle S \subseteq U \rangle$  apply (auto simp: F_def G_def UF_def)
using C0 apply blast
by (metis components_nonoverlap disjoint_iff)
qed
have SUG:  $S \cup \bigcup G \subseteq U - K$ 
using  $\langle S \subseteq U \rangle$  K apply (auto simp: G_def disjoint_iff)
by (meson Diff_iff subsetD in_components_subset)
then have contf': continuous_on ( $S \cup \bigcup G$ ) f
by (rule continuous_on_subset [OF contf])
have contg': continuous_on ( $S \cup UF$ ) g
by (simp add: contg)
have  $\bigwedge x. \llbracket S \subseteq U; x \in S \rrbracket \implies f\ x = g\ x$ 
by (subst gh) (auto simp: ah C0 intro: C0 F)
then have f_eq_g:  $\bigwedge x. x \in S \cup UF \wedge x \in S \cup \bigcup G \implies f\ x = g\ x$ 
using  $\langle S \subseteq U \rangle$  apply (auto simp: F_def G_def UF_def dest: in_components_subset)
using components_eq by blast
have cont: continuous_on ( $S \cup \bigcup G \cup (S \cup UF)$ ) ( $\lambda x. \text{if } x \in S \cup \bigcup G \text{ then } f\ x$ 
 $\text{else } g\ x$ )

```

```

    by (blast intro: continuous_on_cases_local [OF clo1 clo2 contf' contg' f_eq_g,
of  $\lambda x. x \in S \cup \bigcup G$ ])
  show ?thesis
  proof
    have  $UF: \bigcup F - L \subseteq UF$ 
      unfolding F_def UF_def using ah by blast
    have  $U - S - L = \bigcup (\text{components } (U - S)) - L$ 
      by simp
    also have  $\dots = \bigcup F \cup \bigcup G - L$ 
      unfolding F_def G_def by blast
    also have  $\dots \subseteq UF \cup \bigcup G$ 
      using UF by blast
    finally have  $U - L \subseteq S \cup \bigcup G \cup (S \cup UF)$ 
      by blast
    then show continuous_on (U - L) ( $\lambda x. \text{if } x \in S \cup \bigcup G \text{ then } f x \text{ else } g x$ )
      by (rule continuous_on_subset [OF cont])
    have  $((U - L) \cap \{x. x \notin S \wedge (\forall xa \in G. x \notin xa)\}) \subseteq ((U - L) \cap (-S \cap UF))$ 
      using  $\langle U - L \subseteq S \cup \bigcup G \cup (S \cup UF) \rangle$  by auto
    moreover have  $g \text{ ' } ((U - L) \cap (-S \cap UF)) \subseteq T$ 
    proof -
      have  $g x \in T$  if  $x \in U$   $x \notin L$   $x \notin S$   $C \in F$   $x \in C$   $x \neq a$   $C$  for  $x C$ 
    proof (subst gh)
      show  $x \in (S \cup UF) \cap (S \cup (C - \{a\}))$ 
        using that by (auto simp: UF_def)
      show  $h C x \in T$ 
        using ah that by (fastforce simp add: F_def)
    qed (rule that)
    then show ?thesis
      by (force simp: UF_def)
  qed
  ultimately have  $g \text{ ' } ((U - L) \cap \{x. x \notin S \wedge (\forall xa \in G. x \notin xa)\}) \subseteq T$ 
    using image_mono_order_trans by blast
  moreover have  $f \text{ ' } ((U - L) \cap (S \cup \bigcup G)) \subseteq T$ 
    using fim SUG by blast
  ultimately show ( $\lambda x. \text{if } x \in S \cup \bigcup G \text{ then } f x \text{ else } g x$ ) '  $(U - L) \subseteq T$ 
    by force
  show  $\bigwedge x. x \in S \implies (\text{if } x \in S \cup \bigcup G \text{ then } f x \text{ else } g x) = f x$ 
    by (simp add: F_def G_def)
  qed
qed

```

lemma *extend_map_affine_to_sphere2:*

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes compact S convex U bounded U affine T  $S \subseteq T$ 
    and affTU:  $\text{aff\_dim } T \leq \text{aff\_dim } U$ 
    and contf: continuous_on S f
    and fim:  $f \text{ ' } S \subseteq \text{rel\_frontier } U$ 
    and overlap:  $\bigwedge C. C \in \text{components}(T - S) \implies C \cap L \neq \{\}$ 

```

```

    obtains  $K$   $g$  where  $\text{finite } K$   $K \subseteq L$   $K \subseteq T$   $\text{disjnt } K S$ 
       $\text{continuous\_on } (T - K) g$   $g \text{ ' } (T - K) \subseteq \text{rel\_frontier } U$ 
       $\bigwedge x. x \in S \implies g x = f x$ 
proof –
  obtain  $K$   $g$  where  $K: \text{finite } K$   $K \subseteq T$   $\text{disjnt } K S$ 
    and  $\text{contg: continuous\_on } (T - K) g$ 
    and  $\text{gim: } g \text{ ' } (T - K) \subseteq \text{rel\_frontier } U$ 
    and  $\text{gf: } \bigwedge x. x \in S \implies g x = f x$ 
    using  $\text{assms extend\_map\_affine\_to\_sphere\_cofinite\_simple}$  by  $\text{metis}$ 
  have  $(\exists y C. C \in \text{components } (T - S) \wedge x \in C \wedge y \in C \wedge y \in L)$  if  $x \in K$  for
 $x$ 
proof –
  have  $x \in T - S$ 
    using  $\langle K \subseteq T \rangle \langle \text{disjnt } K S \rangle \text{disjnt\_def that}$  by  $\text{fastforce}$ 
  then obtain  $C$  where  $C \in \text{components}(T - S)$   $x \in C$ 
    by  $(\text{metis UnionE Union\_components})$ 
  with  $\text{ovlap [of } C]$  show  $?thesis$ 
    by  $\text{blast}$ 
qed
then obtain  $\xi$  where  $\xi: \bigwedge x. x \in K \implies \exists C. C \in \text{components } (T - S) \wedge x \in$ 
 $C \wedge \xi x \in C \wedge \xi x \in L$ 
    by  $\text{metis}$ 
obtain  $h$  where  $\text{conth: continuous\_on } (T - \xi \text{ ' } K) h$ 
    and  $\text{him: } h \text{ ' } (T - \xi \text{ ' } K) \subseteq \text{rel\_frontier } U$ 
    and  $\text{hg: } \bigwedge x. x \in S \implies h x = g x$ 
proof  $(\text{rule extend\_map\_affine\_to\_sphere1 [OF } \langle \text{finite } K \rangle \langle \text{affine } T \rangle \text{contg gim,}$ 
 $\text{of } S \xi \text{ ' } K])$ 
  show  $\text{cloTS: closedin } (\text{top\_of\_set } T) S$ 
    by  $(\text{simp add: } \langle \text{compact } S \rangle \langle S \subseteq T \rangle \text{closed\_subset compact\_imp\_closed})$ 
  show  $\bigwedge C. \llbracket C \in \text{components } (T - S); C \cap K \neq \{\} \rrbracket \implies C \cap \xi \text{ ' } K \neq \{\}$ 
    using  $\xi \text{ components\_eq}$  by  $\text{blast}$ 
qed  $(\text{use } K \text{ in auto})$ 
show  $?thesis$ 
proof
  show  $*$ :  $\xi \text{ ' } K \subseteq L$ 
    using  $\xi$  by  $\text{blast}$ 
  show  $\text{finite } (\xi \text{ ' } K)$ 
    by  $(\text{simp add: } K)$ 
  show  $\xi \text{ ' } K \subseteq T$ 
    by  $\text{clarify (meson } \xi \text{ Diff\_iff contra\_subsetD in\_components\_subset)}$ 
  show  $\text{continuous\_on } (T - \xi \text{ ' } K) h$ 
    by  $(\text{rule conth})$ 
  show  $\text{disjnt } (\xi \text{ ' } K) S$ 
    using  $K \xi \text{ in\_components\_subset}$  by  $(\text{fastforce simp: disjnt\_def})$ 
qed  $(\text{simp\_all add: him hg gf})$ 
qed

```

proposition $\text{extend\_map\_affine\_to\_sphere\_cofinite\_gen:}$

```

fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
assumes  $SUT: compact\ S\ convex\ U\ bounded\ U\ affine\ T\ S \subseteq T$ 
and  $aff: aff\_dim\ T \leq aff\_dim\ U$ 
and  $contf: continuous\_on\ S\ f$ 
and  $fim: f\ 'S \subseteq rel\_frontier\ U$ 
and  $dis: \bigwedge C. \llbracket C \in components(T - S); bounded\ C \rrbracket \implies C \cap L \neq \{\}$ 
obtains  $K\ g$  where  $finite\ K\ K \subseteq L\ K \subseteq T\ disjoint\ K\ S\ continuous\_on\ (T - K)$ 
 $g$ 
 $g\ ' (T - K) \subseteq rel\_frontier\ U$ 
 $\bigwedge x. x \in S \implies g\ x = f\ x$ 
proof (cases  $S = \{\}$ )
  case  $True$ 
show ?thesis
proof (cases  $rel\_frontier\ U = \{\}$ )
  case  $True$ 
with  $aff$  have  $aff\_dim\ T \leq 0$ 
using  $affine\_bounded\_eq\_lowdim\ \langle bounded\ U \rangle\ order\_trans$ 
by (auto simp add:  $rel\_frontier\_eq\_empty$ )
with  $aff\_dim\_geq\ [of\ T]$  consider  $aff\_dim\ T = -1 \mid aff\_dim\ T = 0$ 
by  $linarith$ 
then show ?thesis
proof cases
  assume  $aff\_dim\ T = -1$ 
then have  $T = \{\}$ 
by (simp add:  $aff\_dim\_empty$ )
then show ?thesis
by (rule_tac  $K=\{\}$  in  $that$ ) auto
next
assume  $aff\_dim\ T = 0$ 
then obtain  $a$  where  $T = \{a\}$ 
using  $aff\_dim\_eq\_0$  by blast
then have  $a \in L$ 
using  $dis\ [of\ \{a\}]\ \langle S = \{\} \rangle$  by (auto simp:  $in\_components\_self$ )
with  $\langle S = \{\} \rangle\ \langle T = \{a\} \rangle$  show ?thesis
by (rule_tac  $K=\{a\}$  and  $g=f$  in  $that$ ) auto
qed
next
case  $False$ 
then obtain  $y$  where  $y \in rel\_frontier\ U$ 
by auto
with  $\langle S = \{\} \rangle$  show ?thesis
by (rule_tac  $K=\{\}$  and  $g=\lambda x. y$  in  $that$ ) (auto)
qed
next
case  $False$ 
have  $bounded\ S$ 
by (simp add:  $assms\ compact\_imp\_bounded$ )
then obtain  $b$  where  $b: S \subseteq cbox\ (-b)\ b$ 
using  $bounded\_subset\_cbox\_symmetric$  by blast

```

```

define  $LU$  where  $LU \equiv L \cup (\bigcup \{C \in \text{components } (T - S). \neg \text{bounded } C\} -$ 
 $\text{cbox } \neg(b+One)) (b+One))$ 
obtain  $K$   $g$  where  $\text{finite } K$   $K \subseteq LU$   $K \subseteq T$   $\text{disjnt } K$   $S$ 
and  $\text{contg: continuous\_on } (T - K)$   $g$ 
and  $\text{gim: } g \text{ ' } (T - K) \subseteq \text{rel\_frontier } U$ 
and  $\text{gf: } \bigwedge x. x \in S \implies g \ x = f \ x$ 
proof ( $\text{rule extend\_map\_affine\_to\_sphere2 } [OF \ SUT \ \text{aff contf fim}]$ )
show  $C \cap LU \neq \{\}$  if  $C \in \text{components } (T - S)$  for  $C$ 
proof ( $\text{cases bounded } C$ )
case  $True$ 
with  $\text{dis}$  that show  $?thesis$ 
unfolding  $LU\_def$  by  $\text{fastforce}$ 
next
case  $False$ 
then have  $\neg \text{bounded } (\bigcup \{C \in \text{components } (T - S). \neg \text{bounded } C\})$ 
by ( $\text{metis } (no\_types, \text{lifting}) \text{Sup\_upper bounded\_subset mem\_Collect\_eq}$ 
 $\text{that})$ 
then show  $?thesis$ 
apply ( $\text{clarsimp simp: } LU\_def \text{Int\_Un\_distrib Diff\_Int\_distrib Int\_UN\_distrib}$ )
by ( $\text{metis } (no\_types, \text{lifting}) \text{False Sup\_upper bounded\_cbox bounded\_subset}$ 
 $\text{inf.orderE mem\_Collect\_eq that})$ 
qed
qed  $\text{blast}$ 
have  $*$ :  $False$  if  $x \in \text{cbox } \neg b - m *_R One) (b + m *_R One)$ 
 $x \notin \text{box } \neg b - n *_R One) (b + n *_R One)$ 
 $0 \leq m$   $m < n$   $n \leq 1$  for  $m$   $n$   $x$ 
using  $\text{that}$  by ( $\text{auto simp: mem\_box algebra\_simps}$ )
have  $\text{disjoint\_family\_on } (\lambda d. \text{frontier } (\text{cbox } \neg b - d *_R One) (b + d *_R One)))$ 
 $\{1 / 2..1\}$ 
by ( $\text{auto simp: disjoint\_family\_on\_def neq\_iff frontier\_def dest: } *$ )
then obtain  $d$  where  $d12: 1/2 \leq d \leq 1$ 
and  $\text{ddis: disjnt } K (\text{frontier } (\text{cbox } \neg(b + d *_R One)) (b + d *_R$ 
 $One)))$ 
using  $\text{disjoint\_family\_elem\_disjnt } [of \ \{1/2..1::\text{real}\} \ K \ \lambda d. \text{frontier } (\text{cbox}$ 
 $\neg(b + d *_R One)) (b + d *_R One))]$ 
by ( $\text{auto simp: } \langle \text{finite } K \rangle$ )
define  $c$  where  $c \equiv b + d *_R One$ 
have  $\text{csub: cbox } \neg b) \ b \subseteq \text{box } \neg c) \ c$ 
 $\text{cbox } \neg b) \ b \subseteq \text{cbox } \neg c) \ c$ 
 $\text{cbox } \neg c) \ c \subseteq \text{cbox } \neg(b+One)) (b+One)$ 
using  $d12$  by ( $\text{simp\_all add: subset\_box } c\_def \text{inner\_diff\_left inner\_left\_distrib}$ )
have  $\text{clo\_cT: closed } (\text{cbox } \neg c) \ c \cap T)$ 
using  $\text{affine\_closed } \langle \text{affine } T \rangle$  by  $\text{blast}$ 
have  $\text{cT\_ne: cbox } \neg c) \ c \cap T \neq \{\}$ 
using  $\langle S \neq \{\} \rangle \ \langle S \subseteq T \rangle \ b \text{csub}$  by  $\text{fastforce}$ 
have  $S\_sub\_cc: S \subseteq \text{cbox } \neg c) \ c$ 
using  $\langle \text{cbox } \neg b) \ b \subseteq \text{cbox } \neg c) \ c \rangle \ b$  by  $\text{auto}$ 
show  $?thesis$ 
proof

```

```

show finite (K ∩ cbox (-(b+One)) (b+One))
  using ⟨finite K⟩ by blast
show K ∩ cbox (-(b + One)) (b + One) ⊆ L
  using ⟨K ⊆ LU⟩ by (auto simp: LU_def)
show K ∩ cbox (-(b + One)) (b + One) ⊆ T
  using ⟨K ⊆ T⟩ by auto
show disjoint (K ∩ cbox (-(b + One)) (b + One)) S
  using ⟨disjoint K S⟩ by (simp add: disjoint_def disjoint_eq_subset_Compl
inf.coboundedI1)
have cloTK: closest_point (cbox (- c) c ∩ T) x ∈ T - K
  if x ∈ T and Knot: x ∈ K ⟶ x ∉ cbox (- b - One) (b + One) for
x
proof (cases x ∈ cbox (- c) c)
  case True
  with ⟨x ∈ T⟩ show ?thesis
    using csub(3) Knot by (force simp: closest_point_self)
  next
  case False
  have clo_in_rf: closest_point (cbox (- c) c ∩ T) x ∈ rel_frontier (cbox (-
c) c ∩ T)
  proof (intro closest_point_in_rel_frontier [OF clo_cT cT_ne] DiffI notI)
    have T ∩ interior (cbox (- c) c) ≠ {}
      using ⟨S ≠ {}⟩ ⟨S ⊆ T⟩ b csub(1) by fastforce
    then show x ∈ affine_hull (cbox (- c) c ∩ T)
      by (simp add: Int_commute affine_hull_affine_Int_nonempty_interior
⟨affine T⟩ hull_inc that(1))
    next
    show False if x ∈ rel_interior (cbox (- c) c ∩ T)
    proof -
      have interior (cbox (- c) c) ∩ T ≠ {}
        using ⟨S ≠ {}⟩ ⟨S ⊆ T⟩ b csub(1) by fastforce
      then have affine_hull (T ∩ cbox (- c) c) = T
        using affine_hull_convex_Int_nonempty_interior [of T cbox (- c) c]
        by (simp add: affine_imp_convex ⟨affine T⟩ inf_commute)
      then show ?thesis
        by (meson subsetD le_inf_iff rel_interior_subset that False)
    qed
  qed
have closest_point (cbox (- c) c ∩ T) x ∉ K
proof
  assume inK: closest_point (cbox (- c) c ∩ T) x ∈ K
  have ∧x. x ∈ K ⟹ x ∉ frontier (cbox (-(b + d *R One)) (b + d *R
One))
    by (metis ddis disjoint_iff)
  then show False
    by (metis DiffI Int_iff ⟨affine T⟩ cT_ne c_def clo_cT clo_in_rf clos-
est_point_in_set
convex_affine_rel_frontier_Int convex_box(1) empty_iff fron-
tier_cbox inK interior_cbox)

```



```

qed
then show ?thesis
  using cT_ne clo_cT closest_point_in_set by blast
qed
show continuous_on (T - K  $\cap$  cbox (- (b + One)) (b + One)) (g  $\circ$  closest_point (cbox (-c) c  $\cap$  T))
  using cloTK
  apply (intro continuous_on_compose continuous_on_closest_point continuous_on_subset [OF contg])
  by (auto simp add: clo_cT affine_imp_convex  $\langle$ affine T $\rangle$  convex_Int cT_ne)
have g (closest_point (cbox (- c) c  $\cap$  T) x)  $\in$  rel_frontier U
  if x  $\in$  T x  $\in$  K  $\longrightarrow$  x  $\notin$  cbox (- b - One) (b + One) for x
  using gim [THEN subsetD] that cloTK by blast
then show (g  $\circ$  closest_point (cbox (- c) c  $\cap$  T)) ' $\{$ (T - K  $\cap$  cbox (- (b + One)) (b + One))  $\subseteq$  rel_frontier U
  by force
show  $\bigwedge x. x \in S \implies (g \circ \text{closest\_point } (\text{cbox } (- c) c \cap T)) x = f x$ 
  by simp (metis (mono_tags, lifting) IntI  $\langle S \subseteq T \rangle$  cT_ne clo_cT closest_point_refl gf subsetD S_sub_cc)
qed
qed

```

```

corollary extend_map_affine_to_sphere_cofinite:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes SUT: compact S affine T S  $\subseteq$  T
    and aff: aff_dim T  $\leq$  DIM('b) and 0  $\leq$  r
    and contf: continuous_on S f
    and fim: f ' $S \subseteq$  sphere a r
    and dis:  $\bigwedge C. \llbracket C \in \text{components}(T - S); \text{bounded } C \rrbracket \implies C \cap L \neq \{\}$ 
  obtains K g where finite K K  $\subseteq$  L K  $\subseteq$  T disjnt K S continuous_on (T - K)
  g
    g ' $(T - K) \subseteq$  sphere a r  $\bigwedge x. x \in S \implies g x = f x$ 
proof (cases r = 0)
  case True
  with fim show ?thesis
    by (rule_tac K= $\{\}$  and g =  $\lambda x. a$  in that) (auto)
next
  case False
  with assms have 0 < r by auto
  then have aff_dim T  $\leq$  aff_dim (cball a r)
    by (simp add: aff_dim_cball)
  then show ?thesis
    apply (rule extend_map_affine_to_sphere_cofinite_gen
      [OF  $\langle$ compact S $\rangle$  convex_cball bounded_cball  $\langle$ affine T $\rangle$   $\langle S \subseteq T \rangle$  _
      contf])
    using fim apply (auto simp: assms False that dest: dis)
    done

```

qed

corollary *extend_map_UNIV_to_sphere_cofinite*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes $\text{DIM}('a) \leq \text{DIM}('b)$ **and** $0 \leq r$
and *compact* S
and *continuous_on* S f
and $f \text{ ` } S \subseteq \text{sphere } a \text{ } r$
and $\bigwedge C. \llbracket C \in \text{components}(- S); \text{bounded } C \rrbracket \Longrightarrow C \cap L \neq \{\}$
obtains K g **where** *finite* K $K \subseteq L$ *disjnt* K S *continuous_on* $(- K)$ g
 $g \text{ ` } (- K) \subseteq \text{sphere } a \text{ } r \bigwedge x. x \in S \Longrightarrow g \text{ } x = f \text{ } x$
using *extend_map_affine_to_sphere_cofinite*
 $[OF \text{ ` } \langle \text{compact } S \rangle \text{ affine\_UNIV subset\_UNIV}] \text{ assms}$
by (*metis* *Compl_eq_Diff_UNIV* *aff_dim_UNIV* *of_nat_le_iff*)

corollary *extend_map_UNIV_to_sphere_no_bounded_component*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes *aff*: $\text{DIM}('a) \leq \text{DIM}('b)$ **and** $0 \leq r$
and *SUT*: *compact* S
and *contf*: *continuous_on* S f
and *fim*: $f \text{ ` } S \subseteq \text{sphere } a \text{ } r$
and *dis*: $\bigwedge C. C \in \text{components}(- S) \Longrightarrow \neg \text{bounded } C$
obtains g **where** *continuous_on* $\text{UNIV } g$ $g \text{ ` } \text{UNIV} \subseteq \text{sphere } a \text{ } r \bigwedge x. x \in S \Longrightarrow$
 $g \text{ } x = f \text{ } x$
apply (*rule* *extend_map_UNIV_to_sphere_cofinite* [*OF* *aff* $\langle 0 \leq r \rangle$ $\langle \text{compact } S \rangle$ *contf* *fim*, *of* $\{\}$])
apply (*auto* *dest*: *dis*)
done

theorem *Borsuk_separation_theorem_gen*:
fixes $S :: 'a::\text{euclidean\_space}$ *set*
assumes *compact* S
shows $(\forall c \in \text{components}(- S). \neg \text{bounded } c) \longleftrightarrow$
 $(\forall f. \text{continuous\_on } S \text{ } f \wedge f \text{ ` } S \subseteq \text{sphere } (0::'a) \text{ } 1$
 $\longrightarrow (\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) \text{ } S \text{ (sphere } 0 \text{ } 1) \text{ } f (\lambda x.$
 $c)))$
(is $?lhs = ?rhs)$

proof
assume L [*rule_format*]: $?lhs$
show $?rhs$
proof *clarify*
fix $f :: 'a \Rightarrow 'a$
assume *contf*: *continuous_on* S f **and** *fim*: $f \text{ ` } S \subseteq \text{sphere } 0 \text{ } 1$
obtain g **where** *contg*: *continuous_on* $\text{UNIV } g$ **and** *gim*: *range* $g \subseteq \text{sphere } 0 \text{ } 1$
and *gf*: $\bigwedge x. x \in S \Longrightarrow g \text{ } x = f \text{ } x$
by (*rule* *extend_map_UNIV_to_sphere_no_bounded_component* [*OF* $\langle \text{compact } S \rangle$ *contf* *fim* L]) *auto*
then obtain c **where** c : *homotopic_with_canon* $(\lambda h. \text{True}) \text{ } \text{UNIV (sphere } 0 \text{ } 1) \text{ } g (\lambda x. c)$

```

    using contractible_UNIV nullhomotopic_from_contractible by blast
  then show  $\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S (\text{sphere } 0 \ 1) f (\lambda x. c)$ 
    by (metis assms compact_imp_closed contf contg contractible_empty fim gf
    gim nullhomotopic_from_contractible nullhomotopic_into_sphere_extension)
  qed
next
  assume R [rule_format]: ?rhs
  show ?lhs
    unfolding components_def
  proof clarify
    fix a
    assume  $a \notin S$  and a: bounded (connected_component_set  $(- S)$  a)
    have  $\forall x \in S. \text{norm } (x - a) \neq 0$ 
      using  $\langle a \notin S \rangle$  by auto
    then have cont: continuous_on S  $(\lambda x. \text{inverse}(\text{norm}(x - a)) *_{\mathbb{R}} (x - a))$ 
      by (intro continuous_intros)
    have im:  $(\lambda x. \text{inverse}(\text{norm}(x - a)) *_{\mathbb{R}} (x - a)) ' S \subseteq \text{sphere } 0 \ 1$ 
      by clarsimp (metis  $\langle a \notin S \rangle$  eq_iff_diff_eq_0 left_inverse_norm_eq_zero)
    show False
      using R cont im Borsuk_map_essential_bounded_component [OF  $\langle \text{compact } S \rangle \langle a \notin S \rangle$ ] a by blast
  qed
qed

```

corollary Borsuk_separation_theorem:

```

  fixes S :: 'a::euclidean_space set
  assumes compact S and 2:  $2 \leq \text{DIM}('a)$ 
  shows connected  $(- S) \longleftrightarrow$ 
    ( $\forall f. \text{continuous\_on } S f \wedge f ' S \subseteq \text{sphere } (0::'a) \ 1$ 
       $\longrightarrow (\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S (\text{sphere } 0 \ 1) f (\lambda x.$ 
c)))
    (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  show ?rhs
    proof (cases  $S = \{\}$ )
      case True
      then show ?thesis by auto
    next
      case False
      then have  $(\forall c \in \text{components } (- S). \neg \text{bounded } c)$ 
        by (metis L assms(1) bounded_empty cobounded_imp_unbounded compact_imp_bounded in_components_maximal order_refl)
      then show ?thesis
        by (simp add: Borsuk_separation_theorem_gen [OF  $\langle \text{compact } S \rangle$ ])
    qed
next
  assume R: ?rhs

```

```

then show ?lhs
  apply (simp add: Borsuk_separation_theorem_gen [OF ⟨compact S⟩, symmetric])
  apply (auto simp: components_def connected_iff_eq_connected_component_set)
  using connected_component_in apply fastforce
  using cobounded_unique_unbounded_component [OF _ 2, of -S] ⟨compact S⟩ compact_eq_bounded_closed by fastforce
qed

```

```

lemma homotopy_eqv_separation:
  fixes S :: 'a::euclidean_space set and T :: 'a set
  assumes S homotopy_eqv T and compact S and compact T
  shows connected(- S)  $\longleftrightarrow$  connected(- T)
proof -
  consider DIM('a) = 1 | 2 ≤ DIM('a)
  by (metis DIM_ge_Suc0 One_nat_def Suc_1 dual_order.antisym not_less_eq_eq)
  then show ?thesis
  proof cases
    case 1
    then show ?thesis
    using bounded_connected_Compl_1 compact_imp_bounded homotopy_eqv_empty1
    homotopy_eqv_empty2 assms by metis
  next
    case 2
    with assms show ?thesis
    by (simp add: Borsuk_separation_theorem homotopy_eqv_cohomotopic_triviality_null)
  qed
qed

```

```

proposition Jordan_Brouwer_separation:
  fixes S :: 'a::euclidean_space set and a::'a
  assumes hom: S homeomorphic sphere a r and 0 < r
  shows ¬ connected(- S)
proof -
  have - sphere a r ∩ ball a r ≠ {}
    using ⟨0 < r⟩ by (simp add: Int_absorb1 subset_eq)
  moreover
  have eq: - sphere a r - ball a r = - cball a r
    by auto
  have - cball a r ≠ {}
  proof -
    have frontier (cball a r) ≠ {}
      using ⟨0 < r⟩ by auto
    then show ?thesis
      by (metis frontier_complement frontier_empty)
  qed
with eq have - sphere a r - ball a r ≠ {}
  by auto

```

```

moreover
have connected ( $- S$ ) = connected ( $- sphere\ a\ r$ )
proof (rule homotopy_eqv_separation)
  show S homotopy_eqv sphere a r
    using hom homeomorphic_imp_homotopy_eqv by blast
  show compact (sphere a r)
    by simp
  then show compact S
    using hom homeomorphic_compactness by blast
qed
ultimately show ?thesis
  using connected_Int_frontier [of  $- sphere\ a\ r\ ball\ a\ r$ ] by (auto simp:  $\langle 0 < r \rangle$ )
qed

proposition Jordan_Brouwer_frontier:
  fixes S :: 'a::euclidean_space set and a::'a
  assumes S: S homeomorphic sphere a r and T: T ∈ components(- S) and 2:
  2 ≤ DIM('a)
  shows frontier T = S
proof (cases r rule: linorder_cases)
  assume r < 0
  with S T show ?thesis by auto
next
  assume r = 0
  with S T card_eq_SucD obtain b where S = {b}
  by (auto simp: homeomorphic_finite [of  $\{a\}\ S$ ])
  have components (- {b}) = { -{b}}
  using T  $\langle S = \{b\} \rangle$  by (auto simp: components_eq_sing_iff connected_punctured_universe
  2)
  with T show ?thesis
  by (metis  $\langle S = \{b\} \rangle$  cball_trivial frontier_cball frontier_complement singletonD
sphere_trivial)
next
  assume r > 0
  have compact S
  using homeomorphic_compactness compact_sphere S by blast
show ?thesis
proof (rule frontier_minimal_separating_closed)
  show closed S
  using  $\langle compact\ S \rangle$  compact_eq_bounded_closed by blast
  show  $\neg connected\ (-\ S)$ 
  using Jordan_Brouwer_separation S  $\langle 0 < r \rangle$  by blast
  obtain f g where hom: homeomorphism S (sphere a r) f g
  using S by (auto simp: homeomorphic_def)
  show connected (- T) if closed T T ⊆ S for T
proof -
  have f ' T ⊆ sphere a r

```

```

    using ⟨T ⊂ S⟩ hom homeomorphism_image1 by blast
  moreover have f ' T ≠ sphere a r
    using ⟨T ⊂ S⟩ hom
    by (metis homeomorphism_image2 homeomorphism_of_subsets order_refl
psubsetE)
  ultimately have f ' T ⊂ sphere a r by blast
  then have connected (− f ' T)
    by (rule psubset_sphere_Compl_connected [OF _ ⟨0 < r⟩ 2])
  moreover have compact T
    using ⟨compact S⟩ bounded_subset compact_eq_bounded_closed that by
blast
  moreover then have compact (f ' T)
    by (meson compact_continuous_image continuous_on_subset hom homeo-
morphism_def psubsetE ⟨T ⊂ S⟩)
  moreover have T homotopy_eqv f ' T
    by (meson ⟨f ' T ⊆ sphere a r⟩ dual_order.strict_implies_order hom homeo-
morphic_def homeomorphic_imp_homotopy_eqv homeomorphism_of_subsets ⟨T
⊂ S⟩)
  ultimately show ?thesis
    using homotopy_eqv_separation [of T f'T] by blast
qed
qed (rule T)
qed

proposition Jordan_Brouwer_nonseparation:
  fixes S :: 'a::euclidean_space set and a::'a
  assumes S: S homeomorphic sphere a r and T ⊂ S and 2: 2 ≤ DIM('a)
  shows connected(− T)
proof −
  have *: connected(C ∪ (S − T)) if C ∈ components(− S) for C
  proof (rule connected_intermediate_closure)
    show connected C
      using in_components_connected that by auto
    have S = frontier C
      using 2 Jordan_Brouwer_frontier S that by blast
    with closure_subset show C ∪ (S − T) ⊆ closure C
      by (auto simp: frontier_def)
  qed auto
  have components(− S) ≠ {}
    by (metis S bounded_empty cobounded_imp_unbounded compact_eq_bounded_closed
compact_sphere
      components_eq_empty homeomorphic_compactness)
  then have − T = (⋃ C ∈ components(− S). C ∪ (S − T))
    using Union_components [of −S] ⟨T ⊂ S⟩ by auto
  moreover have connected ...
    using ⟨T ⊂ S⟩ by (intro connected_Union) (auto simp: *)
  ultimately show ?thesis
    by simp
qed

```

6.42.5 Invariance of domain and corollaries

```

lemma invariance_of_domain_ball:
  fixes f :: 'a  $\Rightarrow$  'a::euclidean_space
  assumes contf: continuous_on (cball a r) f and 0 < r
    and inj: inj_on f (cball a r)
    shows open(f ' ball a r)
proof (cases DIM('a) = 1)
  case True
  obtain h::'a $\Rightarrow$ real and k
    where linear h linear k h ' UNIV = UNIV k ' UNIV = UNIV
       $\bigwedge x. \text{norm}(h\ x) = \text{norm}\ x \bigwedge x. \text{norm}(k\ x) = \text{norm}\ x$ 
      and kh:  $\bigwedge x. k(h\ x) = x$  and  $\bigwedge x. h(k\ x) = x$ 
  proof (rule isomorphisms_UNIV_UNIV)
    show DIM('a) = DIM(real)
      using True by force
  qed (metis UNIV_I UNIV_eq_I imageI)
  have cont: continuous_on S h continuous_on T k for S T
    by (simp_all add: 'linear h' 'linear k' linear_continuous_on linear_linear)
  have continuous_on (h ' cball a r) (h  $\circ$  f  $\circ$  k)
    by (intro continuous_on_compose cont continuous_on_subset [OF contf])
  (auto simp: kh)
  moreover have is_interval (h ' cball a r)
    by (simp add: is_interval_connected_1 'linear h' linear_continuous_on
linear_linear connected_continuous_image)
  moreover have inj_on (h  $\circ$  f  $\circ$  k) (h ' cball a r)
    using inj by (simp add: inj_on_def) (metis '  $\bigwedge x. k(h\ x) = x$  ')
  ultimately have *:  $\bigwedge T. [\text{open } T; T \subseteq h\ ' \text{ cball } a\ r] \Longrightarrow \text{open } ((h \circ f \circ k)\ ' \text{ ball } a\ r)$ 
  T)
    using injective_eq_1d_open_map_UNIV by blast
  have open ((h  $\circ$  f  $\circ$  k) ' (h ' ball a r))
    by (rule *) (auto simp: 'linear h' 'range h = UNIV' open_surjective_linear_image)
  then have open ((h  $\circ$  f) ' ball a r)
    by (simp add: image_comp '  $\bigwedge x. k(h\ x) = x$  ' cong: image_cong)
  then show ?thesis
    unfolding image_comp [symmetric]
    by (metis open_bijective_linear_image_eq 'linear h' kh 'range h = UNIV'
bijI inj_on_def)
next
  case False
  then have 2: DIM('a)  $\geq$  2
    by (metis DIM_ge_Suc0 One_nat_def Suc_1 antisym not_less_eq_eq)
  have fmsub: f ' ball a r  $\subseteq$  - f ' sphere a r
    using inj by clarsimp (metis inj_onD less_eq_real_def mem_cball order_less_irrefl)
  have hom: f ' sphere a r homeomorphic sphere a r
    by (meson compact_sphere contf continuous_on_subset homeomorphic_compact
homeomorphic_sym inj inj_on_subset sphere_cball)
  then have nconn:  $\neg$  connected (- f ' sphere a r)
    by (rule Jordan_Brouwer_separation) (auto simp: '0 < r')
  have bounded (f ' sphere a r)

```

```

    by (meson compact_imp_bounded compact_continuous_image_eq compact_sphere
contf inj sphere_cball)
  then obtain C where C: C ∈ components (− f ‘ sphere a r) and bounded C
    using cobounded_has_bounded_component [OF _ nconn] 2 by auto
  moreover have f ‘ (ball a r) = C
  proof
    have C ≠ {}
      by (rule in_components_nonempty [OF C])
    show C ⊆ f ‘ ball a r
    proof (rule ccontr)
      assume nonsub: ¬ C ⊆ f ‘ ball a r
      have − f ‘ cball a r ⊆ C
      proof (rule components_maximal [OF C])
        have f ‘ cball a r homeomorphic cball a r
          using compact_cball contf homeomorphic_compact homeomorphic_sym
inj by blast
        then show connected (− f ‘ cball a r)
          by (auto intro: connected_complement_homeomorphic_convex_compact
2)
        show − f ‘ cball a r ⊆ − f ‘ sphere a r
          by auto
        then show C ∩ − f ‘ cball a r ≠ {}
          using ⟨C ≠ {}⟩ in_components_subset [OF C] nonsub
          using image_iff by fastforce
        qed
        then have bounded (− f ‘ cball a r)
          using bounded_subset ⟨bounded C⟩ by auto
        then have ¬ bounded (f ‘ cball a r)
          using cobounded_imp_unbounded by blast
        then show False
          using compact_continuous_image [OF contf] compact_cball compact_imp_bounded
by blast
        qed
        with ⟨C ≠ {}⟩ have C ∩ f ‘ ball a r ≠ {}
          by (simp add: inf.absorb_iff1)
        then show f ‘ ball a r ⊆ C
          by (metis components_maximal [OF C _ fmsub] connected_continuous_image
ball_subset_cball connected_ball contf_continuous_on_subset)
        qed
        moreover have open (− f ‘ sphere a r)
          using hom_compact_eq_bounded_closed compact_sphere homeomorphic_compactness
by blast
        ultimately show ?thesis
          using open_components by blast
      qed
    qed
  qed

```

Proved by L. E. J. Brouwer (1912)

theorem *invariance_of_domain*:
 fixes f :: 'a ⇒ 'a::euclidean_space


```

    assumes continuous_on S f open S inj_on f S
    shows open(f ' S)
    unfolding open_subopen [of f'S]
  proof clarify
    fix a
    assume a ∈ S
    obtain δ where δ > 0 and δ: cball a δ ⊆ S
    using ⟨open S⟩ ⟨a ∈ S⟩ open_contains_cball_eq by blast
    show ∃ T. open T ∧ f a ∈ T ∧ T ⊆ f ' S
    proof (intro exI conjI)
      show open (f ' (ball a δ))
      by (meson δ > 0 < δ) assms continuous_on_subset inj_on_subset invari-
ance_of_domain_ball)
      show f a ∈ f ' ball a δ
      by (simp add: ⟨0 < δ⟩)
      show f ' ball a δ ⊆ f ' S
      using δ ball_subset_cball by blast
    qed
  qed

lemma inv_of_domain_ss0:
  fixes f :: 'a ⇒ 'a::euclidean_space
  assumes contf: continuous_on U f and injf: inj_on f U and fim: f ' U ⊆ S
    and subspace S and dimS: dim S = DIM('b::euclidean_space)
    and ope: openin (top_of_set S) U
  shows openin (top_of_set S) (f ' U)
  proof -
    have U ⊆ S
    using ope openin_imp_subset by blast
    have (UNIV::'b set) homeomorphic S
    by (simp add: ⟨subspace S⟩ dimS homeomorphic_subspaces)
    then obtain h k where homhk: homeomorphism (UNIV::'b set) S h k
    using homeomorphic_def by blast
    have homkh: homeomorphism S (k ' S) k h
    using homhk homeomorphism_image2 homeomorphism_sym by fastforce
    have open ((k ∘ f ∘ h) ' k ' U)
    proof (rule invariance_of_domain)
      show continuous_on (k ' U) (k ∘ f ∘ h)
      proof (intro continuous_intros)
        show continuous_on (k ' U) h
        by (meson continuous_on_subset [OF homeomorphism_cont1 [OF homhk]]
top_greatest)
        have h ' k ' U ⊆ U
        by (metis ⟨U ⊆ S⟩ dual_order.eq_iff homeomorphism_image2 homeomor-
phism_of_subsets homkh)
      then show continuous_on (h ' k ' U) f
      by (rule continuous_on_subset [OF contf])
    have f ' h ' k ' U ⊆ S
    using ⟨h ' k ' U ⊆ U⟩ fim by blast

```

```

    then show continuous_on (f ' h ' k ' U) k
    by (rule continuous_on_subset [OF homeomorphism_cont2 [OF homhk]])
  qed
  have ope_iff:  $\bigwedge T. \text{open } T \longleftrightarrow \text{openin } (\text{top\_of\_set } (k ' S)) T$ 
    using homhk homeomorphism_image2 open_openin by fastforce
  show open (k ' U)
    by (simp add: ope_iff homeomorphism_imp_open_map [OF homkh ope])
  show inj_on (k  $\circ$  f  $\circ$  h) (k ' U)
    apply (clarsimp simp: inj_on_def)
    by (metis  $\langle U \subseteq S \rangle$  fim homeomorphism_apply2 homhk image_subset_iff
inj_onD injf subsetD)
  qed
  moreover
  have eq: f ' U = h ' (k  $\circ$  f  $\circ$  h  $\circ$  k) ' U
    unfolding image_comp [symmetric] using  $\langle U \subseteq S \rangle$  fim
    by (metis homeomorphism_image2 homeomorphism_of_subsets homkh subset_image_iff)
  ultimately show ?thesis
    by (metis (no_types, opaque_lifting) homeomorphism_imp_open_map homhk
image_comp open_openin subtopology_UNIV)
  qed

lemma inv_of_domain_ss1:
  fixes f :: 'a  $\Rightarrow$  'a::euclidean_space
  assumes contf: continuous_on U f and injf: inj_on f U and fim: f ' U  $\subseteq$  S
    and subspace S
    and ope: openin (top_of_set S) U
  shows openin (top_of_set S) (f ' U)
proof -
  define S' where S'  $\equiv$  {y.  $\forall x \in S. \text{orthogonal } x y$ }
  have subspace S'
    by (simp add: S'_def subspace_orthogonal_to_vectors)
  define g where g  $\equiv$   $\lambda z::'a * 'a. ((f \circ \text{fst})z, \text{snd } z)$ 
  have openin (top_of_set (S  $\times$  S')) (g ' (U  $\times$  S'))
  proof (rule inv_of_domain_ss0)
    show continuous_on (U  $\times$  S') g
      unfolding g_def
      by (auto intro!: continuous_intros continuous_on_compose2 [OF contf continuous_on_fst])
    show g ' (U  $\times$  S')  $\subseteq$  S  $\times$  S'
      using fim by (auto simp: g_def)
    show inj_on g (U  $\times$  S')
      using injf by (auto simp: g_def inj_on_def)
    show subspace (S  $\times$  S')
      by (simp add:  $\langle \text{subspace } S' \rangle \langle \text{subspace } S \rangle \text{subspace\_Times}$ )
    show openin (top_of_set (S  $\times$  S')) (U  $\times$  S')
      by (simp add: openin_Times [OF ope])
    have dim (S  $\times$  S') = dim S + dim S'
      by (simp add:  $\langle \text{subspace } S' \rangle \langle \text{subspace } S \rangle \text{dim\_Times}$ )
  end

```

```

    also have ... = DIM('a)
    using dim_subspace_orthogonal_to_vectors [OF ‹subspace S› subspace_UNIV]
    by (simp add: add.commute S'_def)
    finally show dim (S × S') = DIM('a) .
qed
moreover have g ' (U × S') = f ' U × S'
  by (auto simp: g_def image_iff)
moreover have 0 ∈ S'
  using ‹subspace S'› subspace_affine by blast
ultimately show ?thesis
  by (auto simp: openin_Times_eq)
qed

corollary invariance_of_domain_subspaces:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes ope: openin (top_of_set U) S
    and subspace U subspace V and VU: dim V ≤ dim U
    and contf: continuous_on S f and fim: f ' S ⊆ V
    and injf: inj_on f S
  shows openin (top_of_set V) (f ' S)
proof -
  obtain V' where subspace V' V' ⊆ U dim V' = dim V
    using choose_subspace_of_subspace [OF VU]
    by (metis span_eq_iff ‹subspace U›)
  then have V homeomorphic V'
    by (simp add: ‹subspace V› homeomorphic_subspaces)
  then obtain h k where homhk: homeomorphism V V' h k
    using homeomorphic_def by blast
  have eq: f ' S = k ' (h ∘ f) ' S
  proof -
    have k ' h ' f ' S = f ' S
    by (meson fim homeomorphism_def homeomorphism_of_subsets homhk subset_refl)
    then show ?thesis
    by (simp add: image_comp)
  qed
  show ?thesis
  unfolding eq
proof (rule homeomorphism_imp_open_map)
  show homkh: homeomorphism V' V k h
    by (simp add: homeomorphism_symD homhk)
  have hfV': (h ∘ f) ' S ⊆ V'
    using fim homeomorphism_image1 homhk by fastforce
  moreover have openin (top_of_set U) ((h ∘ f) ' S)
  proof (rule inv_of_domain_ss1)
    show continuous_on S (h ∘ f)
    by (meson contf continuous_on_compose continuous_on_subset fim homeomorphism_cont1 homhk)
  qed

```

```

    show inj_on (h ∘ f) S
    apply (clarsimp simp: inj_on_def)
    by (metis fin homeomorphism_apply2 [OF homkh] image_subset_iff
inj_onD injf)
    show (h ∘ f) ' S ⊆ U
    using ⟨V' ⊆ U⟩ hfV' by auto
    qed (auto simp: assms)
    ultimately show openin (top_of_set V') ((h ∘ f) ' S)
    using openin_subset_trans ⟨V' ⊆ U⟩ by force
  qed
qed

corollary invariance_of_dimension_subspaces:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes ope: openin (top_of_set U) S
    and subspace U subspace V
    and contf: continuous_on S f and fin: f ' S ⊆ V
    and injf: inj_on f S and S ≠ {}
  shows dim U ≤ dim V
proof -
  have False if dim V < dim U
  proof -
    obtain T where subspace T T ⊆ U dim T = dim V
    using choose_subspace_of_subspace [of dim V U]
    by (metis ⟨dim V < dim U⟩ assms(2) order.strict_implies_order_span_eq_iff)
    then have V homeomorphic T
    by (simp add: ⟨subspace V⟩ homeomorphic_subspaces)
    then obtain h k where homhk: homeomorphism V T h k
    using homeomorphic_def by blast
    have continuous_on S (h ∘ f)
    by (meson contf continuous_on_compose continuous_on_subset fin homeomorphism_cont1 homhk)
    moreover have (h ∘ f) ' S ⊆ U
    using ⟨T ⊆ U⟩ fin homeomorphism_image1 homhk by fastforce
    moreover have inj_on (h ∘ f) S
    apply (clarsimp simp: inj_on_def)
    by (metis fin homeomorphism_apply1 homhk image_subset_iff inj_onD injf)
    ultimately have ope_hf: openin (top_of_set U) ((h ∘ f) ' S)
    using invariance_of_domain_subspaces [OF ope ⟨subspace U⟩ ⟨subspace U⟩]
  by blast
  have (h ∘ f) ' S ⊆ T
  using fin homeomorphism_image1 homhk by fastforce
  then have dim ((h ∘ f) ' S) ≤ dim T
  by (rule dim_subset)
  also have dim ((h ∘ f) ' S) = dim U
  using ⟨S ≠ {}⟩ ⟨subspace U⟩
  by (blast intro: dim_openin_ope_hf)
  finally show False
  using ⟨dim V < dim U⟩ ⟨dim T = dim V⟩ by simp

```

```

qed
then show ?thesis
  using not_less by blast
qed

corollary invariance_of_domain_affine_sets:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes ope: openin (top_of_set U) S
    and aff: affine U affine V aff_dim V  $\leq$  aff_dim U
    and conf: continuous_on S f and fim: f ' S  $\subseteq$  V
    and injf: inj_on f S
  shows openin (top_of_set V) (f ' S)
proof (cases S = {})
  case True
  then show ?thesis by auto
next
  case False
  obtain a b where a  $\in$  S a  $\in$  U b  $\in$  V
  using False fim ope openin_contains_cball by fastforce
  have openin (top_of_set ((+) (- b) ' V)) (((+) (- b)  $\circ$  f  $\circ$  (+) a) ' (+) (- a) ' S)
  proof (rule invariance_of_domain_subspaces)
    show openin (top_of_set ((+) (- a) ' U)) ((+) (- a) ' S)
    by (metis ope homeomorphism_imp_open_map homeomorphism_translation translation_galois)
    show subspace ((+) (- a) ' U)
    by (simp add: 'a  $\in$  U' affine_diffs_subspace_subtract 'affine U' cong: image_cong_simp)
    show subspace ((+) (- b) ' V)
    by (simp add: 'b  $\in$  V' affine_diffs_subspace_subtract 'affine V' cong: image_cong_simp)
    show dim ((+) (- b) ' V)  $\leq$  dim ((+) (- a) ' U)
    by (metis 'a  $\in$  U' 'b  $\in$  V' aff_dim_eq_dim affine_hull_eq_aff_of_nat_le_iff)
    show continuous_on ((+) (- a) ' S) ((+) (- b)  $\circ$  f  $\circ$  (+) a)
    by (metis conf continuous_on_compose homeomorphism_cont2 homeomorphism_translation translation_galois)
    show ((+) (- b)  $\circ$  f  $\circ$  (+) a) ' (+) (- a) ' S  $\subseteq$  (+) (- b) ' V
    using fim by auto
    show inj_on ((+) (- b)  $\circ$  f  $\circ$  (+) a) ((+) (- a) ' S)
    by (auto simp: inj_on_def) (meson inj_onD injf)
  qed
  then show ?thesis
  by (metis (no_types, lifting) homeomorphism_imp_open_map homeomorphism_translation image_comp translation_galois)
qed

corollary invariance_of_dimension_affine_sets:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes ope: openin (top_of_set U) S

```

```

    and aff: affine U affine V
    and conf: continuous_on S f and fim: f ' S ⊆ V
    and injf: inj_on f S and S ≠ {}
    shows aff_dim U ≤ aff_dim V
  proof -
    obtain a b where a ∈ S a ∈ U b ∈ V
    using ⟨S ≠ {}⟩ fim ope openin_contains_cball by fastforce
    have dim ((+) (- a) ' U) ≤ dim ((+) (- b) ' V)
    proof (rule invariance_of_dimension_subspaces)
      show openin (top_of_set ((+) (- a) ' U)) ((+) (- a) ' S)
      by (metis ope homeomorphism_imp_open_map homeomorphism_translation
translation_galois)
      show subspace ((+) (- a) ' U)
      by (simp add: ⟨a ∈ U⟩ affine_diffs_subspace_subtract ⟨affine U⟩ cong:
image_cong_simp)
      show subspace ((+) (- b) ' V)
      by (simp add: ⟨b ∈ V⟩ affine_diffs_subspace_subtract ⟨affine V⟩ cong:
image_cong_simp)
      show continuous_on ((+) (- a) ' S) ((+) (- b) ∘ f ∘ (+) a)
      by (metis conf continuous_on_compose homeomorphism_cont2 homeomor-
phism_translation translation_galois)
      show ((+) (- b) ∘ f ∘ (+) a) ' (+) (- a) ' S ⊆ (+) (- b) ' V
      using fim by auto
      show inj_on ((+) (- b) ∘ f ∘ (+) a) ((+) (- a) ' S)
      by (auto simp: inj_on_def) (meson inj_onD injf)
    qed (use ⟨S ≠ {}⟩ in auto)
    then show ?thesis
    by (metis ⟨a ∈ U⟩ ⟨b ∈ V⟩ aff_dim_eq_dim affine_hull_eq aff_of_nat_le_iff)
  qed

```

```

corollary invariance_of_dimension:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes conf: continuous_on S f and open S
    and injf: inj_on f S and S ≠ {}
  shows DIM('a) ≤ DIM('b)
  using invariance_of_dimension_subspaces [of UNIV S UNIV f] assms
  by auto

```

```

corollary continuous_injective_image_subspace_dim_le:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes subspace S subspace T
    and conf: continuous_on S f and fim: f ' S ⊆ T
    and injf: inj_on f S
  shows dim S ≤ dim T
  using invariance_of_dimension_subspaces [of S S _ f] assms by (auto simp:
subspace_affine)

```

```

lemma invariance_of_dimension_convex_domain:

```

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes convex S
  and contf: continuous_on S f and fim: f ' S  $\subseteq$  affine hull T
  and injf: inj_on f S
shows aff_dim S  $\leq$  aff_dim T
proof (cases S = {})
  case True
    then show ?thesis by (simp add: aff_dim_geq)
  next
    case False
    have aff_dim (affine hull S)  $\leq$  aff_dim (affine hull T)
    proof (rule invariance_of_dimension_affine_sets)
      show openin (top_of_set (affine hull S)) (rel_interior S)
        by (simp add: openin_rel_interior)
      show continuous_on (rel_interior S) f
        using contf continuous_on_subset rel_interior_subset by blast
      show f ' rel_interior S  $\subseteq$  affine hull T
        using fim rel_interior_subset by blast
      show inj_on f (rel_interior S)
        using inj_on_subset injf rel_interior_subset by blast
      show rel_interior S  $\neq$  {}
        by (simp add: False <convex S> rel_interior_eq_empty)
    qed auto
    then show ?thesis
      by simp
  qed

```

```

lemma homeomorphic_convex_sets_le:
  assumes convex S S homeomorphic T
  shows aff_dim S  $\leq$  aff_dim T
proof -
  obtain h k where homhk: homeomorphism S T h k
    using homeomorphic_def assms by blast
  show ?thesis
proof (rule invariance_of_dimension_convex_domain [OF <convex S>])
  show continuous_on S h
    using homeomorphism_def homhk by blast
  show h ' S  $\subseteq$  affine hull T
    by (metis homeomorphism_def homhk hull_subset)
  show inj_on h S
    by (meson homeomorphism_apply1 homhk inj_on_inverseI)
  qed
qed

```

```

lemma homeomorphic_convex_sets:
  assumes convex S convex T S homeomorphic T
  shows aff_dim S = aff_dim T
  by (meson assms dual_order.antisym homeomorphic_convex_sets_le homeomor-

```

phic_sym)

lemma *homeomorphic_convex_compact_sets_eq*:
assumes *convex S compact S convex T compact T*
shows *S homeomorphic T $\longleftrightarrow$ aff_dim S = aff_dim T*
by (*meson assms homeomorphic_convex_compact_sets homeomorphic_convex_sets*)

lemma *invariance_of_domain_gen*:
fixes *f :: 'a::euclidean_space $\Rightarrow$ 'b::euclidean_space*
assumes *open S continuous_on S f inj_on f S DIM('b) $\leq$ DIM('a)*
shows *open(f ' S)*
using *invariance_of_domain_subspaces [of UNIV S UNIV f] assms* **by** *auto*

lemma *injective_into_1d_imp_open_map_UNIV*:
fixes *f :: 'a::euclidean_space $\Rightarrow$ real*
assumes *open T continuous_on S f inj_on f S T $\subseteq$ S*
shows *open (f ' T)*
apply (*rule invariance_of_domain_gen [OF <open T>]*)
using *assms* **by** (*auto simp: elim: continuous_on_subset subset_inj_on*)

lemma *continuous_on_inverse_open*:
fixes *f :: 'a::euclidean_space $\Rightarrow$ 'b::euclidean_space*
assumes *open S continuous_on S f DIM('b) $\leq$ DIM('a) and gf: $\bigwedge x. x \in S \Rightarrow$*
g(f x) = x
shows *continuous_on (f ' S) g*
proof (*clarsimp simp add: continuous_openin_preimage_eq*)
fix *T :: 'a set*
assume *open T*
have *eq: f ' S $\cap$ g - ' T = f ' (S $\cap$ T)*
by (*auto simp: gf*)
have *open (f ' S)*
by (*rule invariance_of_domain_gen*) (*use assms inj_on_inverseI in auto*)
moreover **have** *open (f ' (S $\cap$ T))*
using *assms*
by (*metis <open T> continuous_on_subset inj_onI inj_on_subset invariance_of_domain_gen openin_open openin_open_eq*)
ultimately **show** *openin (top_of_set (f ' S)) (f ' S $\cap$ g - ' T)*
unfolding *eq* **by** (*auto intro: open_openin_trans*)
qed

lemma *invariance_of_domain_homeomorphism*:
fixes *f :: 'a::euclidean_space $\Rightarrow$ 'b::euclidean_space*
assumes *open S continuous_on S f DIM('b) $\leq$ DIM('a) inj_on f S*
obtains *g where homeomorphism S (f ' S) f g*
proof
show *homeomorphism S (f ' S) f (inv_into S f)*
by (*simp add: assms continuous_on_inverse_open homeomorphism_def*)
qed


```

corollary invariance_of_domain_homeomorphic:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes open S continuous_on S f DIM('b) ≤ DIM('a) inj_on f S
  shows S homeomorphic (f ` S)
  using invariance_of_domain_homeomorphism [OF assms]
  by (meson homeomorphic_def)

lemma continuous_image_subset_interior:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes continuous_on S f inj_on f S DIM('b) ≤ DIM('a)
  shows  $f ` (\text{interior } S) \subseteq \text{interior}(f ` S)$ 
proof -
  have open (f ` interior S)
  using assms
  by (intro invariance_of_domain_gen) (auto simp: subset_inj_on interior_subset
continuous_on_subset)
  then show ?thesis
  by (simp add: image_mono interior_maximal interior_subset)
qed

lemma homeomorphic_interiors_same_dimension:
  fixes  $S :: 'a::euclidean\_space \text{ set}$  and  $T :: 'b::euclidean\_space \text{ set}$ 
  assumes S homeomorphic T and dimeq: DIM('a) = DIM('b)
  shows (interior S) homeomorphic (interior T)
  using assms [unfolded homeomorphic_minimal]
  unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
  fix  $f g$ 
  assume  $S: \forall x \in S. f x \in T \wedge g (f x) = x$  and  $T: \forall y \in T. g y \in S \wedge f (g y) = y$ 
  and contf: continuous_on S f and contg: continuous_on T g
  then have fST: f ` S = T and gTS: g ` T = S and inj_on f S inj_on g T
  by (auto simp: inj_on_def intro: rev_image_eqImetis+)
  have fim: f ` interior S ⊆ interior T
  using continuous_image_subset_interior [OF contf ⟨inj_on f S⟩] dimeq fST
by simp
  have gim: g ` interior T ⊆ interior S
  using continuous_image_subset_interior [OF contg ⟨inj_on g T⟩] dimeq gTS
by simp
  show homeomorphism (interior S) (interior T) f g
  unfolding homeomorphism_def
proof (intro conjI ballI)
  show  $\bigwedge x. x \in \text{interior } S \Longrightarrow g (f x) = x$ 
  by (meson ⟨∀ x ∈ S. f x ∈ T ∧ g (f x) = x⟩ subsetD interior_subset)
  have interior T ⊆ f ` interior S
proof
  fix  $x$  assume  $x \in \text{interior } T$ 
  then have  $g x \in \text{interior } S$ 
  using gim by blast
  then show  $x \in f ` \text{interior } S$ 

```

```

      by (metis T ⟨x ∈ interior T⟩ image_iff interior_subset subsetCE)
    qed
  then show f ' interior S = interior T
    using fim by blast
  show continuous_on (interior S) f
    by (metis interior_subset continuous_on_subset contf)
  show  $\bigwedge y. y \in \text{interior } T \implies f (g y) = y$ 
    by (meson T subsetD interior_subset)
  have interior S  $\subseteq$  g ' interior T
  proof
    fix x assume x ∈ interior S
    then have f x ∈ interior T
      using fim by blast
    then show x ∈ g ' interior T
      by (metis S ⟨x ∈ interior S⟩ image_iff interior_subset subsetCE)
  qed
  then show g ' interior T = interior S
    using gim by blast
  show continuous_on (interior T) g
    by (metis interior_subset continuous_on_subset contg)
  qed
qed

lemma homeomorphic_open_imp_same_dimension:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T open S S ≠ {} open T T ≠ {}
  shows DIM('a) = DIM('b)
  using assms
  apply (simp add: homeomorphic_minimal)
  apply (rule order_antisym; metis inj_onI invariance_of_dimension)
  done

proposition homeomorphic_interiors:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T interior S = {}  $\longleftrightarrow$  interior T = {}
  shows (interior S) homeomorphic (interior T)
proof (cases interior T = {})
  case True
  with assms show ?thesis by auto
next
  case False
  then have DIM('a) = DIM('b)
    using assms
    apply (simp add: homeomorphic_minimal)
    apply (rule order_antisym; metis continuous_on_subset inj_onI inj_on_subset
      interior_subset invariance_of_dimension open_interior)
  done
  then show ?thesis
    by (rule homeomorphic_interiors_same_dimension [OF ⟨S homeomorphic T⟩])

```

qed

lemma *homeomorphic_frontiers_same_dimension:*

```

fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
assumes  $S\ homeomorphic\ T$   $closed\ S$   $closed\ T$  and  $dimeq: DIM('a) = DIM('b)$ 
shows  $(frontier\ S)\ homeomorphic\ (frontier\ T)$ 
using assms [unfolded homeomorphic_minimal]
unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
  fix  $f\ g$ 
  assume  $S: \forall x \in S. f\ x \in T \wedge g\ (f\ x) = x$  and  $T: \forall y \in T. g\ y \in S \wedge f\ (g\ y) = y$ 
    and  $contf: continuous\_on\ S\ f$  and  $contg: continuous\_on\ T\ g$ 
  then have  $fST: f\ 'S = T$  and  $gTS: g\ 'T = S$  and  $inj\_on\ f\ S\ inj\_on\ g\ T$ 
    by (auto simp: inj_on_def intro: rev_image_eqI) metis+
  have  $g\ 'interior\ T \subseteq interior\ S$ 
    using continuous_image_subset_interior [OF contg <inj_on g T>] dimeq gTS
by simp
  then have  $fim: f\ 'frontier\ S \subseteq frontier\ T$ 
    unfolding frontier_def
    using continuous_image_subset_interior assms(2) assms(3)  $S$  by auto
  have  $f\ 'interior\ S \subseteq interior\ T$ 
    using continuous_image_subset_interior [OF contf <inj_on f S>] dimeq fST
by simp
  then have  $gim: g\ 'frontier\ T \subseteq frontier\ S$ 
    unfolding frontier_def
    using continuous_image_subset_interior  $T$  assms(2) assms(3) by auto
  show homeomorphism  $(frontier\ S)\ (frontier\ T)\ f\ g$ 
    unfolding homeomorphism_def
proof (intro conjI ballI)
  show  $gf: \bigwedge x. x \in frontier\ S \implies g\ (f\ x) = x$ 
    by (simp add: S assms(2) frontier_def)
  show  $fg: \bigwedge y. y \in frontier\ T \implies f\ (g\ y) = y$ 
    by (simp add: T assms(3) frontier_def)
  have  $frontier\ T \subseteq f\ 'frontier\ S$ 
proof
    fix  $x$  assume  $x \in frontier\ T$ 
    then have  $g\ x \in frontier\ S$ 
      using  $gim$  by blast
    then show  $x \in f\ 'frontier\ S$ 
      by (metis fg <x ∈ frontier T> imageI)
qed
  then show  $f\ 'frontier\ S = frontier\ T$ 
    using  $fim$  by blast
  show continuous_on  $(frontier\ S)\ f$ 
    by (metis Diff_subset assms(2) closure_eq contf continuous_on_subset frontier_def)
  have  $frontier\ S \subseteq g\ 'frontier\ T$ 
proof
  fix  $x$  assume  $x \in frontier\ S$ 

```

```

    then have  $f\ x \in \text{frontier } T$ 
    using  $\text{fim}$  by blast
    then show  $x \in g\ \text{'frontier } T$ 
    by (metis  $gf\ \langle x \in \text{frontier } S \rangle \text{ imageI}$ )
  qed
  then show  $g\ \text{'frontier } T = \text{frontier } S$ 
  using  $\text{gim}$  by blast
  show  $\text{continuous\_on } (\text{frontier } T)\ g$ 
  by (metis  $\text{Diff\_subset assms(3) closure\_closed contg continuous\_on\_subset}$ 
 $\text{frontier\_def}$ )
  qed
qed

```

```

lemma homeomorphic_frontiers:
  fixes  $S :: \text{'a::euclidean\_space set}$  and  $T :: \text{'b::euclidean\_space set}$ 
  assumes  $S \text{ homeomorphic } T$   $\text{closed } S$   $\text{closed } T$ 
     $\text{interior } S = \{\} \longleftrightarrow \text{interior } T = \{\}$ 
  shows  $(\text{frontier } S) \text{ homeomorphic } (\text{frontier } T)$ 
proof (cases  $\text{interior } T = \{\}$ )
  case True
  then show ?thesis
  by (metis  $\text{Diff\_empty assms closure\_eq frontier\_def}$ )
next
  case False
  then have  $\text{DIM}(\text{'a}) = \text{DIM}(\text{'b})$ 
  using  $\text{assms homeomorphic\_interiors homeomorphic\_open\_imp\_same\_dimension}$ 
  by blast
  then show ?thesis
  using  $\text{assms homeomorphic\_frontiers\_same\_dimension}$  by blast
qed

```

```

lemma continuous_image_subset_rel_interior:
  fixes  $f :: \text{'a::euclidean\_space} \Rightarrow \text{'b::euclidean\_space}$ 
  assumes  $\text{contf: continuous\_on } S\ f$  and  $\text{inj: inj\_on } f\ S$  and  $\text{fim: } f\ \text{' } S \subseteq T$ 
    and  $\text{TS: aff\_dim } T \leq \text{aff\_dim } S$ 
  shows  $f\ \text{' } (\text{rel\_interior } S) \subseteq \text{rel\_interior}(f\ \text{' } S)$ 
proof (rule  $\text{rel\_interior\_maximal}$ )
  show  $f\ \text{' } \text{rel\_interior } S \subseteq f\ \text{' } S$ 
  by (simp add:  $\text{image\_mono rel\_interior\_subset}$ )
  show  $\text{openin } (\text{top\_of\_set } (\text{affine hull } f\ \text{' } S))\ (f\ \text{' } \text{rel\_interior } S)$ 
proof (rule  $\text{invariance\_of\_domain\_affine\_sets}$ )
  show  $\text{openin } (\text{top\_of\_set } (\text{affine hull } S))\ (\text{rel\_interior } S)$ 
  by (simp add:  $\text{openin\_rel\_interior}$ )
  show  $\text{aff\_dim } (\text{affine hull } f\ \text{' } S) \leq \text{aff\_dim } (\text{affine hull } S)$ 
  by (metis  $\text{aff\_dim\_affine\_hull aff\_dim\_subset fim TS order\_trans}$ )
  show  $f\ \text{' } \text{rel\_interior } S \subseteq \text{affine hull } f\ \text{' } S$ 
  by (meson  $\langle f\ \text{' } \text{rel\_interior } S \subseteq f\ \text{' } S \rangle \text{ hull\_subset order\_trans}$ )
  show  $\text{continuous\_on } (\text{rel\_interior } S)\ f$ 
  using  $\text{contf continuous\_on\_subset rel\_interior\_subset}$  by blast

```

```

    show inj_on f (rel_interior S)
      using inj_on_subset injf rel_interior_subset by blast
  qed auto
qed

lemma homeomorphic_rel_interiors_same_dimension:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T and aff: aff_dim S = aff_dim T
  shows (rel_interior S) homeomorphic (rel_interior T)
  using assms [unfolded homeomorphic_minimal]
  unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
  fix f g
  assume S:  $\forall x \in S. f x \in T \wedge g(f x) = x$  and T:  $\forall y \in T. g y \in S \wedge f(g y) = y$ 
    and contf: continuous_on S f and contg: continuous_on T g
  then have fST:  $f ' S = T$  and gTS:  $g ' T = S$  and inj_on f S inj_on g T
    by (auto simp: inj_on_def intro: rev_image_eqI) metis+
  have fim:  $f ' \text{rel\_interior } S \subseteq \text{rel\_interior } T$ 
    by (metis <inj_on f S> aff contf continuous_image_subset_rel_interior fST
order_refl)
  have gim:  $g ' \text{rel\_interior } T \subseteq \text{rel\_interior } S$ 
    by (metis <inj_on g T> aff contg continuous_image_subset_rel_interior gTS
order_refl)
  show homeomorphism (rel_interior S) (rel_interior T) f g
    unfolding homeomorphism_def
  proof (intro conjI ballI)
    show gf:  $\bigwedge x. x \in \text{rel\_interior } S \implies g(f x) = x$ 
      using S rel_interior_subset by blast
    show fg:  $\bigwedge y. y \in \text{rel\_interior } T \implies f(g y) = y$ 
      using T mem_rel_interior_ball by blast
    have rel_interior T  $\subseteq f ' \text{rel\_interior } S$ 
    proof
      fix x assume x  $\in \text{rel\_interior } T$ 
      then have g x  $\in \text{rel\_interior } S$ 
        using gim by blast
      then show x  $\in f ' \text{rel\_interior } S$ 
        by (metis fg <x  $\in \text{rel\_interior } T$ > imageI)
    qed
    moreover have f ' rel_interior S  $\subseteq \text{rel\_interior } T$ 
      by (metis <inj_on f S> aff contf continuous_image_subset_rel_interior fST
order_refl)
    ultimately show f ' rel_interior S = rel_interior T
      by blast
    show continuous_on (rel_interior S) f
      using contf continuous_on_subset rel_interior_subset by blast
    have rel_interior S  $\subseteq g ' \text{rel\_interior } T$ 
  proof
    fix x assume x  $\in \text{rel\_interior } S$ 
    then have f x  $\in \text{rel\_interior } T$ 

```

```

    using fim by blast
  then show  $x \in g \text{ ' rel\_interior } T$ 
    by (metis gf  $\langle x \in \text{rel\_interior } S \rangle \text{ imageI}$ )
qed
then show  $g \text{ ' rel\_interior } T = \text{rel\_interior } S$ 
  using gim by blast
show continuous_on (rel_interior T) g
  using contg continuous_on_subset rel_interior_subset by blast
qed
qed

```

```

lemma homeomorphic_aff_dim_le:
  fixes  $S :: 'a::\text{euclidean\_space set}$ 
  assumes  $S \text{ homeomorphic } T \text{ rel\_interior } S \neq \{\}$ 
  shows  $\text{aff\_dim } (\text{affine hull } S) \leq \text{aff\_dim } (\text{affine hull } T)$ 
proof -
  obtain f g
    where  $S: \forall x \in S. f x \in T \wedge g (f x) = x$  and  $T: \forall y \in T. g y \in S \wedge f (g y) = y$ 
    and contf: continuous_on S f and contg: continuous_on T g
    using assms [unfolded homeomorphic_minimal] by auto
  show ?thesis
proof (rule invariance_of_dimension_affine_sets)
  show continuous_on (rel_interior S) f
    using contf continuous_on_subset rel_interior_subset by blast
  show  $f \text{ ' rel\_interior } S \subseteq \text{affine hull } T$ 
    by (meson S hull_subset image_subsetI rel_interior_subset rev_subsetD)
  show inj_on f (rel_interior S)
    by (metis S inj_on_inverseI inj_on_subset rel_interior_subset)
  qed (simp_all add: openin_rel_interior assms)
qed

```

```

lemma homeomorphic_rel_interiors:
  fixes  $S :: 'a::\text{euclidean\_space set}$  and  $T :: 'b::\text{euclidean\_space set}$ 
  assumes  $S \text{ homeomorphic } T \text{ rel\_interior } S = \{\} \longleftrightarrow \text{rel\_interior } T = \{\}$ 
  shows  $(\text{rel\_interior } S) \text{ homeomorphic } (\text{rel\_interior } T)$ 
proof (cases rel_interior T =  $\{\}$ )
  case True
  with assms show ?thesis by auto
next
  case False
  have  $\text{aff\_dim } (\text{affine hull } S) \leq \text{aff\_dim } (\text{affine hull } T)$ 
    using False assms homeomorphic_aff_dim_le by blast
  moreover have  $\text{aff\_dim } (\text{affine hull } T) \leq \text{aff\_dim } (\text{affine hull } S)$ 
    using False assms(1) homeomorphic_aff_dim_le homeomorphic_sym by auto
  ultimately have  $\text{aff\_dim } S = \text{aff\_dim } T$  by force
  then show ?thesis
    by (rule homeomorphic_rel_interiors_same_dimension [OF  $\langle S \text{ homeomorphic } T \rangle$ ])

```

qed

```

lemma homeomorphic_rel_boundaries_same_dimension:
  fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
  assumes  $S\ homeomorphic\ T$  and  $aff: aff\_dim\ S = aff\_dim\ T$ 
  shows  $(S - rel\_interior\ S)\ homeomorphic\ (T - rel\_interior\ T)$ 
  using assms [unfolded homeomorphic_minimal]
  unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
  fix  $f\ g$ 
  assume  $S: \forall x \in S. f\ x \in T \wedge g\ (f\ x) = x$  and  $T: \forall y \in T. g\ y \in S \wedge f\ (g\ y) = y$ 
  and  $contf: continuous\_on\ S\ f$  and  $contg: continuous\_on\ T\ g$ 
  then have  $fST: f\ 'S = T$  and  $gTS: g\ 'T = S$  and  $inj\_on\ f\ S\ inj\_on\ g\ T$ 
  by (auto simp: inj_on_def intro: rev_image_eqI) metis+
  have  $fim: f\ 'rel\_interior\ S \subseteq rel\_interior\ T$ 
  by (metis  $\langle inj\_on\ f\ S \rangle\ aff\ contf\ continuous\_image\_subset\_rel\_interior\ fST$ 
order_refl)
  have  $gim: g\ 'rel\_interior\ T \subseteq rel\_interior\ S$ 
  by (metis  $\langle inj\_on\ g\ T \rangle\ aff\ contg\ continuous\_image\_subset\_rel\_interior\ gTS$ 
order_refl)
  show  $homeomorphism\ (S - rel\_interior\ S)\ (T - rel\_interior\ T)\ f\ g$ 
  unfolding homeomorphism_def
proof (intro conjI ballI)
  show  $gf: \bigwedge x. x \in S - rel\_interior\ S \implies g\ (f\ x) = x$ 
  using  $S\ rel\_interior\_subset$  by blast
  show  $fg: \bigwedge y. y \in T - rel\_interior\ T \implies f\ (g\ y) = y$ 
  using  $T\ mem\_rel\_interior\_ball$  by blast
  show  $f\ '(S - rel\_interior\ S) = T - rel\_interior\ T$ 
  using  $S\ fST\ fim\ gim$  by auto
  show  $continuous\_on\ (S - rel\_interior\ S)\ f$ 
  using  $contf\ continuous\_on\_subset\ rel\_interior\_subset$  by blast
  show  $g\ '(T - rel\_interior\ T) = S - rel\_interior\ S$ 
  using  $T\ gTS\ gim\ fim$  by auto
  show  $continuous\_on\ (T - rel\_interior\ T)\ g$ 
  using  $contg\ continuous\_on\_subset\ rel\_interior\_subset$  by blast
qed
qed

```

```

lemma homeomorphic_rel_boundaries:
  fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
  assumes  $S\ homeomorphic\ T$   $rel\_interior\ S = \{\}$   $\longleftrightarrow rel\_interior\ T = \{\}$ 
  shows  $(S - rel\_interior\ S)\ homeomorphic\ (T - rel\_interior\ T)$ 
proof (cases rel_interior T = {})
  case True
  with assms show ?thesis by auto
next
  case False
  obtain  $f\ g$ 

```

where $S: \forall x \in S. f x \in T \wedge g (f x) = x$ **and** $T: \forall y \in T. g y \in S \wedge f (g y) = y$
and $\text{contf}: \text{continuous\_on } S f$ **and** $\text{contg}: \text{continuous\_on } T g$
using assms [unfolded_homeomorphic_minimal] **by** auto
have $\text{aff\_dim } (\text{affine hull } S) \leq \text{aff\_dim } (\text{affine hull } T)$
using $\text{False assms homeomorphic\_aff\_dim\_le}$ **by** blast
moreover **have** $\text{aff\_dim } (\text{affine hull } T) \leq \text{aff\_dim } (\text{affine hull } S)$
by $(\text{meson False assms}(1) \text{ homeomorphic\_aff\_dim\_le homeomorphic\_sym})$
ultimately **have** $\text{aff\_dim } S = \text{aff\_dim } T$ **by** force
then show $?thesis$
by $(\text{rule homeomorphic\_rel\_boundaries\_same\_dimension } [OF \langle S \text{ homeomor-}$
 $\text{phic } T \rangle])$
qed

proposition $\text{uniformly\_continuous\_homeomorphism\_UNIV\_trivial}$:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'a$
assumes $\text{contf}: \text{uniformly\_continuous\_on } S f$ **and** $\text{hom}: \text{homeomorphism } S$
 $\text{UNIV } f g$
shows $S = \text{UNIV}$
proof $(\text{cases } S = \{\})$
case True
then show $?thesis$
by $(\text{metis UNIV\_I hom empty\_iff homeomorphism\_def image\_eqI})$
next
case False
have $\text{inj } g$
by $(\text{metis UNIV\_I hom homeomorphism\_apply2 injI})$
then **have** $\text{open } (g \text{ ' UNIV})$
by $(\text{blast intro: invariance\_of\_domain hom homeomorphism\_cont2})$
then **have** $\text{open } S$
using $\text{hom homeomorphism\_image2}$ **by** blast
moreover **have** $\text{complete } S$
unfolding complete_def
proof clarify
fix σ
assume $\sigma: \forall n. \sigma n \in S$ **and** $\text{Cauchy } \sigma$
have $\text{Cauchy } (f \circ \sigma)$
using $\text{uniformly\_continuous\_imp\_Cauchy\_continuous } \langle \text{Cauchy } \sigma \rangle \sigma \text{ contf}$
by blast
then **obtain** l **where** $(f \circ \sigma) \longrightarrow l$
by $(\text{auto simp: convergent\_eq\_Cauchy } [\text{symmetric}])$
show $\exists l \in S. \sigma \longrightarrow l$
proof
show $g l \in S$
using $\text{hom homeomorphism\_image2}$ **by** blast
have $(g \circ (f \circ \sigma)) \longrightarrow g l$
by $(\text{meson UNIV\_I } \langle f \circ \sigma \rangle \longrightarrow l \text{ continuous\_on\_sequentially hom}$
 $\text{homeomorphism\_cont2})$
then **show** $\sigma \longrightarrow g l$
proof $-$


```

    have  $\forall n. \sigma \ n = (g \circ (f \circ \sigma)) \ n$ 
    by (metis (no_types)  $\sigma$  comp_eq_dest_lhs hom_homeomorphism_apply1)
  then show ?thesis
    by (metis (no_types) LIMSEQ_iff  $\langle (g \circ (f \circ \sigma)) \longrightarrow g \ l \rangle$ )
qed
qed
qed
then have closed S
  by (simp add: complete_eq_closed)
ultimately show ?thesis
  using clopen [of S] False by simp
qed

```

6.42.6 Formulation of loop homotopy in terms of maps out of type complex

```

lemma homotopic_circlemaps_imp_homotopic_loops:
  assumes homotopic_with_canon ( $\lambda h. \text{True}$ ) (sphere 0 1) S f g
  shows homotopic_loops S (f  $\circ$  exp  $\circ$  ( $\lambda t. 2 * \text{of\_real } \pi * \text{of\_real } t * i$ ))
    (g  $\circ$  exp  $\circ$  ( $\lambda t. 2 * \text{of\_real } \pi * \text{of\_real } t * i$ ))
proof -
  have homotopic_with_canon ( $\lambda f. \text{True}$ ) {z. cmod z = 1} S f g
    using assms by (auto simp: sphere_def)
  moreover have continuous_on {0..1} (exp  $\circ$  ( $\lambda t. 2 * \text{of\_real } \pi * \text{of\_real } t * i$ 
    i))
    by (intro continuous_intros)
  moreover have (exp  $\circ$  ( $\lambda t. 2 * \text{of\_real } \pi * \text{of\_real } t * i$ )) ' {0..1}  $\subseteq$  {z. cmod
    z = 1}
    by (auto simp: norm_mult)
  ultimately
  show ?thesis
    apply (simp add: homotopic_loops_def comp_assoc)
    apply (rule homotopic_with_compose_continuous_right)
    apply (auto simp: pathstart_def pathfinish_def)
    done
qed

```

```

lemma homotopic_loops_imp_homotopic_circlemaps:
  assumes homotopic_loops S p q
  shows homotopic_with_canon ( $\lambda h. \text{True}$ ) (sphere 0 1) S
    (p  $\circ$  ( $\lambda z. (\text{Arg2pi } z / (2 * \pi)$ )))
    (q  $\circ$  ( $\lambda z. (\text{Arg2pi } z / (2 * \pi)$ )))
proof -
  obtain h where conth: continuous_on ({0..1::real}  $\times$  {0..1}) h
    and him: h ' ({0..1}  $\times$  {0..1})  $\subseteq$  S
    and h0: ( $\forall x. h \ (0, x) = p \ x$ )
    and h1: ( $\forall x. h \ (1, x) = q \ x$ )
    and h01: ( $\forall t \in \{0..1\}. h \ (t, 1) = h \ (t, 0)$ )
  using assms

```

```

    by (auto simp: homotopic_loops_def sphere_def homotopic_with_def path-
start_def pathfinish_def)
    define j where j  $\equiv \lambda z. \text{if } 0 \leq \text{Im } (\text{snd } z) \text{ then } h \text{ (fst } z, \text{Arg2pi } (\text{snd } z) / (2 * \text{pi}))$ 
    else  $h \text{ (fst } z, 1 - \text{Arg2pi } (\text{cnj } (\text{snd } z)) / (2 * \text{pi}))$ 
    have Arg2pi_eq:  $1 - \text{Arg2pi } (\text{cnj } y) / (2 * \text{pi}) = \text{Arg2pi } y / (2 * \text{pi}) \vee \text{Arg2pi } y = 0 \wedge \text{Arg2pi } (\text{cnj } y) = 0$  if  $\text{cmod } y = 1$  for y
    using that Arg2pi_eq_0_pi Arg2pi_eq_pi by (force simp: Arg2pi_cnj_field_split_simps)
    show ?thesis
    proof (simp add: homotopic_with; intro conjI ballI exI)
      show continuous_on  $(\{0..1\} \times \text{sphere } 0 \ 1)$   $(\lambda w. h \text{ (fst } w, \text{Arg2pi } (\text{snd } w) / (2 * \text{pi})))$ 
    proof (rule continuous_on_eq)
      show j:  $j \ x = h \text{ (fst } x, \text{Arg2pi } (\text{snd } x) / (2 * \text{pi}))$  if  $x \in \{0..1\} \times \text{sphere } 0 \ 1$ 
    for x
      using Arg2pi_eq that h01 by (force simp: j_def)
      have eq:  $S = S \cap (\text{UNIV} \times \{z. 0 \leq \text{Im } z\}) \cup S \cap (\text{UNIV} \times \{z. \text{Im } z \leq 0\})$  for  $S :: (\text{real} * \text{complex}) \text{set}$ 
      by auto
      have c1: continuous_on  $(\{0..1\} \times \text{sphere } 0 \ 1 \cap \text{UNIV} \times \{z. 0 \leq \text{Im } z\})$ 
 $(\lambda x. h \text{ (fst } x, \text{Arg2pi } (\text{snd } x) / (2 * \text{pi})))$ 
      apply (intro continuous_intros continuous_on_compose2 [OF conth] continuous_on_compose2 [OF continuous_on_upperhalf_Arg2pi])
      apply (auto simp: Arg2pi)
      apply (meson Arg2pi_lt_2pi linear not_le)
      done
      have c2: continuous_on  $(\{0..1\} \times \text{sphere } 0 \ 1 \cap \text{UNIV} \times \{z. \text{Im } z \leq 0\})$ 
 $(\lambda x. h \text{ (fst } x, 1 - \text{Arg2pi } (\text{cnj } (\text{snd } x)) / (2 * \text{pi})))$ 
      apply (intro continuous_intros continuous_on_compose2 [OF conth] continuous_on_compose2 [OF continuous_on_upperhalf_Arg2pi])
      apply (auto simp: Arg2pi)
      apply (meson Arg2pi_lt_2pi linear not_le)
      done
      show continuous_on  $(\{0..1\} \times \text{sphere } 0 \ 1)$  j
      apply (simp add: j_def)
      apply (subst eq)
      apply (rule continuous_on_cases_local)
      using Arg2pi_eq h01
      by (force simp add: eq [symmetric] closedin_closed_Int closed_Times closed_halfspace_Im_le closed_halfspace_Im_ge c1 c2)+
    qed
    have  $(\lambda w. h \text{ (fst } w, \text{Arg2pi } (\text{snd } w) / (2 * \text{pi}))) ' (\{0..1\} \times \text{sphere } 0 \ 1) \subseteq h ' (\{0..1\} \times \{0..1\})$ 
    by (auto simp: Arg2pi_ge_0 Arg2pi_lt_2pi less_imp_le)
    also have  $\dots \subseteq S$ 
    using him by blast
    finally show  $(\lambda w. h \text{ (fst } w, \text{Arg2pi } (\text{snd } w) / (2 * \text{pi}))) ' (\{0..1\} \times \text{sphere } 0 \ 1) \subseteq S$  .
  qed (auto simp: h0 h1)

```

qed

lemma *simply_connected_homotopic_loops*:

simply_connected $S \longleftrightarrow$
 $(\forall p\ q. \text{homotopic\_loops } S\ p\ p \wedge \text{homotopic\_loops } S\ q\ q \longrightarrow \text{homotopic\_loops } S\ p\ q)$
unfolding *simply_connected_def* **using** *homotopic_loops_refl* **by** *metis*

lemma *simply_connected_eq_homotopic_circlemaps1*:

fixes $f :: \text{complex} \Rightarrow 'a::\text{topological\_space}$ **and** $g :: \text{complex} \Rightarrow 'a$
assumes S : *simply_connected* S
and *contf*: *continuous_on* (*sphere* 0 1) f **and** *fim*: $f \text{ ' } (\text{sphere } 0\ 1) \subseteq S$
and *contg*: *continuous_on* (*sphere* 0 1) g **and** *gim*: $g \text{ ' } (\text{sphere } 0\ 1) \subseteq S$
shows *homotopic_with_canon* ($\lambda h. \text{True}$) (*sphere* 0 1) $S\ f\ g$
proof –
have *homotopic_loops* $S\ (f \circ \exp \circ (\lambda t. \text{of\_real}(2 * \pi * t) * i))\ (g \circ \exp \circ (\lambda t. \text{of\_real}(2 * \pi * t) * i))$
apply (*rule* S [*unfolded simply_connected_homotopic_loops*, *rule_format*])
apply (*simp* *add*: *homotopic_circlemaps_imp_homotopic_loops* *contf* *fim* *contg* *gim*)
done
then show *?thesis*
apply (*rule* *homotopic_with_eq* [*OF* *homotopic_loops_imp_homotopic_circlemaps*])
apply (*auto* *simp*: *o_def* *complex_norm_eq_1_exp* *mult.commute*)
done
qed

lemma *simply_connected_eq_homotopic_circlemaps2a*:

fixes $h :: \text{complex} \Rightarrow 'a::\text{topological\_space}$
assumes *conth*: *continuous_on* (*sphere* 0 1) h **and** *him*: $h \text{ ' } (\text{sphere } 0\ 1) \subseteq S$
and *hom*: $\bigwedge f\ g::\text{complex} \Rightarrow 'a.$
 $\llbracket \text{continuous\_on } (\text{sphere } 0\ 1)\ f; f \text{ ' } (\text{sphere } 0\ 1) \subseteq S;$
 $\text{continuous\_on } (\text{sphere } 0\ 1)\ g; g \text{ ' } (\text{sphere } 0\ 1) \subseteq S \rrbracket$
 $\implies \text{homotopic\_with\_canon } (\lambda h. \text{True})\ (\text{sphere } 0\ 1)\ S\ f\ g$
shows $\exists a. \text{homotopic\_with\_canon } (\lambda h. \text{True})\ (\text{sphere } 0\ 1)\ S\ h\ (\lambda x. a)$
apply (*rule_tac* $x=h\ 1$ **in** *exI*)
apply (*rule* *hom*)
using *assms* **by** (*auto*)

lemma *simply_connected_eq_homotopic_circlemaps2b*:

fixes $S :: 'a::\text{real\_normed\_vector\_set}$
assumes $\bigwedge f\ g::\text{complex} \Rightarrow 'a.$
 $\llbracket \text{continuous\_on } (\text{sphere } 0\ 1)\ f; f \text{ ' } (\text{sphere } 0\ 1) \subseteq S;$
 $\text{continuous\_on } (\text{sphere } 0\ 1)\ g; g \text{ ' } (\text{sphere } 0\ 1) \subseteq S \rrbracket$
 $\implies \text{homotopic\_with\_canon } (\lambda h. \text{True})\ (\text{sphere } 0\ 1)\ S\ f\ g$
shows *path_connected* S
proof (*clarsimp* *simp* *add*: *path_connected_eq_homotopic_points*)
fix $a\ b$

```

assume  $a \in S \ b \in S$ 
then show homotopic_loops  $S$  (linepath  $a$   $a$ ) (linepath  $b$   $b$ )
  using homotopic_circlemaps_imp_homotopic_loops [OF assms [of  $\lambda x. a \ \lambda x. b$ ]]
  by (auto simp: o_def linepath_def)
qed

```

lemma *simply_connected_eq_homotopic_circlemaps3*:

```

fixes  $h :: \text{complex} \Rightarrow 'a :: \text{real\_normed\_vector}$ 
assumes path_connected  $S$ 
  and  $hom: \bigwedge f :: \text{complex} \Rightarrow 'a. \llbracket \text{continuous\_on } (\text{sphere } 0 \ 1) \ f; f \ '(\text{sphere } 0 \ 1) \subseteq S \rrbracket$ 
     $\implies \exists a. \text{homotopic\_with\_canon } (\lambda h. \text{True}) (\text{sphere } 0 \ 1) \ S \ f \ (\lambda x. a)$ 
shows simply_connected  $S$ 
proof (clarsimp simp add: simply_connected_eq_contractible_loop_some assms)
  fix  $p$ 
  assume  $p: \text{path } p \ \text{path\_image } p \subseteq S \ \text{pathfinish } p = \text{pathstart } p$ 
  then have homotopic_loops  $S \ p \ p$ 
    by (simp add: homotopic_loops_refl)
  then obtain  $a$  where  $homp: \text{homotopic\_with\_canon } (\lambda h. \text{True}) (\text{sphere } 0 \ 1) \ S$ 
     $(p \circ (\lambda z. \text{Arg2pi } z / (2 * \text{pi}))) (\lambda x. a)$ 
  by (metis homotopic_with_imp_subset2 homotopic_loops_imp_homotopic_circlemaps
    homotopic_with_imp_continuous hom)
  show  $\exists a. a \in S \wedge \text{homotopic\_loops } S \ p \ (\text{linepath } a \ a)$ 
  proof (intro exI conjI)
    show  $a \in S$ 
      using homotopic_with_imp_subset2 [OF homp]
      by (metis dist_0_norm image_subset_iff mem_sphere norm_one)
    have  $teq: \bigwedge t. \llbracket 0 \leq t; t \leq 1 \rrbracket$ 
       $\implies t = \text{Arg2pi } (\exp (2 * \text{of\_real } \text{pi} * \text{of\_real } t * i)) / (2 * \text{pi}) \vee t=1$ 
       $\wedge \text{Arg2pi } (\exp (2 * \text{of\_real } \text{pi} * \text{of\_real } t * i)) = 0$ 
      using Arg2pi_of_real [of 1] by (force simp: Arg2pi_exp)
    have homotopic_loops  $S \ p \ (p \circ (\lambda z. \text{Arg2pi } z / (2 * \text{pi}))) \circ \exp \circ (\lambda t. 2 * \text{complex\_of\_real } \text{pi} * \text{complex\_of\_real } t * i)$ 
      using  $p \ teq$  by (fastforce simp: pathfinish_def pathstart_def intro: homotopic_loops_eq [OF  $p$ ])
    then show homotopic_loops  $S \ p \ (\text{linepath } a \ a)$ 
      by (simp add: linepath_refl homotopic_loops_trans [OF homotopic_circlemaps_imp_homotopic_loops
        [OF homp, simplified K_record_comp]])
    qed
  qed

```

proposition *simply_connected_eq_homotopic_circlemaps*:

```

fixes  $S :: 'a :: \text{real\_normed\_vector} \text{ set}$ 
shows simply_connected  $S \longleftrightarrow$ 
   $(\forall f \ g :: \text{complex} \Rightarrow 'a. \text{continuous\_on } (\text{sphere } 0 \ 1) \ f \wedge f \ '(\text{sphere } 0 \ 1) \subseteq S \wedge$ 
     $\text{continuous\_on } (\text{sphere } 0 \ 1) \ g \wedge g \ '(\text{sphere } 0 \ 1) \subseteq S$ 

```

```

       $\longrightarrow$  homotopic_with_canon ( $\lambda h. \text{True}$ ) ( $\text{sphere } 0 \ 1$ )  $S \ f \ g$ )
    apply (rule iffI)
    apply (blast dest: simply_connected_eq_homotopic_circlemaps1)
    by (simp add: simply_connected_eq_homotopic_circlemaps2a simply_connected_eq_homotopic_circlemaps2b
        simply_connected_eq_homotopic_circlemaps3)

proposition simply_connected_eq_contractible_circlemap:
  fixes  $S :: 'a::\text{real\_normed\_vector\_set}$ 
  shows simply_connected  $S \longleftrightarrow$ 
    path_connected  $S \wedge$ 
    ( $\forall f::\text{complex} \Rightarrow 'a. \text{continuous\_on } (\text{sphere } 0 \ 1) \ f \wedge f \text{ `}(\text{sphere } 0 \ 1) \subseteq S$ 
       $\longrightarrow (\exists a. \text{homotopic\_with\_canon } (\lambda h. \text{True}) (\text{sphere } 0 \ 1) \ S \ f \ (\lambda x. a)))$ )
  apply (rule iffI)
  apply (simp add: simply_connected_eq_homotopic_circlemaps1 simply_connected_eq_homotopic_circlemaps2a
    simply_connected_eq_homotopic_circlemaps2b)
  using simply_connected_eq_homotopic_circlemaps3 by blast

corollary homotopy_eqv_simple_connectedness:
  fixes  $S :: 'a::\text{real\_normed\_vector\_set}$  and  $T :: 'b::\text{real\_normed\_vector\_set}$ 
  shows  $S \text{ homotopy\_eqv } T \implies \text{simply\_connected } S \longleftrightarrow \text{simply\_connected } T$ 
  by (simp add: simply_connected_eq_homotopic_circlemaps homotopy_eqv_homotopic_triviality)

```

6.42.7 Homeomorphism of simple closed curves to circles

```

proposition homeomorphic_simple_path_image_circle:
  fixes  $a :: \text{complex}$  and  $\gamma :: \text{real} \Rightarrow 'a::\text{t2\_space}$ 
  assumes simple_path  $\gamma$  and loop: pathfinish  $\gamma = \text{pathstart } \gamma$  and  $0 < r$ 
  shows (path_image  $\gamma$ ) homeomorphic sphere  $a \ r$ 
proof –
  have homotopic_loops (path_image  $\gamma$ )  $\gamma \ \gamma$ 
    by (simp add: asms homotopic_loops_refl simple_path_imp_path)
  then have hom: homotopic_with_canon ( $\lambda h. \text{True}$ ) ( $\text{sphere } 0 \ 1$ ) (path_image
 $\gamma$ )
    ( $\gamma \circ (\lambda z. \text{Arg2pi } z / (2*\text{pi}))$ ) ( $\gamma \circ (\lambda z. \text{Arg2pi } z / (2*\text{pi}))$ )
  by (rule homotopic_loops_imp_homotopic_circlemaps)
  have  $\exists g. \text{homeomorphism } (\text{sphere } 0 \ 1) (\text{path\_image } \gamma) (\gamma \circ (\lambda z. \text{Arg2pi } z /$ 
 $(2*\text{pi}))) \ g$ 
proof (rule homeomorphism_compact)
  show continuous_on ( $\text{sphere } 0 \ 1$ ) ( $\gamma \circ (\lambda z. \text{Arg2pi } z / (2*\text{pi}))$ )
    using hom homotopic_with_imp_continuous by blast
  show inj_on ( $\gamma \circ (\lambda z. \text{Arg2pi } z / (2*\text{pi}))$ ) ( $\text{sphere } 0 \ 1$ )
proof
  fix  $x \ y$ 
  assume  $xy: x \in \text{sphere } 0 \ 1 \ y \in \text{sphere } 0 \ 1$ 
  and eq: ( $\gamma \circ (\lambda z. \text{Arg2pi } z / (2*\text{pi}))$ )  $x = (\gamma \circ (\lambda z. \text{Arg2pi } z / (2*\text{pi}))$ )  $y$ 
  then have ( $\text{Arg2pi } x / (2*\text{pi})$ ) = ( $\text{Arg2pi } y / (2*\text{pi})$ )
proof –
  have ( $\text{Arg2pi } x / (2*\text{pi})$ )  $\in \{0..1\}$  ( $\text{Arg2pi } y / (2*\text{pi})$ )  $\in \{0..1\}$ 

```

```

    using Arg2pi_ge_0 Arg2pi_lt_2pi dual_order.strict_iff_order by fast-
force+
    with eq show ?thesis
    using ⟨simple_path γ⟩ Arg2pi_lt_2pi unfolding simple_path_def o_def
    by (metis eq_divide_eq_1 not_less_iff_gr_or_eq)
  qed
  with xy show x = y
  by (metis is_Arg_def Arg2pi Arg2pi_0 dist_0_norm divide_cancel_right
dual_order.strict_iff_order mem_sphere)
  qed
  have  $\bigwedge z. \text{cmod } z = 1 \implies \exists x \in \{0..1\}. \gamma (\text{Arg2pi } z / (2\pi)) = \gamma x$ 
  by (metis Arg2pi_ge_0 Arg2pi_lt_2pi atLeastAtMost_iff divide_less_eq_1
less_eq_real_def zero_less_mult_iff pi_gt_zero zero_le_divide_iff zero_less_numeral)
  moreover have  $\exists z \in \text{sphere } 0 \ 1. \gamma x = \gamma (\text{Arg2pi } z / (2\pi))$  if  $0 \leq x \leq 1$ 
for x
  proof (cases x=1)
  case True
  with Arg2pi_of_real [of 1] loop show ?thesis
  by (rule_tac x=1 in bexI) (auto simp: pathfinish_def pathstart_def ⟨0 ≤
x⟩)
  next
  case False
  then have *:  $(\text{Arg2pi } (\exp (i * (2\pi \text{ of\_real } \pi * \text{ of\_real } x)))) / (2\pi) = x$ 
  using that by (auto simp: Arg2pi_exp field_split_simps)
  show ?thesis
  by (rule_tac x= $\exp(i * \text{ of\_real}(2\pi * x))$  in bexI) (auto simp: *)
  qed
  ultimately show  $(\gamma \circ (\lambda z. \text{Arg2pi } z / (2\pi))) \text{ `sphere } 0 \ 1 = \text{path\_image } \gamma$ 
  by (auto simp: path_image_def image_iff)
  qed auto
  then have path_image γ homeomorphic sphere (0::complex) 1
  using homeomorphic_def homeomorphic_sym by blast
  also have ... homeomorphic sphere a r
  by (simp add: assms homeomorphic_spheres)
  finally show ?thesis .
  qed

```

```

lemma homeomorphic_simple_path_images:
  fixes γ1 :: real ⇒ 'a::t2_space and γ2 :: real ⇒ 'b::t2_space
  assumes simple_path γ1 and loop: pathfinish γ1 = pathstart γ1
  assumes simple_path γ2 and loop: pathfinish γ2 = pathstart γ2
  shows (path_image γ1) homeomorphic (path_image γ2)
  by (meson assms homeomorphic_simple_path_image_circle homeomorphic_sym
homeomorphic_trans loop pi_gt_zero)

```

6.42.8 Dimension-based conditions for various homeomorphisms

```

lemma homeomorphic_subspaces_eq:

```

```

fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
assumes  $subspace\ S\ subspace\ T$ 
shows  $S\ homeomorphic\ T \longleftrightarrow dim\ S = dim\ T$ 
proof
  assume  $S\ homeomorphic\ T$ 
  then obtain  $f\ g$  where  $hom: homeomorphism\ S\ T\ f\ g$ 
    using  $homeomorphic\_def$  by  $blast$ 
  show  $dim\ S = dim\ T$ 
  proof ( $rule\ order\_antisym$ )
    show  $dim\ S \leq dim\ T$ 
    by ( $metis\ assms\ dual\_order.refl\ inj\_onI\ homeomorphism\_cont1\ [OF\ hom]\$ 
 $homeomorphism\_apply1\ [OF\ hom]\ homeomorphism\_image1\ [OF\ hom]\ continu-$ 
 $ous\_injective\_image\_subspace\_dim\_le$ )
    show  $dim\ T \leq dim\ S$ 
    by ( $metis\ assms\ dual\_order.refl\ inj\_onI\ homeomorphism\_cont2\ [OF\ hom]\$ 
 $homeomorphism\_apply2\ [OF\ hom]\ homeomorphism\_image2\ [OF\ hom]\ continu-$ 
 $ous\_injective\_image\_subspace\_dim\_le$ )
  qed
next
  assume  $dim\ S = dim\ T$ 
  then show  $S\ homeomorphic\ T$ 
    by ( $simp\ add: assms\ homeomorphic\_subspaces$ )
qed

lemma  $homeomorphic\_affine\_sets\_eq$ :
  fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
  assumes  $affine\ S\ affine\ T$ 
  shows  $S\ homeomorphic\ T \longleftrightarrow aff\_dim\ S = aff\_dim\ T$ 
proof ( $cases\ S = \{\} \vee T = \{\}$ )
  case  $True$ 
    then show  $?thesis$ 
    using  $assms\ homeomorphic\_affine\_sets$  by  $force$ 
next
  case  $False$ 
    then obtain  $a\ b$  where  $a \in S\ b \in T$ 
    by  $blast$ 
    then have  $subspace\ ((+)\ (-\ a)\ 'S)\ subspace\ ((+)\ (-\ b)\ 'T)$ 
    using  $affine\_diffs\_subspace\ assms$  by  $blast+$ 
    then show  $?thesis$ 
    by ( $metis\ affine\_imp\_convex\ assms\ homeomorphic\_affine\_sets\ homeomor-$ 
 $phic\_convex\_sets$ )
qed

lemma  $homeomorphic\_hyperplanes\_eq$ :
  fixes  $a :: 'a::euclidean\_space$  and  $c :: 'b::euclidean\_space$ 
  assumes  $a \neq 0\ c \neq 0$ 
  shows  $(\{x. a \cdot x = b\}\ homeomorphic\ \{x. c \cdot x = d\}) \longleftrightarrow DIM('a) = DIM('b)$ 
  apply ( $auto\ simp: homeomorphic\_affine\_sets\_eq\ affine\_hyperplane\ assms$ )
  by ( $metis\ DIM\_positive\ Suc\_pred$ )

```

```

lemma homeomorphic_UNIV_UNIV:
  shows (UNIV::'a set) homeomorphic (UNIV::'b set)  $\longleftrightarrow$ 
     $DIM('a::euclidean\_space) = DIM('b::euclidean\_space)$ 
  by (simp add: homeomorphic_subspaces_eq)

lemma simply_connected_sphere_gen:
  assumes convex S bounded S and  $\exists \mathfrak{z}. \mathfrak{z} \leq \text{aff\_dim } S$ 
  shows simply_connected(rel_frontier S)
proof –
  have pa: path_connected (rel_frontier S)
    using assms by (simp add: path_connected_sphere_gen)
  show ?thesis
  proof (clarsimp simp add: simply_connected_eq_contractible_circlemap pa)
    fix f
    assume f: continuous_on (sphere (0::complex) 1) f  $f \restriction \text{sphere } 0 \ 1 \subseteq \text{rel\_frontier } S$ 
    have eq: sphere (0::complex) 1 = rel_frontier(cball 0 1)
      by simp
    have convex (cball (0::complex) 1)
      by (rule convex_cball)
    then obtain c where homotopic_with_canon ( $\lambda z. \text{True}$ ) (sphere (0::complex) 1) (rel_frontier S) f ( $\lambda x. c$ )
    apply (rule inessential_spheremap_lowdim_gen [OF __ bounded_cball <convex S> <bounded S>, where f=f])
    using f  $\exists$ 
    apply (auto simp: aff_dim_cball)
    done
    then show  $\exists a. \text{homotopic\_with\_canon } (\lambda h. \text{True}) (\text{sphere } 0 \ 1) (\text{rel\_frontier } S) f (\lambda x. a)$ 
      by blast
  qed
qed

```

6.42.9 more invariance of domain

```

proposition invariance_of_domain_sphere_affine_set_gen:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes contf: continuous_on S f and inj: inj_on f S and fm:  $f \restriction S \subseteq T$ 
    and U: bounded U convex U
    and affine T and affTU: aff_dim T < aff_dim U
    and ope: openin (top_of_set (rel_frontier U)) S
  shows openin (top_of_set T) (f  $\restriction S$ )
proof (cases rel_frontier U = {})
  case True
  then show ?thesis
    using ope openin_subset by force
next
  case False

```



```

obtain b c where b:  $b \in \text{rel\_frontier } U$  and c:  $c \in \text{rel\_frontier } U$  and  $b \neq c$ 
using  $\langle \text{bounded } U \rangle$  rel_frontier_not_sing [of U] subset_singletonD False by
fastforce
obtain V :: 'a set where affine V and affV:  $\text{aff\_dim } V = \text{aff\_dim } U - 1$ 
proof (rule choose_affine_subset [OF affine_UNIV])
show  $-1 \leq \text{aff\_dim } U - 1$ 
by (metis aff_dim_empty aff_dim_geq aff_dim_negative_iff affTU diff_0
diff_right_mono not_le)
show  $\text{aff\_dim } U - 1 \leq \text{aff\_dim } (UNIV :: 'a \text{ set})$ 
by (metis aff_dim_UNIV aff_dim_le_DIM le_cases not_le zle_diff1_eq)
qed auto
have SU:  $S \subseteq \text{rel\_frontier } U$ 
using ope_openin_imp_subset by auto
have homb:  $\text{rel\_frontier } U - \{b\}$  homeomorphic V
and homc:  $\text{rel\_frontier } U - \{c\}$  homeomorphic V
using homeomorphic_punctured_sphere_affine_gen [of U _ V]
by (simp_all add:  $\langle \text{affine } V \rangle$  affV U b c)
then obtain g h j k
where gh: homeomorphism  $(\text{rel\_frontier } U - \{b\})$  V g h
and jk: homeomorphism  $(\text{rel\_frontier } U - \{c\})$  V j k
by (auto simp: homeomorphic_def)
with SU have hgsub:  $(h \circ g \circ (S - \{b\})) \subseteq S$  and kjsub:  $(k \circ j \circ (S - \{c\})) \subseteq S$ 
by (simp_all add: homeomorphism_def subset_eq)
have [simp]:  $\text{aff\_dim } T \leq \text{aff\_dim } V$ 
by (simp add: affTU affV)
have openin  $(\text{top\_of\_set } T)$   $((f \circ h) \circ g \circ (S - \{b\}))$ 
proof (rule invariance_of_domain_affine_sets [OF _  $\langle \text{affine } V \rangle$ ])
have openin  $(\text{top\_of\_set } (\text{rel\_frontier } U - \{b\}))$   $(S - \{b\})$ 
by (meson Diff_mono Diff_subset SU ope_openin_delete openin_subset_trans
order_refl)
then show openin  $(\text{top\_of\_set } V)$   $(g \circ (S - \{b\}))$ 
by (rule homeomorphism_imp_open_map [OF gh])
show continuous_on  $(g \circ (S - \{b\}))$   $(f \circ h)$ 
proof (rule continuous_on_compose)
show continuous_on  $(g \circ (S - \{b\}))$  h
by (meson Diff_mono SU homeomorphism_def homeomorphism_of_subsets
gh set_eq_subset)
qed (use contf continuous_on_subset hgsub in blast)
show inj_on  $(f \circ h)$   $(g \circ (S - \{b\}))$ 
using kjsub
apply (clarsimp simp add: inj_on_def)
by (metis SU b homeomorphism_def inj_onD injf insert_Diff insert_iff gh
rev_subsetD)
show  $(f \circ h) \circ g \circ (S - \{b\}) \subseteq T$ 
by (metis fim image_comp image_mono hgsub subset_trans)
qed (auto simp: assms)
moreover
have openin  $(\text{top\_of\_set } T)$   $((f \circ k) \circ j \circ (S - \{c\}))$ 
proof (rule invariance_of_domain_affine_sets [OF _  $\langle \text{affine } V \rangle$ ])

```

```

show openin (top_of_set V) (j ‘ (S - {c}))
  by (meson Diff_mono Diff_subset SU ope openin_delete openin_subset_trans
order_refl homeomorphism_imp_open_map [OF jk])
show continuous_on (j ‘ (S - {c})) (f ∘ k)
proof (rule continuous_on_compose)
  show continuous_on (j ‘ (S - {c})) k
  by (meson Diff_mono SU homeomorphism_def homeomorphism_of_subsets
jk set_eq_subset)
qed (use contf continuous_on_subset kjsub in blast)
show inj_on (f ∘ k) (j ‘ (S - {c}))
  using kjsub
apply (clarsimp simp add: inj_on_def)
  by (metis SU c homeomorphism_def inj_onD inj insert_Diff insert_iff jk
rev_subsetD)
show (f ∘ k) ‘ j ‘ (S - {c}) ⊆ T
  by (metis fim image_comp image_mono kjsub subset_trans)
qed (auto simp: assms)
ultimately have openin (top_of_set T) ((f ∘ h) ‘ g ‘ (S - {b}) ∪ ((f ∘ k) ‘ j
‘ (S - {c})))
  by (rule openin_Un)
moreover have (f ∘ h) ‘ g ‘ (S - {b}) = f ‘ (S - {b})
proof -
  have h ‘ g ‘ (S - {b}) = (S - {b})
  proof
    show h ‘ g ‘ (S - {b}) ⊆ S - {b}
    using homeomorphism_apply1 [OF gh] SU
    by (fastforce simp add: image_iff image_subset_iff)
    show S - {b} ⊆ h ‘ g ‘ (S - {b})
    apply clarify
    by (metis SU subsetD homeomorphism_apply1 [OF gh] image_iff mem-
ber_remove remove_def)
  qed
then show ?thesis
  by (metis image_comp)
qed
moreover have (f ∘ k) ‘ j ‘ (S - {c}) = f ‘ (S - {c})
proof -
  have k ‘ j ‘ (S - {c}) = (S - {c})
  proof
    show k ‘ j ‘ (S - {c}) ⊆ S - {c}
    using homeomorphism_apply1 [OF jk] SU
    by (fastforce simp add: image_iff image_subset_iff)
    show S - {c} ⊆ k ‘ j ‘ (S - {c})
    apply clarify
    by (metis SU subsetD homeomorphism_apply1 [OF jk] image_iff mem-
ber_remove remove_def)
  qed
then show ?thesis
  by (metis image_comp)

```

```

qed
moreover have  $f^{-1}(S - \{b\}) \cup f^{-1}(S - \{c\}) = f^{-1}(S)$ 
  using  $\langle b \neq c \rangle$  by blast
ultimately show ?thesis
  by simp
qed

```

```

lemma invariance_of_domain_sphere_affine_set:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $\text{contf: continuous\_on } S \text{ f}$  and  $\text{injf: inj\_on } f \text{ } S$  and  $\text{fim: } f^{-1} S \subseteq T$ 
    and  $r \neq 0$  affine  $T$  and  $\text{affTU: aff\_dim } T < \text{DIM}('a)$ 
    and  $\text{oep: openin (top\_of\_set (sphere a r)) } S$ 
  shows  $\text{openin (top\_of\_set } T) (f^{-1} S)$ 
proof (cases  $\text{sphere a r} = \{\}$ )
  case True
  then show ?thesis
    using ope openin_subset by force
next
  case False
  show ?thesis
  proof (rule invariance_of_domain_sphere_affine_set_gen [OF contf injf fim
    bounded_cball convex_cball  $\langle \text{affine } T \rangle$ ])
    show  $\text{aff\_dim } T < \text{aff\_dim (cball a r)}$ 
      by (metis False affTU aff_dim_cball assms(4) linorder_cases sphere_empty)
    show  $\text{openin (top\_of\_set (rel\_frontier (cball a r))) } S$ 
      by (simp add:  $\langle r \neq 0 \rangle$  ope)
  qed
qed

```

```

lemma no_embedding_sphere_lowdim:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $\text{contf: continuous\_on (sphere a r) } f$  and  $\text{injf: inj\_on } f \text{ (sphere a r)}$ 
  and  $r > 0$ 
  shows  $\text{DIM}('a) \leq \text{DIM}('b)$ 
proof -
  have False if  $\text{DIM}('a) > \text{DIM}('b)$ 
  proof -
    have compact  $(f^{-1} \text{ sphere a r})$ 
      using compact_continuous_image
      by (simp add: compact_continuous_image contf)
    then have  $\neg \text{open } (f^{-1} \text{ sphere a r})$ 
      using compact_open
      by (metis assms(3) image_is_empty not_less_iff_gr_or_eq sphere_eq_empty)
    then show False
      using invariance_of_domain_sphere_affine_set [OF contf injf subset_UNIV]
       $\langle r > 0 \rangle$ 
      by (metis aff_dim_UNIV affine_UNIV less_irrefl of_nat_less_iff open_openin
        openin_subtopology_self subtopology_UNIV that)
  qed

```

3120

```

qed
then show ?thesis
  using not_less by blast
qed

```

```

lemma simply_connected_sphere:
  fixes a :: 'a::euclidean_space
  assumes  $3 \leq \text{DIM}('a)$ 
  shows simply_connected(sphere a r)
proof (cases rule: linorder_cases [of r 0])
  case less
  then show ?thesis by simp
next
  case equal
  then show ?thesis by (auto simp: convex_imp_simply_connected)
next
  case greater
  then show ?thesis
    using simply_connected_sphere_gen [of cball a r] assms
    by (simp add: aff_dim_cball)
qed

```

```

lemma simply_connected_sphere_eq:
  fixes a :: 'a::euclidean_space
  shows simply_connected(sphere a r)  $\longleftrightarrow 3 \leq \text{DIM}('a) \vee r \leq 0$  (is ?lhs =
  ?rhs)
proof (cases r  $\leq 0$ )
  case True
  have simply_connected (sphere a r)
    using True less_eq_real_def by (auto intro: convex_imp_simply_connected)
  with True show ?thesis by auto
next
  case False
  show ?thesis
  proof
    assume L: ?lhs
    have False if  $\text{DIM}('a) = 1 \vee \text{DIM}('a) = 2$ 
      using that
    proof
      assume  $\text{DIM}('a) = 1$ 
      with L show False
        using connected_sphere_eq simply_connected_imp_connected
        by (metis False Suc_1 not_less_eq_eq order_refl)
    next
      assume  $\text{DIM}('a) = 2$ 
      then have sphere a r homeomorphic_sphere (0::complex) 1
        by (metis DIM_complex False homeomorphic_spheres_gen not_less zero_less_one)
      then have simply_connected(sphere (0::complex) 1)
        using L homeomorphic_simply_connected_eq by blast
    end
  end

```

```

    then obtain  $a::\text{complex}$  where homotopic_with_canon ( $\lambda h. \text{True}$ ) (sphere 0
1) (sphere 0 1) id ( $\lambda x. a$ )
    by (metis continuous_on_id' id_apply image_id subset_refl simply_connected_eq_contractible_circlemap)
    then show False
    using contractible_sphere contractible_def not_one_le_zero by blast
qed
with False show ?rhs
  apply simp
  by (metis DIM_ge_Suc0 le_antisym not_less_eq_eq numeral_2_eq_2 numeral_3_eq_3)
next
  assume ?rhs
  with False show ?lhs by (simp add: simply_connected_sphere)
qed
qed

```

```

lemma simply_connected_punctured_universe_eq:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  shows simply_connected( $- \{a\}$ )  $\longleftrightarrow 3 \leq \text{DIM}('a)$ 
proof -
  have [simp]:  $a \in \text{rel\_interior} (\text{cball } a \ 1)$ 
  by (simp add: rel_interior_nonempty_interior)
  have [simp]:  $\text{affine hull cball } a \ 1 - \{a\} = -\{a\}$ 
  by (metis Compl_eq_Diff_UNIV aff_dim_cball aff_dim_lt_full not_less_iff_gr_or_eq zero_less_one)
  have sphere  $a \ 1$  homotopy_eqv  $- \{a\}$ 
  using homotopy_eqv_rel_frontier_punctured_affine_hull [of cball  $a \ 1 \ a$ ] by
auto
  then have simply_connected( $- \{a\}$ )  $\longleftrightarrow$  simply_connected(sphere  $a \ 1$ )
  using homotopy_eqv_simple_connectedness by blast
  also have ...  $\longleftrightarrow 3 \leq \text{DIM}('a)$ 
  by (simp add: simply_connected_sphere_eq)
  finally show ?thesis .
qed

```

```

lemma not_simply_connected_circle:
  fixes  $a :: \text{complex}$ 
  shows  $0 < r \implies \neg \text{simply\_connected}(\text{sphere } a \ r)$ 
by (simp add: simply_connected_sphere_eq)

```

```

proposition simply_connected_punctured_convex:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  assumes convex  $S$  and  $3: 3 \leq \text{aff\_dim } S$ 
  shows simply_connected( $S - \{a\}$ )
proof (cases  $a \in \text{rel\_interior } S$ )
  case True
  then obtain  $e$  where  $a \in S$   $0 < e$  and  $e: \text{cball } a \ e \cap \text{affine hull } S \subseteq S$ 

```

```

    by (auto simp: rel_interior_cball)
  have con: convex (cball a e ∩ affine hull S)
    by (simp add: convex_Int)
  have bo: bounded (cball a e ∩ affine hull S)
    by (simp add: bounded_Int)
  have affine_hull_S ∩ interior (cball a e) ≠ {}
    using ⟨0 < e⟩ ⟨a ∈ S⟩ hull_subset by fastforce
  then have 3 ≤ aff_dim (affine hull S ∩ cball a e)
    by (simp add: 3_aff_dim_convex_Int_nonempty_interior [OF convex_affine_hull])
  also have ... = aff_dim (cball a e ∩ affine hull S)
    by (simp add: Int_commute)
  finally have 3 ≤ aff_dim (cball a e ∩ affine hull S) .
  moreover have rel_frontier (cball a e ∩ affine hull S) homotopy_eqv S - {a}
  proof (rule homotopy_eqv_rel_frontier_punctured_convex)
    show a ∈ rel_interior (cball a e ∩ affine hull S)
      by (meson IntI Int_mono ⟨a ∈ S⟩ ⟨0 < e⟩ e ⟨cball a e ∩ affine hull S
        ⊆ S⟩ ball_subset_cball centre_in_cball dual_order.strict_implies_order hull_inc
        hull_mono mem_rel_interior_ball)
    have closed (cball a e ∩ affine hull S)
      by blast
    then show rel_frontier (cball a e ∩ affine hull S) ⊆ S
      by (metis Diff_subset closure_closed dual_order.trans e rel_frontier_def)
    show S ⊆ affine hull (cball a e ∩ affine hull S)
      by (metis (no_types, lifting) IntI ⟨a ∈ S⟩ ⟨0 < e⟩ affine_hull_convex_Int_nonempty_interior
        centre_in_ball convex_affine_hull empty_iff hull_subset inf_commute interior_cball
        subsetCE subsetI)
    qed (auto simp: assms con bo)
  ultimately show ?thesis
    using homotopy_eqv_simple_connectedness simply_connected_sphere_gen [OF
    con bo]
    by blast
next
case False
then have rel_interior S ⊆ S - {a}
  by (simp add: False rel_interior_subset subset_Diff_insert)
moreover have S - {a} ⊆ closure S
  by (meson Diff_subset closure_subset subset_trans)
ultimately show ?thesis
  by (metis contractible_imp_simply_connected contractible_convex_tweak_boundary_points
  [OF ⟨convex S⟩])
qed

corollary simply_connected_punctured_universe:
  fixes a :: 'a::euclidean_space
  assumes 3 ≤ DIM('a)
  shows simply_connected(− {a})
proof −
  have [simp]: affine hull cball a 1 = UNIV
    by (simp add: aff_dim_cball affine_hull_UNIV)

```

```

have a ∈ rel_interior (cball a 1)
  by (simp add: rel_interior_interior)
then
  have simply_connected (rel_frontier (cball a 1)) = simply_connected (affine hull
cball a 1 - {a})
  using homotopy_eqv_rel_frontier_punctured_affine_hull homotopy_eqv_simple_connectedness
by blast
  then show ?thesis
  using simply_connected_sphere [of a 1, OF assms] by (auto simp: Compl_eq_Diff_UNIV)
qed

```

6.42.10 The power, squaring and exponential functions as covering maps

```

proposition covering_space_power_punctured_plane:
  assumes 0 < n
  shows covering_space (- {0}) (λz::complex. z^n) (- {0})
proof -
  consider n = 1 | 2 ≤ n using assms by linarith
  then obtain e where 0 < e
    and e: ⋀w z. cmod(w - z) < e * cmod z ⟹ (w^n = z^n ⟷ w =
z)
  proof cases
    assume n = 1 then show ?thesis
      by (rule_tac e=1 in that) auto
  next
    assume 2 ≤ n
    have eq_if_pow_eq:
      w = z if lt: cmod (w - z) < 2 * sin (pi / real n) * cmod z
      and eq: w^n = z^n for w z
    proof (cases z = 0)
      case True with eq assms show ?thesis by (auto simp: power_0_left)
    next
      case False
      then have z ≠ 0 by auto
      have (w/z)^n = 1
        by (metis False divide_self_if eq power_divide power_one)
      then obtain j where j: w / z = exp (2 * of_real pi * i * j / n) and j < n
        using Suc_leI assms ⟨2 ≤ n⟩ complex_roots_unity [THEN eqset_imp_iff,
of n w/z]
      by force
      have cmod (w/z - 1) < 2 * sin (pi / real n)
        using lt assms ⟨z ≠ 0⟩ by (simp add: field_split_simps norm_divide)
      then have cmod (exp (i * of_real (2 * pi * j / n)) - 1) < 2 * sin (pi / real
n)
        by (simp add: j field_simps)
      then have 2 * |sin((2 * pi * j / n) / 2)| < 2 * sin (pi / real n)
        by (simp only: dist_exp_i_1)
      then have sin_less: sin((pi * j / n)) < sin (pi / real n)

```

```

    by (simp add: field_simps)
  then have  $w / z = 1$ 
  proof (cases  $j = 0$ )
    case True then show ?thesis by (auto simp: j)
  next
    case False
    then have  $\sin (pi / \text{real } n) \leq \sin((pi * j / n))$ 
    proof (cases  $j / n \leq 1/2$ )
      case True
      show ?thesis
        using  $\langle j \neq 0 \rangle \langle j < n \rangle$  True
        by (intro sin_monotone_2pi_le) (auto simp: field_simps intro: order_trans
[of _ 0])
    next
      case False
      then have seq:  $\sin(pi * j / n) = \sin(pi * (n - j) / n)$ 
      using  $\langle j < n \rangle$  by (simp add: algebra_simps diff_divide_distrib
of_nat_diff)
      show ?thesis
        unfolding seq
        using  $\langle j < n \rangle$  False
        by (intro sin_monotone_2pi_le) (auto simp: field_simps intro: order_trans
[of _ 0])
    qed
  qed
  with sin_less show ?thesis by force
qed
then show ?thesis by simp
qed
show ?thesis
proof
  show  $0 < 2 * \sin (pi / \text{real } n)$ 
  by (force simp:  $\langle 2 \leq n \rangle$  sin_pi_divide_n_gt_0)
qed (meson eq_if_pow_eq)
qed
have zn1: continuous_on  $(-\{0\})$   $(\lambda z::\text{complex}. z^n)$ 
  by (rule continuous_intros)+
have zn2:  $(\lambda z::\text{complex}. z^n) '(-\{0\}) = -\{0\}$ 
  using assms by (auto simp: image_def elim: exists_complex_root_nonzero
[where  $n = n$ ])
have zn3:  $\exists T. z^n \in T \wedge \text{open } T \wedge 0 \notin T \wedge$ 
   $(\exists v. \bigcup v = -\{0\} \cap (\lambda z. z^n) 'T \wedge$ 
   $(\forall u \in v. \text{open } u \wedge 0 \notin u) \wedge$ 
  pairwise disjoint  $v \wedge$ 
   $(\forall u \in v. \exists (\text{homeomorphism } u \ T \ (\lambda z. z^n))))$ 
  if  $z \neq 0$  for  $z::\text{complex}$ 
proof -
  define d where  $d \equiv \min (1/2) (e/4) * \text{norm } z$ 
  have  $0 < d$ 
  by (simp add: d_def  $\langle 0 < e \rangle \langle z \neq 0 \rangle$ )

```



```

have iff  $x \text{ eq } y: \hat{x}^n = \hat{y}^n \longleftrightarrow x = y$ 
  if eq:  $\hat{w}^n = \hat{z}^n$  and  $x: x \in \text{ball } w \ d$  and  $y: y \in \text{ball } w \ d$  for  $w \ x \ y$ 
proof -
  have [simp]:  $\text{norm } z = \text{norm } w$  using that
    by (simp add: assms power_eq_imp_eq_norm)
  show ?thesis
  proof (cases  $w = 0$ )
    case True with  $\langle z \neq 0 \rangle$  assms eq
      show ?thesis by (auto simp: power_0_left)
    next
      case False
      have  $\text{cmod } (x - y) < 2 * d$ 
        using  $x \ y$ 
        by (simp add: dist_norm [symmetric]) (metis dist_commute mult_2
dist_triangle_less_add)
      also have  $\dots \leq 2 * e / 4 * \text{norm } w$ 
        using  $\langle e > 0 \rangle$  by (simp add: d_def min_mult_distrib_right)
      also have  $\dots = e * (\text{cmod } w / 2)$ 
        by simp
      also have  $\dots \leq e * \text{cmod } y$ 
        proof (rule mult_left_mono)
          have  $\text{cmod } (w - y) < \text{cmod } w / 2 \implies \text{cmod } w / 2 \leq \text{cmod } y$ 
            by (metis (no_types) dist_0_norm dist_norm norm_triangle_half_l
not_le order_less_irrefl)
          then show  $\text{cmod } w / 2 \leq \text{cmod } y$ 
            using  $y$  by (simp add: dist_norm d_def min_mult_distrib_right)
        qed (use  $\langle e > 0 \rangle$  in auto)
      finally have  $\text{cmod } (x - y) < e * \text{cmod } y$  .
      then show ?thesis by (rule e)
    qed
  qed
then have inj:  $\text{inj\_on } (\lambda w. \hat{w}^n) (\text{ball } z \ d)$ 
  by (simp add: inj_on_def)
have cont:  $\text{continuous\_on } (\text{ball } z \ d) (\lambda w. \hat{w}^n)$ 
  by (intro continuous_intros)
have noncon:  $\neg (\lambda w::\text{complex}. \hat{w}^n) \text{ constant\_on } \text{UNIV}$ 
  by (metis UNIV_I assms constant_on_def power_one zero_neq_one zero_power)
have im_eq:  $(\lambda w. \hat{w}^n) ' \text{ball } z' \ d = (\lambda w. \hat{w}^n) ' \text{ball } z \ d$ 
  if  $z': \hat{z}'^n = \hat{z}^n$  for  $z'$ 
proof -
  have  $\text{nz}': \text{norm } z' = \text{norm } z$  using that assms power_eq_imp_eq_norm by
blast
  have  $(w \in (\lambda w. \hat{w}^n) ' \text{ball } z' \ d) = (w \in (\lambda w. \hat{w}^n) ' \text{ball } z \ d)$  for  $w$ 
  proof (cases  $w=0$ )
    case True with assms show ?thesis
      by (simp add: image_def ball_def nz')
    next
      case False
      have  $z' \neq 0$  using  $\langle z \neq 0 \rangle \ \text{nz}'$  by force

```

```

have 1: (z*x / z')^n = x^n if x ≠ 0 for x
  using z' that by (simp add: field_simps ⟨z ≠ 0⟩)
have 2: cmod (z - z * x / z') = cmod (z' - x) if x ≠ 0 for x
proof -
  have cmod (z - z * x / z') = cmod z * cmod (1 - x / z')
  by (metis (no_types) ab_semigroup_mult_class.mult_ac(1) divide_complex_def
mult.right_neutral norm_mult right_diff_distrib')
  also have ... = cmod z' * cmod (1 - x / z')
  by (simp add: nz')
  also have ... = cmod (z' - x)
  by (simp add: ⟨z' ≠ 0⟩ diff_divide_eq_iff norm_divide)
  finally show ?thesis .
qed
have 3: (z'*x / z)^n = x^n if x ≠ 0 for x
  using z' that by (simp add: field_simps ⟨z ≠ 0⟩)
have 4: cmod (z' - z' * x / z) = cmod (z - x) if x ≠ 0 for x
proof -
  have cmod (z * (1 - x * inverse z)) = cmod (z - x)
  by (metis ⟨z ≠ 0⟩ diff_divide_distrib divide_complex_def divide_self_if
nonzero_eq_divide_eq semiring_normalization_rules(7))
  then show ?thesis
  by (metis (no_types) mult.assoc divide_complex_def mult.right_neutral
norm_mult nz' right_diff_distrib')
qed
show ?thesis
  by (simp add: set_eq_iff image_def ball_def) (metis 1 2 3 4 diff_zero
dist_norm nz')
qed
then show ?thesis by blast
qed

have ex_ball: ∃ B. (∃ z'. B = ball z' d ∧ z'^n = z^n) ∧ x ∈ B
  if x ≠ 0 and eq: x^n = w^n and dzw: dist z w < d for x w
proof -
  have w ≠ 0 by (metis assms power_eq_0_iff that(1) that(2))
  have [simp]: cmod x = cmod w
  using assms power_eq_imp_eq_norm eq by blast
  have [simp]: cmod (x * z / w - x) = cmod (z - w)
  proof -
    have cmod (x * z / w - x) = cmod x * cmod (z / w - 1)
    by (metis (no_types) mult.right_neutral norm_mult right_diff_distrib'
times_divide_eq_right)
    also have ... = cmod w * cmod (z / w - 1)
    by simp
    also have ... = cmod (z - w)
    by (simp add: ⟨w ≠ 0⟩ divide_diff_eq_iff nonzero_norm_divide)
  finally show ?thesis .
qed
show ?thesis

```

```

proof (intro exI conjI)
  show  $(z / w * x)^{\wedge n} = z^{\wedge n}$ 
    by (metis  $\langle w \neq 0 \rangle$  eq nonzero_eq_divide_eq power_mult_distrib)
  show  $x \in \text{ball } (z / w * x) \ d$ 
    using  $\langle d > 0 \rangle$  that
    by (simp add: ball_eq_ball_iff  $\langle z \neq 0 \rangle \langle w \neq 0 \rangle$  field_simps) (simp add:
dist_norm)
  qed auto
qed

show ?thesis
proof (rule exI, intro conjI)
  show  $z^{\wedge n} \in (\lambda w. w^{\wedge n}) \text{ `ball } z \ d$ 
    using  $\langle d > 0 \rangle$  by simp
  show open  $((\lambda w. w^{\wedge n}) \text{ `ball } z \ d)$ 
    by (rule invariance_of_domain [OF cont_open_ball inj])
  show  $0 \notin (\lambda w. w^{\wedge n}) \text{ `ball } z \ d$ 
    using  $\langle z \neq 0 \rangle$  assms by (force simp: d_def)
  show  $\exists v. \bigcup v = -\{0\} \cap (\lambda z. z^{\wedge n}) - (\lambda w. w^{\wedge n}) \text{ `ball } z \ d \wedge$ 
     $(\forall u \in v. \text{open } u \wedge 0 \notin u) \wedge$ 
    disjoint v  $\wedge$ 
     $(\forall u \in v. \text{Ex } (\text{homeomorphism } u ((\lambda w. w^{\wedge n}) \text{ `ball } z \ d) (\lambda z. z^{\wedge n})))$ 
proof (rule exI, intro ballI conjI)
  show  $\bigcup \{\text{ball } z' \ d \mid z'. z'^{\wedge n} = z^{\wedge n}\} = -\{0\} \cap (\lambda z. z^{\wedge n}) - (\lambda w. w^{\wedge n}) \text{ `ball } z \ d$ 
    (is ?l = ?r)
proof
  have  $\bigwedge z'. \text{cmod } z' < d \implies z'^{\wedge n} \neq z^{\wedge n}$ 
    by (auto simp add: assms d_def power_eq_imp_eq_norm that)
  then show  $?l \subseteq ?r$ 
    by auto (metis im_eq_image_eqI mem_ball)
  show  $?r \subseteq ?l$ 
    by auto (meson ex_ball)
qed
show  $\bigwedge u. u \in \{\text{ball } z' \ d \mid z'. z'^{\wedge n} = z^{\wedge n}\} \implies 0 \notin u$ 
    by (force simp add: assms d_def power_eq_imp_eq_norm that)

show disjoint  $\{\text{ball } z' \ d \mid z'. z'^{\wedge n} = z^{\wedge n}\}$ 
proof (clarsimp simp add: pairwise_def disjnt_iff)
  fix  $\xi \ \zeta \ x$ 
  assume  $\xi^{\wedge n} = z^{\wedge n} \ \zeta^{\wedge n} = z^{\wedge n} \ \text{ball } \xi \ d \neq \text{ball } \zeta \ d$ 
  and  $\text{dist } \xi \ x < d \ \text{dist } \zeta \ x < d$ 
  then have  $\text{dist } \xi \ \zeta < d + d$ 
    using dist_triangle_less_add by blast
  then have  $\text{cmod } (\xi - \zeta) < 2 * d$ 
    by (simp add: dist_norm)
  also have  $\dots \leq e * \text{cmod } z$ 
    using mult_right_mono  $\langle 0 < e \rangle$  that by (auto simp: d_def)
  finally have  $\text{cmod } (\xi - \zeta) < e * \text{cmod } z$  .
  with e have  $\xi = \zeta$ 

```

```

    by (metis ⟨ $\xi^n = z^n$ ⟩ ⟨ $\zeta^n = z^n$ ⟩ assms power_eq_imp_eq_norm)
  then show False
    using ⟨ball  $\xi$   $d \neq$  ball  $\zeta$   $d$ ⟩ by blast
qed
show Ex (homeomorphism u (( $\lambda w. w^n$ ) ‘ ball  $z$   $d$ ) ( $\lambda z. z^n$ ))
  if  $u \in \{\text{ball } z' \ d \mid z'. z'^n = z^n\}$  for u
proof (rule invariance_of_domain_homeomorphism [of u  $\lambda z. z^n$ ])
  show open u
    using that by auto
  show continuous_on u ( $\lambda z. z^n$ )
    by (intro continuous_intros)
  show inj_on ( $\lambda z. z^n$ ) u
    using that by (auto simp: iff_x_eq_y inj_on_def)
  show  $\bigwedge g. \text{homeomorphism } u ((\lambda z. z^n) ' u) (\lambda z. z^n) \ g \implies \text{Ex}$ 
    (homeomorphism u (( $\lambda w. w^n$ ) ‘ ball  $z$   $d$ ) ( $\lambda z. z^n$ ))
    using im_eq that by clarify metis
  qed auto
  qed auto
  qed
  qed
show ?thesis
  using assms
  apply (simp add: covering_space_def zn1 zn2)
  apply (subst zn2 [symmetric])
  apply (simp add: openin_open_eq open_Compl zn3)
  done
qed

corollary covering_space_square_punctured_plane:
  covering_space (– {0}) ( $\lambda z::\text{complex}. z^2$ ) (– {0})
  by (simp add: covering_space_power_punctured_plane)

proposition covering_space_exp_punctured_plane:
  covering_space UNIV ( $\lambda z::\text{complex}. \exp z$ ) (– {0})
proof (simp add: covering_space_def, intro conjI ballI)
  show continuous_on UNIV ( $\lambda z::\text{complex}. \exp z$ )
    by (rule continuous_on_exp [OF continuous_on_id])
  show range exp = – {0::complex}
    by auto (metis exp_Ln range_eqI)
  show  $\exists T. z \in T \wedge \text{openin } (\text{top\_of\_set } (– \{0\})) \ T \wedge$ 
    ( $\exists v. \bigcup v = \exp - ' T \wedge (\forall u \in v. \text{open } u) \wedge \text{disjoint } v \wedge$ 
    ( $\forall u \in v. \exists q. \text{homeomorphism } u \ T \ \exp \ q$ ))
    if  $z \in - \{0::\text{complex}\}$  for z
  proof –
    have  $z \neq 0$ 
      using that by auto
    have ball (Ln  $z$ ) 1  $\subseteq$  ball (Ln  $z$ )  $\pi$ 
      using pi_ge_two by (simp add: ball_subset_ball_iff)

```

```

then have inj_exp: inj_on exp (ball (Ln z) 1)
  using inj_on_exp_pi inj_on_subset by blast
define V where V  $\equiv$  range ( $\lambda n. (\lambda x. x + \text{of\_real } (2 * \text{of\_int } n * \text{pi}) * i)$  '
(ball(Ln z) 1))
show ?thesis
proof (intro exI conjI)
  show  $z \in \text{exp}$  ' (ball(Ln z) 1)
    by (metis  $\langle z \neq 0 \rangle$  centre_in_ball exp_Ln rev_image_eqI zero_less_one)
  have open ( $-\{0::\text{complex}\}$ )
    by blast
  with inj_exp show openin (top_of_set ( $-\{0\}$ )) (exp ' ball (Ln z) 1)
    by (auto simp: openin_open_eq invariance_of_domain continuous_on_exp
[OF continuous_on_id])
  show  $\bigcup V = \text{exp} - \text{exp ' ball (Ln z) 1}$ 
    by (force simp: V_def Complex_Transcendental.exp_eq_image_iff)
  show  $\forall V \in V. \text{open } V$ 
    by (auto simp: V_def inj_on_def continuous_intros invariance_of_domain)
  have xy:  $2 \leq \text{cmod } (2 * \text{of\_int } x * \text{of\_real } \text{pi} * i - 2 * \text{of\_int } y * \text{of\_real } \text{pi} * i)$ 
    if  $x < y$  for  $x y$ 
  proof -
    have  $1 \leq \text{abs } (x - y)$ 
      using that by linarith
    then have  $1 \leq \text{cmod } (\text{of\_int } x - \text{of\_int } y) * 1$ 
      by (metis mult.right_neutral norm_of_int of_int_1_le_iff of_int_abs
of_int_diff)
    also have  $\dots \leq \text{cmod } (\text{of\_int } x - \text{of\_int } y) * \text{of\_real } \text{pi}$ 
      using pi_ge_two
      by (intro mult_left_mono) auto
    also have  $\dots \leq \text{cmod } ((\text{of\_int } x - \text{of\_int } y) * \text{of\_real } \text{pi} * i)$ 
      by (simp add: norm_mult)
    also have  $\dots \leq \text{cmod } (\text{of\_int } x * \text{of\_real } \text{pi} * i - \text{of\_int } y * \text{of\_real } \text{pi} * i)$ 
      by (simp add: algebra_simps)
    finally have  $1 \leq \text{cmod } (\text{of\_int } x * \text{of\_real } \text{pi} * i - \text{of\_int } y * \text{of\_real } \text{pi} * i)$ 
      .
    then have  $2 * 1 \leq \text{cmod } (2 * (\text{of\_int } x * \text{of\_real } \text{pi} * i - \text{of\_int } y * \text{of\_real } \text{pi} * i))$ 
      by (metis mult_le_cancel_left_pos norm_mult_numeral1 zero_less_numeral)
    then show ?thesis
      by (simp add: algebra_simps)
  qed
show disjoint V
  apply (clarify simp add: V_def pairwise_def disjnt_def add commute [of
_ x*y for x y]
ball_eq_ball_iff intro!: disjoint_ballI)
  apply (auto simp: dist_norm neq_iff)
  by (metis norm_minus_commute xy)+
show  $\forall u \in V. \exists q. \text{homeomorphism } u (\text{exp ' ball (Ln z) 1}) \text{exp } q$ 
proof

```

```

fix u
assume u ∈ V
then obtain n where n: u = (λx. x + of_real (2 * of_int n * pi) * i) '
(ball (Ln z) 1)
  by (auto simp: V_def)
have compact (cball (Ln z) 1)
  by simp
moreover have continuous_on (cball (Ln z) 1) exp
  by (rule continuous_on_exp [OF continuous_on_id])
moreover have inj_on exp (cball (Ln z) 1)
  apply (rule inj_on_subset [OF inj_on_exp_pi [of Ln z]])
  using pi_ge_two by (simp add: cball_subset_ball_iff)
ultimately obtain γ where hom: homeomorphism (cball (Ln z) 1) (exp '
cball (Ln z) 1) exp γ
  using homeomorphism_compact by blast
have eq1: exp ' u = exp ' ball (Ln z) 1
  apply (auto simp: algebra_simps n)
  apply (rule_tac x = _ + i * (of_int n * (of_real pi * 2)) in image_eqI)
  apply (auto simp: image_iff)
  done
have γexp: γ (exp x) + 2 * of_int n * of_real pi * i = x if x ∈ u for x
proof -
  have exp x = exp (x - 2 * of_int n * of_real pi * i)
    by (simp add: exp_eq)
  then have γ (exp x) = γ (exp (x - 2 * of_int n * of_real pi * i))
    by simp
  also have ... = x - 2 * of_int n * of_real pi * i
    using ⟨x ∈ u⟩ by (auto simp: n intro: homeomorphism_apply1 [OF hom])
  finally show ?thesis
    by simp
qed
have exp2n: exp (γ (exp x) + 2 * of_int n * complex_of_real pi * i) = exp
x
  if dist (Ln z) x < 1 for x
  using that by (auto simp: exp_eq homeomorphism_apply1 [OF hom])
have continuous_on (exp ' ball (Ln z) 1) γ
  by (meson ball_subset_cball continuous_on_subset hom homeomor-
phism_cont2 image_mono)
then have cont: continuous_on (exp ' ball (Ln z) 1) (λx. γ x + 2 * of_int
n * complex_of_real pi * i)
  by (intro continuous_intros)
show ∃ q. homeomorphism u (exp ' ball (Ln z) 1) exp q
  apply (rule_tac x=(λx. x + of_real(2 * n * pi) * i) ∘ γ in exI)
  unfolding homeomorphism_def
  apply (intro conjI ballI eq1 continuous_on_exp [OF continuous_on_id])
  apply (auto simp: γexp exp2n cont n)
  apply (force simp: image_iff homeomorphism_apply1 [OF hom])
  done
qed

```

qed
qed
qed

6.42.11 Hence the Borsukian results about mappings into circles

lemma *inessential_eq_continuous_logarithm*:
fixes $f :: 'a::\text{real\_normed\_vector} \Rightarrow \text{complex}$
shows $(\exists a. \text{homotopic\_with\_canon } (\lambda h. \text{True}) S (-\{0\}) f (\lambda t. a)) \longleftrightarrow$
 $(\exists g. \text{continuous\_on } S g \wedge (\forall x \in S. f x = \exp(g x)))$
(is ?lhs $\longleftrightarrow$?rhs)
proof
assume ?lhs **thus** ?rhs
by (metis covering_space_lift_inessential_function covering_space_exp_punctured_plane)
next
assume ?rhs
then obtain g **where** $\text{contg}: \text{continuous\_on } S g$ **and** $f: \bigwedge x. x \in S \implies f x = \exp(g x)$
by metis
obtain a **where** $\text{homotopic\_with\_canon } (\lambda h. \text{True}) S (-\{\text{of\_real } 0\}) (\exp \circ g) (\lambda x. a)$
proof (rule nullhomotopic_through_contractible [OF contg subset_UNIV _ contractible_UNIV])
show $\text{continuous\_on } (UNIV::\text{complex set}) \exp$
by (intro continuous_intros)
show $\text{range } \exp \subseteq -\{0\}$
by auto
qed force
then have $\text{homotopic\_with\_canon } (\lambda h. \text{True}) S (-\{0\}) f (\lambda t. a)$
using f homotopic_with_eq **by** fastforce
then show ?lhs ..
qed

corollary *inessential_imp_continuous_logarithm_circle*:
fixes $f :: 'a::\text{real\_normed\_vector} \Rightarrow \text{complex}$
assumes $\text{homotopic\_with\_canon } (\lambda h. \text{True}) S (\text{sphere } 0 \ 1) f (\lambda t. a)$
obtains g **where** $\text{continuous\_on } S g$ **and** $\bigwedge x. x \in S \implies f x = \exp(g x)$
proof –
have $\text{homotopic\_with\_canon } (\lambda h. \text{True}) S (-\{0\}) f (\lambda t. a)$
using assms $\text{homotopic\_with\_subset\_right}$ **by** fastforce
then show ?thesis
by (metis inessential_eq_continuous_logarithm that)
qed

lemma *inessential_eq_continuous_logarithm_circle*:
fixes $f :: 'a::\text{real\_normed\_vector} \Rightarrow \text{complex}$
shows $(\exists a. \text{homotopic\_with\_canon } (\lambda h. \text{True}) S (\text{sphere } 0 \ 1) f (\lambda t. a)) \longleftrightarrow$

```

      (∃ g. continuous_on S g ∧ (∀ x ∈ S. f x = exp(i * of_real(g x))))
    (is ?lhs ←→ ?rhs)
  proof
    assume L: ?lhs
    then obtain g where contg: continuous_on S g and g: ∧x. x ∈ S ⇒ f x =
      exp(g x)
    using inessential_imp_continuous_logarithm_circle by blast
    have f ' S ⊆ sphere 0 1
    by (metis L homotopic_with_imp_subset1)
    then have ∧x. x ∈ S ⇒ Re (g x) = 0
    using g by auto
    then show ?rhs
    by (rule_tac x=Im ∘ g in exI) (auto simp: Euler g intro: contg continu-
      ous_intros)
  next
    assume ?rhs
    then obtain g where contg: continuous_on S g and g: ∧x. x ∈ S ⇒ f x =
      exp(i * of_real(g x))
    by metis
    obtain a where homotopic_with_canon (λh. True) S (sphere 0 1) ((exp ∘ (λz.
      i * z)) ∘ (of_real ∘ g)) (λx. a)
    proof (rule nullhomotopic_through_contractible)
      show continuous_on S (complex_of_real ∘ g)
      by (intro conjI contg continuous_intros)
      show (complex_of_real ∘ g) ' S ⊆ ℝ
      by auto
      show continuous_on ℝ (exp ∘ (*i))
      by (intro continuous_intros)
      show (exp ∘ (*i)) ' ℝ ⊆ sphere 0 1
      by (auto simp: complex_is_Real_iff)
    qed (auto simp: convex_Reals convex_imp_contractible)
    moreover have ∧x. x ∈ S ⇒ (exp ∘ (*i)) ∘ (complex_of_real ∘ g) x = f x
    by (simp add: g)
    ultimately have homotopic_with_canon (λh. True) S (sphere 0 1) f (λt. a)
    using homotopic_with_eq by force
    then show ?lhs ..
  qed

```

proposition *homotopic_with_sphere_times:*

```

  fixes f :: 'a::real_normed_vector ⇒ complex
  assumes hom: homotopic_with_canon (λx. True) S (sphere 0 1) f g and conth:
    continuous_on S h
    and hin: ∧x. x ∈ S ⇒ h x ∈ sphere 0 1
    shows homotopic_with_canon (λx. True) S (sphere 0 1) (λx. f x * h x) (λx.
      g x * h x)
  proof -
    obtain k where contk: continuous_on ({0..1::real} × S) k
      and kim: k ' ({0..1} × S) ⊆ sphere 0 1
      and k0: ∧x. k(0, x) = f x

```



```

      and k1:  $\bigwedge x. k(1, x) = g x$ 
    using hom by (auto simp: homotopic_with_def)
  show ?thesis
    apply (simp add: homotopic_with)
    apply (rule_tac x= $\lambda z. k z * (h \circ \text{snd}) z$  in exI)
    using kim hin by (fastforce simp: conth norm_mult k0 k1 intro!: contk continuous_intros)+
  qed

```

proposition *homotopic_circlemaps_divide:*

```

  fixes f :: 'a::real_normed_vector  $\Rightarrow$  complex
  shows homotopic_with_canon ( $\lambda x. \text{True}$ ) S (sphere 0 1) f g  $\longleftrightarrow$ 
    continuous_on S f  $\wedge$  f ' S  $\subseteq$  sphere 0 1  $\wedge$ 
    continuous_on S g  $\wedge$  g ' S  $\subseteq$  sphere 0 1  $\wedge$ 
    ( $\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S (\text{sphere } 0 \ 1) (\lambda x. f x / g x)$ 
      ( $\lambda x. c$ ))
  proof -
    have homotopic_with_canon ( $\lambda x. \text{True}$ ) S (sphere 0 1) ( $\lambda x. f x / g x$ ) ( $\lambda x. 1$ )
      if homotopic_with_canon ( $\lambda x. \text{True}$ ) S (sphere 0 1) ( $\lambda x. f x / g x$ ) ( $\lambda x. c$ )
    for c
    proof -
      have S = {}  $\vee$  path_component (sphere 0 1) 1 c
      using homotopic_with_imp_subset2 [OF that] path_connected_sphere [of 0::complex 1]
      by (auto simp: path_connected_component)
      then have homotopic_with_canon ( $\lambda x. \text{True}$ ) S (sphere 0 1) ( $\lambda x. 1$ ) ( $\lambda x. c$ )
      by (simp add: homotopic_constant_maps)
      then show ?thesis
        using homotopic_with_symD homotopic_with_trans that by blast
    qed
    then have *: ( $\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S (\text{sphere } 0 \ 1) (\lambda x. f x / g x)$ 
      ( $\lambda x. c$ ))  $\longleftrightarrow$ 
      homotopic_with_canon ( $\lambda x. \text{True}$ ) S (sphere 0 1) ( $\lambda x. f x / g x$ ) ( $\lambda x. 1$ )
    by auto
    have homotopic_with_canon ( $\lambda x. \text{True}$ ) S (sphere 0 1) f g  $\longleftrightarrow$ 
      continuous_on S f  $\wedge$  f ' S  $\subseteq$  sphere 0 1  $\wedge$ 
      continuous_on S g  $\wedge$  g ' S  $\subseteq$  sphere 0 1  $\wedge$ 
      homotopic_with_canon ( $\lambda x. \text{True}$ ) S (sphere 0 1) ( $\lambda x. f x / g x$ ) ( $\lambda x. 1$ )
    (is ?lhs  $\longleftrightarrow$  ?rhs)
  proof
    assume L: ?lhs
    have geq1 [simp]:  $\bigwedge x. x \in S \implies \text{cmod } (g x) = 1$ 
      using homotopic_with_imp_subset2 [OF L]
      by (simp add: image_subset_iff)
    have cont: continuous_on S (inverse  $\circ$  g)
    proof (rule continuous_intros)
      show continuous_on S g
        using homotopic_with_imp_continuous [OF L] by blast
    qed
  end

```

```

      show continuous_on (g ' S) inverse
      by (rule continuous_on_subset [of sphere 0 1, OF continuous_on_inverse])
    auto
  qed
  have [simp]:  $\bigwedge x. x \in S \implies g\ x \neq 0$ 
  using geq1 by fastforce
  have homotopic_with_canon ( $\lambda x. \text{True}$ ) S (sphere 0 1) ( $\lambda x. f\ x / g\ x$ ) ( $\lambda x. 1$ )
  apply (rule homotopic_with_eq [OF homotopic_with_sphere_times [OF L
cont]])
  by (auto simp: divide_inverse norm_inverse)
  with L show ?rhs
  by (auto simp: homotopic_with_imp_continuous dest: homotopic_with_imp_subset1
homotopic_with_imp_subset2)
  next
  assume ?rhs then show ?lhs
  by (elim conjE homotopic_with_eq [OF homotopic_with_sphere_times];
force)
  qed
  then show ?thesis
  by (simp add: *)
qed

```

6.42.12 Upper and lower hemicontinuous functions

And relation in the case of preimage map to open and closed maps, and fact that upper and lower hemicontinuity together imply continuity in the sense of the Hausdorff metric (at points where the function gives a bounded and nonempty set).

Many similar proofs below.

lemma *upper_hemicontinuous*:

```

  assumes  $\bigwedge x. x \in S \implies f\ x \subseteq T$ 
  shows  $((\forall U. \text{openin } (\text{top\_of\_set } T) \ U$ 
 $\implies \text{openin } (\text{top\_of\_set } S) \ \{x \in S. f\ x \subseteq U\}) \longleftrightarrow$ 
 $(\forall U. \text{closedin } (\text{top\_of\_set } T) \ U$ 
 $\implies \text{closedin } (\text{top\_of\_set } S) \ \{x \in S. f\ x \cap U \neq \{\}\}))$ 
  (is ?lhs = ?rhs)

```

proof (*intro iffI allI impI*)

fix U

assume * [rule_format]: ?lhs and closedin (top_of_set T) U

then have openin (top_of_set T) (T - U)

by (simp add: openin_diff)

then have openin (top_of_set S) $\{x \in S. f\ x \subseteq T - U\}$

using * [of T-U] by blast

moreover have $S - \{x \in S. f\ x \subseteq T - U\} = \{x \in S. f\ x \cap U \neq \{\}\}$

using assms by blast

ultimately show closedin (top_of_set S) $\{x \in S. f\ x \cap U \neq \{\}\}$

by (simp add: openin_closedin_eq)

next

```

fix U
assume * [rule_format]: ?rhs and openin (top_of_set T) U
then have closedin (top_of_set T) (T - U)
  by (simp add: closedin_diff)
then have closedin (top_of_set S) {x ∈ S. f x ∩ (T - U) ≠ {}}
  using * [of T-U] by blast
moreover have {x ∈ S. f x ∩ (T - U) ≠ {}} = S - {x ∈ S. f x ⊆ U}
  using assms by auto
ultimately show openin (top_of_set S) {x ∈ S. f x ⊆ U}
  by (simp add: openin_closedin_eq)
qed

```

lemma *lower_hemicontinuous*:

```

assumes  $\bigwedge x. x \in S \implies f x \subseteq T$ 
shows (( $\forall U. \text{closedin } (\text{top\_of\_set } T) U$ 
 $\longrightarrow \text{closedin } (\text{top\_of\_set } S) \{x \in S. f x \subseteq U\}$ )  $\longleftrightarrow$ 
 $(\forall U. \text{openin } (\text{top\_of\_set } T) U$ 
 $\longrightarrow \text{openin } (\text{top\_of\_set } S) \{x \in S. f x \cap U \neq \{\}\}$ ))
(is ?lhs = ?rhs))
proof (intro iffI allI impI)
fix U
assume * [rule_format]: ?lhs and openin (top_of_set T) U
then have closedin (top_of_set T) (T - U)
  by (simp add: closedin_diff)
then have closedin (top_of_set S) {x ∈ S. f x ⊆ T-U}
  using * [of T-U] by blast
moreover have {x ∈ S. f x ⊆ T-U} = S - {x ∈ S. f x ∩ U ≠ {}}
  using assms by auto
ultimately show openin (top_of_set S) {x ∈ S. f x ∩ U ≠ {}}
  by (simp add: openin_closedin_eq)
next
fix U
assume * [rule_format]: ?rhs and closedin (top_of_set T) U
then have openin (top_of_set T) (T - U)
  by (simp add: openin_diff)
then have openin (top_of_set S) {x ∈ S. f x ∩ (T - U) ≠ {}}
  using * [of T-U] by blast
moreover have S - {x ∈ S. f x ∩ (T - U) ≠ {}} = {x ∈ S. f x ⊆ U}
  using assms by blast
ultimately show closedin (top_of_set S) {x ∈ S. f x ⊆ U}
  by (simp add: openin_closedin_eq)
qed

```

lemma *open_map_iff_lower_hemicontinuous_preimage*:

```

assumes  $f \text{ ' } S \subseteq T$ 
shows (( $\forall U. \text{openin } (\text{top\_of\_set } S) U$ 
 $\longrightarrow \text{openin } (\text{top\_of\_set } T) (f \text{ ' } U)$ )  $\longleftrightarrow$ 
 $(\forall U. \text{closedin } (\text{top\_of\_set } S) U$ 
 $\longrightarrow \text{closedin } (\text{top\_of\_set } T) \{y \in T. \{x. x \in S \wedge f x = y\} \subseteq U\})$ )

```

```

      (is ?lhs = ?rhs)
proof (intro iffI allI impI)
  fix U
  assume * [rule_format]: ?lhs and closedin (top_of_set S) U
  then have openin (top_of_set S) (S - U)
    by (simp add: openin_diff)
  then have openin (top_of_set T) (f ' (S - U))
    using * [of S-U] by blast
  moreover have  $T - (f ' (S - U)) = \{y \in T. \{x \in S. f x = y\} \subseteq U\}$ 
    using assms by blast
  ultimately show closedin (top_of_set T)  $\{y \in T. \{x \in S. f x = y\} \subseteq U\}$ 
    by (simp add: openin_closedin_eq)
next
fix U
assume * [rule_format]: ?rhs and opeSU: openin (top_of_set S) U
then have closedin (top_of_set S) (S - U)
  by (simp add: closedin_diff)
then have closedin (top_of_set T)  $\{y \in T. \{x \in S. f x = y\} \subseteq S - U\}$ 
  using * [of S-U] by blast
moreover have  $\{y \in T. \{x \in S. f x = y\} \subseteq S - U\} = T - (f ' U)$ 
  using assms openin_imp_subset [OF opeSU] by auto
ultimately show openin (top_of_set T) (f ' U)
  using assms openin_imp_subset [OF opeSU] by (force simp: openin_closedin_eq)
qed

```

lemma *closed_map_iff_upper_hemicontinuous_preimage:*

```

assumes f ' S  $\subseteq$  T
shows (( $\forall U. \text{closedin } (\text{top\_of\_set } S) U$ 
 $\longrightarrow \text{closedin } (\text{top\_of\_set } T) (f ' U)$ )  $\longleftrightarrow$ 
( $\forall U. \text{openin } (\text{top\_of\_set } S) U$ 
 $\longrightarrow \text{openin } (\text{top\_of\_set } T) \{y \in T. \{x. x \in S \wedge f x = y\} \subseteq U\}$ ))
(is ?lhs = ?rhs)
proof (intro iffI allI impI)
  fix U
  assume * [rule_format]: ?lhs and opeSU: openin (top_of_set S) U
  then have closedin (top_of_set S) (S - U)
    by (simp add: closedin_diff)
  then have closedin (top_of_set T) (f ' (S - U))
    using * [of S-U] by blast
  moreover have  $f ' (S - U) = T - \{y \in T. \{x. x \in S \wedge f x = y\} \subseteq U\}$ 
    using assms openin_imp_subset [OF opeSU] by auto
  ultimately show openin (top_of_set T)  $\{y \in T. \{x. x \in S \wedge f x = y\} \subseteq U\}$ 
    using assms openin_imp_subset [OF opeSU] by (force simp: openin_closedin_eq)
next
fix U
assume * [rule_format]: ?rhs and cloSU: closedin (top_of_set S) U
then have openin (top_of_set S) (S - U)
  by (simp add: openin_diff)
then have openin (top_of_set T)  $\{y \in T. \{x \in S. f x = y\} \subseteq S - U\}$ 

```

```

    using * [of S-U] by blast
  moreover have (f ' U) = T - {y ∈ T. {x ∈ S. f x = y} ⊆ S - U}
    using assms closedin_imp_subset [OF cloSU] by auto
  ultimately show closedin (top_of_set T) (f ' U)
    by (simp add: openin_closedin_eq)
qed

proposition upper_lower_hemicontinuous_explicit:
  fixes T :: ('b::{real_normed_vector,heine_borel}) set
  assumes fST:  $\bigwedge x. x \in S \implies f x \subseteq T$ 
    and ope:  $\bigwedge U. \text{openin } (\text{top\_of\_set } T) U \implies \text{openin } (\text{top\_of\_set } S) \{x \in S. f x \subseteq U\}$ 
    and clo:  $\bigwedge U. \text{closedin } (\text{top\_of\_set } T) U \implies \text{closedin } (\text{top\_of\_set } S) \{x \in S. f x \subseteq U\}$ 
    and x ∈ S 0 < e and bofx: bounded(f x) and fx_ne: f x ≠ {}
  obtains d where 0 < d
     $\bigwedge x'. \llbracket x' \in S; \text{dist } x x' < d \rrbracket \implies (\forall y \in f x. \exists y'. y' \in f x' \wedge \text{dist } y y' < e) \wedge$ 
     $(\forall y' \in f x'. \exists y. y \in f x \wedge \text{dist } y' y < e)$ 

proof -
  have openin (top_of_set T) (T ∩ (⋃ a ∈ f x. ⋃ b ∈ ball 0 e. {a + b}))
    by (auto simp: open_sums openin_open_Int)
  with ope have openin (top_of_set S)
    {u ∈ S. f u ⊆ T ∩ (⋃ a ∈ f x. ⋃ b ∈ ball 0 e. {a + b})} by blast
  with ⟨0 < e⟩ ⟨x ∈ S⟩ obtain d1 where d1 > 0 and
    d1:  $\bigwedge x'. \llbracket x' \in S; \text{dist } x' x < d1 \rrbracket \implies f x' \subseteq T \wedge f x' \subseteq (\bigcup a \in f x. \bigcup b \in \text{ball } 0 e. \{a + b\})$ 
  by (force simp: openin_euclidean_subtopology_iff dest: fST)
  have oo:  $\bigwedge U. \text{openin } (\text{top\_of\_set } T) U \implies \text{openin } (\text{top\_of\_set } S) \{x \in S. f x \cap U \neq \{\}\}$ 
    apply (rule lower_hemicontinuous [THEN iffD1, rule_format])
    using fST clo by auto
  have compact (closure(f x))
    by (simp add: bofx)
  moreover have closure(f x) ⊆ (⋃ a ∈ f x. ball a (e/2))
    using ⟨0 < e⟩ by (force simp: closure_approachable simp del: divide_const_simps)
  ultimately obtain C where C ⊆ f x finite C closure(f x) ⊆ (⋃ a ∈ C. ball a (e/2))
    apply (rule compactE, force)
    by (metis finite_subset_image)
  then have fx_cover: f x ⊆ (⋃ a ∈ C. ball a (e/2))
    by (meson closure_subset order_trans)
  with fx_ne have C ≠ {}
    by blast
  have xin: x ∈ (⋂ a ∈ C. {x ∈ S. f x ∩ T ∩ ball a (e/2) ≠ {}})
    using ⟨x ∈ S⟩ ⟨0 < e⟩ fST ⟨C ⊆ f x⟩ by force
  have openin (top_of_set S) {x ∈ S. f x ∩ (T ∩ ball a (e/2)) ≠ {} for a
    by (simp add: openin_open_Int oo)
  then have openin (top_of_set S) (⋂ a ∈ C. {x ∈ S. f x ∩ T ∩ ball a (e/2) ≠ {}

```

```

{}})
  by (simp add: Int_assoc openin_INT2 [OF ⟨finite C⟩ ⟨C ≠ {}⟩])
  with xin obtain d2 where d2 > 0
    and d2:  $\bigwedge u v. \llbracket u \in S; \text{dist } u \ x < d2; v \in C \rrbracket \implies f u \cap T \cap \text{ball } v$ 
    (e/2) ≠ {}
  unfolding openin_euclidean_subtopology_iff using xin by fastforce
  show ?thesis
  proof (intro that conjI ballI)
    show 0 < min d1 d2
      using ⟨0 < d1⟩ ⟨0 < d2⟩ by linarith
  next
    fix x' y
    assume x' ∈ S dist x x' < min d1 d2 y ∈ f x
    then have dd2: dist x' x < d2
      by (auto simp: dist_commute)
    obtain a where a ∈ C y ∈ ball a (e/2)
      using fx_cover ⟨y ∈ f x⟩ by auto
    then show  $\exists y'. y' \in f x' \wedge \text{dist } y \ y' < e$ 
      using d2 [OF ⟨x' ∈ S⟩ dd2] dist_triangle_half_r by fastforce
  next
    fix x' y'
    assume x' ∈ S dist x x' < min d1 d2 y' ∈ f x'
    then have dist x' x < d1
      by (auto simp: dist_commute)
    then have y' ∈  $(\bigcup a \in f x. \bigcup b \in \text{ball } 0 \ e. \{a + b\})$ 
      using d1 [OF ⟨x' ∈ S⟩] ⟨y' ∈ f x'⟩ by force
    then show  $\exists y. y \in f x \wedge \text{dist } y' \ y < e$ 
      by clarsimp (metis add_diff_cancel_left' dist_norm)
  qed
qed

```

6.42.13 Complex logs exist on various "well-behaved" sets

```

lemma continuous_logarithm_on_contractible:
  fixes f :: 'a::real_normed_vector ⇒ complex
  assumes continuous_on S f contractible S  $\bigwedge z. z \in S \implies f z \neq 0$ 
  obtains g where continuous_on S g  $\bigwedge x. x \in S \implies f x = \exp(g x)$ 
proof -
  obtain c where hom: homotopic_with_canon (λh. True) S (-{0}) f (λx. c)
    using nullhomotopic_from_contractible assms
    by (metis imageE subset_Compl_singleton)
  then show ?thesis
    by (metis inessential_eq_continuous_logarithm that)
qed

lemma continuous_logarithm_on_simply_connected:
  fixes f :: 'a::real_normed_vector ⇒ complex
  assumes contf: continuous_on S f and S: simply_connected S locally_path_connected S

```

and $f: \bigwedge z. z \in S \implies f z \neq 0$
 obtains g where $\text{continuous\_on } S g \bigwedge x. x \in S \implies f x = \exp(g x)$
 using $\text{covering\_space\_lift } [OF \text{ covering\_space\_exp\_punctured\_plane } S \text{ contf}]$
 by $(\text{metis } (\text{full\_types}) f \text{ imageE subset\_Compl\_singleton})$

lemma *continuous_logarithm_on_cball*:

fixes $f :: 'a::\text{real\_normed\_vector} \Rightarrow \text{complex}$
 assumes $\text{continuous\_on } (\text{cball } a r) f$ and $\bigwedge z. z \in \text{cball } a r \implies f z \neq 0$
 obtains h where $\text{continuous\_on } (\text{cball } a r) h \bigwedge z. z \in \text{cball } a r \implies f z = \exp(h z)$
 using *assms continuous_logarithm_on_contractible convex_imp_contractible* by *blast*

lemma *continuous_logarithm_on_ball*:

fixes $f :: 'a::\text{real\_normed\_vector} \Rightarrow \text{complex}$
 assumes $\text{continuous\_on } (\text{ball } a r) f$ and $\bigwedge z. z \in \text{ball } a r \implies f z \neq 0$
 obtains h where $\text{continuous\_on } (\text{ball } a r) h \bigwedge z. z \in \text{ball } a r \implies f z = \exp(h z)$
 using *assms continuous_logarithm_on_contractible convex_imp_contractible* by *blast*

lemma *continuous_sqrt_on_contractible*:

fixes $f :: 'a::\text{real\_normed\_vector} \Rightarrow \text{complex}$
 assumes $\text{continuous\_on } S f$ contractible S
 and $\bigwedge z. z \in S \implies f z \neq 0$
 obtains g where $\text{continuous\_on } S g \bigwedge x. x \in S \implies f x = (g x) ^ 2$
proof –
 obtain g where $\text{contg}: \text{continuous\_on } S g$ and $\text{feq}: \bigwedge x. x \in S \implies f x = \exp(g x)$
 using *continuous_logarithm_on_contractible* [*OF assms*] by *blast*
 show ?thesis
proof
 show $\text{continuous\_on } S (\lambda z. \exp (g z / 2))$
 by (*rule continuous_on_compose2* [*of UNIV exp*]; *intro continuous_intros contg subset_UNIV*) *auto*
 show $\bigwedge x. x \in S \implies f x = (\exp (g x / 2))^2$
 by (*metis exp_double feq nonzero_mult_div_cancel_left times_divide_eq_right zero_neq_numeral*)
 qed
 qed

lemma *continuous_sqrt_on_simply_connected*:

fixes $f :: 'a::\text{real\_normed\_vector} \Rightarrow \text{complex}$
 assumes $\text{contf}: \text{continuous\_on } S f$ and $S: \text{simply\_connected } S$ locally *path_connected* S
 and $f: \bigwedge z. z \in S \implies f z \neq 0$
 obtains g where $\text{continuous\_on } S g \bigwedge x. x \in S \implies f x = (g x) ^ 2$
proof –
 obtain g where $\text{contg}: \text{continuous\_on } S g$ and $\text{feq}: \bigwedge x. x \in S \implies f x = \exp(g x)$

```

x)
  using continuous_logarithm_on_simply_connected [OF assms] by blast
show ?thesis
proof
  show continuous_on S ( $\lambda z. \exp (g z / 2)$ )
  by (rule continuous_on_compose2 [of UNIV exp]; intro continuous_intros
contg_subset_UNIV) auto
  show  $\bigwedge x. x \in S \implies f x = (\exp (g x / 2))^2$ 
  by (metis exp_double feq nonzero_mult_div_cancel_left times_divide_eq_right
zero_neq_numeral)
qed
qed

```

6.42.14 Another simple case where sphere maps are nullhomotopic

```

lemma inessential_spheremap_2_aux:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  complex
  assumes 2:  $2 < \text{DIM}('a)$  and contf: continuous_on (sphere a r) f
  and fim:  $f \text{ `}( \text{sphere } a \text{ } r ) \subseteq ( \text{sphere } 0 \text{ } 1 )$ 
  obtains c where homotopic_with_canon ( $\lambda z. \text{True}$ ) (sphere a r) (sphere 0 1) f
  ( $\lambda x. c$ )
proof -
  obtain g where contg: continuous_on (sphere a r) g
  and feq:  $\bigwedge x. x \in \text{sphere } a \text{ } r \implies f x = \exp(g x)$ 
proof (rule continuous_logarithm_on_simply_connected [OF contf])
  show simply_connected (sphere a r)
  using 2 by (simp add: simply_connected_sphere_eq)
  show locally_path_connected (sphere a r)
  by (simp add: locally_path_connected_sphere)
  show  $\bigwedge z. z \in \text{sphere } a \text{ } r \implies f z \neq 0$ 
  using fim by force
qed auto
  have  $\exists g. \text{continuous\_on } (\text{sphere } a \text{ } r) \text{ } g \wedge (\forall x \in \text{sphere } a \text{ } r. f x = \exp (i * \text{complex\_of\_real } (g x)))$ 
proof (intro exI conjI)
  show continuous_on (sphere a r) (Im  $\circ$  g)
  by (intro contg continuous_intros continuous_on_compose)
  show  $\forall x \in \text{sphere } a \text{ } r. f x = \exp (i * \text{complex\_of\_real } ((\text{Im} \circ g) x))$ 
  using exp_eq_polar feq fim norm_exp_eq_Re by auto
qed
  with inessential_eq_continuous_logarithm_circle that show ?thesis
  by metis
qed

```

```

lemma inessential_spheremap_2:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes a2:  $2 < \text{DIM}('a)$  and b2:  $\text{DIM}('b) = 2$ 
  and contf: continuous_on (sphere a r) f and fim:  $f \text{ `}( \text{sphere } a \text{ } r ) \subseteq ( \text{sphere } 0 \text{ } 1 )$ 

```



```

b s)
  obtains c where homotopic_with_canon ( $\lambda z. \text{True}$ ) (sphere a r) (sphere b s) f
  ( $\lambda x. c$ )
proof (cases  $s \leq 0$ )
  case True
  then show ?thesis
    using contf contractible_sphere fim nullhomotopic_into_contractible that by
    blast
  next
  case False
  then have sphere b s homeomorphic_sphere (0::complex) 1
    using assms by (simp add: homeomorphic_spheres_gen)
  then obtain h k where hk: homeomorphism (sphere b s) (sphere (0::complex)
  1) h k
    by (auto simp: homeomorphic_def)
  then have conth: continuous_on (sphere b s) h
    and contk: continuous_on (sphere 0 1) k
    and him:  $h \text{ ` sphere b s } \subseteq \text{ sphere 0 1}$ 
    and kim:  $k \text{ ` sphere 0 1 } \subseteq \text{ sphere b s}$ 
    by (simp_all add: homeomorphism_def)
  obtain c where homotopic_with_canon ( $\lambda z. \text{True}$ ) (sphere a r) (sphere 0 1) (h
   $\circ f$ ) ( $\lambda x. c$ )
  proof (rule inessential_spheremap_2_aux [OF a2])
    show continuous_on (sphere a r) (h  $\circ f$ )
    by (meson continuous_on_compose [OF contf] conth continuous_on_subset
    fim)
    show (h  $\circ f$ ) ` sphere a r  $\subseteq$  sphere 0 1
    using fim him by force
  qed auto
  then have homotopic_with_canon ( $\lambda f. \text{True}$ ) (sphere a r) (sphere b s) (k  $\circ$  (h
   $\circ f$ )) (k  $\circ$  ( $\lambda x. c$ ))
    by (rule homotopic_with_compose_continuous_left [OF _ contk kim])
  then have homotopic_with_canon ( $\lambda z. \text{True}$ ) (sphere a r) (sphere b s) f ( $\lambda x. k$ 
  c)
    apply (rule homotopic_with_eq, auto)
    by (metis fim hk homeomorphism_def image_subset_iff mem_sphere)
  then show ?thesis
    by (metis that)
qed

```

6.42.15 Holomorphic logarithms and square roots

```

lemma g_imp_holomorphic_log:
  assumes holf: f holomorphic_on S
    and contg: continuous_on S g and feq:  $\bigwedge x. x \in S \implies f x = \exp (g x)$ 
    and fnz:  $\bigwedge z. z \in S \implies f z \neq 0$ 
  obtains g where g holomorphic_on S  $\bigwedge z. z \in S \implies f z = \exp(g z)$ 
proof -
  have contf: continuous_on S f

```

```

    by (simp add: holf holomorphic_on_imp_continuous_on)
    have g field_differentiable at z within S if f field_differentiable at z within S z ∈
    S for z
    proof -
      obtain f' where f': ((λy. (f y - f z) / (y - z)) → f') (at z within S)
      using ⟨f field_differentiable at z within S⟩ by (auto simp: field_differentiable_def
      has_field_derivative_iff)
      then have ee: ((λx. (exp(g x) - exp(g z)) / (x - z)) → f') (at z within S)
      by (simp add: feq ⟨z ∈ S⟩ Lim_transform_within [OF zero_less_one])
      have (((λy. if y = g z then exp(g z) else (exp y - exp(g z)) / (y - g z)) ∘ g)
      → exp(g z))
      (at z within S)
      proof (rule tendsto_compose_at)
        show (g → g z) (at z within S)
        using contg continuous_on ⟨z ∈ S⟩ by blast
        show (λy. if y = g z then exp(g z) else (exp y - exp(g z)) / (y - g z)) -g
      z → exp(g z)
      by (simp add: LIM_offset_zero_iff DERIV_D cong: if_cong Lim_cong_within)
      qed auto
      then have dd: ((λx. if g x = g z then exp(g z) else (exp(g x) - exp(g z)) / (g
      x - g z)) → exp(g z)) (at z within S)
      by (simp add: o_def)
      have continuous (at z within S) g
      using contg continuous_on_eq_continuous_within ⟨z ∈ S⟩ by blast
      then have (∀F x in at z within S. dist (g x) (g z) < 2*π)
      by (simp add: continuous_within tendsto_iff)
      then have ∀F x in at z within S. exp(g x) = exp(g z) → g x ≠ g z → x
      = z
      by (rule eventually_mono) (auto simp: exp_eq dist_norm norm_mult)
      then have ((λy. (g y - g z) / (y - z)) → f' / exp(g z)) (at z within S)
      by (auto intro!: Lim_transform_eventually [OF tendsto_divide [OF ee dd]])
      then show ?thesis
      by (auto simp: field_differentiable_def has_field_derivative_iff)
      qed
      then have g holomorphic_on S
      using holf holomorphic_on_def by auto
      then show ?thesis
      using feq that by auto
    qed

lemma contractible_imp_holomorphic_log:
  assumes holf: f holomorphic_on S
  and S: contractible S
  and fnz: ∧z. z ∈ S ⇒ f z ≠ 0
  obtains g where g holomorphic_on S ∧z. z ∈ S ⇒ f z = exp(g z)
proof -
  have contf: continuous_on S f
  by (simp add: holf holomorphic_on_imp_continuous_on)
  obtain g where contg: continuous_on S g and feq: ∧x. x ∈ S ⇒ f x = exp(g

```

```

x)
  by (metis continuous_logarithm_on_contractible [OF contf S fnz])
  then show thesis
    using fnz g_imp_holomorphic_log holg that by blast
qed

lemma simply_connected_imp_holomorphic_log:
  assumes holg: f holomorphic_on S
    and S: simply_connected S locally_path_connected S
    and fnz:  $\bigwedge z. z \in S \implies f z \neq 0$ 
  obtains g where g holomorphic_on S  $\bigwedge z. z \in S \implies f z = \exp(g z)$ 
proof -
  have contf: continuous_on S f
    by (simp add: holg holomorphic_on_imp_continuous_on)
  obtain g where contg: continuous_on S g and feq:  $\bigwedge x. x \in S \implies f x = \exp(g x)$ 
x)
  by (metis continuous_logarithm_on_simply_connected [OF contf S fnz])
  then show thesis
    using fnz g_imp_holomorphic_log holg that by blast
qed

lemma contractible_imp_holomorphic_sqrt:
  assumes holg: f holomorphic_on S
    and S: contractible S
    and fnz:  $\bigwedge z. z \in S \implies f z \neq 0$ 
  obtains g where g holomorphic_on S  $\bigwedge z. z \in S \implies f z = g z^2$ 
proof -
  obtain g where holg: g holomorphic_on S and feq:  $\bigwedge z. z \in S \implies f z = \exp(g z)$ 
z)
  using contractible_imp_holomorphic_log [OF assms] by blast
  show ?thesis
  proof
    show  $\exp \circ (\lambda z. z / 2) \circ g$  holomorphic_on S
      by (intro holomorphic_on_compose holg holomorphic_intros) auto
    show  $\bigwedge z. z \in S \implies f z = ((\exp \circ (\lambda z. z / 2) \circ g) z)^2$ 
      by (simp add: feq flip: exp_double)
  qed
qed

lemma simply_connected_imp_holomorphic_sqrt:
  assumes holg: f holomorphic_on S
    and S: simply_connected S locally_path_connected S
    and fnz:  $\bigwedge z. z \in S \implies f z \neq 0$ 
  obtains g where g holomorphic_on S  $\bigwedge z. z \in S \implies f z = g z^2$ 
proof -
  obtain g where holg: g holomorphic_on S and feq:  $\bigwedge z. z \in S \implies f z = \exp(g z)$ 
z)
  using simply_connected_imp_holomorphic_log [OF assms] by blast
  show ?thesis

```

```

proof
  show  $\exp \circ (\lambda z. z / 2) \circ g$  holomorphic_on  $S$ 
    by (intro holomorphic_on_compose holg holomorphic_intros) auto
  show  $\bigwedge z. z \in S \implies f z = ((\exp \circ (\lambda z. z / 2) \circ g) z)^2$ 
    by (simp add: feq flip: exp_double)
qed
qed

```

Related theorems about holomorphic inverse cosines.

```

lemma contractible_imp_holomorphic_arccos:
  assumes holf:  $f$  holomorphic_on  $S$  and  $S$ : contractible  $S$ 
    and non1:  $\bigwedge z. z \in S \implies f z \neq 1 \wedge f z \neq -1$ 
  obtains  $g$  where  $g$  holomorphic_on  $S$   $\bigwedge z. z \in S \implies f z = \cos(g z)$ 
proof -
  have hol1f:  $(\lambda z. 1 - f z \wedge 2)$  holomorphic_on  $S$ 
    by (intro holomorphic_intros holf)
  obtain  $g$  where holg:  $g$  holomorphic_on  $S$  and eq:  $\bigwedge z. z \in S \implies 1 - (f z)^2 =$ 
     $(g z)^2$ 
    using contractible_imp_holomorphic_sqrt [OF hol1f S]
    by (metis eq_iff_diff_eq_0 non1 power2_eq_1_iff)
  have holfg:  $(\lambda z. f z + i * g z)$  holomorphic_on  $S$ 
    by (intro holf holg holomorphic_intros)
  have  $\bigwedge z. z \in S \implies f z + i * g z \neq 0$ 
    by (metis Arccos_body_lemma eq add.commute add.inverse_unique complex_i_mult_minus
    power2_csqr power2_eq_iff)
  then obtain  $h$  where holh:  $h$  holomorphic_on  $S$  and fgeq:  $\bigwedge z. z \in S \implies f z$ 
     $+ i * g z = \exp (h z)$ 
    using contractible_imp_holomorphic_log [OF holfg S] by metis
  show ?thesis
proof
  show  $(\lambda z. -i * h z)$  holomorphic_on  $S$ 
    by (intro holh holomorphic_intros)
  show  $f z = \cos (-i * h z)$  if  $z \in S$  for  $z$ 
proof -
  have  $(f z + i * g z) * (f z - i * g z) = 1$ 
    using that eq by (auto simp: algebra_simps power2_eq_square)
  then have  $f z - i * g z = \text{inverse} (f z + i * g z)$ 
    using inverse_unique by force
  also have  $\dots = \exp (-h z)$ 
    by (simp add: exp_minus fgeq that)
  finally have  $f z = \exp (-h z) + i * g z$ 
    by (simp add: diff_eq_eq)
  then show ?thesis
    apply (simp add: cos_exp_eq)
    by (metis fgeq add.assoc mult_2_right that)
qed
qed
qed

```

```

lemma contractible_imp_holomorphic_arccos_bounded:
  assumes hol:  $f$  holomorphic_on  $S$  and  $S$ : contractible  $S$  and  $a \in S$ 
    and non1:  $\bigwedge z. z \in S \implies f z \neq 1 \wedge f z \neq -1$ 
  obtains  $g$  where  $g$  holomorphic_on  $S$   $\text{norm}(g a) \leq \pi + \text{norm}(f a)$   $\bigwedge z. z \in S$ 
 $\implies f z = \cos(g z)$ 
proof -
  obtain  $g$  where holg:  $g$  holomorphic_on  $S$  and feq:  $\bigwedge z. z \in S \implies f z = \cos(g z)$ 
  using contractible_imp_holomorphic_arccos [OF holg  $S$  non1] by blast
  obtain  $b$  where  $\cos b = f a$   $\text{norm } b \leq \pi + \text{norm}(f a)$ 
  using cos_Arccos norm_Arccos_bounded by blast
  then have  $\cos b = \cos(g a)$ 
  by (simp add:  $\langle a \in S \rangle$  feq)
  then consider  $n$  where  $n \in \mathbb{Z}$   $b = g a + \text{of\_real}(2*n*\pi) \mid n$  where  $n \in \mathbb{Z}$   $b$ 
 $= -g a + \text{of\_real}(2*n*\pi)$ 
  by (auto simp: complex_cos_eq)
  then show ?thesis
proof cases
  case 1
  show ?thesis
proof
  show  $(\lambda z. g z + \text{of\_real}(2*n*\pi))$  holomorphic_on  $S$ 
  by (intro holomorphic_intros holg)
  show  $\text{cmod}(g a + \text{of\_real}(2*n*\pi)) \leq \pi + \text{cmod}(f a)$ 
  using 1  $\langle \text{cmod } b \leq \pi + \text{cmod}(f a) \rangle$  by blast
  show  $\bigwedge z. z \in S \implies f z = \cos(g z + \text{complex\_of\_real}(2*n*\pi))$ 
  by (metis  $\langle n \in \mathbb{Z} \rangle$  complex_cos_eq feq)
qed
next
  case 2
  show ?thesis
proof
  show  $(\lambda z. -g z + \text{of\_real}(2*n*\pi))$  holomorphic_on  $S$ 
  by (intro holomorphic_intros holg)
  show  $\text{cmod}(-g a + \text{of\_real}(2*n*\pi)) \leq \pi + \text{cmod}(f a)$ 
  using 2  $\langle \text{cmod } b \leq \pi + \text{cmod}(f a) \rangle$  by blast
  show  $\bigwedge z. z \in S \implies f z = \cos(-g z + \text{complex\_of\_real}(2*n*\pi))$ 
  by (metis  $\langle n \in \mathbb{Z} \rangle$  complex_cos_eq feq)
qed
qed
qed

```

6.42.16 The "Borsukian" property of sets

This doesn't have a standard name. Kuratowski uses "contractible with respect to $[S^1]$ " while Whyburn uses "property b". It's closely related to unicoherence.

definition *Borsukian* where

Borsukian $S \equiv$
 $\forall f. \text{continuous\_on } S \ f \wedge f' \ S \subseteq (-\{0::\text{complex}\})$
 $\longrightarrow (\exists a. \text{homotopic\_with\_canon } (\lambda h. \text{True}) \ S \ (-\{0\}) \ f \ (\lambda x. a))$

lemma *Borsukian_retraction_gen*:
assumes *Borsukian* S *continuous_on* S h $h' \ S = T$
 $\text{continuous\_on } T \ k \ k' \ T \subseteq S \ \wedge y. y \in T \implies h(k \ y) = y$
shows *Borsukian* T
proof –
interpret R : *Retracts* S h T k
using *assms* **by** (*simp* *add*: *Retracts.intro*)
show *?thesis*
using *assms*
apply (*clarsimp* *simp* *add*: *Borsukian_def*)
apply (*rule* *R.cohomotopically_trivial_retraction_null_gen* [*OF* *TrueI* *TrueI*
refl, *of* $-\{0\}$], *auto*)
done
qed

lemma *retract_of_Borsukian*: $\llbracket \text{Borsukian } T; S \text{ retract\_of } T \rrbracket \implies \text{Borsukian } S$
apply (*auto* *simp*: *retract_of_def* *retraction_def*)
apply (*erule* (1) *Borsukian_retraction_gen*)
apply (*meson* *retraction* *retraction_def*)
apply (*auto*)
done

lemma *homeomorphic_Borsukian*: $\llbracket \text{Borsukian } S; S \text{ homeomorphic } T \rrbracket \implies \text{Borsukian } T$
using *Borsukian_retraction_gen* *order_refl*
by (*fastforce* *simp* *add*: *homeomorphism_def* *homeomorphic_def*)

lemma *homeomorphic_Borsukian_eq*:
 $S \text{ homeomorphic } T \implies \text{Borsukian } S \longleftrightarrow \text{Borsukian } T$
by (*meson* *homeomorphic_Borsukian* *homeomorphic_sym*)

lemma *Borsukian_translation*:
fixes $S :: 'a::\text{real\_normed\_vector\_set}$
shows *Borsukian* (*image* $(\lambda x. a + x)$ S) $\longleftrightarrow \text{Borsukian } S$
using *homeomorphic_Borsukian_eq* *homeomorphic_translation* **by** *blast*

lemma *Borsukian_injective_linear_image*:
fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$
assumes *linear* f *inj* f
shows *Borsukian*($f' \ S$) $\longleftrightarrow \text{Borsukian } S$
using *assms* *homeomorphic_Borsukian_eq* *linear_homeomorphic_image* **by** *blast*

lemma *homotopy_eqv_Borsukianness*:
fixes $S :: 'a::\text{real\_normed\_vector\_set}$
and $T :: 'b::\text{real\_normed\_vector\_set}$

```

assumes  $S$  homotopy_eqv  $T$ 
shows ( $Borsukian\ S \longleftrightarrow Borsukian\ T$ )
by (meson Borsukian_def assms homotopy_eqv_cohomotopic_triviality_null)

lemma Borsukian_alt:
fixes  $S :: 'a::real\_normed\_vector\ set$ 
shows
   $Borsukian\ S \longleftrightarrow$ 
    ( $\forall f\ g. continuous\_on\ S\ f \wedge f\ 'S \subseteq -\{0\} \wedge$ 
       $continuous\_on\ S\ g \wedge g\ 'S \subseteq -\{0\}$ 
       $\longrightarrow homotopic\_with\_canon\ (\lambda h. True)\ S\ (-\{0::complex\})\ f\ g$ )
unfolding Borsukian_def homotopic_triviality
by (simp add: path_connected_punctured_universe)

lemma Borsukian_continuous_logarithm:
fixes  $S :: 'a::real\_normed\_vector\ set$ 
shows  $Borsukian\ S \longleftrightarrow$ 
  ( $\forall f. continuous\_on\ S\ f \wedge f\ 'S \subseteq (-\{0::complex\})$ 
     $\longrightarrow (\exists g. continuous\_on\ S\ g \wedge (\forall x \in S. f\ x = exp(g\ x))))$ 
by (simp add: Borsukian_def inessential_eq_continuous_logarithm)

lemma Borsukian_continuous_logarithm_circle:
fixes  $S :: 'a::real\_normed\_vector\ set$ 
shows  $Borsukian\ S \longleftrightarrow$ 
  ( $\forall f. continuous\_on\ S\ f \wedge f\ 'S \subseteq sphere\ (0::complex)\ 1$ 
     $\longrightarrow (\exists g. continuous\_on\ S\ g \wedge (\forall x \in S. f\ x = exp(g\ x))))$ 
  (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
    by (force simp: Borsukian_continuous_logarithm)
next
  assume RHS [rule_format]: ?rhs
  show ?lhs
proof (clarsimp simp: Borsukian_continuous_logarithm)
    fix  $f :: 'a \Rightarrow complex$ 
    assume contf:  $continuous\_on\ S\ f$  and  $0: 0 \notin f\ 'S$ 
    then have  $continuous\_on\ S\ (\lambda x. f\ x / complex\_of\_real\ (cmod\ (f\ x)))$ 
      by (intro continuous_intros) auto
    moreover have  $(\lambda x. f\ x / complex\_of\_real\ (cmod\ (f\ x)))\ 'S \subseteq sphere\ 0\ 1$ 
      using  $0$  by (auto simp: norm_divide)
    ultimately obtain  $g$  where contg:  $continuous\_on\ S\ g$ 
      and  $fg: \forall x \in S. f\ x / complex\_of\_real\ (cmod\ (f\ x)) = exp(g\ x)$ 
    using RHS [of  $\lambda x. f\ x / of\_real(norm(f\ x))$ ] by auto
    show  $\exists g. continuous\_on\ S\ g \wedge (\forall x \in S. f\ x = exp(g\ x))$ 
proof (intro exI ballI conjI)
    show  $continuous\_on\ S\ (\lambda x. (Ln \circ of\_real \circ norm \circ f)x + g\ x)$ 
      by (intro continuous_intros contf contg conjI) (use 0 in auto)
    show  $f\ x = exp\ ((Ln \circ complex\_of\_real \circ cmod \circ f)\ x + g\ x)$  if  $x \in S$  for  $x$ 
      using  $0$  that

```

```

    apply (simp add: exp_add)
  by (metis div_by_0 exp_Ln exp_not_eq_zero fg mult.commute nonzero_eq_divide_eq)
qed
qed
qed

```

lemma *Borsukian_continuous_logarithm_circle_real*:

```

  fixes S :: 'a::real_normed_vector set
  shows Borsukian S  $\longleftrightarrow$ 
    ( $\forall f. \text{continuous\_on } S f \wedge f' S \subseteq \text{sphere } (0::\text{complex}) 1$ 
       $\longrightarrow (\exists g. \text{continuous\_on } S (\text{complex\_of\_real} \circ g) \wedge (\forall x \in S. f x =$ 
 $\exp(i * \text{of\_real}(g x))))$ 
    (is ?lhs = ?rhs)
proof
  assume LHS: ?lhs
  show ?rhs
  proof (clarify)
    fix f :: 'a  $\Rightarrow$  complex
    assume continuous_on S f and f01:  $f' S \subseteq \text{sphere } 0 1$ 
    then obtain g where contg: continuous_on S g and  $\bigwedge x. x \in S \implies f x =$ 
 $\exp(g x)$ 
    using LHS by (auto simp: Borsukian_continuous_logarithm_circle)
    then have  $\forall x \in S. f x = \exp(i * \text{complex\_of\_real} ((\text{Im} \circ g) x))$ 
    using f01 exp_eq_polar norm_exp_eq_Re by auto
    then show  $\exists g. \text{continuous\_on } S (\text{complex\_of\_real} \circ g) \wedge (\forall x \in S. f x = \exp$ 
 $(i * \text{complex\_of\_real} (g x)))$ 
    by (rule_tac x= $\text{Im} \circ g$  in exI) (force intro: continuous_intros contg)
  qed
next
  assume RHS [rule_format]: ?rhs
  show ?lhs
  proof (clarsimp simp: Borsukian_continuous_logarithm_circle)
    fix f :: 'a  $\Rightarrow$  complex
    assume continuous_on S f and f01:  $f' S \subseteq \text{sphere } 0 1$ 
    then obtain g where contg: continuous_on S ( $\text{complex\_of\_real} \circ g$ ) and  $\bigwedge x.$ 
 $x \in S \implies f x = \exp(i * \text{of\_real}(g x))$ 
    by (metis RHS)
    then show  $\exists g. \text{continuous\_on } S g \wedge (\forall x \in S. f x = \exp(g x))$ 
    by (rule_tac x= $\lambda x. i * \text{of\_real}(g x)$  in exI) (auto simp: continuous_intros
contg)
  qed
qed

```

lemma *Borsukian_circle*:

```

  fixes S :: 'a::real_normed_vector set
  shows Borsukian S  $\longleftrightarrow$ 
    ( $\forall f. \text{continuous\_on } S f \wedge f' S \subseteq \text{sphere } (0::\text{complex}) 1$ 
       $\longrightarrow (\exists a. \text{homotopic\_with\_canon } (\lambda h. \text{True}) S (\text{sphere } (0::\text{complex}) 1)$ 

```



```

f (λx. a)))
by (simp add: inessential_eq_continuous_logarithm_circle Borsukian_continuous_logarithm_circle_real)

lemma contractible_imp_Borsukian: contractible S ⟹ Borsukian S
  by (meson Borsukian_def nullhomotopic_from_contractible)

lemma simply_connected_imp_Borsukian:
  fixes S :: 'a::real_normed_vector set
  shows ⟦simply_connected S; locally_path_connected S⟧ ⟹ Borsukian S
  by (metis (no_types, lifting) Borsukian_continuous_logarithm continuous_logarithm_on_simply_connected
    image_eqI subset_Compl_singleton)

lemma starlike_imp_Borsukian:
  fixes S :: 'a::real_normed_vector set
  shows starlike S ⟹ Borsukian S
  by (simp add: contractible_imp_Borsukian starlike_imp_contractible)

lemma Borsukian_empty: Borsukian {}
  by (auto simp: contractible_imp_Borsukian)

lemma Borsukian_UNIV: Borsukian (UNIV :: 'a::real_normed_vector set)
  by (auto simp: contractible_imp_Borsukian)

lemma convex_imp_Borsukian:
  fixes S :: 'a::real_normed_vector set
  shows convex S ⟹ Borsukian S
  by (meson Borsukian_def convex_imp_contractible nullhomotopic_from_contractible)

proposition Borsukian_sphere:
  fixes a :: 'a::euclidean_space
  shows  $3 \leq \text{DIM}(a) \implies \text{Borsukian}(\text{sphere } a \text{ } r)$ 
  using ENR_sphere
  by (blast intro: simply_connected_imp_Borsukian ENR_imp_locally_path_connected
    simply_connected_sphere)

lemma Borsukian_Un_lemma:
  fixes S :: 'a::real_normed_vector set
  assumes BS: Borsukian S and BT: Borsukian T and ST: connected(S ∩ T)
  and *:  $\bigwedge f g :: 'a \Rightarrow \text{complex}.$ 
     $\llbracket \text{continuous\_on } S \text{ } f; \text{continuous\_on } T \text{ } g; \bigwedge x. x \in S \wedge x \in T \implies f$ 
 $x = g \text{ } x \rrbracket$ 
     $\implies \text{continuous\_on } (S \cup T) (\lambda x. \text{if } x \in S \text{ then } f \text{ } x \text{ else } g \text{ } x)$ 
  shows Borsukian(S ∪ T)
proof (clarsimp simp add: Borsukian_continuous_logarithm)
  fix f :: 'a ⇒ complex
  assume conf: continuous_on (S ∪ T) f and 0: 0 ∉ f ` (S ∪ T)
  then have confS: continuous_on S f and confT: continuous_on T f
    using continuous_on_subset by auto
  have ⟦continuous_on S f; f ` S ⊆ −{0}⟧ ⟹ ∃ g. continuous_on S g ∧ (∀ x ∈

```

```

S. f x = exp(g x)
  using BS by (auto simp: Borsukian_continuous_logarithm)
  then obtain g where contg: continuous_on S g and fg:  $\bigwedge x. x \in S \implies f x = \exp(g x)$ 
    using 0 contfS by blast
  have  $\llbracket \text{continuous\_on } T f; f \text{ ' } T \subseteq -\{0\} \rrbracket \implies \exists g. \text{continuous\_on } T g \wedge (\forall x \in T. f x = \exp(g x))$ 
    using BT by (auto simp: Borsukian_continuous_logarithm)
  then obtain h where conth: continuous_on T h and fh:  $\bigwedge x. x \in T \implies f x = \exp(h x)$ 
    using 0 contfT by blast
  show  $\exists g. \text{continuous\_on } (S \cup T) g \wedge (\forall x \in S \cup T. f x = \exp(g x))$ 
  proof (cases  $S \cap T = \{\}$ )
    case True
      show ?thesis
      proof (intro exI conjI)
        show continuous_on (S  $\cup$  T) ( $\lambda x. \text{if } x \in S \text{ then } g x \text{ else } h x$ )
          using True * [OF contg conth]
          by (meson disjoint_iff)
        show  $\forall x \in S \cup T. f x = \exp(\text{if } x \in S \text{ then } g x \text{ else } h x)$ 
          using fg fh by auto
      qed
    next
      case False
        have ( $\lambda x. g x - h x$ ) constant_on S  $\cap$  T
        proof (rule continuous_discrete_range_constant [OF ST])
          show continuous_on (S  $\cap$  T) ( $\lambda x. g x - h x$ )
          proof (intro continuous_intros)
            show continuous_on (S  $\cap$  T) g
              by (meson contg continuous_on_subset inf_le1)
            show continuous_on (S  $\cap$  T) h
              by (meson conth continuous_on_subset inf_sup_ord(2))
          qed
          show  $\exists e > 0. \forall y. y \in S \cap T \wedge g y - h y \neq g x - h x \longrightarrow e \leq \text{cmod}(g y - h y - (g x - h x))$ 
            if  $x \in S \cap T$  for x
          proof -
            have  $g y - g x = h y - h x$ 
              if  $y \in S y \in T$  cmod (g y - g x - (h y - h x)) < 2 * pi for y
            proof (rule exp_complex_eqI)
              have  $|\text{Im}(g y - g x) - \text{Im}(h y - h x)| \leq \text{cmod}(g y - g x - (h y - h x))$ 
                by (metis abs_Im_le_cmod minus_complex.simps(2))
              then show  $|\text{Im}(g y - g x) - \text{Im}(h y - h x)| < 2 * pi$ 
                using that by linarith
              have  $\exp(g x) = \exp(h x) \exp(g y) = \exp(h y)$ 
                using fg fh that  $\langle x \in S \cap T \rangle$  by fastforce+
              then show  $\exp(g y - g x) = \exp(h y - h x)$ 
                by (simp add: exp_diff)
            qed
          qed
        qed

```

```

      qed
    then show ?thesis
      by (rule_tac x=2*pi in exI) (fastforce simp add: algebra_simps)
    qed
  qed
  then obtain a where a:  $\bigwedge x. x \in S \cap T \implies g\ x - h\ x = a$ 
    by (auto simp: constant_on_def)
  with False have exp a = 1
    by (metis IntI disjoint_iff_not_equal divide_self_if exp_diff exp_not_eq_zero
fg fh)
  with a show ?thesis
    apply (rule_tac x= $\lambda x. \text{if } x \in S \text{ then } g\ x \text{ else } a + h\ x$  in exI)
    apply (intro * contg conth continuous_intros conjI)
    apply (auto simp: algebra_simps fg fh exp_add)
    done
  qed
qed

proposition Borsukian_open_Un:
  fixes S :: 'a::real_normed_vector set
  assumes opeS: openin (top_of_set (S  $\cup$  T)) S
    and opeT: openin (top_of_set (S  $\cup$  T)) T
    and BS: Borsukian S and BT: Borsukian T and ST: connected(S  $\cap$  T)
  shows Borsukian(S  $\cup$  T)
  by (force intro: Borsukian_Un_lemma [OF BS BT ST] continuous_on_cases_local_open
[OF opeS opeT])

lemma Borsukian_closed_Un:
  fixes S :: 'a::real_normed_vector set
  assumes cloS: closedin (top_of_set (S  $\cup$  T)) S
    and cloT: closedin (top_of_set (S  $\cup$  T)) T
    and BS: Borsukian S and BT: Borsukian T and ST: connected(S  $\cap$  T)
  shows Borsukian(S  $\cup$  T)
  by (force intro: Borsukian_Un_lemma [OF BS BT ST] continuous_on_cases_local
[OF cloS cloT])

lemma Borsukian_separation_compact:
  fixes S :: complex set
  assumes compact S
  shows Borsukian S  $\longleftrightarrow$  connected(- S)
  by (simp add: Borsuk_separation_theorem Borsukian_circle assms)

lemma Borsukian_monotone_image_compact:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes Borsukian S and contf: continuous_on S f and fim: f ' S = T
    and compact S and conn:  $\bigwedge y. y \in T \implies \text{connected } \{x. x \in S \wedge f\ x = y\}$ 
  shows Borsukian T

proof (clarsimp simp add: Borsukian_continuous_logarithm)
  fix g :: 'b  $\Rightarrow$  complex

```

```

assume contg: continuous_on T g and 0: 0 ∉ g ` T
have continuous_on S (g ∘ f)
  using contf contg continuous_on_compose fim by blast
moreover have (g ∘ f) ` S ⊆ -{0}
  using fim 0 by auto
ultimately obtain h where conth: continuous_on S h and gfh:  $\bigwedge x. x \in S \implies$ 
(g ∘ f) x = exp(h x)
  using ⟨Borsukian S⟩ by (auto simp: Borsukian_continuous_logarithm)
have  $\bigwedge y. \exists x. y \in T \longrightarrow x \in S \wedge f x = y$ 
  using fim by auto
then obtain f' where f':  $\bigwedge y. y \in T \longrightarrow f' y \in S \wedge f (f' y) = y$ 
  by metis
have *: ( $\lambda x. h x - h(f' y)$ ) constant_on {x. x ∈ S ∧ f x = y} if y ∈ T for y
proof (rule continuous_discrete_range_constant [OF conn [OF that], of  $\lambda x. h$ 
x - h (f' y)], simp_all add: algebra_simps)
  show continuous_on {x ∈ S. f x = y} ( $\lambda x. h x - h (f' y)$ )
    by (intro continuous_intros continuous_on_subset [OF conth]) auto
  show  $\exists e > 0. \forall u. u \in S \wedge f u = y \wedge h u \neq h x \longrightarrow e \leq cmod (h u - h x)$ 
    if x: x ∈ S ∧ f x = y for x
  proof -
    have h u = h x if u ∈ S f u = y cmod (h u - h x) < 2 * pi for u
    proof (rule exp_complex_eqI)
      have |Im (h u) - Im (h x)| ≤ cmod (h u - h x)
        by (metis abs_Im_le_cmod minus_complex.simps(2))
      then show |Im (h u) - Im (h x)| < 2 * pi
        using that by linarith
      show exp (h u) = exp (h x)
        by (simp add: gfh [symmetric] x that)
    qed
    then show ?thesis
      by (rule_tac x=2*pi in exI) (fastforce simp add: algebra_simps)
  qed
qed
show  $\exists h. \text{continuous\_on } T h \wedge (\forall x \in T. g x = \text{exp } (h x))$ 
proof (intro exI conjI)
  show continuous_on T (h ∘ f')
  proof (rule continuous_from_closed_graph [of h ` S])
    show compact (h ` S)
      by (simp add: ⟨compact S⟩ compact_continuous_image conth)
    show (h ∘ f') ` T ⊆ h ` S
      by (auto simp: f')
    have h x = h (f' (f x)) if x ∈ S for x
      using * [of f x] fim that unfolding constant_on_def by clarsimp (metis f'
imageI right_minus_eq)
    moreover have  $\bigwedge x. x \in T \implies \exists u. u \in S \wedge x = f u \wedge h (f' x) = h u$ 
      using f' by fastforce
    ultimately
    have eq: (( $\lambda x. (x, (h \circ f') x)$ ) ` T) =
      {p.  $\exists x. x \in S \wedge (x, p) \in (S \times UNIV) \cap ((\lambda z. snd z - ((f \circ fst) z,$ 

```

```

(h ∘ fst) z)) - ' {0} }}
  using fim by (auto simp: image_iff)
  moreover have closed ...
  apply (intro closed_compact_projection [OF <compact S>] continuous_closed_preimage
    continuous_intros continuous_on_subset [OF contf] continu-
ous_on_subset [OF conth])
    by (auto simp: <compact S> closed_Times compact_imp_closed)
  ultimately show closed ((λx. (x, (h ∘ f') x)) ' T)
    by simp
qed
qed (use f' gfh in fastforce)
qed

```

```

lemma Borsukian_open_map_image_compact:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes Borsukian S and contf: continuous_on S f and fim: f ' S = T and
compact S
  and ope:  $\bigwedge U. \text{openin } (\text{top\_of\_set } S) U \implies \text{openin } (\text{top\_of\_set } T) (f ' U)$ 
  shows Borsukian T
proof (clarsimp simp add: Borsukian_continuous_logarithm_circle_real)
  fix g :: 'b ⇒ complex
  assume contg: continuous_on T g and gim: g ' T ⊆ sphere 0 1
  have continuous_on S (g ∘ f)
    using contf contg continuous_on_compose fim by blast
  moreover have (g ∘ f) ' S ⊆ sphere 0 1
    using fim gim by auto
  ultimately obtain h where cont_cxh: continuous_on S (complex_of_real ∘ h)
    and gfh:  $\bigwedge x. x \in S \implies (g \circ f) x = \exp(i * \text{of\_real}(h x))$ 
  using <Borsukian S> Borsukian_continuous_logarithm_circle_real by metis
  then have conth: continuous_on S h
    by simp
  have  $\exists x. x \in S \wedge f x = y \wedge (\forall x' \in S. f x' = y \longrightarrow h x \leq h x')$  if  $y \in T$  for  $y$ 
  proof -
    have 1: compact (h ' {x ∈ S. f x = y})
    proof (rule compact_continuous_image)
      show continuous_on {x ∈ S. f x = y} h
        by (rule continuous_on_subset [OF conth]) auto
      have compact (S ∩ f - ' {y})
        by (rule proper_map_from_compact [OF contf _ <compact S>, of T])
        (simp_all add: fim that)
      then show compact {x ∈ S. f x = y}
        by (auto simp: vimage_def Int_def)
    qed
    have 2: h ' {x ∈ S. f x = y} ≠ {}
      using fim that by auto
    have  $\exists s \in h ' \{x \in S. f x = y\}. \forall t \in h ' \{x \in S. f x = y\}. s \leq t$ 
      using compact_attains_inf [OF 1 2] by blast
  qed

```

```

    then show ?thesis by auto
qed
then obtain k where kTS:  $\bigwedge y. y \in T \implies k y \in S$ 
    and fk:  $\bigwedge y. y \in T \implies f (k y) = y$ 
    and hle:  $\bigwedge x' y. \llbracket y \in T; x' \in S; f x' = y \rrbracket \implies h (k y) \leq h x'$ 
  by metis
have continuous_on T (h o k)
proof (clarsimp simp add: continuous_on_iff)
  fix y and e::real
  assume y  $\in T$  0 < e
  moreover have uniformly_continuous_on S (complex_of_real o h)
    using <compact S> cont_cxh compact_uniformly_continuous by blast
  ultimately obtain d where 0 < d
    and d:  $\bigwedge x x'. \llbracket x \in S; x' \in S; \text{dist } x' x < d \rrbracket \implies \text{dist } (h x') (h x) < e$ 
  by (force simp: uniformly_continuous_on_def)
  obtain  $\delta$  where 0 <  $\delta$  and  $\delta$ :
     $\bigwedge x'. \llbracket x' \in T; \text{dist } y x' < \delta \rrbracket$ 
     $\implies (\forall v \in \{z \in S. f z = y\}. \exists v'. v' \in \{z \in S. f z = x'\} \wedge \text{dist } v v' <$ 
d)  $\wedge$ 
     $(\forall v' \in \{z \in S. f z = x'\}. \exists v. v \in \{z \in S. f z = y\} \wedge \text{dist } v' v < d)$ 
  proof (rule upper_lower_hemicontinuous_explicit [of T  $\lambda y. \{z \in S. f z = y\}$ 
S])
    show  $\bigwedge U. \text{openin } (\text{top\_of\_set } S) U$ 
       $\implies \text{openin } (\text{top\_of\_set } T) \{x \in T. \{z \in S. f z = x\} \subseteq U\}$ 
    using closed_map_iff_upper_hemicontinuous_preimage [OF fim [THEN
equalityD1]]
    by (simp add: Abstract_Topology_2.continuous_imp_closed_map <compact
S> contf fim)
    show  $\bigwedge U. \text{closedin } (\text{top\_of\_set } S) U \implies$ 
       $\text{closedin } (\text{top\_of\_set } T) \{x \in T. \{z \in S. f z = x\} \subseteq U\}$ 
    using ope_open_map_iff_lower_hemicontinuous_preimage [OF fim [THEN
equalityD1]]
    by meson
    show bounded  $\{z \in S. f z = y\}$ 
    by (metis (no_types, lifting) compact_imp_bounded [OF <compact S>]
bounded_subset mem_Collect_eq subsetI)
  qed (use <y  $\in T$ > <0 < d> fk kTS in <force+>)
  have  $\text{dist } (h (k y')) (h (k y)) < e$  if  $y' \in T$   $\text{dist } y y' < \delta$  for  $y'$ 
  proof -
    have k1:  $k y \in S$   $f (k y) = y$  and k2:  $k y' \in S$   $f (k y') = y'$ 
    by (auto simp: <y  $\in T$ > <y'  $\in T$ > kTS fk)
    have 1:  $\bigwedge v. \llbracket v \in S; f v = y \rrbracket \implies \exists v'. v' \in \{z \in S. f z = y'\} \wedge \text{dist } v v' < d$ 
    and 2:  $\bigwedge v'. \llbracket v' \in S; f v' = y \rrbracket \implies \exists v. v \in \{z \in S. f z = y'\} \wedge \text{dist } v' v < d$ 
    using  $\delta$  [OF that] by auto
    then obtain  $w' w$  where  $w' \in S$   $f w' = y'$   $\text{dist } (k y) w' < d$ 
    and  $w \in S$   $f w = y$   $\text{dist } (k y') w < d$ 
    using 1 [OF k1] 2 [OF k2] by auto
    then show ?thesis
    using d [of w k y] d [of w' k y] k1 k2 <y'  $\in T$ > <y  $\in T$ > hle

```

```

    by (fastforce simp: dist_norm abs_diff_less_iff algebra_simps)
  qed
  then show  $\exists d > 0. \forall x' \in T. \text{dist } x' y < d \longrightarrow \text{dist } (h (k x')) (h (k y)) < e$ 
    using  $\langle 0 < \delta \rangle$  by (auto simp: dist_commute)
  qed
  then show  $\exists h. \text{continuous\_on } T h \wedge (\forall x \in T. g x = \exp (i * \text{complex\_of\_real } (h x)))$ 
    using fk gfh kTS by force
  qed

```

If two points are separated by a closed set, there's a minimal one.

proposition *closed_irreducible_separator:*

```

  fixes  $a :: 'a :: \text{real\_normed\_vector}$ 
  assumes closed S and  $ab: \neg \text{connected\_component } (- S) a b$ 
  obtains T where  $T \subseteq S$  closed T  $T \neq \{\}$   $\neg \text{connected\_component } (- T) a b$ 
     $\bigwedge U. U \subset T \implies \text{connected\_component } (- U) a b$ 
proof (cases  $a \in S \vee b \in S$ )
  case True
  then show ?thesis
  proof
    assume *:  $a \in S$ 
    show ?thesis
    proof
      show  $\{a\} \subseteq S$ 
      using * by blast
      show  $\neg \text{connected\_component } (- \{a\}) a b$ 
      using connected_component_in by auto
      show  $\bigwedge U. U \subset \{a\} \implies \text{connected\_component } (- U) a b$ 
      by (metis connected_component_UNIV UNIV_I compl_bot_eq connected_component_eq_eq
        less_le_not_le subset_singletonD)
    qed auto
  next
    assume *:  $b \in S$ 
    show ?thesis
    proof
      show  $\{b\} \subseteq S$ 
      using * by blast
      show  $\neg \text{connected\_component } (- \{b\}) a b$ 
      using connected_component_in by auto
      show  $\bigwedge U. U \subset \{b\} \implies \text{connected\_component } (- U) a b$ 
      by (metis connected_component_UNIV UNIV_I compl_bot_eq connected_component_eq_eq
        less_le_not_le subset_singletonD)
    qed auto
  qed
next
  case False
  define A where  $A \equiv \text{connected\_component\_set } (- S) a$ 
  define B where  $B \equiv \text{connected\_component\_set } (- (\text{closure } A)) b$ 
  have  $a \in A$ 

```

```

    using False A_def by auto
  have b ∈ B
    unfolding A_def B_def closure_Un_frontier
    using ab False ⟨closed S⟩ frontier_complement frontier_of_connected_component_subset
frontier_subset_closed by force
  have frontier B ⊆ frontier (connected_component_set (− closure A) b)
    using B_def by blast
  also have frsub: ... ⊆ frontier A
  proof −
    have ∧A. closure (− closure (− A)) ⊆ closure A
      by (metis (no_types) closure_mono closure_subset compl_le_compl_iff double_compl)
    then show ?thesis
      by (metis (no_types) closure_closure double_compl frontier_closures frontier_of_connected_component_subset le_inf_iff subset_trans)
  qed
  finally have frBA: frontier B ⊆ frontier A .
  show ?thesis
  proof
    show frontier B ⊆ S
    proof −
      have frontier S ⊆ S
        by (simp add: ⟨closed S⟩ frontier_subset_closed)
      then show ?thesis
        using frsub frontier_complement frontier_of_connected_component_subset
        unfolding A_def B_def by blast
    qed
    show closed (frontier B)
      by simp
    show ¬ connected_component (− frontier B) a b
      unfolding connected_component_def
    proof clarify
      fix T
      assume connected T and TB: T ⊆ − frontier B and a ∈ T and b ∈ T
      have a ∉ B
        by (metis A_def B_def ComplD ⟨a ∈ A⟩ assms(1) closed_open connected_component_subset in_closure_connected_component subsetD)
      have T ∩ B ≠ {}
        using ⟨b ∈ B⟩ ⟨b ∈ T⟩ by blast
      moreover have T − B ≠ {}
        using ⟨a ∉ B⟩ ⟨a ∈ T⟩ by blast
      ultimately show False
        using connected_Int_frontier [of T B] TB ⟨connected T⟩ by blast
    qed
    moreover have connected_component (− frontier B) a b if frontier B = {}
      using connected_component_eq_UNIV that by auto
    ultimately show frontier B ≠ {}
      by blast
    show connected_component (− U) a b if U ⊂ frontier B for U

```



```

proof –
  obtain  $p$  where  $U_{\text{sub}}: U \subseteq \text{frontier } B$  and  $p: p \in \text{frontier } B \ p \notin U$ 
  using  $\langle U \subset \text{frontier } B \rangle$  by blast
  show ?thesis
  unfolding connected_component_def
  proof (intro exI conjI)
    have connected  $((\text{insert } p \ A) \cup (\text{insert } p \ B))$ 
    proof (rule connected_Un)
      show connected  $(\text{insert } p \ A)$ 
      by (metis A_def IntD1 frBA  $\langle p \in \text{frontier } B \rangle$  closure_insert closure_subset
connected_connected_component connected_intermediate_closure frontier_closures
insert_absorb subsetCE subset_insertI)
      show connected  $(\text{insert } p \ B)$ 
      by (metis B_def IntD1  $\langle p \in \text{frontier } B \rangle$  closure_insert closure_subset con-
nected_connected_component connected_intermediate_closure frontier_closures in-
sert_absorb subset_insertI)
    qed blast
    then show connected  $(\text{insert } p \ (B \cup A))$ 
    by (simp add: sup.commute)
    have  $A \subseteq - \ U$ 
    using  $A\_def \ U_{\text{sub}} \ \langle \text{frontier } B \subseteq S \rangle$  connected_component_subset by
fastforce
    moreover have  $B \subseteq - \ U$ 
    using  $B\_def \ U_{\text{sub}} \ \text{connected\_component\_subset } frBA \ \text{frontier\_closures}$ 
by fastforce
    ultimately show  $\text{insert } p \ (B \cup A) \subseteq - \ U$ 
    using  $p$  by auto
    qed (auto simp:  $\langle a \in A \rangle \langle b \in B \rangle$ )
  qed
qed
qed

```

lemma *frontier_minimal_separating_closed_pointwise*:

```

fixes  $S :: 'a::\text{real\_normed\_vector\_set}$ 
assumes  $S$ : closed  $S \ a \notin S$  and  $nconn$ :  $\neg \text{connected\_component } (- \ S) \ a \ b$ 
and  $conn$ :  $\bigwedge T. \llbracket \text{closed } T; T \subset S \rrbracket \implies \text{connected\_component } (- \ T) \ a \ b$ 
shows  $\text{frontier}(\text{connected\_component\_set } (- \ S) \ a) = S$  (is  $?F = S$ )
proof –
  have  $?F \subseteq S$ 
  by (simp add: S componentsI frontier_of_components_closed_complement)
  moreover have  $\text{False}$  if  $?F \subset S$ 
  proof –
    have connected_component  $(- \ ?F) \ a \ b$ 
    by (simp add: conn that)
    then obtain  $T$  where connected  $T \ T \subseteq - \ ?F \ a \in T \ b \in T$ 
    by (auto simp: connected_component_def)
    moreover have  $T \cap ?F \neq \{\}$ 
    proof (rule connected_Int_frontier [OF  $\langle \text{connected } T \rangle$ ])
      show  $T \cap \text{connected\_component\_set } (- \ S) \ a \neq \{\}$ 

```

```

      using ⟨a ∉ S⟩ ⟨a ∈ T⟩ by fastforce
    show T = connected_component_set (− S) a ≠ {}
      using ⟨b ∈ T⟩ nconn by blast
  qed
  ultimately show ?thesis
    by blast
qed
ultimately show ?thesis
  by blast
qed

```

6.42.17 Unicoherence (closed)

definition *unicoherent* where

$$\begin{aligned} \text{unicoherent } U &\equiv \\ \forall S T. \text{ connected } S \wedge \text{ connected } T \wedge S \cup T = U \wedge \\ &\quad \text{closedin } (\text{top\_of\_set } U) S \wedge \text{closedin } (\text{top\_of\_set } U) T \\ &\quad \longrightarrow \text{ connected } (S \cap T) \end{aligned}$$

lemma *unicoherentI* [intro?]:

```

  assumes ∧S T. ⟦connected S; connected T; U = S ∪ T; closedin (top_of_set
    U) S; closedin (top_of_set U) T⟧
    ⟹ connected (S ∩ T)
  shows unicoherent U
  using assms unfolding unicoherent_def by blast

```

lemma *unicoherentD*:

```

  assumes unicoherent U connected S connected T U = S ∪ T closedin (top_of_set
    U) S closedin (top_of_set U) T
  shows connected (S ∩ T)
  using assms unfolding unicoherent_def by blast

```

proposition *homeomorphic_unicoherent*:

```

  assumes ST: S homeomorphic T and S: unicoherent S
  shows unicoherent T

```

proof –

```

  obtain f g where gf: ∧x. x ∈ S ⟹ g (f x) = x and fim: T = f ‘ S and gfm:
    g ‘ f ‘ S = S
  and contf: continuous_on S f and contg: continuous_on (f ‘ S) g
  using ST by (auto simp: homeomorphic_def homeomorphism_def)
  show ?thesis

```

proof

```

  fix U V

```

```

  assume connected U connected V and T: T = U ∪ V

```

```

    and cloU: closedin (top_of_set T) U

```

```

    and cloV: closedin (top_of_set T) V

```

```

  have f ‘ (g ‘ U ∩ g ‘ V) ⊆ U f ‘ (g ‘ U ∩ g ‘ V) ⊆ V

```

```

    using gf fim T by auto (metis UnCI image_iff)+

```

```

  moreover have U ∩ V ⊆ f ‘ (g ‘ U ∩ g ‘ V)

```

```

    using gf fim by (force simp: image_iff T)
    ultimately have  $U \cap V = f \, ' (g \, ' U \cap g \, ' V)$  by blast
    moreover have connected (f \, ' (g \, ' U \cap g \, ' V))
    proof (rule connected_continuous_image)
      show continuous_on (g \, ' U \cap g \, ' V) f
        using T fim gfim by (metis Un_upper1 contf continuous_on_subset image_mono inf_le1)
      show connected (g \, ' U \cap g \, ' V)
      proof (intro conjI unicoherentD [OF S])
        show connected (g \, ' U) connected (g \, ' V)
          using <connected U> cloU <connected V> cloV
        by (metis Topological_Spaces.connected_continuous_image closedin_imp_subset
          contg continuous_on_subset fim)+
      show  $S = g \, ' U \cup g \, ' V$ 
        using T fim gfim by auto
      have hom: homeomorphism T S g f
        by (simp add: contf contg fim gf gfim homeomorphism_def)
      have closedin (top_of_set T) U closedin (top_of_set T) V
        by (simp_all add: cloU cloV)
      then show closedin (top_of_set S) (g \, ' U)
        closedin (top_of_set S) (g \, ' V)
        by (blast intro: homeomorphism_imp_closed_map [OF hom])+
    qed
  qed
  ultimately show connected (U \cap V) by metis
qed
qed

```

```

lemma homeomorphic_unicoherent_eq:
   $S \text{ homeomorphic } T \implies (\text{unicoherent } S \longleftrightarrow \text{unicoherent } T)$ 
  by (meson homeomorphic_sym homeomorphic_unicoherent)

```

```

lemma unicoherent_translation:
  fixes S :: 'a::real_normed_vector set
  shows
     $\text{unicoherent } (\text{image } (\lambda x. a + x) S) \longleftrightarrow \text{unicoherent } S$ 
  using homeomorphic_translation homeomorphic_unicoherent_eq by blast

```

```

lemma unicoherent_injective_linear_image:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes linear f inj f
  shows  $(\text{unicoherent}(f \, ' S) \longleftrightarrow \text{unicoherent } S)$ 
  using assms homeomorphic_unicoherent_eq linear_homeomorphic_image by
  blast

```

```

lemma Borsukian_imp_unicoherent:
  fixes U :: 'a::euclidean_space set

```

```

    assumes Borsukian U shows unicoherent U
  unfolding unicoherent_def
proof clarify
  fix S T
  assume connected S connected T  $U = S \cup T$ 
    and cloS: closedin (top_of_set ( $S \cup T$ )) S
    and cloT: closedin (top_of_set ( $S \cup T$ )) T
  show connected ( $S \cap T$ )
    unfolding connected_closedin_eq
proof clarify
  fix V W
  assume closedin (top_of_set ( $S \cap T$ )) V
    and closedin (top_of_set ( $S \cap T$ )) W
    and VW:  $V \cup W = S \cap T$   $V \cap W = \{\}$  and  $V \neq \{\}$   $W \neq \{\}$ 
  then have cloV: closedin (top_of_set U) V and cloW: closedin (top_of_set
U) W
    using  $\langle U = S \cup T \rangle$  cloS cloT closedin_trans by blast+
  obtain q where contq: continuous_on U q
    and q01:  $\bigwedge x. x \in U \implies q\ x \in \{0..1::real\}$ 
    and qV:  $\bigwedge x. x \in V \implies q\ x = 0$  and qW:  $\bigwedge x. x \in W \implies q\ x = 1$ 
  by (rule Urysohn_local [OF cloV cloW  $\langle V \cap W = \{\} \rangle$ , of 0 1])
    (fastforce simp: closed_segment_eq_real_ivl)
  let ?h =  $\lambda x. \text{if } x \in S \text{ then } \exp(\pi * i * q\ x) \text{ else } 1 / \exp(\pi * i * q\ x)$ 
  have eqST:  $\exp(\pi * i * q\ x) = 1 / \exp(\pi * i * q\ x)$  if  $x \in S \cap T$  for x
proof –
    have  $x \in V \cup W$ 
      using that  $\langle V \cup W = S \cap T \rangle$  by blast
    with qV qW show ?thesis by force
  qed
  obtain g where contg: continuous_on U g
    and circle:  $g\ ` U \subseteq \text{sphere } 0\ 1$ 
    and S:  $\bigwedge x. x \in S \implies g\ x = \exp(\pi * i * q\ x)$ 
    and T:  $\bigwedge x. x \in T \implies g\ x = 1 / \exp(\pi * i * q\ x)$ 
proof
  show continuous_on U ?h
    unfolding  $\langle U = S \cup T \rangle$ 
  proof (rule continuous_on_cases_local [OF cloS cloT])
    show continuous_on S ( $\lambda x. \exp(\pi * i * q\ x)$ )
      proof (intro continuous_intros)
        show continuous_on S q
          using  $\langle U = S \cup T \rangle$  continuous_on_subset contq by blast
        qed
      show continuous_on T ( $\lambda x. 1 / \exp(\pi * i * q\ x)$ )
        proof (intro continuous_intros)
          show continuous_on T q
            using  $\langle U = S \cup T \rangle$  continuous_on_subset contq by auto
          qed auto
        qed (use eqST in auto)
      qed (use eqST in  $\langle \text{auto simp: norm\_divide} \rangle$ )

```

```

    then obtain h where conth: continuous_on U h and heq:  $\bigwedge x. x \in U \implies g$ 
    x = exp (h x)
    by (metis Borsukian_continuous_logarithm_circle assms)
  obtain v w where v  $\in V$  w  $\in W$ 
    using  $\langle V \neq \{\} \rangle \langle W \neq \{\} \rangle$  by blast
  then have vw: v  $\in S \cap T$  w  $\in S \cap T$ 
    using VW by auto
  have iff:  $2 * pi \leq cmod (2 * of\_int m * of\_real pi * i - 2 * of\_int n * of\_real pi * i)$ 
     $\longleftrightarrow 1 \leq abs (m - n)$  for m n
  proof -
    have  $2 * pi \leq cmod (2 * of\_int m * of\_real pi * i - 2 * of\_int n * of\_real pi * i)$ 
     $\longleftrightarrow 2 * pi \leq cmod ((2 * pi * i) * (of\_int m - of\_int n))$ 
    by (simp add: algebra_simps)
    also have ...  $\longleftrightarrow 2 * pi \leq 2 * pi * cmod (of\_int m - of\_int n)$ 
    by (simp add: norm_mult)
    also have ...  $\longleftrightarrow 1 \leq abs (m - n)$ 
    by simp (metis norm_of_int of_int_1_le_iff of_int_abs of_int_diff)
    finally show ?thesis .
  qed
  have *:  $\exists n::int. h x - (pi * i * q x) = (of\_int(2*n) * pi) * i$  if x  $\in S$  for x
    using that S  $\langle U = S \cup T \rangle$  heq exp_eq [symmetric] by (simp add: algebra_simps)
  moreover have  $(\lambda x. h x - (pi * i * q x))$  constant_on S
  proof (rule continuous_discrete_range_constant [OF  $\langle connected S \rangle$ ])
    have continuous_on S h continuous_on S q
      using  $\langle U = S \cup T \rangle$  continuous_on_subset conth contq by blast+
    then show continuous_on S  $(\lambda x. h x - (pi * i * q x))$ 
      by (intro continuous_intros)
    have  $2*pi \leq cmod (h y - (pi * i * q y) - (h x - (pi * i * q x)))$ 
      if x  $\in S$  y  $\in S$  and ne:  $h y - (pi * i * q y) \neq h x - (pi * i * q x)$  for x y
      using * [OF  $\langle x \in S \rangle$ ] * [OF  $\langle y \in S \rangle$ ] ne by (auto simp: iff)
    then show  $\bigwedge x. x \in S \implies$ 
       $\exists e>0. \forall y. y \in S \wedge h y - (pi * i * q y) \neq h x - (pi * i * q x) \longrightarrow$ 
       $e \leq cmod (h y - (pi * i * q y) - (h x - (pi * i * q x)))$ 
      by (rule_tac x=2*pi in exI) auto
  qed
  ultimately
  obtain m where m:  $\bigwedge x. x \in S \implies h x - (pi * i * q x) = (of\_int(2*m) * pi) * i$ 
  * i
    using vw by (force simp: constant_on_def)
  have *:  $\exists n::int. h x = - (pi * i * q x) + (of\_int(2*n) * pi) * i$  if x  $\in T$  for x
    unfolding exp_eq [symmetric]
    using that T  $\langle U = S \cup T \rangle$  by (simp add: exp_minus field_simps heq [symmetric])
  moreover have  $(\lambda x. h x + (pi * i * q x))$  constant_on T
  proof (rule continuous_discrete_range_constant [OF  $\langle connected T \rangle$ ])
    have continuous_on T h continuous_on T q

```

```

    using ⟨U = S ∪ T⟩ continuous_on_subset conth contq by blast+
  then show continuous_on T (λx. h x + (pi * i * q x))
  by (intro continuous_intros)
  have 2*pi ≤ cmod (h y + (pi * i * q y) - (h x + (pi * i * q x)))
  if x ∈ T y ∈ T and ne: h y + (pi * i * q y) ≠ h x + (pi * i * q x) for x y
  using * [OF ⟨x ∈ T⟩] * [OF ⟨y ∈ T⟩] ne by (auto simp: iff)
  then show ∧x. x ∈ T ⇒
    ∃ e > 0. ∀ y. y ∈ T ∧ h y + (pi * i * q y) ≠ h x + (pi * i * q x) →
      e ≤ cmod (h y + (pi * i * q y) - (h x + (pi * i * q x)))
  by (rule_tac x=2*pi in exI) auto
qed
ultimately
obtain n where n: ∧x. x ∈ T ⇒ h x + (pi * i * q x) = (of_int(2*n) * pi)
* i
  using vw by (force simp: constant_on_def)
  show False
  using m [of v] m [of w] n [of v] n [of w] vw
  by (auto simp: algebra_simps ⟨v ∈ V⟩ ⟨w ∈ W⟩ qV qW)
qed
qed

```

```

corollary contractible_imp_unicoherent:
  fixes U :: 'a::euclidean_space set
  assumes contractible U shows unicoherent U
  by (simp add: Borsukian_imp_unicoherent assms contractible_imp_Borsukian)

```

```

corollary convex_imp_unicoherent:
  fixes U :: 'a::euclidean_space set
  assumes convex U shows unicoherent U
  by (simp add: Borsukian_imp_unicoherent assms convex_imp_Borsukian)

```

If the type class constraint can be relaxed, I don't know how!

```

corollary unicoherent_UNIV: unicoherent (UNIV :: 'a :: euclidean_space set)
  by (simp add: convex_imp_unicoherent)

```

```

lemma unicoherent_monotone_image_compact:
  fixes T :: 'b :: t2_space set
  assumes S: unicoherent S compact S and conf: continuous_on S f and fim: f
    ' S = T
  and conn: ∧y. y ∈ T ⇒ connected (S ∩ f - ' {y})
  shows unicoherent T
proof
  fix U V
  assume UV: connected U connected V T = U ∪ V
    and cloU: closedin (top_of_set T) U
    and cloV: closedin (top_of_set T) V
  moreover have compact T

```

```

    using ⟨compact S⟩ compact_continuous_image contf fim by blast
  ultimately have closed U closed V
    by (auto simp: closedin_closed_eq compact_imp_closed)
  let ?SUV = (S ∩ f -‘ U) ∩ (S ∩ f -‘ V)
  have UV_eq: f ‘ ?SUV = U ∩ V
    using ⟨T = U ∪ V⟩ fim by force+
  have connected (f ‘ ?SUV)
  proof (rule connected_continuous_image)
    show continuous_on ?SUV f
      by (meson contf continuous_on_subset inf_le1)
    show connected ?SUV
  proof (rule unicoherentD [OF ⟨unicoherent S⟩, of S ∩ f -‘ U S ∩ f -‘ V])
    have  $\bigwedge C$ . closedin (top_of_set S) C  $\implies$  closedin (top_of_set T) (f ‘ C)
      by (metis ⟨compact S⟩ closed_subset closedin_compact closedin_imp_subset
        compact_continuous_image compact_imp_closed contf continuous_on_subset fim
        image_mono)
    then show connected (S ∩ f -‘ U) connected (S ∩ f -‘ V)
      using UV by (auto simp: conn intro: connected_closed_monotone_preimage
        [OF contf fim])
    show S = (S ∩ f -‘ U) ∪ (S ∩ f -‘ V)
      using UV fim by blast
    show closedin (top_of_set S) (S ∩ f -‘ U)
      closedin (top_of_set S) (S ∩ f -‘ V)
      by (auto simp: continuous_on_imp_closedin cloU cloV contf fim)
  qed
  qed
  with UV_eq show connected (U ∩ V)
    by simp
  qed

```

6.42.18 Several common variants of unicoherence

lemma *connected_frontier_simple:*

fixes $S :: 'a :: \text{euclidean\_space set}$

assumes $\text{connected } S$ $\text{connected}(- S)$ **shows** $\text{connected}(\text{frontier } S)$

unfolding *frontier_closures*

by (rule unicoherentD [OF unicoherent_UNIV]; simp add: assms connected_imp_connected_closure flip: closure_Un)

lemma *connected_frontier_component_complement:*

fixes $S :: 'a :: \text{euclidean\_space set}$

assumes $\text{connected } S$ $C \in \text{components}(- S)$ **shows** $\text{connected}(\text{frontier } C)$

by (meson assms component_complement_connected connected_frontier_simple in_components_connected)

lemma *connected_frontier_disjoint:*

fixes $S :: 'a :: \text{euclidean\_space set}$

assumes $\text{connected } S$ $\text{connected } T$ $\text{disjnt } S T$ **and** ST : $\text{frontier } S \subseteq \text{frontier } T$

shows $\text{connected}(\text{frontier } S)$

```

proof (cases  $S = UNIV$ )
  case True then show ?thesis
    by simp
next
  case False
  then have  $-S \neq \{\}$ 
    by blast
  then obtain  $C$  where  $C: C \in \text{components}(-S)$  and  $T \subseteq C$ 
    by (metis ComplI disjoint_iff subsetI exists_component_superset disjoint S T)
  moreover have  $\text{frontier } S = \text{frontier } C$ 
  proof -
    have  $\text{frontier } C \subseteq \text{frontier } S$ 
      using  $C \text{ frontier\_complement frontier\_of\_components\_subset}$  by blast
    moreover have  $x \in \text{frontier } C \text{ if } x \in \text{frontier } S \text{ for } x$ 
    proof -
      have  $x \in \text{closure } C$ 
        using that unfolding frontier_def
        by (metis (no_types) Diff_eq ST T C closure_mono contra_subsetD frontier_def le_inf_iff that)
      moreover have  $x \notin \text{interior } C$ 
        using that unfolding frontier_def
        by (metis  $C \text{ Compl\_eq\_Diff\_UNIV Diff\_iff subsetD in\_components\_subset interior\_diff interior\_mono}$ )
      ultimately show ?thesis
        by (auto simp: frontier_def)
    qed
  ultimately show ?thesis
    by blast
qed
ultimately show ?thesis
  using  $\langle \text{connected } S \rangle \text{ connected\_frontier\_component\_complement}$  by auto
qed

```

6.42.19 Some separation results

```

lemma separation_by_component_closed_pointwise:
  fixes  $S :: 'a :: \text{euclidean\_space set}$ 
  assumes  $\text{closed } S \neg \text{connected\_component } (-S) a b$ 
  obtains  $C$  where  $C \in \text{components } S \neg \text{connected\_component}(-C) a b$ 
proof (cases  $a \in S \vee b \in S$ )
  case True
  then show ?thesis
    using  $\text{connected\_component\_in componentsI}$  that by fastforce
next
  case False
  obtain  $T$  where  $T \subseteq S \text{ closed } T T \neq \{\}$ 
    and  $\text{nab}: \neg \text{connected\_component } (-T) a b$ 
    and  $\text{conn}: \bigwedge U. U \subset T \implies \text{connected\_component } (-U) a b$ 

```



```

    using closed_irreducible_separator [OF assms] by metis
  moreover have connected T
  proof -
    have ab: frontier (connected_component_set (- T) a) = T frontier (connected_component_set
      (- T) b) = T
      using frontier_minimal_separating_closed_pointwise
      by (metis False  $\langle T \subseteq S \rangle$   $\langle \text{closed } T \rangle$  connected_component_sym conn connected_component_eq_empty
        connected_component_intermediate_subset empty_subsetI nab)+
    have connected (frontier (connected_component_set (- T) a))
    proof (rule connected_frontier_disjoint)
      show disjnt (connected_component_set (- T) a) (connected_component_set
        (- T) b)
        unfolding disjnt_iff
        by (metis connected_component_eq connected_component_eq_empty connected_component_idemp mem_Collect_eq nab)
      show frontier (connected_component_set (- T) a)  $\subseteq$  frontier (connected_component_set
        (- T) b)
        by (simp add: ab)
    qed auto
    with ab  $\langle \text{closed } T \rangle$  show ?thesis
    by simp
  qed
  ultimately obtain C where  $C \in \text{components } S$   $T \subseteq C$ 
  using exists_component_superset [of T S] by blast
  then show ?thesis
  by (meson Compl_anti_mono connected_component_of_subset nab that)
qed

```

lemma separation_by_component_closed:

```

  fixes S :: 'a :: euclidean_space set
  assumes closed S  $\neg$  connected(- S)
  obtains C where  $C \in \text{components } S$   $\neg$  connected(- C)
  proof -
    obtain x y where closed S  $x \notin S$   $y \notin S$  and  $\neg$  connected_component (- S) x y
      using assms by (auto simp: connected_iff_connected_component)
    then obtain C where  $C \in \text{components } S$   $\neg$  connected_component(- C) x y
      using separation_by_component_closed_pointwise by metis
    then show thesis
      by (metis Compl_iff  $\langle x \notin S \rangle$   $\langle y \notin S \rangle$  connected_component_eq_self_in_components_subset
        mem_Collect_eq subsetD that)
  qed

```

lemma separation_by_Un_closed_pointwise:

```

  fixes S :: 'a :: euclidean_space set
  assumes ST: closed S closed T  $S \cap T = \{\}$ 
    and conS: connected_component (- S) a b and conT: connected_component
      (- T) a b

```

```

    shows connected_component ( $-(S \cup T)$ )  $a \ b$ 
  proof (rule ccontr)
    have  $a \notin S \ b \notin S \ a \notin T \ b \notin T$ 
    using conS conT connected_component_in by auto
    assume  $\neg \text{connected\_component } (-(S \cup T)) \ a \ b$ 
    then obtain  $C$  where  $C \in \text{components } (S \cup T)$  and  $C: \neg \text{connected\_component } (-(C)) \ a \ b$ 
    using separation_by_component_closed_pointwise assms by blast
    then have  $C \subseteq S \vee C \subseteq T$ 
    proof -
      have connected  $C \ C \subseteq S \cup T$ 
      using  $\langle C \in \text{components } (S \cup T) \rangle$  in_components_subset by (blast elim:
componentsE)+
      moreover then have  $C \cap T = \{\} \vee C \cap S = \{\}$ 
      by (metis Int_empty_right ST inf commute connected_closed)
      ultimately show ?thesis
      by blast
    qed
    then show False
    by (meson Compl_anti_mono C conS conT connected_component_of_subset)
  qed

```

```

lemma separation_by_Un_closed:
  fixes  $S :: 'a :: \text{euclidean\_space set}$ 
  assumes  $ST: \text{closed } S \ \text{closed } T \ S \cap T = \{\}$  and conS: connected( $- S$ ) and
conT: connected( $- T$ )
  shows connected( $-(S \cup T)$ )
  using assms separation_by_Un_closed_pointwise
  by (fastforce simp add: connected_iff_connected_component)

```

```

lemma open_unicoherent_UNIV:
  fixes  $S :: 'a :: \text{euclidean\_space set}$ 
  assumes open  $S \ \text{open } T \ \text{connected } S \ \text{connected } T \ S \cup T = \text{UNIV}$ 
  shows connected( $S \cap T$ )
  proof -
    have connected( $-( -S \cup -T)$ )
    by (metis closed_Compl compl_sup compl_top_eq double_compl separation_by_Un_closed
assms)
    then show ?thesis
    by simp
  qed

```

```

lemma separation_by_component_open_aux:
  fixes  $S :: 'a :: \text{euclidean\_space set}$ 
  assumes  $ST: \text{closed } S \ \text{closed } T \ S \cap T = \{\}$ 
    and  $S \neq \{\} \ T \neq \{\}$ 
  obtains  $C$  where  $C \in \text{components } (-(S \cup T)) \ C \neq \{\} \ \text{frontier } C \cap S \neq \{\}$ 
frontier  $C \cap T \neq \{\}$ 
  proof (rule ccontr)

```

```

let ?S = S  $\cup$   $\bigcup \{C \in \text{components}(- (S \cup T)). \text{frontier } C \subseteq S\}$ 
let ?T = T  $\cup$   $\bigcup \{C \in \text{components}(- (S \cup T)). \text{frontier } C \subseteq T\}$ 
assume  $\neg \text{thesis}$ 
with that have *:  $\text{frontier } C \cap S = \{\} \vee \text{frontier } C \cap T = \{\}$ 
      if C:  $C \in \text{components}(- (S \cup T)) \ C \neq \{\}$  for C
    using C by blast
have  $\exists A \ B::'a \text{ set. } \text{closed } A \wedge \text{closed } B \wedge \text{UNIV} \subseteq A \cup B \wedge A \cap B = \{\} \wedge A$ 
 $\neq \{\} \wedge B \neq \{\}$ 
proof (intro exI conjI)
  have  $\text{frontier} (\bigcup \{C \in \text{components}(- S \cap - T). \text{frontier } C \subseteq S\}) \subseteq S$ 
    using subset_trans [OF frontier_Union_subset_closure]
  by (metis (no_types, lifting) SUP_least  $\langle \text{closed } S \rangle$  closure_minimal mem_Collect_eq)
  then have  $\text{frontier } ?S \subseteq S$ 
  by (simp add: frontier_subset_eq assms subset_trans [OF frontier_Un_subset])
  then show  $\text{closed } ?S$ 
    using frontier_subset_eq by fastforce
  have  $\text{frontier} (\bigcup \{C \in \text{components}(- S \cap - T). \text{frontier } C \subseteq T\}) \subseteq T$ 
    using subset_trans [OF frontier_Union_subset_closure]
  by (metis (no_types, lifting) SUP_least  $\langle \text{closed } T \rangle$  closure_minimal mem_Collect_eq)
  then have  $\text{frontier } ?T \subseteq T$ 
  by (simp add: frontier_subset_eq assms subset_trans [OF frontier_Un_subset])
  then show  $\text{closed } ?T$ 
    using frontier_subset_eq by fastforce
  have  $\text{UNIV} \subseteq (S \cup T) \cup \bigcup (\text{components}(- (S \cup T)))$ 
    using Union_components by blast
  also have  $\dots \subseteq ?S \cup ?T$ 
proof -
    have  $C \in \text{components}(- (S \cup T)) \wedge \text{frontier } C \subseteq S \vee$ 
 $C \in \text{components}(- (S \cup T)) \wedge \text{frontier } C \subseteq T$ 
    if C  $\in \text{components}(- (S \cup T)) \ C \neq \{\}$  for C
    using * [OF that] that
    by clarify (metis (no_types, lifting) UnE  $\langle \text{closed } S \rangle \langle \text{closed } T \rangle$  closed_Un
disjoint_iff_not_equal frontier_of_components_closed_complement subsetCE)
    then show ?thesis
      by blast
qed
finally show  $\text{UNIV} \subseteq ?S \cup ?T$  .
have  $\bigcup \{C \in \text{components}(- (S \cup T)). \text{frontier } C \subseteq S\} \cup$ 
 $\bigcup \{C \in \text{components}(- (S \cup T)). \text{frontier } C \subseteq T\} \subseteq - (S \cup T)$ 
    using in_components_subset by fastforce
moreover have  $\bigcup \{C \in \text{components}(- (S \cup T)). \text{frontier } C \subseteq S\} \cap$ 
 $\bigcup \{C \in \text{components}(- (S \cup T)). \text{frontier } C \subseteq T\} = \{\}$ 
proof -
  have  $C \cap C' = \{\}$  if C  $\in \text{components}(- (S \cup T)) \text{frontier } C \subseteq S$ 
 $C' \in \text{components}(- (S \cup T)) \text{frontier } C' \subseteq T$  for C C'
proof -
    have NUN:  $- S \cap - T \neq \text{UNIV}$ 
      using  $\langle T \neq \{\} \rangle$  by blast
    have  $C \neq C'$ 

```

```

proof
  assume  $C = C'$ 
  with that have frontier  $C' \subseteq S \cap T$ 
    by simp
  also have  $\dots = \{\}$ 
    using  $\langle S \cap T = \{\} \rangle$  by blast
  finally have  $C' = \{\} \vee C' = \text{UNIV}$ 
    using frontier_eq_empty by auto
  then show False
    using  $\langle C = C' \rangle$  NUN that by (force simp: dest: in_components_nonempty
in_components_subset)
  qed
  with that show ?thesis
    by (simp add: components_nonoverlap [of _  $-(S \cup T)$ ])
  qed
  then show ?thesis
    by blast
  qed
  ultimately show  $?S \cap ?T = \{\}$ 
    using ST by blast
  show  $?S \neq \{\} \wedge ?T \neq \{\}$ 
    using  $\langle S \neq \{\} \rangle \langle T \neq \{\} \rangle$  by blast+
  qed
  then show False
    by (metis Compl_disjoint connected_UNIV compl_bot_eq compl_unique
connected_closedD inf_sup_absorb sup_compl_top_left1 top.extremum_uniqueI)
  qed

```

```

proposition separation_by_component_open:
  fixes  $S :: 'a :: \text{euclidean\_space}$  set
  assumes open  $S$  and non:  $\neg \text{connected}(-S)$ 
  obtains  $C$  where  $C \in \text{components } S \wedge \neg \text{connected}(-C)$ 
proof -
  obtain  $T \ U$ 
    where closed  $T$  closed  $U$  and  $TU$ :  $T \cup U = -S \wedge T \cap U = \{\} \wedge T \neq \{\} \wedge U \neq \{\}$ 
    using assms by (auto simp: connected_closed_set closed_def)
  then obtain  $C$  where  $C \in \text{components}(- (T \cup U)) \wedge C \neq \{\}$ 
    and frontier  $C \cap T \neq \{\} \wedge \text{frontier } C \cap U \neq \{\}$ 
    using separation_by_component_open_aux [OF  $\langle \text{closed } T \rangle \langle \text{closed } U \rangle \langle T \cap U = \{\} \rangle$ ] by force
  show thesis
proof
  show  $C \in \text{components } S$ 
    using  $C(1) \ TU(1)$  by auto
  show  $\neg \text{connected}(-C)$ 
proof
  assume connected  $(-C)$ 

```

```

    then have connected (frontier C)
    using connected_frontier_simple [of C]  $\langle C \in \text{components } S \rangle$  in_components_connected
  by blast
  then show False
    unfolding connected_closed
    by (metis C(1) TU(2)  $\langle \text{closed } T \rangle \langle \text{closed } U \rangle \langle \text{frontier } C \cap T \neq \{\} \rangle$ 
 $\langle \text{frontier } C \cap U \neq \{\} \rangle$  closed_Un frontier_of_components_closed_complement
    inf_bot_right inf_commute)
  qed
  qed
  qed

```

```

lemma separation_by_Un_open:
  fixes S :: 'a :: euclidean_space set
  assumes open S open T  $S \cap T = \{\}$  and cS: connected( $-S$ ) and cT: connected( $-T$ )
  shows connected( $-(S \cup T)$ )
  using assms uncoherent_UNIV unfolding uncoherent_def by force

```

```

lemma nonseparation_by_component_eq:
  fixes S :: 'a :: euclidean_space set
  assumes open S  $\vee$  closed S
  shows  $((\forall C \in \text{components } S. \text{connected}(-C)) \longleftrightarrow \text{connected}(-S))$  (is ?lhs = ?rhs)
proof
  assume ?lhs with assms show ?rhs
    by (meson separation_by_component_closed separation_by_component_open)
next
  assume ?rhs with assms show ?lhs
    using component_complement_connected by force
qed

```

Another interesting equivalent of an inessential mapping into C-0

```

proposition inessential_eq_extensible:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  complex
  assumes closed S
  shows  $(\exists a. \text{homotopic\_with\_canon } (\lambda h. \text{True}) S (-\{0\}) f (\lambda t. a)) \longleftrightarrow$ 
 $(\exists g. \text{continuous\_on UNIV } g \wedge (\forall x \in S. g\ x = f\ x) \wedge (\forall x. g\ x \neq 0))$ 
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain a where a: homotopic_with_canon  $(\lambda h. \text{True}) S (-\{0\}) f (\lambda t. a)$ 
  ..
  show ?rhs
  proof (cases  $S = \{\}$ )
    case True
    with a show ?thesis by force
  next

```

```

case False
have anr: ANR ( $-\{0::\text{complex}\}$ )
  by (simp add: ANR_delete open_Cmpl open_imp_ANR)
obtain g where contg: continuous_on UNIV g and gim:  $g \restriction \text{UNIV} \subseteq -\{0\}$ 
  and gf:  $\bigwedge x. x \in S \implies g x = f x$ 
proof (rule Borsuk_homotopy_extension_homotopic [OF __ continuous_on_const
  _ homotopic_with_symD [OF a]])
  show closedin (top_of_set UNIV) S
  using assms by auto
  show range ( $\lambda t. a$ )  $\subseteq -\{0\}$ 
  using a homotopic_with_imp_subset2 False by blast
qed (use anr that in <force+>)
then show ?thesis
  by force
qed
next
assume ?rhs
then obtain g where contg: continuous_on UNIV g
  and gf:  $\bigwedge x. x \in S \implies g x = f x$  and non0:  $\bigwedge x. g x \neq 0$ 
  by metis
obtain h k::' $a \Rightarrow a$ ' where hk: homeomorphism (ball 0 1) UNIV h k
  using homeomorphic_ball01_UNIV homeomorphic_def by blast
then have continuous_on (ball 0 1) (g  $\circ$  h)
  by (meson contg continuous_on_compose continuous_on_subset homeomor-
  phism_cont1 top_greatest)
then obtain j where contj: continuous_on (ball 0 1) j
  and j:  $\bigwedge z. z \in \text{ball } 0 \ 1 \implies \exp(j z) = (g \circ h) z$ 
  by (metis (mono_tags, opaque_lifting) continuous_logarithm_on_ball comp_apply
  non0)
have [simp]:  $\bigwedge x. x \in S \implies h (k x) = x$ 
  using hk homeomorphism_apply2 by blast
have  $\exists \zeta. \text{continuous\_on } S \ \zeta \wedge (\forall x \in S. f x = \exp (\zeta x))$ 
proof (intro exI conjI ballI)
  show continuous_on S (j  $\circ$  k)
  proof (rule continuous_on_compose)
    show continuous_on S k
    by (meson continuous_on_subset hk homeomorphism_cont2 top_greatest)
  show continuous_on (k  $\restriction$  S) j
    by (auto intro: continuous_on_subset [OF contj] simp flip: homeomor-
    phism_image2 [OF hk])
  qed
  show f x = exp ((j  $\circ$  k) x) if x  $\in$  S for x
proof -
  have f x = (g  $\circ$  h) (k x)
    by (simp add: gf that)
  also have ... = exp (j (k x))
    by (metis rangeI homeomorphism_image2 [OF hk] j)
  finally show ?thesis by simp
qed

```

```

qed
then show ?lhs
  by (simp add: inessential_eq_continuous_logarithm)
qed

lemma inessential_on_clopen_Union:
  fixes  $\mathcal{F} :: 'a::euclidean\_space \text{ set set}$ 
  assumes  $T: \text{path\_connected } T$ 
    and  $\bigwedge S. S \in \mathcal{F} \implies \text{closedin } (\text{top\_of\_set } (\bigcup \mathcal{F})) S$ 
    and  $\bigwedge S. S \in \mathcal{F} \implies \text{openin } (\text{top\_of\_set } (\bigcup \mathcal{F})) S$ 
    and  $\text{hom}: \bigwedge S. S \in \mathcal{F} \implies \exists a. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S T f (\lambda x.$ 
a)
  obtains  $a$  where  $\text{homotopic\_with\_canon } (\lambda x. \text{True}) (\bigcup \mathcal{F}) T f (\lambda x. a)$ 
proof (cases  $\bigcup \mathcal{F} = \{\}$ )
  case True
    with that show ?thesis
      by force
  next
  case False
    then obtain  $C$  where  $C \in \mathcal{F} \ C \neq \{\}$ 
      by blast
    then obtain  $a$  where  $\text{clo}: \text{closedin } (\text{top\_of\_set } (\bigcup \mathcal{F})) C$ 
      and  $\text{ope}: \text{openin } (\text{top\_of\_set } (\bigcup \mathcal{F})) C$ 
      and  $\text{homotopic\_with\_canon } (\lambda x. \text{True}) C T f (\lambda x. a)$ 
      using  $\text{assms}$  by blast
    with  $\langle C \neq \{\} \rangle$  have  $f \cdot C \subseteq T \ a \in T$ 
      using  $\text{homotopic\_with\_imp\_subset1 homotopic\_with\_imp\_subset2}$  by blast+
    have  $\text{homotopic\_with\_canon } (\lambda x. \text{True}) (\bigcup \mathcal{F}) T f (\lambda x. a)$ 
    proof (rule  $\text{homotopic\_on\_clopen\_Union}$ )
      show  $\bigwedge S. S \in \mathcal{F} \implies \text{closedin } (\text{top\_of\_set } (\bigcup \mathcal{F})) S$ 
        and  $\bigwedge S. S \in \mathcal{F} \implies \text{openin } (\text{top\_of\_set } (\bigcup \mathcal{F})) S$ 
        by (simp_all add:  $\text{assms}$ )
      show  $\text{homotopic\_with\_canon } (\lambda x. \text{True}) S T f (\lambda x. a)$  if  $S \in \mathcal{F}$  for  $S$ 
      proof (cases  $S = \{\}$ )
        case True
          then show ?thesis
            by auto
        next
        case False
          then obtain  $b$  where  $b \in S$ 
            by blast
          obtain  $c$  where  $c: \text{homotopic\_with\_canon } (\lambda x. \text{True}) S T f (\lambda x. c)$ 
            using  $\langle S \in \mathcal{F} \rangle \text{ hom}$  by blast
          then have  $c \in T$ 
            using  $\langle b \in S \rangle \text{ homotopic\_with\_imp\_subset2}$  by blast
          then have  $\text{homotopic\_with\_canon } (\lambda x. \text{True}) S T (\lambda x. a) (\lambda x. c)$ 
            using  $T \ \langle a \in T \rangle \text{ homotopic\_constant\_maps path\_connected\_component}$ 
            by (simp add:  $\text{homotopic\_constant\_maps path\_connected\_component}$ )
          then show ?thesis

```

```

      using c homotopic_with_symD homotopic_with_trans by blast
    qed
  qed
  then show ?thesis ..
qed

proposition Janiszewski_dual:
  fixes S :: complex set
  assumes
    compact S compact T connected S connected T connected(¬ (S ∪ T))
  shows connected(S ∩ T)
proof −
  have ST: compact (S ∪ T)
    by (simp add: assms compact_Un)
  with Borsukian_imp_unicoherent [of S ∪ T] ST assms
  show ?thesis
    by (auto simp: closed_subset compact_imp_closed Borsukian_separation_compact
        unicoherent_def)
  qed

end

```

6.43 The Jordan Curve Theorem and Applications

```

theory Jordan_Curve
  imports Arcwise_Connected Further_Topology
begin

```

6.43.1 Janiszewski's theorem

```

lemma Janiszewski_weak:
  fixes a b :: complex
  assumes compact S compact T and conST: connected(S ∩ T)
    and ccS: connected_component (¬ S) a b and ccT: connected_component
(¬ T) a b
  shows connected_component (¬ (S ∪ T)) a b
proof −
  have [simp]: a ∉ S a ∉ T b ∉ S b ∉ T
    by (meson ComplD ccS ccT connected_component_in)+
  have clo: closedin (top_of_set (S ∪ T)) S closedin (top_of_set (S ∪ T)) T
    by (simp_all add: assms closed_subset compact_imp_closed)
  obtain g where contg: continuous_on S g
    and g:  $\bigwedge x. x \in S \implies \exp(i * \text{of\_real}(g\ x)) = (x - a) /_R \text{cmod}(x - a) / ((x - b) /_R \text{cmod}(x - b))$ 
  using ccS ⟨compact S⟩
  apply (simp add: Borsuk_maps_homotopic_in_connected_component_eq [symmetric])
  apply (subst (asm) homotopic_circlemaps_divide)
  apply (auto simp: inessential_eq_continuous_logarithm_circle)
  done

```



```

obtain  $h$  where  $conth$ :  $continuous\_on\ T\ h$ 
  and  $h$ :  $\bigwedge x. x \in T \implies \exp\ (i * of\_real\ (h\ x)) = (x - a) /_R\ cmod\ (x -$ 
 $a) / ((x - b) /_R\ cmod\ (x - b))$ 
  using  $ccT\ \langle compact\ T \rangle$ 
  apply ( $simp\ add$ :  $Borsuk\_maps\_homotopic\_in\_connected\_component\_eq\ [symmetric]$ )
  apply ( $subst\ (asm)\ homotopic\_circlemaps\_divide$ )
  apply ( $auto\ simp$ :  $inessential\_eq\_continuous\_logarithm\_circle$ )
  done
  have  $continuous\_on\ (S \cup T)\ (\lambda x. (x - a) /_R\ cmod\ (x - a))\ continuous\_on\ (S$ 
 $\cup T)\ (\lambda x. (x - b) /_R\ cmod\ (x - b))$ 
  by ( $intro\ continuous\_intros$ ;  $force$ ) $+$ 
  moreover have  $(\lambda x. (x - a) /_R\ cmod\ (x - a))\ '\ (S \cup T) \subseteq sphere\ 0\ 1\ (\lambda x. (x$ 
 $- b) /_R\ cmod\ (x - b))\ '\ (S \cup T) \subseteq sphere\ 0\ 1$ 
  by ( $auto\ simp$ :  $divide\_simps$ )
  moreover have  $\exists g. continuous\_on\ (S \cup T)\ g \wedge$ 
 $(\forall x \in S \cup T. (x - a) /_R\ cmod\ (x - a) / ((x - b) /_R\ cmod\ (x -$ 
 $b)) = \exp\ (i * complex\_of\_real\ (g\ x)))$ 
  proof ( $cases\ S \cap T = \{\}$ )
  case  $True$ 
  have  $continuous\_on\ (S \cup T)\ (\lambda x. if\ x \in S\ then\ g\ x\ else\ h\ x)$ 
  apply ( $rule\ continuous\_on\_cases\_local\ [OF\ clo\ contg\ conth]$ )
  using  $True$  by  $auto$ 
  then show  $?thesis$ 
  by ( $rule\_tac\ x = (\lambda x. if\ x \in S\ then\ g\ x\ else\ h\ x)\ in\ exI$ ) ( $auto\ simp$ :  $g\ h$ )
next
case  $False$ 
have  $diffpi$ :  $\exists n. g\ x = h\ x + 2 * of\_int\ n * pi$  if  $x \in S \cap T$  for  $x$ 
proof  $-$ 
  have  $\exp\ (i * of\_real\ (g\ x)) = \exp\ (i * of\_real\ (h\ x))$ 
  using  $that$  by ( $simp\ add$ :  $g\ h$ )
  then obtain  $n$  where  $complex\_of\_real\ (g\ x) = complex\_of\_real\ (h\ x) + 2 *$ 
 $of\_int\ n * complex\_of\_real\ pi$ 
  apply ( $auto\ simp$ :  $exp\_eq$ )
  by ( $metis\ complex\_i\_not\_zero\ distrib\_left\ mult.commute\ mult\_cancel\_left$ )
  then show  $?thesis$ 
  apply ( $rule\_tac\ x = n\ in\ exI$ )
  using  $of\_real\_eq\_iff$  by  $fastforce$ 
qed
  have  $contgh$ :  $continuous\_on\ (S \cap T)\ (\lambda x. g\ x - h\ x)$ 
  by ( $intro\ continuous\_intros\ continuous\_on\_subset\ [OF\ contg]\ continu-$ 
 $ous\_on\_subset\ [OF\ conth]\ auto$ )
  moreover have  $disc$ :
   $\exists e > 0. \forall y. y \in S \cap T \wedge g\ y - h\ y \neq g\ x - h\ x \longrightarrow e \leq norm\ ((g\ y - h$ 
 $y) - (g\ x - h\ x))$ 
  if  $x \in S \cap T$  for  $x$ 
proof  $-$ 
  obtain  $nx$  where  $nx$ :  $g\ x = h\ x + 2 * of\_int\ nx * pi$ 
  using  $\langle x \in S \cap T \rangle\ diffpi$  by  $blast$ 
  have  $2 * pi \leq norm\ (g\ y - h\ y - (g\ x - h\ x))$  if  $y: y \in S \cap T$  and  $neg$ :  $g\ y$ 

```

```

- h y ≠ g x - h x for y
  proof -
    obtain ny where ny: g y = h y + 2* of_int ny*pi
    using ⟨y ∈ S ∩ T⟩ diffpi by blast
    { assume nx ≠ ny
      then have 1 ≤ |real_of_int ny - real_of_int nx|
        by linarith
      then have (2*pi)*1 ≤ (2*pi)*|real_of_int ny - real_of_int nx|
        by simp
      also have ... = |2*real_of_int ny*pi - 2*real_of_int nx*pi|
        by (simp add: algebra_simps abs_if)
      finally have 2*pi ≤ |2*real_of_int ny*pi - 2*real_of_int nx*pi| by
simp
    }
    with neq show ?thesis
      by (simp add: nx ny)
  qed
  then show ?thesis
    by (rule_tac x=2*pi in exI) auto
  qed
  ultimately have (λx. g x - h x) constant_on S ∩ T
    using continuous_discrete_range_constant [OF conST contgh] by blast
  then obtain z where z: ∧x. x ∈ S ∩ T ⟹ g x - h x = z
    by (auto simp: constant_on_def)
  obtain w where exp(i * of_real(h w)) = exp (i * of_real(z + h w))
    using disc z False
    by auto (metis diff_add_cancel g h of_real_add)
  then have [simp]: exp (i * of_real z) = 1
    by (metis cis_conv_exp cis_mult exp_not_eq_zero mult_cancel_right1)
  show ?thesis
  proof (intro exI conjI)
    show continuous_on (S ∪ T) (λx. if x ∈ S then g x else z + h x)
      apply (intro continuous_intros continuous_on_cases_local [OF clo contg]
conth)
        using z by fastforce
    qed (auto simp: g h algebra_simps exp_add)
  qed
  ultimately have *: homotopic_with_canon (λx. True) (S ∪ T) (sphere 0 1)
    (λx. (x - a) /R cmod (x - a)) (λx. (x - b) /R cmod (x - b))
    by (subst homotopic_circlemaps_divide) (auto simp: inessential_eq_continuous_logarithm_circle)
  show ?thesis
    apply (rule Borsuk_maps_homotopic_in_connected_component_eq [THEN
iffD1])
      using assms by (auto simp: *)
  qed

```

theorem Janiszewski:
 fixes a b :: complex

```

    assumes compact S closed T and conST: connected (S ∩ T)
      and ccS: connected_component (− S) a b and ccT: connected_component
(− T) a b
    shows connected_component (− (S ∪ T)) a b
  proof −
    have path_component(− T) a b
    by (simp add: ⟨closed T⟩ ccT open_Compl open_path_connected_component)
    then obtain g where g: path g path_image g ⊆ − T pathstart g = a pathfinish
g = b
    by (auto simp: path_component_def)
    obtain C where C: compact C connected C a ∈ C b ∈ C C ∩ T = {}
  proof
    show compact (path_image g)
    by (simp add: ⟨path g⟩ compact_path_image)
    show connected (path_image g)
    by (simp add: ⟨path g⟩ connected_path_image)
  qed (use g in auto)
  obtain r where 0 < r and r: C ∪ S ⊆ ball 0 r
  by (metis ⟨compact C⟩ ⟨compact S⟩ bounded_Un compact_imp_bounded bounded_subset_ballD)
  have connected_component (− (S ∪ (T ∩ cball 0 r ∪ sphere 0 r))) a b
  proof (rule Janiszewski_weak [OF ⟨compact S⟩])
    show comT': compact ((T ∩ cball 0 r) ∪ sphere 0 r)
    by (simp add: ⟨closed T⟩ closed_Int_compact_compact_Un)
    have S ∩ (T ∩ cball 0 r ∪ sphere 0 r) = S ∩ T
    using r by auto
    with conST show connected (S ∩ (T ∩ cball 0 r ∪ sphere 0 r))
    by simp
    show connected_component (− (T ∩ cball 0 r ∪ sphere 0 r)) a b
    using conST C r
    apply (simp add: connected_component_def)
    apply (rule_tac x=C in exI)
    by auto
  qed (simp add: ccS)
  then obtain U where U: connected U U ⊆ − S U ⊆ − T ∪ − cball 0 r U ⊆
− sphere 0 r a ∈ U b ∈ U
  by (auto simp: connected_component_def)
  show ?thesis
    unfolding connected_component_def
  proof (intro exI conjI)
    show U ⊆ − (S ∪ T)
    using U r ⟨0 < r⟩ ⟨a ∈ C⟩ connected_Int_frontier [of U cball 0 r]
    apply simp
    by (metis ball_subset_cball compl_inf_disjoint_eq_subset_Compl_disjoint_iff_not_equal
inf.orderE inf_sup_aci(3) subsetCE)
  qed (auto simp: U)
qed

```

lemma Janiszewski_connected:
 fixes S :: complex set

```

assumes ST: compact S closed T connected(S  $\cap$  T)
and notST: connected ( $-$  S) connected ( $-$  T)
shows connected( $-$  (S  $\cup$  T))
using Janiszewski [OF ST]
by (metis IntD1 IntD2 notST compl_sup connected_iff_connected_component)

```

6.43.2 The Jordan Curve theorem

lemma *exists_double_arc*:

```

fixes g :: real  $\Rightarrow$  'a::real_normed_vector
assumes simple_path g pathfinish g = pathstart g a  $\in$  path_image g b  $\in$  path_image
g a  $\neq$  b

```

```

obtains u d where arc u arc d pathstart u = a pathfinish u = b
pathstart d = b pathfinish d = a
(path_image u)  $\cap$  (path_image d) = {a, b}
(path_image u)  $\cup$  (path_image d) = path_image g

```

proof –

```

obtain u where u: 0  $\leq$  u u  $\leq$  1 g u = a
using assms by (auto simp: path_image_def)
define h where h  $\equiv$  shiftpath u g
have simple_path h
using  $\langle$ simple_path g $\rangle$  simple_path_shiftpath  $\langle$ 0  $\leq$  u $\rangle$   $\langle$ u  $\leq$  1 $\rangle$  assms(2) h_def
by blast
have pathstart h = g u
by (simp add:  $\langle$ u  $\leq$  1 $\rangle$  h_def pathstart_shiftpath)
have pathfinish h = g u
by (simp add:  $\langle$ 0  $\leq$  u $\rangle$  assms h_def pathfinish_shiftpath)
have pihg: path_image h = path_image g
by (simp add:  $\langle$ 0  $\leq$  u $\rangle$   $\langle$ u  $\leq$  1 $\rangle$  assms h_def path_image_shiftpath)
then obtain v where v: 0  $\leq$  v v  $\leq$  1 h v = b
using assms by (metis (mono_tags, lifting) atLeastAtMost_iff imageE path_image_def)
show ?thesis

```

proof

```

show arc (subpath 0 v h)
by (metis (no_types)  $\langle$ pathstart h = g u $\rangle$   $\langle$ simple_path h $\rangle$  arc_simple_path_subpath
 $\langle$ a  $\neq$  b $\rangle$  atLeastAtMost_iff zero_le_one order_refl pathstart_def u(3) v)
show arc (subpath v 1 h)
by (metis (no_types)  $\langle$ pathfinish h = g u $\rangle$   $\langle$ simple_path h $\rangle$  arc_simple_path_subpath
 $\langle$ a  $\neq$  b $\rangle$  atLeastAtMost_iff zero_le_one order_refl pathfinish_def u(3) v)
show pathstart (subpath 0 v h) = a
by (metis  $\langle$ pathstart h = g u $\rangle$  pathstart_def pathstart_subpath u(3))
show pathfinish (subpath 0 v h) = b pathstart (subpath v 1 h) = b
by (simp_all add: v(3))
show pathfinish (subpath v 1 h) = a
by (metis  $\langle$ pathfinish h = g u $\rangle$  pathfinish_def pathfinish_subpath u(3))
show path_image (subpath 0 v h)  $\cap$  path_image (subpath v 1 h) = {a, b}

```

proof

```

show path_image (subpath 0 v h)  $\cap$  path_image (subpath v 1 h)  $\subseteq$  {a, b}
using v  $\langle$ pathfinish (subpath v 1 h) = a $\rangle$   $\langle$ simple_path h $\rangle$ 

```

```

    apply (auto simp: simple_path_def path_image_subpath image_iff
Ball_def)
    by (metis (full_types) less_eq_real_def less_irrefl less_le_trans)
  show {a, b}  $\subseteq$  path_image (subpath 0 v h)  $\cap$  path_image (subpath v 1 h)
    using v  $\langle$ pathstart (subpath 0 v h) = a $\rangle$   $\langle$ pathfinish (subpath v 1 h) = a $\rangle$ 
    apply (auto simp: path_image_subpath image_iff)
    by (metis atLeastAtMost_iff order_refl)
  qed
  show path_image (subpath 0 v h)  $\cup$  path_image (subpath v 1 h) = path_image
g
    using v apply (simp add: path_image_subpath pihg [symmetric])
    using path_image_def by fastforce
  qed
qed

```

theorem *Jordan_curve*:

```

  fixes c :: real  $\Rightarrow$  complex
  assumes simple_path c and loop: pathfinish c = pathstart c
  obtains inner outer where
    inner  $\neq$  {} open inner connected inner
    outer  $\neq$  {} open outer connected outer
    bounded inner  $\neg$  bounded outer inner  $\cap$  outer = {}
    inner  $\cup$  outer =  $-$  path_image c
    frontier inner = path_image c
    frontier outer = path_image c

```

proof $-$

```

  have path c
    by (simp add: asms simple_path_imp_path)
  have hom: (path_image c) homeomorphic (sphere(0::complex) 1)
    by (simp add: asms homeomorphic_simple_path_image_circle)
  with Jordan_Brouwer_separation have  $\neg$  connected ( $-$  (path_image c))
    by fastforce
  then obtain inner where inner: inner  $\in$  components ( $-$  path_image c) and
bounded inner
    using cobounded_has_bounded_component [of  $-$  (path_image c)]
    using  $\langle \neg$  connected ( $-$  path_image c)  $\rangle$   $\langle$ simple_path c $\rangle$  bounded_simple_path_image
  by force
  obtain outer where outer: outer  $\in$  components ( $-$  path_image c) and  $\neg$  bounded
outer
    using cobounded_unbounded_components [of  $-$  (path_image c)]
    using  $\langle$ path c $\rangle$  bounded_path_image by auto
  show ?thesis

```

proof

```

  show inner  $\neq$  {}
    using inner in_components_nonempty by auto
  show open inner
    by (meson  $\langle$ simple_path c $\rangle$  compact_imp_closed compact_simple_path_image
inner open_Compl open_components)

```

```

show connected inner
  using in_components_connected inner by blast
show outer  $\neq \{\}$ 
  using outer in_components_nonempty by auto
show open outer
  by (meson  $\langle \text{simple\_path } c \rangle$  compact_imp_closed compact_simple_path_image
outer open_Cmpl open_components)
show connected outer
  using in_components_connected outer by blast
show inner  $\cap$  outer  $= \{\}$ 
  by (meson  $\langle \neg \text{bounded outer} \rangle \langle \text{bounded inner} \rangle \langle \text{connected outer} \rangle$  bounded_subset
components_maximal in_components_subset inner outer)
show fro_inner: frontier inner  $=$  path_image c
  by (simp add: Jordan_Brouwer_frontier [OF hom inner])
show fro_outer: frontier outer  $=$  path_image c
  by (simp add: Jordan_Brouwer_frontier [OF hom outer])
have False if m: middle  $\in$  components  $(- \text{path\_image } c)$  and middle  $\neq$  inner
middle  $\neq$  outer for middle
proof  $-$ 
  have frontier middle  $=$  path_image c
    by (simp add: Jordan_Brouwer_frontier [OF hom] that)
  have middle: open middle connected middle middle  $\neq \{\}$ 
  apply (meson  $\langle \text{simple\_path } c \rangle$  compact_imp_closed compact_simple_path_image
m open_Cmpl open_components)
    using in_components_connected in_components_nonempty m by blast+
  obtain a0 b0 where a0  $\in$  path_image c b0  $\in$  path_image c a0  $\neq$  b0
    using simple_path_image_uncountable [OF  $\langle \text{simple\_path } c \rangle$ ]
  by (metis Diff_cancel countable_Diff_eq countable_empty insert_iff subsetI
subset_singleton_iff)
  obtain a b g where ab: a  $\in$  path_image c b  $\in$  path_image c a  $\neq$  b
    and arc g pathstart g  $=$  a pathfinish g  $=$  b
    and pag_sub: path_image g  $- \{a, b\} \subseteq$  middle
  proof (rule dense_accessible_frontier_point_pairs [OF  $\langle \text{open middle} \rangle \langle \text{con-$ 
nected middle} \rangle, of path_image c  $\cap$  ball a0 (dist a0 b0) path_image c  $\cap$  ball b0
(dist a0 b0)]])
    show openin (top_of_set (frontier middle)) (path_image c  $\cap$  ball a0 (dist
a0 b0))
      openin (top_of_set (frontier middle)) (path_image c  $\cap$  ball b0 (dist a0
b0))
        by (simp_all add:  $\langle \text{frontier middle} = \text{path\_image } c \rangle$  openin_open_Int)
    show path_image c  $\cap$  ball a0 (dist a0 b0)  $\neq$  path_image c  $\cap$  ball b0 (dist
a0 b0)
      using  $\langle a0 \neq b0 \rangle \langle b0 \in \text{path\_image } c \rangle$  by auto
      show path_image c  $\cap$  ball a0 (dist a0 b0)  $\neq \{\}$ 
        using  $\langle a0 \in \text{path\_image } c \rangle \langle a0 \neq b0 \rangle$  by auto
      show path_image c  $\cap$  ball b0 (dist a0 b0)  $\neq \{\}$ 
        using  $\langle b0 \in \text{path\_image } c \rangle \langle a0 \neq b0 \rangle$  by auto
    qed (use arc_distinct_ends arc_imp_simple_path simple_path_endless that
in fastforce)

```

```

obtain  $u\ d$  where  $\text{arc } u\ \text{arc } d$ 
    and  $\text{pathstart } u = a\ \text{pathfinish } u = b\ \text{pathstart } d = b\ \text{pathfinish } d$ 
=  $a$ 
    and  $\text{ud\_ab}: (\text{path\_image } u) \cap (\text{path\_image } d) = \{a,b\}$ 
    and  $\text{ud\_Un}: (\text{path\_image } u) \cup (\text{path\_image } d) = \text{path\_image } c$ 
using  $\text{exists\_double\_arc } [OF\ \text{assms } ab]$  by  $\text{blast}$ 
obtain  $x\ y$  where  $x \in \text{inner}\ y \in \text{outer}$ 
using  $\langle \text{inner} \neq \{\} \rangle\ \langle \text{outer} \neq \{\} \rangle$  by  $\text{auto}$ 
have  $\text{inner} \cap \text{middle} = \{\}\ \text{middle} \cap \text{outer} = \{\}$ 
using  $\text{components\_nonoverlap } \text{inner } \text{outer } m\ \text{that}$  by  $\text{blast+}$ 
have  $\text{connected\_component } (- (\text{path\_image } u \cup \text{path\_image } g \cup (\text{path\_image } d \cup \text{path\_image } g)))\ x\ y$ 
proof ( $\text{rule } \text{Janiszewski}$ )
  show  $\text{compact } (\text{path\_image } u \cup \text{path\_image } g)$ 
    by ( $\text{simp add: } \langle \text{arc } g \rangle\ \langle \text{arc } u \rangle\ \text{compact\_Un } \text{compact\_arc\_image}$ )
  show  $\text{closed } (\text{path\_image } d \cup \text{path\_image } g)$ 
    by ( $\text{simp add: } \langle \text{arc } d \rangle\ \langle \text{arc } g \rangle\ \text{closed\_Un } \text{closed\_arc\_image}$ )
  show  $\text{connected } ((\text{path\_image } u \cup \text{path\_image } g) \cap (\text{path\_image } d \cup \text{path\_image } g))$ 
    by ( $\text{metis } \text{Un\_Diff\_cancel } \langle \text{arc } g \rangle\ \langle \text{path\_image } u \cap \text{path\_image } d = \{a, b\} \rangle\ \langle \text{pathfinish } g = b \rangle\ \langle \text{pathstart } g = a \rangle\ \text{connected\_arc\_image } \text{insert\_Diff1 } \text{pathfinish\_in\_path\_image } \text{pathstart\_in\_path\_image } \text{sup\_bot.right\_neutral } \text{sup\_commute } \text{sup\_inf\_distrib1}$ )
  show  $\text{connected\_component } (- (\text{path\_image } u \cup \text{path\_image } g))\ x\ y$ 
    unfolding  $\text{connected\_component\_def}$ 
proof ( $\text{intro } \text{exI } \text{conjI}$ )
    have  $\text{connected } ((\text{inner} \cup (\text{path\_image } c - \text{path\_image } u)) \cup (\text{outer} \cup (\text{path\_image } c - \text{path\_image } u)))$ 
    proof ( $\text{rule } \text{connected\_Un}$ )
      show  $\text{connected } (\text{inner} \cup (\text{path\_image } c - \text{path\_image } u))$ 
        apply ( $\text{rule } \text{connected\_intermediate\_closure } [OF\ \langle \text{connected } \text{inner} \rangle]$ )
        using  $\text{fro\_inner } [\text{symmetric}]$  apply ( $\text{auto simp: } \text{closure\_subset } \text{frontier\_def}$ )
      done
      show  $\text{connected } (\text{outer} \cup (\text{path\_image } c - \text{path\_image } u))$ 
        apply ( $\text{rule } \text{connected\_intermediate\_closure } [OF\ \langle \text{connected } \text{outer} \rangle]$ )
        using  $\text{fro\_outer } [\text{symmetric}]$  apply ( $\text{auto simp: } \text{closure\_subset } \text{frontier\_def}$ )
      done
      have  $(\text{inner} \cap \text{outer}) \cup (\text{path\_image } c - \text{path\_image } u) \neq \{\}$ 
        by ( $\text{metis } \langle \text{arc } d \rangle\ \text{ud\_ab } \text{Diff\_Int } \text{Diff\_cancel } \text{Un\_Diff } \langle \text{inner} \cap \text{outer} = \{\} \rangle\ \langle \text{pathfinish } d = a \rangle\ \langle \text{pathstart } d = b \rangle\ \text{arc\_simple\_path } \text{insert\_commute } \text{nonempty\_simple\_path\_endless } \text{sup\_bot\_left } \text{ud\_Un}$ )
        then show  $(\text{inner} \cup (\text{path\_image } c - \text{path\_image } u)) \cap (\text{outer} \cup (\text{path\_image } c - \text{path\_image } u)) \neq \{\}$ 
          by  $\text{auto}$ 
    qed
then show  $\text{connected } (\text{inner} \cup \text{outer} \cup (\text{path\_image } c - \text{path\_image } u))$ 
  by ( $\text{metis } \text{sup\_right\_idem } \text{sup\_assoc } \text{sup\_commute}$ )

```

```

have inner  $\subseteq$  - path_image u outer  $\subseteq$  - path_image u
  using in_components_subset inner outer ud_Un by auto
moreover have inner  $\subseteq$  - path_image g outer  $\subseteq$  - path_image g
  using  $\langle$ inner  $\cap$  middle =  $\{\}$  $\rangle$   $\langle$ inner  $\subseteq$  - path_image u $\rangle$ 
  using  $\langle$ middle  $\cap$  outer =  $\{\}$  $\rangle$   $\langle$ outer  $\subseteq$  - path_image u $\rangle$  pag_sub ud_ab
by fastforce+
  moreover have path_image c - path_image u  $\subseteq$  - path_image g
    using in_components_subset m pag_sub ud_ab by fastforce
  ultimately show inner  $\cup$  outer  $\cup$  (path_image c - path_image u)  $\subseteq$  -
    (path_image u  $\cup$  path_image g)
    by force
  show  $x \in$  inner  $\cup$  outer  $\cup$  (path_image c - path_image u)
    by (auto simp:  $\langle x \in$  inner $\rangle$ )
  show  $y \in$  inner  $\cup$  outer  $\cup$  (path_image c - path_image u)
    by (auto simp:  $\langle y \in$  outer $\rangle$ )
qed
show connected_component (- (path_image d  $\cup$  path_image g)) x y
  unfolding connected_component_def
proof (intro exI conjI)
  have connected ((inner  $\cup$  (path_image c - path_image d))  $\cup$  (outer  $\cup$ 
    (path_image c - path_image d)))
  proof (rule connected_Un)
    show connected (inner  $\cup$  (path_image c - path_image d))
      apply (rule connected_intermediate_closure [OF  $\langle$ connected inner $\rangle$ ])
      using fro_inner [symmetric] apply (auto simp: closure_subset
frontier_def)
    done
    show connected (outer  $\cup$  (path_image c - path_image d))
      apply (rule connected_intermediate_closure [OF  $\langle$ connected outer $\rangle$ ])
      using fro_outer [symmetric] apply (auto simp: closure_subset
frontier_def)
    done
    have (inner  $\cap$  outer)  $\cup$  (path_image c - path_image d)  $\neq$   $\{\}$ 
      using  $\langle$ arc u $\rangle$   $\langle$ pathfinish u = b $\rangle$   $\langle$ pathstart u = a $\rangle$  arc_imp_simple_path
nonempty_simple_path_endless ud_Un ud_ab by fastforce
    then show (inner  $\cup$  (path_image c - path_image d))  $\cap$  (outer  $\cup$ 
    (path_image c - path_image d))  $\neq$   $\{\}$ 
      by auto
  qed
then show connected (inner  $\cup$  outer  $\cup$  (path_image c - path_image d))
  by (metis sup.right_idem sup_assoc sup_commute)
have inner  $\subseteq$  - path_image d outer  $\subseteq$  - path_image d
  using in_components_subset inner outer ud_Un by auto
moreover have inner  $\subseteq$  - path_image g outer  $\subseteq$  - path_image g
  using  $\langle$ inner  $\cap$  middle =  $\{\}$  $\rangle$   $\langle$ inner  $\subseteq$  - path_image d $\rangle$ 
  using  $\langle$ middle  $\cap$  outer =  $\{\}$  $\rangle$   $\langle$ outer  $\subseteq$  - path_image d $\rangle$  pag_sub ud_ab
by fastforce+
  moreover have path_image c - path_image d  $\subseteq$  - path_image g
    using in_components_subset m pag_sub ud_ab by fastforce

```



```

      ultimately show  $inner \cup outer \cup (path\_image\ c - path\_image\ d) \subseteq -$ 
       $(path\_image\ d \cup path\_image\ g)$ 
      by force
      show  $x \in inner \cup outer \cup (path\_image\ c - path\_image\ d)$ 
      by (auto simp:  $\langle x \in inner \rangle$ )
      show  $y \in inner \cup outer \cup (path\_image\ c - path\_image\ d)$ 
      by (auto simp:  $\langle y \in outer \rangle$ )
    qed
  qed
  then have connected_component  $(- (path\_image\ u \cup path\_image\ d \cup$ 
   $path\_image\ g))\ x\ y$ 
  by (simp add: Un_ac)
  moreover have  $\neg (connected\_component\ (- (path\_image\ c))\ x\ y)$ 
  by (metis (no_types, lifting)  $\langle \neg bounded\ outer \rangle\ \langle bounded\ inner \rangle\ \langle x \in inner \rangle$ 
   $\langle y \in outer \rangle\ componentsE\ connected\_component\_eq\ inner\ mem\_Collect\_eq\ outer$ )
  ultimately show False
  by (auto simp: ud_Un [symmetric] connected_component_def)
  qed
  then have components  $(- path\_image\ c) = \{inner, outer\}$ 
  using inner outer by blast
  then have Union  $(components\ (- path\_image\ c)) = inner \cup outer$ 
  by simp
  then show  $inner \cup outer = - path\_image\ c$ 
  by auto
  qed (auto simp:  $\langle bounded\ inner \rangle\ \langle \neg bounded\ outer \rangle$ )
  qed

```

corollary *Jordan_disconnected:*

```

  fixes  $c :: real \Rightarrow complex$ 
  assumes simple_path  $c\ pathfinish\ c = pathstart\ c$ 
  shows  $\neg connected(- path\_image\ c)$ 
  using Jordan_curve [OF assms]
  by (metis Jordan_Brouwer_separation assms homeomorphic_simple_path_image_circle
  zero_less_one)

```

corollary *Jordan_inside_outside:*

```

  fixes  $c :: real \Rightarrow complex$ 
  assumes simple_path  $c\ pathfinish\ c = pathstart\ c$ 
  shows  $inside(path\_image\ c) \neq \{\} \wedge$ 
   $open(inside(path\_image\ c)) \wedge$ 
   $connected(inside(path\_image\ c)) \wedge$ 
   $outside(path\_image\ c) \neq \{\} \wedge$ 
   $open(outside(path\_image\ c)) \wedge$ 
   $connected(outside(path\_image\ c)) \wedge$ 
   $bounded(inside(path\_image\ c)) \wedge$ 
   $\neg bounded(outside(path\_image\ c)) \wedge$ 
   $inside(path\_image\ c) \cap outside(path\_image\ c) = \{\} \wedge$ 

```

```

    inside(path_image c) ∪ outside(path_image c) =
    - path_image c ∧
    frontier(inside(path_image c)) = path_image c ∧
    frontier(outside(path_image c)) = path_image c
  proof -
    obtain inner outer
    where *: inner ≠ {} open inner connected inner
           outer ≠ {} open outer connected outer
           bounded inner ⊃ bounded outer inner ∩ outer = {}
           inner ∪ outer = - path_image c
           frontier inner = path_image c
           frontier outer = path_image c
    using Jordan_curve [OF assms] by blast
    then have inner: inside(path_image c) = inner
    by (metis dual_order.antisym inside_subset interior_eq interior_inside_frontier)
    have outer: outside(path_image c) = outer
    using ⟨inner ∪ outer = - path_image c⟩ ⟨inside (path_image c) = inner⟩
         outside_inside ⟨inner ∩ outer = {}⟩ by auto
    show ?thesis
    using * by (auto simp: inner outer)
  qed

```

Triple-curve or "theta-curve" theorem

Proof that there is no fourth component taken from Kuratowski's Topology
vol 2, para 61, II.

theorem *split_inside_simple_closed_curve*:

```

  fixes c :: real ⇒ complex
  assumes simple_path c1 and c1: pathstart c1 = a pathfinish c1 = b
    and simple_path c2 and c2: pathstart c2 = a pathfinish c2 = b
    and simple_path c and c: pathstart c = a pathfinish c = b
    and a ≠ b
    and c1c2: path_image c1 ∩ path_image c2 = {a,b}
    and c1c: path_image c1 ∩ path_image c = {a,b}
    and c2c: path_image c2 ∩ path_image c = {a,b}
    and ne_12: path_image c ∩ inside(path_image c1 ∪ path_image c2) ≠ {}
  obtains inside(path_image c1 ∪ path_image c) ∩ inside(path_image c2 ∪
    path_image c) = {}
    inside(path_image c1 ∪ path_image c) ∪ inside(path_image c2 ∪
    path_image c) ∪
    (path_image c - {a,b}) = inside(path_image c1 ∪ path_image c2)
  proof -
    let ?Θ = path_image c let ?Θ1 = path_image c1 let ?Θ2 = path_image c2
    have sp: simple_path (c1 +++ reversepath c2) simple_path (c1 +++ re-
    versepath c) simple_path (c2 +++ reversepath c)
    using assms by (auto simp: simple_path_join_loop_eq arc_simple_path sim-
    ple_path_reversepath)
    then have op_in12: open (inside (?Θ1 ∪ ?Θ2))
    and op_out12: open (outside (?Θ1 ∪ ?Θ2))

```

```

and op_in1c: open (inside (?Θ1 ∪ ?Θ))
and op_in2c: open (inside (?Θ2 ∪ ?Θ))
and op_out1c: open (outside (?Θ1 ∪ ?Θ))
and op_out2c: open (outside (?Θ2 ∪ ?Θ))
and co_in1c: connected (inside (?Θ1 ∪ ?Θ))
and co_in2c: connected (inside (?Θ2 ∪ ?Θ))
and co_out12c: connected (outside (?Θ1 ∪ ?Θ2))
and co_out1c: connected (outside (?Θ1 ∪ ?Θ))
and co_out2c: connected (outside (?Θ2 ∪ ?Θ))
and pa_c: ?Θ - {pathstart c, pathfinish c} ⊆ - ?Θ1
      ?Θ - {pathstart c, pathfinish c} ⊆ - ?Θ2
and pa_c1: ?Θ1 - {pathstart c1, pathfinish c1} ⊆ - ?Θ2
      ?Θ1 - {pathstart c1, pathfinish c1} ⊆ - ?Θ
and pa_c2: ?Θ2 - {pathstart c2, pathfinish c2} ⊆ - ?Θ1
      ?Θ2 - {pathstart c2, pathfinish c2} ⊆ - ?Θ
and co_c: connected(?Θ - {pathstart c, pathfinish c})
and co_c1: connected(?Θ1 - {pathstart c1, pathfinish c1})
and co_c2: connected(?Θ2 - {pathstart c2, pathfinish c2})
and fr_in: frontier(inside(?Θ1 ∪ ?Θ2)) = ?Θ1 ∪ ?Θ2
      frontier(inside(?Θ2 ∪ ?Θ)) = ?Θ2 ∪ ?Θ
      frontier(inside(?Θ1 ∪ ?Θ)) = ?Θ1 ∪ ?Θ
and fr_out: frontier(outside(?Θ1 ∪ ?Θ2)) = ?Θ1 ∪ ?Θ2
      frontier(outside(?Θ2 ∪ ?Θ)) = ?Θ2 ∪ ?Θ
      frontier(outside(?Θ1 ∪ ?Θ)) = ?Θ1 ∪ ?Θ
using Jordan_inside_outside [of c1 +++ reversepath c2]
using Jordan_inside_outside [of c1 +++ reversepath c]
using Jordan_inside_outside [of c2 +++ reversepath c] assms
apply (simp_all add: path_image_join closed_Un closed_simple_path_image
open_inside open_outside)
apply (blast elim: |metis connected_simple_path_endless)+
done
have inout_12: inside (?Θ1 ∪ ?Θ2) ∩ (?Θ - {pathstart c, pathfinish c}) ≠ {}
by (metis (no_types, lifting) c c1c ne_12 Diff_Int_distrib Diff_empty_Int_empty_right
Int_left_commute inf_sup_absorb inf_sup_aci(1) inside_no_overlap)
have pi_disjoint: ?Θ ∩ outside(?Θ1 ∪ ?Θ2) = {}
proof (rule ccontr)
assume ?Θ ∩ outside(?Θ1 ∪ ?Θ2) ≠ {}
then show False
using connectedD [OF co_c, of inside(?Θ1 ∪ ?Θ2) outside(?Θ1 ∪ ?Θ2)]
using c c1c2 pa_c op_in12 op_out12 inout_12
apply auto
apply (metis Un_Diff_cancel2 Un_iff compl_sup disjoint_insert(1) inf_commute
inf_compl_bot_left2 inside_Un_outside mk_disjoint_insert sup_inf_absorb)
done
qed
have out_sub12: outside(?Θ1 ∪ ?Θ2) ⊆ outside(?Θ1 ∪ ?Θ) outside(?Θ1 ∪
?Θ2) ⊆ outside(?Θ2 ∪ ?Θ)
by (metis Un_commute pi_disjoint outside_Un_outside_Un)+
have pa1_disj_in2: ?Θ1 ∩ inside (?Θ2 ∪ ?Θ) = {}

```

```

proof (rule ccontr)
  assume ne: ?Θ1 ∩ inside (?Θ2 ∪ ?Θ) ≠ {}
  have 1: inside (?Θ ∪ ?Θ2) ∩ ?Θ = {}
  by (metis (no_types) Diff_Int_distrib Diff_cancel inf_sup_absorb inf_sup_aci(3)
inside_no_overlap)
  have 2: outside (?Θ ∪ ?Θ2) ∩ ?Θ = {}
  by (metis (no_types) Int_empty_right Int_left_commute inf_sup_absorb
outside_no_overlap)
  have outside (?Θ2 ∪ ?Θ) ⊆ outside (?Θ1 ∪ ?Θ2)
  apply (subst Un_commute, rule outside_Un_outside_Un)
  using connectedD [OF co_c1, of inside(?Θ2 ∪ ?Θ) outside(?Θ2 ∪ ?Θ)]
  pa_c1 op_in2c op_out2c ne c1 c2c 1 2 by (auto simp: inf_sup_aci)
with out_sub12
have outside(?Θ1 ∪ ?Θ2) = outside(?Θ2 ∪ ?Θ) by blast
then have frontier(outside(?Θ1 ∪ ?Θ2)) = frontier(outside(?Θ2 ∪ ?Θ))
  by simp
then show False
  using inout_12 pi_disjoint c c1c c2c fr_out by auto
qed
have pa2_disj_in1: ?Θ2 ∩ inside(?Θ1 ∪ ?Θ) = {}
proof (rule ccontr)
  assume ne: ?Θ2 ∩ inside(?Θ1 ∪ ?Θ) ≠ {}
  have 1: inside (?Θ ∪ ?Θ1) ∩ ?Θ = {}
  by (metis (no_types) Diff_Int_distrib Diff_cancel inf_sup_absorb inf_sup_aci(3)
inside_no_overlap)
  have 2: outside (?Θ ∪ ?Θ1) ∩ ?Θ = {}
  by (metis (no_types) Int_empty_right Int_left_commute inf_sup_absorb
outside_no_overlap)
  have outside (?Θ1 ∪ ?Θ) ⊆ outside (?Θ1 ∪ ?Θ2)
  apply (rule outside_Un_outside_Un)
  using connectedD [OF co_c2, of inside(?Θ1 ∪ ?Θ) outside(?Θ1 ∪ ?Θ)]
  pa_c2 op_in1c op_out1c ne c2 c1c 1 2 by (auto simp: inf_sup_aci)
with out_sub12
have outside(?Θ1 ∪ ?Θ2) = outside(?Θ1 ∪ ?Θ)
  by blast
then have frontier(outside(?Θ1 ∪ ?Θ2)) = frontier(outside(?Θ1 ∪ ?Θ))
  by simp
then show False
  using inout_12 pi_disjoint c c1c c2c fr_out by auto
qed
have in_sub_in1: inside(?Θ1 ∪ ?Θ) ⊆ inside(?Θ1 ∪ ?Θ2)
  using pa2_disj_in1 out_sub12 by (auto simp: inside_outside)
have in_sub_in2: inside(?Θ2 ∪ ?Θ) ⊆ inside(?Θ1 ∪ ?Θ2)
  using pa1_disj_in2 out_sub12 by (auto simp: inside_outside)
have in_sub_out12: inside(?Θ1 ∪ ?Θ) ⊆ outside(?Θ2 ∪ ?Θ)
proof
  fix x
  assume x: x ∈ inside (?Θ1 ∪ ?Θ)
  then have xnot: x ∉ ?Θ

```

```

    by (simp add: inside_def)
  obtain z where zim:  $z \in ?\Theta 1$  and zout:  $z \in \text{outside}(?\Theta 2 \cup ?\Theta)$ 
  apply (auto simp: outside_inside)
  using nonempty_simple_path_endless [OF ‹simple_path c1›]
  by (metis Diff_Diff_Int Diff_iff ex_in_conv c1 c1c c1c2 pa1_disj_in2)
  obtain e where  $e > 0$  and e:  $\text{ball } z \ e \subseteq \text{outside}(?\Theta 2 \cup ?\Theta)$ 
  using zout op_out2c open_contains_ball_eq by blast
  have  $z \in \text{frontier}(\text{inside}(?\Theta 1 \cup ?\Theta))$ 
  using zim by (auto simp: fr_in)
  then obtain w where w1:  $w \in \text{inside}(?\Theta 1 \cup ?\Theta)$  and dwz:  $\text{dist } w \ z < e$ 
  using zim ‹ $e > 0$ › by (auto simp: frontier_def closure_approachable)
  then have w2:  $w \in \text{outside}(?\Theta 2 \cup ?\Theta)$ 
  by (metis e dist_commute mem_ball subsetCE)
  then have connected_component ( $-\ ?\Theta 2 \cap - ?\Theta$ ) z w
  apply (simp add: connected_component_def)
  apply (rule_tac  $x = \text{outside}(?\Theta 2 \cup ?\Theta)$  in exI)
  using zout apply (auto simp: co_out2c)
  apply (simp_all add: outside_inside)
  done
  moreover have connected_component ( $-\ ?\Theta 2 \cap - ?\Theta$ ) w x
  unfolding connected_component_def
  using pa2_disj_in1 co_in1c x w1 union_with_outside by fastforce
  ultimately have eq:  $\text{connected\_component\_set}(\text{outside}(?\Theta 2 \cup ?\Theta)) x =$ 
     $\text{connected\_component\_set}(\text{outside}(?\Theta 2 \cup ?\Theta)) z$ 
  by (metis (mono_tags, lifting) connected_component_eq mem_Collect_eq)
  show  $x \in \text{outside}(?\Theta 2 \cup ?\Theta)$ 
  using zout x pa2_disj_in1 by (auto simp: outside_def eq xnot)
qed
have in_sub_out21:  $\text{inside}(?\Theta 2 \cup ?\Theta) \subseteq \text{outside}(?\Theta 1 \cup ?\Theta)$ 
proof
  fix x
  assume x:  $x \in \text{inside}(?\Theta 2 \cup ?\Theta)$ 
  then have xnot:  $x \notin ?\Theta$ 
  by (simp add: inside_def)
  obtain z where zim:  $z \in ?\Theta 2$  and zout:  $z \in \text{outside}(?\Theta 1 \cup ?\Theta)$ 
  apply (auto simp: outside_inside)
  using nonempty_simple_path_endless [OF ‹simple_path c2›]
  by (metis (no_types, opaque_lifting) Diff_Diff_Int Diff_iff c1c2 c2 c2c
    ex_in_conv pa2_disj_in1)
  obtain e where  $e > 0$  and e:  $\text{ball } z \ e \subseteq \text{outside}(?\Theta 1 \cup ?\Theta)$ 
  using zout op_out1c open_contains_ball_eq by blast
  have  $z \in \text{frontier}(\text{inside}(?\Theta 2 \cup ?\Theta))$ 
  using zim by (auto simp: fr_in)
  then obtain w where w2:  $w \in \text{inside}(?\Theta 2 \cup ?\Theta)$  and dwz:  $\text{dist } w \ z < e$ 
  using zim ‹ $e > 0$ › by (auto simp: frontier_def closure_approachable)
  then have w1:  $w \in \text{outside}(?\Theta 1 \cup ?\Theta)$ 
  by (metis e dist_commute mem_ball subsetCE)
  then have connected_component ( $-\ ?\Theta 1 \cap - ?\Theta$ ) z w
  apply (simp add: connected_component_def)

```

```

apply (rule_tac  $x = \text{outside}(\Theta1 \cup \Theta)$  in  $exI$ )
using  $zout$  apply (auto simp:  $co\_out1c$ )
apply (simp_all add:  $outside\_inside$ )
done
moreover have  $\text{connected\_component}(-\Theta1 \cap -\Theta) \ w \ x$ 
unfolding  $\text{connected\_component\_def}$ 
using  $pa1\_disj\_in2 \ co\_in2c \ x \ w2 \ union\_with\_outside$  by fastforce
ultimately have  $eq: \text{connected\_component\_set}(-\Theta1 \cap -\Theta) \ x =$ 
 $\text{connected\_component\_set}(-\Theta1 \cap -\Theta) \ z$ 
by (metis (no_types, lifting)  $\text{connected\_component\_eq} \ mem\_Collect\_eq$ )
show  $x \in \text{outside}(\Theta1 \cup \Theta)$ 
using  $zout \ x \ pa1\_disj\_in2$  by (auto simp:  $outside\_def \ eq \ xnot$ )
qed
show ?thesis
proof
show  $\text{inside}(\Theta1 \cup \Theta) \cap \text{inside}(\Theta2 \cup \Theta) = \{\}$ 
by (metis  $Int\_Un\_distrib \ in\_sub\_out12 \ bot\_eq\_sup\_iff \ disjoint\_eq\_subset\_Compl$ 
 $outside\_inside$ )
have *:  $\text{outside}(\Theta1 \cup \Theta) \cap \text{outside}(\Theta2 \cup \Theta) \subseteq \text{outside}(\Theta1 \cup \Theta2)$ 
proof (rule  $components\_maximal$ )
show  $out\_in: \text{outside}(\Theta1 \cup \Theta2) \in components(-(\Theta1 \cup \Theta2))$ 
apply (simp only:  $outside\_in\_components \ co\_out12c$ )
by (metis  $bounded\_empty \ fr\_out(1) \ frontier\_empty \ unbounded\_outside$ )
have  $conn\_U: \text{connected}(-(\text{closure}(\text{inside}(\Theta1 \cup \Theta)) \cup \text{closure}(\text{inside}$ 
 $(\Theta2 \cup \Theta))))$ 
proof (rule  $Janiszewski\_connected, \ simp\_all$ )
show  $bounded(\text{inside}(\Theta1 \cup \Theta))$ 
by (simp add:  $\langle simple\_path \ c1 \rangle \langle simple\_path \ c \rangle \ bounded\_inside \ bounded\_simple\_path\_image$ )
have  $if1: -(\text{inside}(\Theta1 \cup \Theta) \cup \text{frontier}(\text{inside}(\Theta1 \cup \Theta))) = -\Theta1$ 
 $\cap -\Theta \cap -\text{inside}(\Theta1 \cup \Theta)$ 
by (metis (no_types, lifting)  $Int\_commute \ Jordan\_inside\_outside \ c$ 
 $c1 \ compl\_sup \ path\_image\_join \ path\_image\_reversepath \ pathfinish\_join \ pathfin-$ 
 $ish\_reversepath \ pathstart\_join \ pathstart\_reversepath \ sp(2) \ closure\_Un\_frontier$ 
 $fr\_out(3)$ )
then show  $\text{connected}(-\text{closure}(\text{inside}(\Theta1 \cup \Theta)))$ 
by (metis  $Compl\_Un \ outside\_inside \ co\_out1c \ closure\_Un\_frontier$ )
have  $if2: -(\text{inside}(\Theta2 \cup \Theta) \cup \text{frontier}(\text{inside}(\Theta2 \cup \Theta))) = -\Theta2$ 
 $\cap -\Theta \cap -\text{inside}(\Theta2 \cup \Theta)$ 
by (metis (no_types, lifting)  $Int\_commute \ Jordan\_inside\_outside \ c$ 
 $c2 \ compl\_sup \ path\_image\_join \ path\_image\_reversepath \ pathfinish\_join \ pathfin-$ 
 $ish\_reversepath \ pathstart\_join \ pathstart\_reversepath \ sp(3) \ closure\_Un\_frontier$ 
 $fr\_out(2)$ )
then show  $\text{connected}(-\text{closure}(\text{inside}(\Theta2 \cup \Theta)))$ 
by (metis  $Compl\_Un \ outside\_inside \ co\_out2c \ closure\_Un\_frontier$ )
have  $\text{connected}(\Theta)$ 
by (metis  $\langle simple\_path \ c \rangle \ connected\_simple\_path\_image$ )
moreover
have  $\text{closure}(\text{inside}(\Theta1 \cup \Theta)) \cap \text{closure}(\text{inside}(\Theta2 \cup \Theta)) = \Theta$ 
(is ?lhs = ?rhs)

```

```

proof
  show ?lhs  $\subseteq$  ?rhs
  proof clarify
    fix x
    assume x:  $x \in \text{closure } (\text{inside } (? \Theta 1 \cup ? \Theta))$   $x \in \text{closure } (\text{inside } (? \Theta 2 \cup$ 
?  $\Theta))$ 
    then have  $x \notin \text{inside } (? \Theta 1 \cup ? \Theta)$ 
    by (meson closure_iff_nhds_not_empty in_sub_out12 inside_Int_outside
op_in1c)
    with fr_in x show  $x \in ? \Theta$ 
    by (metis c1c c1c2 closure_Un_frontier pa1_disj_in2 Int_iff Un_iff
insert_disjoint(2) insert_subset subsetI subset_antisym)
    qed
    show ?rhs  $\subseteq$  ?lhs
    using if1 if2 closure_Un_frontier by fastforce
    qed
  ultimately
    show connected (closure (inside (?  $\Theta$  1  $\cup$  ?  $\Theta$ ))  $\cap$  closure (inside (?  $\Theta$  2  $\cup$ 
?  $\Theta$ )))
    by auto
  qed
  show connected (outside (?  $\Theta$  1  $\cup$  ?  $\Theta$ )  $\cap$  outside (?  $\Theta$  2  $\cup$  ?  $\Theta$ ))
  using fr_in conn_U by (simp add: closure_Un_frontier outside_inside
Un_commute)
  show outside (?  $\Theta$  1  $\cup$  ?  $\Theta$ )  $\cap$  outside (?  $\Theta$  2  $\cup$  ?  $\Theta$ )  $\subseteq$   $-$  (?  $\Theta$  1  $\cup$  ?  $\Theta$  2)
  by clarify (metis Diff_Compl Diff_iff Un_iff inf_sup_absorb outside_inside)
  show outside (?  $\Theta$  1  $\cup$  ?  $\Theta$  2)  $\cap$ 
    (outside (?  $\Theta$  1  $\cup$  ?  $\Theta$ )  $\cap$  outside (?  $\Theta$  2  $\cup$  ?  $\Theta$ ))  $\neq \{\}$ 
  by (metis Int_assoc out_in inf.orderE out_sub12(1) out_sub12(2) out-
side_in_components)
  qed
  show inside (?  $\Theta$  1  $\cup$  ?  $\Theta$ )  $\cup$  inside (?  $\Theta$  2  $\cup$  ?  $\Theta$ )  $\cup$  (?  $\Theta$   $-$  {a, b}) = inside (?  $\Theta$  1
 $\cup$  ?  $\Theta$  2)
  (is ?lhs = ?rhs)
proof
  show ?lhs  $\subseteq$  ?rhs
  apply (simp add: in_sub_in1 in_sub_in2)
  using c1c c2c inside_outside pi_disjoint by fastforce
  have inside (?  $\Theta$  1  $\cup$  ?  $\Theta$  2)  $\subseteq$  inside (?  $\Theta$  1  $\cup$  ?  $\Theta$ )  $\cup$  inside (?  $\Theta$  2  $\cup$  ?  $\Theta$ )  $\cup$  (?  $\Theta$ )
  using Compl_anti_mono [OF *] by (force simp: inside_outside)
  moreover have inside (?  $\Theta$  1  $\cup$  ?  $\Theta$  2)  $\subseteq$   $-$ {a, b}
  using c1 union_with_outside by fastforce
  ultimately show ?rhs  $\subseteq$  ?lhs by auto
qed
qed
qed
end

```

6.44 Polynomial Functions: Extremal Behaviour and Root Counts

```
theory Poly_Roots
imports Complex_Main
begin
```

6.44.1 Basics about polynomial functions: extremal behaviour and root counts

```
lemma sub_polyfun:
  fixes x :: 'a::{comm_ring,monoid_mult}
  shows  $(\sum_{i \leq n}. a\ i * x^i) - (\sum_{i \leq n}. a\ i * y^i) =$ 
 $(x - y) * (\sum_{j < n}. \sum_{k = \text{Suc } j..n}. a\ k * y^{k - \text{Suc } j} * x^j)$ 
proof -
  have  $(\sum_{i \leq n}. a\ i * x^i) - (\sum_{i \leq n}. a\ i * y^i) =$ 
 $(\sum_{i \leq n}. a\ i * (x^i - y^i))$ 
  by (simp add: algebra_simps sum_subtractf [symmetric])
  also have ... =  $(\sum_{i \leq n}. a\ i * (x - y) * (\sum_{j < i}. y^{i - \text{Suc } j} * x^j))$ 
  by (simp add: power_diff_sumr2 ac_simps)
  also have ... =  $(x - y) * (\sum_{i \leq n}. (\sum_{j < i}. a\ i * y^{i - \text{Suc } j} * x^j))$ 
  by (simp add: sum_distrib_left ac_simps)
  also have ... =  $(x - y) * (\sum_{j < n}. (\sum_{i = \text{Suc } j..n}. a\ i * y^{i - \text{Suc } j} * x^j))$ 
  by (simp add: sum.nested_swap')
  finally show ?thesis .
qed
```

```
lemma sub_polyfun_alt:
  fixes x :: 'a::{comm_ring,monoid_mult}
  shows  $(\sum_{i \leq n}. a\ i * x^i) - (\sum_{i \leq n}. a\ i * y^i) =$ 
 $(x - y) * (\sum_{j < n}. \sum_{k < n - j}. a\ (j + k + 1) * y^k * x^j)$ 
proof -
  { fix j
    have  $(\sum_{k = \text{Suc } j..n}. a\ k * y^{k - \text{Suc } j} * x^j) =$ 
 $(\sum_{k < n - j}. a\ (\text{Suc } (j + k)) * y^k * x^j)$ 
    by (rule sum.reindex_bij_witness[where i= $\lambda i. i + \text{Suc } j$  and j= $\lambda i. i - \text{Suc } j$ ]) auto }
  then show ?thesis
  by (simp add: sub_polyfun)
qed
```

```
lemma polyfun_linear_factor:
  fixes a :: 'a::{comm_ring,monoid_mult}
  shows  $\exists b. \forall z. (\sum_{i \leq n}. c\ i * z^i) =$ 
 $(z - a) * (\sum_{i < n}. b\ i * z^i) + (\sum_{i \leq n}. c\ i * a^i)$ 
proof -
  { fix z
    have  $(\sum_{i \leq n}. c\ i * z^i) - (\sum_{i \leq n}. c\ i * a^i) =$ 
 $(z - a) * (\sum_{j < n}. (\sum_{k = \text{Suc } j..n}. c\ k * a^{k - \text{Suc } j})) * z^j$ 
```



```

    by (simp add: sub_polyfun sum_distrib_right)
  then have  $(\sum_{i \leq n}. c\ i * z^{\wedge} i) =$ 
     $(z - a) * (\sum_{j < n}. (\sum_{k = \text{Suc } j..n}. c\ k * a^{\wedge}(k - \text{Suc } j)) * z^{\wedge} j)$ 
     $+ (\sum_{i \leq n}. c\ i * a^{\wedge} i)$ 
    by (simp add: algebra_simps) }
  then show ?thesis
    by (intro exI allI)
qed

```

```

lemma polyfun_linear_factor_root:
  fixes a :: 'a::{comm_ring,monoid_mult}
  assumes  $(\sum_{i \leq n}. c\ i * a^{\wedge} i) = 0$ 
  shows  $\exists b. \forall z. (\sum_{i \leq n}. c\ i * z^{\wedge} i) = (z - a) * (\sum_{i < n}. b\ i * z^{\wedge} i)$ 
  using polyfun_linear_factor [of c n a] assms
  by simp

```

```

lemma adhoc_norm_triangle:  $a + \text{norm}(y) \leq b \implies \text{norm}(x) \leq a \implies \text{norm}(x + y) \leq b$ 
  by (metis norm_triangle_mono order.trans order_refl)

```

```

proposition polyfun_extremal_lemma:
  fixes c :: nat  $\Rightarrow$  'a::real_normed_div_algebra
  assumes  $e > 0$ 
  shows  $\exists M. \forall z. M \leq \text{norm } z \longrightarrow \text{norm}(\sum_{i \leq n}. c\ i * z^{\wedge} i) \leq e * \text{norm}(z)^{\wedge} \text{Suc } n$ 
  proof (induction n)
    case 0
    show ?case
      by (rule exI [where x = norm (c 0) / e]) (auto simp: mult.commute pos_divide_le_eq assms)
    next
      case (Suc n)
      then obtain M where  $M: \forall z. M \leq \text{norm } z \longrightarrow \text{norm}(\sum_{i \leq n}. c\ i * z^{\wedge} i) \leq e * \text{norm } z^{\wedge} \text{Suc } n ..$ 
      show ?case
        proof (rule exI [where x = max 1 (max M ((e + norm(c(Suc n))) / e)], clarify)
          fix z::'a
          assume  $\max 1 (\max M ((e + \text{norm}(c(\text{Suc } n))) / e)) \leq \text{norm } z$ 
          then have norm1:  $0 < \text{norm } z \wedge M \leq \text{norm } z \wedge (e + \text{norm}(c(\text{Suc } n))) / e \leq \text{norm } z$ 
            by auto
          then have norm2:  $(e + \text{norm}(c(\text{Suc } n))) \leq e * \text{norm } z \wedge (\text{norm } z * \text{norm } z^{\wedge} n) > 0$ 
            by (metis assms less_divide_eq mult.commute not_le)
          using norm1 apply (metis mult_pos_pos zero_less_power)
          done
          have  $e * (\text{norm } z * \text{norm } z^{\wedge} n) + \text{norm}(c(\text{Suc } n) * (z * z^{\wedge} n)) =$ 
             $(e + \text{norm}(c(\text{Suc } n))) * (\text{norm } z * \text{norm } z^{\wedge} n)$ 
            by (simp add: norm_mult norm_power algebra_simps)

```

```

    also have ... ≤ (e * norm z) * (norm z * norm z ^ n)
      using norm2
      using assms mult_mono by fastforce
    also have ... = e * (norm z * (norm z * norm z ^ n))
      by (simp add: algebra_simps)
    finally have e * (norm z * norm z ^ n) + norm (c (Suc n) * (z * z ^ n))
      ≤ e * (norm z * (norm z * norm z ^ n)) .
    then show norm (∑ i ≤ Suc n. c i * z ^ i) ≤ e * norm z ^ Suc (Suc n) using
M norm1
      by (drule_tac x=z in spec) (auto simp: intro!: adhoc_norm_triangle)
    qed
  qed

```

```

lemma norm_lemma_xy: assumes |b| + 1 ≤ norm(y) - a norm(x) ≤ a shows
b ≤ norm(x + y)
proof -
  have b ≤ norm y - norm x
    using assms by linarith
  then show ?thesis
    by (metis (no_types) add.commute norm_diff_ineq order_trans)
qed

```

```

proposition polyfun_extremal:
  fixes c :: nat ⇒ 'a::real_normed_div_algebra
  assumes ∃ k. k ≠ 0 ∧ k ≤ n ∧ c k ≠ 0
  shows eventually (λz. norm(∑ i ≤ n. c i * z ^ i) ≥ B) at_infinity
using assms
proof (induction n)
  case 0 then show ?case
    by simp
next
  case (Suc n)
  show ?case
  proof (cases c (Suc n) = 0)
    case True
    with Suc show ?thesis
      by auto (metis diff_is_0_eq diff0_imp_equal less_Suc_eq_le not_less_eq)
  next
    case False
    with polyfun_extremal_lemma [of norm(c (Suc n)) / 2 c n]
    obtain M where M: ⋀ z. M ≤ norm z ⇒
      norm (∑ i ≤ n. c i * z ^ i) ≤ norm (c (Suc n)) / 2 * norm z ^ Suc n
    by auto
    show ?thesis
    unfolding eventually_at_infinity
    proof (rule exI [where x=max M (max 1 ((|B| + 1) / (norm (c (Suc n)) / 2)))] , clarsimp)
      fix z::'a
      assume les: M ≤ norm z 1 ≤ norm z (|B| * 2 + 2) / norm (c (Suc n)) ≤

```

```

norm z
  then have  $|B| * 2 + 2 \leq \text{norm } z * \text{norm } (c \text{ (Suc } n))$ 
    by (metis False pos_divide_le_eq zero_less_norm_iff)
  then have  $|B| * 2 + 2 \leq \text{norm } z ^{\wedge} (\text{Suc } n) * \text{norm } (c \text{ (Suc } n))$ 
    by (metis  $\langle 1 \leq \text{norm } z \rangle$  order.trans mult_right_mono norm_ge_zero
self_le_power zero_less_Suc)
  then show  $B \leq \text{norm } ((\sum_{i \leq n}. c \ i * z^{\wedge} i) + c \text{ (Suc } n) * (z * z^{\wedge} n))$  using
M les
    apply auto
    apply (rule norm_lemma_xy [where  $a = \text{norm } (c \text{ (Suc } n)) * \text{norm } z ^{\wedge}$ 
(Suc n) / 2])
    apply (simp_all add: norm_mult norm_power)
    done
  qed
qed
qed

```

proposition polyfun_rootbound:

```

fixes  $c :: \text{nat} \Rightarrow 'a :: \{\text{comm\_ring, real\_normed\_div\_algebra}\}$ 
assumes  $\exists k. k \leq n \wedge c \ k \neq 0$ 
shows  $\text{finite } \{z. (\sum_{i \leq n}. c \ i * z^{\wedge} i) = 0\} \wedge \text{card } \{z. (\sum_{i \leq n}. c \ i * z^{\wedge} i) = 0\}$ 
 $\leq n$ 
using assms
proof (induction n arbitrary: c)
case (Suc n) show ?case
proof (cases  $\{z. (\sum_{i \leq \text{Suc } n}. c \ i * z^{\wedge} i) = 0\} = \{\}$ )
case False
  then obtain a where  $a: (\sum_{i \leq \text{Suc } n}. c \ i * a^{\wedge} i) = 0$ 
    by auto
  from polyfun_linear_factor_root [OF this]
  obtain b where  $\bigwedge z. (\sum_{i \leq \text{Suc } n}. c \ i * z^{\wedge} i) = (z - a) * (\sum_{i < \text{Suc } n}. b \ i * z^{\wedge} i)$ 
    by auto
  then have  $b: \bigwedge z. (\sum_{i \leq \text{Suc } n}. c \ i * z^{\wedge} i) = (z - a) * (\sum_{i \leq n}. b \ i * z^{\wedge} i)$ 
    by (metis lessThan_Suc_atMost)
  then have  $\text{ins\_ab}: \{z. (\sum_{i \leq \text{Suc } n}. c \ i * z^{\wedge} i) = 0\} = \text{insert } a \ \{z. (\sum_{i \leq n}. b \ i * z^{\wedge} i) = 0\}$ 
    by auto
  have  $c \ 0 = - (a * b \ 0)$  using b [of 0]
    by simp
  then have  $\text{extr\_prem}: \neg (\exists k \leq n. b \ k \neq 0) \implies \exists k. k \neq 0 \wedge k \leq \text{Suc } n \wedge c \ k \neq 0$ 
    by (metis Suc.prem le0 minus_zero mult_zero_right)
  have  $\exists k \leq n. b \ k \neq 0$ 
    apply (rule ccontr)
    using polyfun_extremal [OF extr_prem, of 1]
    apply (auto simp: eventually_at_infinity b simp del: sum.atMost_Suc)
    apply (drule_tac  $x = \text{of\_real } ba$  in spec, simp)
    done

```

```

    then show ?thesis using Suc.IH [of b] ins_ab
    by (auto simp: card_insert_if)
  qed simp
qed simp

```

corollary

```

  fixes c :: nat  $\Rightarrow$  'a::{comm_ring,real_normed_div_algebra}
  assumes  $\exists k. k \leq n \wedge c\ k \neq 0$ 
  shows polyfun_rootbound_finite: finite {z.  $(\sum_{i \leq n}. c\ i * z^i) = 0$ }
  and polyfun_rootbound_card: card {z.  $(\sum_{i \leq n}. c\ i * z^i) = 0$ }  $\leq n$ 
using polyfun_rootbound [OF assms] by auto

```

proposition polyfun_finite_roots:

```

  fixes c :: nat  $\Rightarrow$  'a::{comm_ring,real_normed_div_algebra}
  shows finite {z.  $(\sum_{i \leq n}. c\ i * z^i) = 0$ }  $\longleftrightarrow (\exists k. k \leq n \wedge c\ k \neq 0)$ 
proof (cases  $\exists k \leq n. c\ k \neq 0$ )
  case True then show ?thesis
    by (blast intro: polyfun_rootbound_finite)
  next
  case False then show ?thesis
    by (auto simp: infinite_UNIV_char_0)
qed

```

lemma polyfun_eq_0:

```

  fixes c :: nat  $\Rightarrow$  'a::{comm_ring,real_normed_div_algebra}
  shows  $(\forall z. (\sum_{i \leq n}. c\ i * z^i) = 0) \longleftrightarrow (\forall k. k \leq n \longrightarrow c\ k = 0)$ 
proof (cases  $(\forall z. (\sum_{i \leq n}. c\ i * z^i) = 0)$ )
  case True
  then have  $\neg$  finite {z.  $(\sum_{i \leq n}. c\ i * z^i) = 0$ }
    by (simp add: infinite_UNIV_char_0)
  with True show ?thesis
    by (metis (poly_guards_query) polyfun_rootbound_finite)
  next
  case False
  then show ?thesis
    by auto
qed

```

theorem polyfun_eq_const:

```

  fixes c :: nat  $\Rightarrow$  'a::{comm_ring,real_normed_div_algebra}
  shows  $(\forall z. (\sum_{i \leq n}. c\ i * z^i) = k) \longleftrightarrow c\ 0 = k \wedge (\forall k. k \neq 0 \wedge k \leq n \longrightarrow c\ k = 0)$ 
proof -
  {fix z
    have  $(\sum_{i \leq n}. c\ i * z^i) = (\sum_{i \leq n}. (\text{if } i = 0 \text{ then } c\ 0 - k \text{ else } c\ i) * z^i) + k$ 
      by (induct n) auto
    } then
  have  $(\forall z. (\sum_{i \leq n}. c\ i * z^i) = k) \longleftrightarrow (\forall z. (\sum_{i \leq n}. (\text{if } i = 0 \text{ then } c\ 0 - k \text{ else } c\ i) * z^i) = 0)$ 

```

```

    by auto
  also have ...  $\longleftrightarrow$   $c\ 0 = k \wedge (\forall k. k \neq 0 \wedge k \leq n \longrightarrow c\ k = 0)$ 
    by (auto simp: polyfun_eq_0)
  finally show ?thesis .
qed

end

```

6.45 Generalised Binomial Theorem

The proof of the Generalised Binomial Theorem and related results. We prove the generalised binomial theorem for complex numbers, following the proof at: https://proofwiki.org/wiki/Binomial_Theorem/General_Binomial_Theorem

```

theory Generalised_Binomial_Theorem
imports
  Complex_Main
  Complex_Transcendental
  Summation_Tests
begin

```

```

lemma gbinomial_ratio_limit:
  fixes a :: 'a :: real_normed_field
  assumes a  $\notin$   $\mathbb{N}$ 
  shows  $(\lambda n. (a\ gchoose\ n) / (a\ gchoose\ Suc\ n)) \longrightarrow -1$ 
proof (rule Lim_transform_eventually)
  let ?f =  $\lambda n. inverse\ (a / of\_nat\ (Suc\ n) - of\_nat\ n / of\_nat\ (Suc\ n))$ 
  from eventually_gt_at_top[of 0::nat]
  show eventually  $(\lambda n. ?f\ n = (a\ gchoose\ n) / (a\ gchoose\ Suc\ n))$  sequentially
proof eventually_elim
  fix n :: nat assume n:  $n > 0$ 
  then obtain q where  $q: n = Suc\ q$  by (cases n) blast
  let ?P =  $\prod_{i=0..<n.} a - of\_nat\ i$ 
  from n have  $(a\ gchoose\ n) / (a\ gchoose\ Suc\ n) = (of\_nat\ (Suc\ n) :: 'a) * ( ?P / (\prod_{i=0..n.} a - of\_nat\ i))$ 
    by (simp add: gbinomial_prod_rev atLeastLessThanSuc_atLeastAtMost)
  also from q have  $(\prod_{i=0..n.} a - of\_nat\ i) = ?P * (a - of\_nat\ n)$ 
    by (simp add: prod.atLeast0_atMost_Suc atLeastLessThanSuc_atLeastAtMost)
  also have  $?P / \dots = (?P / ?P) / (a - of\_nat\ n)$  by (rule divide_divide_eq_left[symmetric])
  also from assms have  $?P / ?P = 1$  by auto
  also have  $of\_nat\ (Suc\ n) * (1 / (a - of\_nat\ n)) =$ 
     $inverse\ (inverse\ (of\_nat\ (Suc\ n)) * (a - of\_nat\ n))$  by (simp add:
field_simps)
  also have  $inverse\ (of\_nat\ (Suc\ n)) * (a - of\_nat\ n) = a / of\_nat\ (Suc\ n) -$ 
 $of\_nat\ n / of\_nat\ (Suc\ n)$ 
    by (simp add: field_simps del: of_nat_Suc)
  finally show ?f n =  $(a\ gchoose\ n) / (a\ gchoose\ Suc\ n)$  by simp
qed

```

```

have (λn. norm a / (of_nat (Suc n))) → 0
  unfolding divide_inverse
  by (intro tendsto_mult_right_zero LIMSEQ_inverse_real_of_nat)
hence (λn. a / of_nat (Suc n)) → 0
  by (subst tendsto_norm_zero_iff[symmetric]) (simp add: norm_divide del:
of_nat_Suc)
hence ?f → inverse (0 - 1)
  by (intro tendsto_inverse tendsto_diff LIMSEQ_n_over_Suc_n) simp_all
thus ?f → -1 by simp
qed

```

```

lemma conv_radius_gchoose:
  fixes a :: 'a :: {real_normed_field,banach}
  shows conv_radius (λn. a gchoose n) = (if a ∈ ℕ then ∞ else 1)
proof (cases a ∈ ℕ)
  assume a: a ∈ ℕ
  have eventually (λn. (a gchoose n) = 0) sequentially
    using eventually_gt_at_top[of nat ‖norm a‖]
  by eventually_elim (insert a, auto elim!: Nats_cases simp: binomial_gbinomial[symmetric])
  from conv_radius_cong'[OF this] a show ?thesis by simp
next
  assume a: a ∉ ℕ
  from tendsto_norm[OF gbinomial_ratio_limit[OF this]]
  have conv_radius (λn. a gchoose n) = 1
    by (intro conv_radius_ratio_limit_nonzero[of _ 1]) (simp_all add: norm_divide)
  with a show ?thesis by simp
qed

```

```

theorem gen_binomial_complex:
  fixes z :: complex
  assumes norm z < 1
  shows (λn. (a gchoose n) * z^n) sums (1 + z) powr a
proof -
  define K where K = 1 - (1 - norm z) / 2
  from assms have K: K > 0 K < 1 norm z < K
    unfolding K_def by (auto simp: field_simps intro!: add_pos_nonneg)
  let ?f = λn. a gchoose n and ?f' = diffs (λn. a gchoose n)
  have summable_strong: summable (λn. ?f n * z^n) if norm z < 1 for z using
that
  by (intro summable_in_conv_radius) (simp_all add: conv_radius_gchoose)
  with K have summable: summable (λn. ?f n * z^n) if norm z < K for z using
that by auto
  hence summable': summable (λn. ?f' n * z^n) if norm z < K for z using that
  by (intro termdiff_converges[of _ K]) simp_all

  define f f' where [abs_def]: f z = (∑ n. ?f n * z^n) f' z = (∑ n. ?f' n * z^n)
  for z
  {

```

```

fix z :: complex assume z: norm z < K
from summable_mult2[OF summable'[OF z], of z]
  have summable1: summable (λn. ?f' n * z ^ Suc n) by (simp add: mult_ac)
hence summable2: summable (λn. of_nat n * ?f n * z ^ n)
  unfolding diffs_def by (subst (asm) summable_Suc_iff)

have (1 + z) * f' z = (∑ n. ?f' n * z ^ n) + (∑ n. ?f' n * z ^ Suc n)
  unfolding f_f'_def using summable' z by (simp add: algebra_simps sum-
inf_mult)
also have (∑ n. ?f' n * z ^ n) = (∑ n. of_nat (Suc n) * ?f (Suc n) * z ^ n)
  by (intro suminf_cong) (simp add: diffs_def)
also have (∑ n. ?f' n * z ^ Suc n) = (∑ n. of_nat n * ?f n * z ^ n)
  using summable1 suminf_split_initial_segment[OF summable1] unfolding
diffs_def
  by (subst suminf_split_head, subst (asm) summable_Suc_iff) simp_all
also have (∑ n. of_nat (Suc n) * ?f (Suc n) * z ^ n) + (∑ n. of_nat n * ?f n
* z ^ n) =
  (∑ n. a * ?f n * z ^ n)
  by (subst gbinomial_mult_1, subst suminf_add)
  (insert summable'[OF z] summable2,
  simp_all add: summable_powser_split_head algebra_simps diffs_def)
also have ... = a * f z unfolding f_f'_def
  by (subst suminf_mult[symmetric]) (simp_all add: summable[OF z] mult_ac)
finally have a * f z = (1 + z) * f' z by simp
} note deriv = this

have [derivative_intros]: (f has_field_derivative f' z) (at z) if norm z < of_real
K for z
  unfolding f_f'_def using K that
  by (intro termdiffs_strong[of ?f K z] summable_strong) simp_all
have f 0 = (∑ n. if n = 0 then 1 else 0) unfolding f_f'_def by (intro sum-
inf_cong) simp
also have ... = 1 using sums_single[of 0 λ_. 1::complex] unfolding sums_iff
by simp
finally have [simp]: f 0 = 1 .

have ∃ c. ∀ z ∈ ball 0 K. f z * (1 + z) powr (-a) = c
proof (rule has_field_derivative_zero_constant)
  fix z :: complex assume z': z ∈ ball 0 K
  hence z: norm z < K by simp
  with K have nz: 1 + z ≠ 0 by (auto dest!: minus_unique)
  from z K have norm z < 1 by simp
  hence (1 + z) ∉ ℝ≤0 by (cases z) (auto simp: Complex_eq complex_nonpos_Reals_iff)
  hence ((λz. f z * (1 + z) powr (-a)) has_field_derivative
    f' z * (1 + z) powr (-a) - a * f z * (1 + z) powr (-a-1)) (at z)
using z
  by (auto intro!: derivative_eq_intros)
also from z have a * f z = (1 + z) * f' z by (rule deriv)
  finally show ((λz. f z * (1 + z) powr (-a)) has_field_derivative 0) (at z)

```

```

within ball 0 K)
  using nz by (simp add: field_simps powr_diff at_within_open[OF z'])
qed simp_all
then obtain c where c:  $\bigwedge z. z \in \text{ball } 0 K \implies f z * (1 + z) \text{ powr } (-a) = c$  by
blast
from c[of 0] and K have c = 1 by simp
with c[of z] have  $f z = (1 + z) \text{ powr } a$  using K
  by (simp add: powr_minus field_simps dist_complex_def)
with summable K show ?thesis unfolding f'_def by (simp add: sums_iff)
qed

```

```

lemma gen_binomial_complex':
  fixes x y :: real and a :: complex
  assumes  $|x| < |y|$ 
  shows  $(\lambda n. (a \text{ gchoose } n) * \text{of\_real } x^n * \text{of\_real } y \text{ powr } (a - \text{of\_nat } n)) \text{ sums } \text{of\_real } (x + y) \text{ powr } a$  (is ?P x y)
proof -
  {
    fix x y :: real assume xy:  $|x| < |y|$   $y \geq 0$ 
    hence  $y > 0$  by simp
    note xy = xy this
    from xy have  $(\lambda n. (a \text{ gchoose } n) * \text{of\_real } (x / y)^n) \text{ sums } (1 + \text{of\_real } (x / y)) \text{ powr } a$ 
      by (intro gen_binomial_complex) (simp add: norm_divide)
    hence  $(\lambda n. (a \text{ gchoose } n) * \text{of\_real } (x / y)^n * y \text{ powr } a) \text{ sums } ((1 + \text{of\_real } (x / y)) \text{ powr } a * y \text{ powr } a)$ 
      by (rule sums_mult2)
    also have  $(1 + \text{complex\_of\_real } (x / y)) = \text{complex\_of\_real } (1 + x/y)$  by
simp
    also from xy have  $\dots \text{ powr } a * \text{of\_real } y \text{ powr } a = (\dots * y) \text{ powr } a$ 
      by (subst powr_times_real[symmetric]) (simp_all add: field_simps)
    also from xy have  $\text{complex\_of\_real } (1 + x / y) * \text{complex\_of\_real } y = \text{of\_real } (x + y)$ 
      by (simp add: field_simps)
    finally have ?P x y using xy by (simp add: field_simps powr_diff powr_nat)
  } note A = this

```

show ?thesis

proof (cases $y < 0$)

assume $y: y < 0$

with assms have $xy: x + y < 0$ by simp

with assms have $|x| < |-y|$ $-y \geq 0$ by simp_all

note A[OF this]

also have $\text{complex\_of\_real } (-x + -y) = - \text{complex\_of\_real } (x + y)$ by simp

also from xy assms have $\dots \text{ powr } a = (-1) \text{ powr } -a * \text{of\_real } (x + y) \text{ powr } a$

by (subst powr_neg_real_complex) (simp add: abs_real_def split: if_split_asm)

also {

fix $n :: \text{nat}$

from y have $(a \text{ gchoose } n) * \text{of\_real } (-x)^n * \text{of\_real } (-y) \text{ powr } (a -$


```

of_nat n) =
  (a gchoose n) * (-of_real x / -of_real y) ^ n * (- of_real y)
powr a
  by (subst power_divide) (simp add: powr_diff powr_nat)
  also from y have (- of_real y) powr a = (-1) powr -a * of_real y powr a
  by (subst powr_neg_real_complex) simp
  also have -complex_of_real x / -complex_of_real y = complex_of_real x
/ complex_of_real y
  by simp
  also have ... ^ n = of_real x ^ n / of_real y ^ n by (simp add: power_divide)
  also have (a gchoose n) * ... * ((-1) powr -a * of_real y powr a) =
    (-1) powr -a * ((a gchoose n) * of_real x ^ n * of_real y powr
(a - n))
  by (simp add: algebra_simps powr_diff powr_nat)
  finally have (a gchoose n) * of_real (- x) ^ n * of_real (- y) powr (a -
of_nat n) =
    (-1) powr -a * ((a gchoose n) * of_real x ^ n * of_real y powr
(a - of_nat n)) .
}
note sums_cong[OF this]
finally show ?thesis by (simp add: sums_mult_iff)
qed (insert A[of x y] assms, simp_all add: not_less)
qed

```

```

lemma gen_binomial_complex'':
  fixes x y :: real and a :: complex
  assumes |y| < |x|
  shows (λn. (a gchoose n) * of_real x powr (a - of_nat n) * of_real y ^ n)
sums
  of_real (x + y) powr a
  using gen_binomial_complex'[OF assms] by (simp add: mult_ac add.commute)

```

```

lemma gen_binomial_real:
  fixes z :: real
  assumes |z| < 1
  shows (λn. (a gchoose n) * z ^ n) sums (1 + z) powr a
proof -
  from assms have norm (of_real z :: complex) < 1 by simp
  from gen_binomial_complex'[OF this]
  have (λn. (of_real a gchoose n :: complex) * of_real z ^ n) sums
    (of_real (1 + z)) powr (of_real a) by simp
  also have (of_real (1 + z) :: complex) powr (of_real a) = of_real ((1 + z)
powr a)
  using assms by (subst powr_of_real) simp_all
  also have (of_real a gchoose n :: complex) = of_real (a gchoose n) for n
  by (simp add: gbinomial_prod_rev)
  hence (λn. (of_real a gchoose n :: complex) * of_real z ^ n) =
    (λn. of_real ((a gchoose n) * z ^ n)) by (intro ext) simp
  finally show ?thesis by (simp only: sums_of_real_iff)

```

3198

qed

```

lemma gen_binomial_real':
  fixes x y a :: real
  assumes  $|x| < y$ 
  shows  $(\lambda n. (a \text{ gchoose } n) * x^n * y \text{ powr } (a - \text{of\_nat } n)) \text{ sums } (x + y) \text{ powr } a$ 
a
proof -
  from assms have  $y > 0$  by simp
  note xy = this assms
  from assms have  $|x / y| < 1$  by simp
  hence  $(\lambda n. (a \text{ gchoose } n) * (x / y)^n) \text{ sums } (1 + x / y) \text{ powr } a$ 
    by (rule gen_binomial_real)
  hence  $(\lambda n. (a \text{ gchoose } n) * (x / y)^n * y \text{ powr } a) \text{ sums } ((1 + x / y) \text{ powr } a * y \text{ powr } a)$ 
    by (rule sums_mult2)
  with xy show ?thesis
    by (simp add: field_simps powr_divide powr_diff powr_realpow)
qed

```

```

lemma one_plus_neg_powr_powser:
  fixes z s :: complex
  assumes  $\text{norm } (z :: \text{complex}) < 1$ 
  shows  $(\lambda n. (-1)^n * ((s + n - 1) \text{ gchoose } n) * z^n) \text{ sums } (1 + z) \text{ powr } (-s)$ 
    using gen_binomial_complex[OF assms, of  $-s$ ] by (simp add: gbinomial_minus)

```

```

lemma gen_binomial_real'':
  fixes x y a :: real
  assumes  $|y| < x$ 
  shows  $(\lambda n. (a \text{ gchoose } n) * x \text{ powr } (a - \text{of\_nat } n) * y^n) \text{ sums } (x + y) \text{ powr } a$ 
a
  using gen_binomial_real'[OF assms] by (simp add: mult_ac add.commute)

```

```

lemma sqrt_series':
   $|z| < a \implies (\lambda n. ((1/2) \text{ gchoose } n) * a \text{ powr } (1/2 - \text{real\_of\_nat } n) * z^n) \text{ sums }$ 
 $\text{sqrt } (a + z :: \text{real})$ 
  using gen_binomial_real''[of  $z \ a \ 1/2$ ] by (simp add: powr_half_sqrt)

```

```

lemma sqrt_series:
   $|z| < 1 \implies (\lambda n. ((1/2) \text{ gchoose } n) * z^n) \text{ sums sqrt } (1 + z)$ 
  using gen_binomial_real''[of  $z \ 1/2$ ] by (simp add: powr_half_sqrt)

```

end

6.46 Vitali Covering Theorem and an Application to Negligibility

```

theory Vitali_Covering_Theorem
imports
  HOL-Combinatorics.Permutations
  Equivalence_Lebesgue_Henstock_Integration
begin

lemma stretch_Galois:
  fixes  $x :: \text{real}^n$ 
  shows  $(\bigwedge k. m\ k \neq 0) \implies ((y = (\chi\ k. m\ k * x\$k)) \longleftrightarrow (\chi\ k. y\$k / m\ k) = x)$ 
  by auto

lemma lambda_swap_Galois:
   $(x = (\chi\ i. y\ \$\ \text{Transposition.transpose}\ m\ n\ i) \longleftrightarrow (\chi\ i. x\ \$\ \text{Transposition.transpose}\ m\ n\ i) = y)$ 
  by (auto; simp add: pointfree_idE vec_eq_iff)

lemma lambda_add_Galois:
  fixes  $x :: \text{real}^n$ 
  shows  $m \neq n \implies (x = (\chi\ i. \text{if } i = m \text{ then } y\$m + y\$n \text{ else } y\$i) \longleftrightarrow (\chi\ i. \text{if } i = m \text{ then } x\$m - x\$n \text{ else } x\$i) = y)$ 
  by (safe; simp add: vec_eq_iff)

lemma Vitali_covering_lemma_cballs_balls:
  fixes  $a :: 'a \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes  $\bigwedge i. i \in K \implies 0 < r\ i \wedge r\ i \leq B$ 
  obtains  $C$  where  $\text{countable } C\ C \subseteq K$ 
   $\text{pairwise } (\lambda i\ j. \text{disjnt } (\text{cball } (a\ i)\ (r\ i))\ (\text{cball } (a\ j)\ (r\ j)))\ C$ 
   $\bigwedge i. i \in K \implies \exists j. j \in C \wedge$ 
     $\neg \text{disjnt } (\text{cball } (a\ i)\ (r\ i))\ (\text{cball } (a\ j)\ (r\ j)) \wedge$ 
     $\text{cball } (a\ i)\ (r\ i) \subseteq \text{ball } (a\ j)\ (5 * r\ j)$ 
proof (cases  $K = \{\}$ )
  case True
  with that show ?thesis
  by auto
next
  case False
  then have  $B > 0$ 
  using assms less_le_trans by auto
  have rgt0[simp]:  $\bigwedge i. i \in K \implies 0 < r\ i$ 
  using assms by auto
  let ?djnt =  $\text{pairwise } (\lambda i\ j. \text{disjnt } (\text{cball } (a\ i)\ (r\ i))\ (\text{cball } (a\ j)\ (r\ j)))$ 
  have  $\exists C. \forall n. (C\ n \subseteq K \wedge$ 
     $(\forall i \in C\ n. B/2 \wedge n \leq r\ i) \wedge ?djnt\ (C\ n) \wedge$ 
     $(\forall i \in K. B/2 \wedge n < r\ i \longrightarrow (\exists j. j \in C\ n \wedge$ 

```

```

       $\neg \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j)) \wedge$ 
       $\text{cball } (a \ i) \ (r \ i) \subseteq \text{ball } (a \ j) \ (5 * r \ j))) \wedge (C \ n \subseteq C(\text{Suc } n))$ 
proof (rule dependent_nat_choice, safe)
  fix C n
  define D where  $D \equiv \{i \in K. B/2 \wedge \text{Suc } n < r \ i \wedge (\forall j \in C. \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j))))\}$ 
  let ?cover_ar =  $\lambda i \ j. \neg \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j)) \wedge$ 
       $\text{cball } (a \ i) \ (r \ i) \subseteq \text{ball } (a \ j) \ (5 * r \ j)$ 
  assume  $C \subseteq K$ 
  and Ble:  $\forall i \in C. B/2 \wedge n \leq r \ i$ 
  and djntC: ?djnt C
  and cov_n:  $\forall i \in K. B/2 \wedge n < r \ i \longrightarrow (\exists j. j \in C \wedge ?cover\_ar \ i \ j)$ 
  have *:  $\forall C \in \text{chains } \{C. C \subseteq D \wedge ?djnt \ C\}. \bigcup C \in \{C. C \subseteq D \wedge ?djnt \ C\}$ 
  proof (clarsimp simp: chains_def)
  fix C
  assume C:  $C \subseteq \{C. C \subseteq D \wedge ?djnt \ C\}$  and chain $\subseteq$  C
  show  $\bigcup C \subseteq D \wedge ?djnt \ (\bigcup C)$ 
  unfolding pairwise_def
  proof (intro ballI conjI impI)
  show  $\bigcup C \subseteq D$ 
  using C by blast
  next
  fix x y
  assume  $x \in \bigcup C$  and  $y \in \bigcup C$  and  $x \neq y$ 
  then obtain X Y where XY:  $x \in X \ X \in C \ y \in Y \ Y \in C$ 
  by blast
  then consider  $X \subseteq Y \mid Y \subseteq X$ 
  by (meson <chain $\subseteq$  C> chain_subset_def)
  then show  $\text{disjnt } (\text{cball } (a \ x) \ (r \ x)) \ (\text{cball } (a \ y) \ (r \ y))$ 
  proof cases
  case 1
  with C XY <x ≠ y> show ?thesis
  unfolding pairwise_def by blast
  next
  case 2
  with C XY <x ≠ y> show ?thesis
  unfolding pairwise_def by blast
  qed
  qed
  obtain E where  $E \subseteq D$  and djntE: ?djnt E and maximalE:  $\bigwedge X. \llbracket X \subseteq D; ?djnt \ X; E \subseteq X \rrbracket \Longrightarrow X = E$ 
  using Zorn_Lemma [OF *] by safe blast
  show  $\exists L. (L \subseteq K \wedge$ 
       $(\forall i \in L. B/2 \wedge \text{Suc } n \leq r \ i) \wedge ?djnt \ L \wedge$ 
       $(\forall i \in K. B/2 \wedge \text{Suc } n < r \ i \longrightarrow (\exists j. j \in L \wedge ?cover\_ar \ i \ j))) \wedge C \subseteq L$ 
  proof (intro exI conjI ballI)
  show  $C \cup E \subseteq K$ 
  using D_def <C ⊆ K> <E ⊆ D> by blast

```

```

show  $B/2 \wedge \text{Suc } n \leq r \ i$  if  $i: i \in C \cup E$  for  $i$ 
  using  $i$ 
proof
  assume  $i \in C$ 
  have  $B/2 \wedge \text{Suc } n \leq B/2 \wedge n$ 
    using  $\langle B > 0 \rangle$  by (simp add: field_split_simps)
  also have  $\dots \leq r \ i$ 
    using  $\text{Ble } \langle i \in C \rangle$  by blast
  finally show ?thesis .
qed (use  $D\_def \ \langle E \subseteq D \rangle$  in auto)
show ?djnt  $(C \cup E)$ 
  using  $D\_def \ \langle C \subseteq K \rangle \ \langle E \subseteq D \rangle \ \text{djnt } C \ \text{djnt } E$ 
  unfolding pairwise_def disjnt_def by blast
next
fix  $i$ 
assume  $i \in K$ 
show  $B/2 \wedge \text{Suc } n < r \ i \longrightarrow (\exists j. j \in C \cup E \wedge ?cover\_ar \ i \ j)$ 
proof (cases  $r \ i \leq B/2 \wedge n$ )
  case False
  then show ?thesis
    using cov_n  $\langle i \in K \rangle$  by auto
next
case True
have cball  $(a \ i) \ (r \ i) \subseteq \text{ball } (a \ j) \ (5 * r \ j)$ 
  if less:  $B/2 \wedge \text{Suc } n < r \ i$  and  $j: j \in C \cup E$ 
  and nondis:  $\neg \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j))$  for  $j$ 
proof -
  obtain  $x$  where  $x: \text{dist } (a \ i) \ x \leq r \ i \ \text{dist } (a \ j) \ x \leq r \ j$ 
    using nondis by (force simp: disjnt_def)
  have  $\text{dist } (a \ i) \ (a \ j) \leq \text{dist } (a \ i) \ x + \text{dist } x \ (a \ j)$ 
    by (simp add: dist_triangle)
  also have  $\dots \leq r \ i + r \ j$ 
    by (metis add_mono_thms_linordered_semiring(1) dist_commute x)
  finally have  $\text{aij}: \text{dist } (a \ i) \ (a \ j) + r \ i < 5 * r \ j$  if  $r \ i < 2 * r \ j$ 
    using that by auto
  show ?thesis
    using  $j$ 
  proof
    assume  $j \in C$ 
    have  $B/2 \wedge n < 2 * r \ j$ 
      using  $\text{Ble True } \langle j \in C \rangle$  less by auto
    with aij True show  $\text{cball } (a \ i) \ (r \ i) \subseteq \text{ball } (a \ j) \ (5 * r \ j)$ 
      by (simp add: cball_subset_ball_iff)
  next
    assume  $j \in E$ 
    then have  $B/2 \wedge n < 2 * r \ j$ 
      using  $D\_def \ \langle E \subseteq D \rangle$  by auto
    with True have  $r \ i < 2 * r \ j$ 
      by auto
  end
end

```

```

      with aij show  $cball\ (a\ i)\ (r\ i) \subseteq ball\ (a\ j)\ (5 * r\ j)$ 
      by (simp add: cball_subset_ball_iff)
    qed
  qed
  moreover have  $\exists j. j \in C \cup E \wedge \neg disjoint\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j))$ 
  if  $B/2 \wedge Suc\ n < r\ i$ 
  proof (rule classical)
    assume NON:  $\neg ?thesis$ 
    show ?thesis
    proof (cases  $i \in D$ )
      case True
      have  $insert\ i\ E = E$ 
      proof (rule maximalE)
        show  $insert\ i\ E \subseteq D$ 
        by (simp add: True  $\langle E \subseteq D \rangle$ )
      show pairwise  $(\lambda i\ j. disjoint\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)))\ (insert\ i$ 
E)
        using False NON by (auto simp: pairwise_insert djntE disjoint_sym)
      qed auto
    then show ?thesis
      using  $\langle i \in K \rangle$  assms by fastforce
    next
      case False
      with that show ?thesis
      by (auto simp: D_def disjoint_def  $\langle i \in K \rangle$ )
    qed
  qed
  ultimately
  show  $B/2 \wedge Suc\ n < r\ i \longrightarrow$ 
     $(\exists j. j \in C \cup E \wedge$ 
       $\neg disjoint\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)) \wedge$ 
       $cball\ (a\ i)\ (r\ i) \subseteq ball\ (a\ j)\ (5 * r\ j))$ 
    by blast
  qed
  qed auto
  qed (use assms in force)
  then obtain F where FK:  $\bigwedge n. F\ n \subseteq K$ 
    and Fle:  $\bigwedge n\ i. i \in F\ n \implies B/2 \wedge n \leq r\ i$ 
    and Fdjnt:  $\bigwedge n. ?djnt\ (F\ n)$ 
    and FF:  $\bigwedge n\ i. \llbracket i \in K; B/2 \wedge n < r\ i \rrbracket$ 
       $\implies \exists j. j \in F\ n \wedge \neg disjoint\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)) \wedge$ 
       $cball\ (a\ i)\ (r\ i) \subseteq ball\ (a\ j)\ (5 * r\ j)$ 
    and inc:  $\bigwedge n. F\ n \subseteq F(Suc\ n)$ 
    by (force simp: all_conj_distrib)
  show thesis
  proof
    have *: countable I
    if  $I \subseteq K$  and pw: pairwise  $(\lambda i\ j. disjoint\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)))$ 
    I for I

```

```

proof -
  show ?thesis
proof (rule countable_image_inj_on [of  $\lambda i. cball(a\ i)(r\ i)$ ])
  show countable (( $\lambda i. cball\ (a\ i)\ (r\ i)$ ) '  $I$ )
proof (rule countable_disjoint_nonempty_interior_subsets)
  show disjoint (( $\lambda i. cball\ (a\ i)\ (r\ i)$ ) '  $I$ )
    by (auto simp: dest: pairwiseD [OF pw] intro: pairwise_imageI)
  show  $\bigwedge S. \llbracket S \in (\lambda i. cball\ (a\ i)\ (r\ i))\ ' I; interior\ S = \{\} \rrbracket \implies S = \{\}$ 
    using  $\langle I \subseteq K \rangle$ 
    by (auto simp: not_less [symmetric])
qed
next
have  $\bigwedge x\ y. \llbracket x \in I; y \in I; a\ x = a\ y; r\ x = r\ y \rrbracket \implies x = y$ 
  using pw  $\langle I \subseteq K \rangle$  assms
  apply (clarsimp simp: pairwise_def disjoint_def)
by (metis assms centre_in_cball subsetD empty_iff inf.idem less_eq_real_def)
then show inj_on ( $\lambda i. cball\ (a\ i)\ (r\ i)$ )  $I$ 
  using  $\langle I \subseteq K \rangle$  by (fastforce simp: inj_on_def cball_eq_cball_iff dest:
assms)
qed
qed
show (Union(range  $F$ ))  $\subseteq K$ 
  using FK by blast
moreover show pairwise ( $\lambda i\ j. disjoint\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j))$ )
  (Union(range  $F$ ))
proof (rule pairwise_chain_Union)
  show chain $\subseteq$  (range  $F$ )
    unfolding chain_subset_def by clarify (meson inc lift_Suc_mono_le linear
subsetCE)
qed (use Fdjnt in blast)
ultimately show countable (Union(range  $F$ ))
  by (blast intro: *)
next
fix  $i$  assume  $i \in K$ 
then obtain  $n$  where  $(1/2)^\wedge n < r\ i / B$ 
  using  $\langle B > 0 \rangle$  assms real_arch_pow_inv by fastforce
then have  $B2: B/2^\wedge n < r\ i$ 
  using  $\langle B > 0 \rangle$  by (simp add: field_split_simps)
have  $0 < r\ i\ r\ i \leq B$ 
  by (auto simp:  $\langle i \in K \rangle$  assms)
show  $\exists j. j \in (Union(range\ F)) \wedge$ 
   $\neg disjoint\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)) \wedge$ 
   $cball\ (a\ i)\ (r\ i) \subseteq ball\ (a\ j)\ (5 * r\ j)$ 
  using FF [OF  $\langle i \in K \rangle\ B2$ ] by auto
qed
qed

```

6.46.1 Vitali covering theorem

lemma *Vitali_covering_lemma_cballs:*

fixes $a :: 'a \Rightarrow 'b::\text{euclidean\_space}$

assumes $S: S \subseteq (\bigcup i \in K. \text{cball } (a \ i) \ (r \ i))$

and $r: \bigwedge i. i \in K \implies 0 < r \ i \wedge r \ i \leq B$

obtains C **where** *countable* $C \ C \subseteq K$

pairwise $(\lambda i \ j. \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j))) \ C$

$S \subseteq (\bigcup i \in C. \text{cball } (a \ i) \ (5 * r \ i))$

proof –

obtain C **where** $C: \text{countable } C \ C \subseteq K$

pairwise $(\lambda i \ j. \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j))) \ C$

and $\text{cov}: \bigwedge i. i \in K \implies \exists j. j \in C \wedge \neg \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j)) \wedge$

$\text{cball } (a \ i) \ (r \ i) \subseteq \text{ball } (a \ j) \ (5 * r \ j)$

by (rule *Vitali_covering_lemma_cballs_balls* [OF r , **where** $a=a$]) (blast intro: that)+

show ?thesis

proof

have $(\bigcup i \in K. \text{cball } (a \ i) \ (r \ i)) \subseteq (\bigcup i \in C. \text{cball } (a \ i) \ (5 * r \ i))$

using *cov subset_iff* **by** *fastforce*

with S **show** $S \subseteq (\bigcup i \in C. \text{cball } (a \ i) \ (5 * r \ i))$

by *blast*

qed (use C in *auto*)

qed

lemma *Vitali_covering_lemma_balls:*

fixes $a :: 'a \Rightarrow 'b::\text{euclidean\_space}$

assumes $S: S \subseteq (\bigcup i \in K. \text{ball } (a \ i) \ (r \ i))$

and $r: \bigwedge i. i \in K \implies 0 < r \ i \wedge r \ i \leq B$

obtains C **where** *countable* $C \ C \subseteq K$

pairwise $(\lambda i \ j. \text{disjnt } (\text{ball } (a \ i) \ (r \ i)) \ (\text{ball } (a \ j) \ (r \ j))) \ C$

$S \subseteq (\bigcup i \in C. \text{ball } (a \ i) \ (5 * r \ i))$

proof –

obtain C **where** $C: \text{countable } C \ C \subseteq K$

and $\text{pw}: \text{pairwise } (\lambda i \ j. \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j))) \ C$

and $\text{cov}: \bigwedge i. i \in K \implies \exists j. j \in C \wedge \neg \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j)) \wedge$

$\text{cball } (a \ i) \ (r \ i) \subseteq \text{ball } (a \ j) \ (5 * r \ j)$

by (rule *Vitali_covering_lemma_cballs_balls* [OF r , **where** $a=a$]) (blast intro: that)+

show ?thesis

proof

have $(\bigcup i \in K. \text{ball } (a \ i) \ (r \ i)) \subseteq (\bigcup i \in C. \text{ball } (a \ i) \ (5 * r \ i))$

using *cov subset_iff*

by *clarsimp* (*meson less_imp_le mem_ball mem_cball subset_eq*)

with S **show** $S \subseteq (\bigcup i \in C. \text{ball } (a \ i) \ (5 * r \ i))$

by *blast*

show *pairwise* $(\lambda i \ j. \text{disjnt } (\text{ball } (a \ i) \ (r \ i)) \ (\text{ball } (a \ j) \ (r \ j))) \ C$

using pw


```

    by (clarsimp simp: pairwise_def) (meson ball_subset_cball disjnt_subset1
disjnt_subset2)
  qed (use C in auto)
qed

```

theorem Vitali_covering_theorem_cballs:

```

  fixes a :: 'a  $\Rightarrow$  'n::euclidean_space
  assumes r:  $\bigwedge i. i \in K \implies 0 < r\ i$ 
    and S:  $\bigwedge x\ d. \llbracket x \in S; 0 < d \rrbracket \implies \exists i. i \in K \wedge x \in \text{cball } (a\ i) (r\ i) \wedge r\ i < d$ 
  obtains C where countable C C  $\subseteq$  K
    pairwise ( $\lambda i\ j. \text{disjnt } (\text{cball } (a\ i) (r\ i)) (\text{cball } (a\ j) (r\ j))$ ) C
    negligible(S - ( $\bigcup i \in C. \text{cball } (a\ i) (r\ i)$ ))
proof -
  let ? $\mu$  = measure lebesgue
  have *:  $\exists C. \text{countable } C \wedge C \subseteq K \wedge$ 
    pairwise ( $\lambda i\ j. \text{disjnt } (\text{cball } (a\ i) (r\ i)) (\text{cball } (a\ j) (r\ j))$ ) C  $\wedge$ 
    negligible(S - ( $\bigcup i \in C. \text{cball } (a\ i) (r\ i)$ ))
  if r01:  $\bigwedge i. i \in K \implies 0 < r\ i \wedge r\ i \leq 1$ 
    and Sd:  $\bigwedge x\ d. \llbracket x \in S; 0 < d \rrbracket \implies \exists i. i \in K \wedge x \in \text{cball } (a\ i) (r\ i) \wedge r\ i < d$ 
  d
    for K r and a :: 'a  $\Rightarrow$  'n
  proof -
    obtain C where C: countable C C  $\subseteq$  K
      and pwC: pairwise ( $\lambda i\ j. \text{disjnt } (\text{cball } (a\ i) (r\ i)) (\text{cball } (a\ j) (r\ j))$ ) C
      and cov:  $\bigwedge i. i \in K \implies \exists j. j \in C \wedge \neg \text{disjnt } (\text{cball } (a\ i) (r\ i)) (\text{cball } (a\ j) (r\ j)) \wedge$ 
         $\text{cball } (a\ i) (r\ i) \subseteq \text{ball } (a\ j) (5 * r\ j)$ 
    by (rule Vitali_covering_lemma_cballs_balls [of K r 1 a]) (auto simp: r01)
    have ar_injective:  $\bigwedge x\ y. \llbracket x \in C; y \in C; a\ x = a\ y; r\ x = r\ y \rrbracket \implies x = y$ 
      using  $\langle C \subseteq K \rangle$  pwC cov
    by (force simp: pairwise_def disjnt_def)
    show ?thesis
    proof (intro exI conjI)
      show negligible (S - ( $\bigcup i \in C. \text{cball } (a\ i) (r\ i)$ ))
    proof (clarsimp simp: negligible_on_intervals [of S - T for T])
      fix l u
      show negligible ((S - ( $\bigcup i \in C. \text{cball } (a\ i) (r\ i)$ ))  $\cap$  cbox l u)
        unfolding negligible_outer_le
    proof (intro allI impI)
      fix e::real
      assume e > 0
      define D where D  $\equiv \{i \in C. \neg \text{disjnt } (\text{ball } (a\ i) (5 * r\ i)) (\text{cbox } l\ u)\}$ 
      then have D  $\subseteq C$ 
        by auto
      have countable D
        unfolding D_def using  $\langle \text{countable } C \rangle$  by simp
      have UD: ( $\bigcup i \in D. \text{cball } (a\ i) (r\ i)$ )  $\in$  lmeasurable

```

```

proof (rule fmeasurableI2)
  show  $\text{cbox } (l - 6 *_{\mathcal{R}} \text{One}) (u + 6 *_{\mathcal{R}} \text{One}) \in \text{lmeasurable}$ 
    by blast
  have  $y \in \text{cbox } (l - 6 *_{\mathcal{R}} \text{One}) (u + 6 *_{\mathcal{R}} \text{One})$ 
    if  $i \in C$  and  $x: x \in \text{cbox } l \ u$  and  $ai: \text{dist } (a \ i) \ y \leq r \ i \ \text{dist } (a \ i) \ x < 5$ 
    *  $r \ i$ 
    for  $i \ x \ y$ 
  proof -
    have  $d6: \text{dist } y \ x < 6 * r \ i$ 
      using  $\text{dist\_triangle3} \ [of \ y \ x \ a \ i] \ \text{that}$  by  $\text{linarith}$ 
    show ?thesis
  proof (clarsimp simp: mem_box algebra_simps)
    fix  $j::'n$ 
    assume  $j: j \in \text{Basis}$ 
    then have  $xyj: |x \cdot j - y \cdot j| \leq \text{dist } y \ x$ 
    by (metis Basis_le_norm dist_commute dist_norm inner_diff_left)
    have  $l \cdot j \leq x \cdot j$ 
      using  $\langle j \in \text{Basis} \rangle \ \text{mem\_box } \langle x \in \text{cbox } l \ u \rangle$  by blast
    also have  $\dots \leq y \cdot j + 6 * r \ i$ 
      using  $d6 \ xyj$  by (auto simp: algebra_simps)
    also have  $\dots \leq y \cdot j + 6$ 
      using  $r01 \ [of \ i] \ \langle C \subseteq K \rangle \ \langle i \in C \rangle$  by auto
    finally have  $l \cdot j \leq y \cdot j + 6$  .
    have  $y \cdot j \leq x \cdot j + 6 * r \ i$ 
      using  $d6 \ xyj$  by (auto simp: algebra_simps)
    also have  $\dots \leq u \cdot j + 6 * r \ i$ 
      using  $j \ x$  by (auto simp: mem_box)
    also have  $\dots \leq u \cdot j + 6$ 
      using  $r01 \ [of \ i] \ \langle C \subseteq K \rangle \ \langle i \in C \rangle$  by auto
    finally have  $u \cdot j \leq y \cdot j + 6$  .
    show  $l \cdot j \leq y \cdot j + 6 \wedge y \cdot j \leq u \cdot j + 6$ 
      using  $l \ u$  by blast
    qed
  qed
then show  $(\bigcup_{i \in D}. \text{cball } (a \ i) (r \ i)) \subseteq \text{cbox } (l - 6 *_{\mathcal{R}} \text{One}) (u + 6 *_{\mathcal{R}} \text{One})$ 
  One)
    by (force simp: D_def disjnt_def)
  show  $(\bigcup_{i \in D}. \text{cball } (a \ i) (r \ i)) \in \text{sets lebesgue}$ 
    using  $\langle \text{countable } D \rangle$  by auto
  qed
obtain  $D1$  where  $D1 \subseteq D$  finite  $D1$ 
  and  $\text{meas} D1: ?\mu (\bigcup_{i \in D}. \text{cball } (a \ i) (r \ i)) - e / 5 \wedge \text{DIM}('n) < ?\mu$ 
   $(\bigcup_{i \in D1}. \text{cball } (a \ i) (r \ i))$ 
  proof (rule measure_countable_Union_approachable [where  $e = e / 5 \wedge$ 
   $(\text{DIM}('n))$ ])
    show  $\text{countable } ((\lambda i. \text{cball } (a \ i) (r \ i)) \cdot D)$ 
      using  $\langle \text{countable } D \rangle$  by auto
    show  $\bigwedge d. d \in (\lambda i. \text{cball } (a \ i) (r \ i)) \cdot D \implies d \in \text{lmeasurable}$ 
      by auto

```

```

show  $\bigwedge D'. \llbracket D' \subseteq (\lambda i. \text{cball } (a \ i) \ (r \ i)) \text{ ' } D; \text{finite } D \rrbracket \implies ?\mu (\bigcup D') \leq$ 
 $?\mu (\bigcup_{i \in D. \text{cball } (a \ i) \ (r \ i)})$ 
by (fastforce simp add: intro!: measure_mono_fmeasurable UD)
qed (use <e > 0> in <auto dest: finite_subset_image>)
show  $\exists T. (S - (\bigcup_{i \in C. \text{cball } (a \ i) \ (r \ i))) \cap$ 
 $\text{cbox } l \ u \subseteq T \wedge T \in \text{lmeasurable} \wedge ?\mu \ T \leq e$ 
proof (intro exI conjI)
show  $(S - (\bigcup_{i \in C. \text{cball } (a \ i) \ (r \ i))) \cap \text{cbox } l \ u \subseteq (\bigcup_{i \in D - D1. \text{ball}}$ 
 $(a \ i) \ (5 * r \ i))$ 
proof clarify
fix x
assume x:  $x \in \text{cbox } l \ u \ x \in S \ x \notin (\bigcup_{i \in C. \text{cball } (a \ i) \ (r \ i)})$ 
have closed  $(\bigcup_{i \in D1. \text{cball } (a \ i) \ (r \ i)})$ 
using <finite D1> by blast
moreover have  $x \notin (\bigcup_{j \in D1. \text{cball } (a \ j) \ (r \ j)})$ 
using x <D1  $\subseteq D$ > unfolding D_def by blast
ultimately obtain q where  $q > 0$  and  $q: \text{ball } x \ q \subseteq - (\bigcup_{i \in D1. \text{cball}}$ 
 $(a \ i) \ (r \ i))$ 
by (metis (no_types, lifting) ComplI open_contains_ball closed_def)
obtain i where  $i \in K$  and  $xi: x \in \text{cball } (a \ i) \ (r \ i)$  and  $ri: r \ i < q/2$ 
using Sd [OF <x  $\in S$ >] <q > 0> half_gt_zero by blast
then obtain j where  $j \in C$ 
and nondisj:  $\neg \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j))$ 
and sub5j:  $\text{cball } (a \ i) \ (r \ i) \subseteq \text{ball } (a \ j) \ (5 * r \ j)$ 
using cov [OF <i  $\in K$ >] by metis
show  $x \in (\bigcup_{i \in D - D1. \text{ball } (a \ i) \ (5 * r \ i)})$ 
proof
show  $j \in D - D1$ 
proof
show  $j \in D$ 
using <j  $\in C$ > sub5j <x  $\in \text{cbox } l \ u$ > xi by (auto simp: D_def
disjnt_def)
obtain y where  $yi: \text{dist } (a \ i) \ y \leq r \ i$  and  $yj: \text{dist } (a \ j) \ y \leq r \ j$ 
using disjnt_def nondisj by fastforce
have  $\text{dist } x \ y \leq r \ i + r \ i$ 
by (metis add_mono dist_commute dist_triangle_le mem_cball
xi yi)
also have  $\dots < q$ 
using ri by linarith
finally have  $y \in \text{ball } x \ q$ 
by simp
with yj q show  $j \notin D1$ 
by (auto simp: disjoint_UN_iff)
qed
show  $x \in \text{ball } (a \ j) \ (5 * r \ j)$ 
using xi sub5j by blast
qed
qed
have  $\exists: ?\mu (\bigcup_{i \in D2. \text{ball } (a \ i) \ (5 * r \ i)}) \leq e$ 

```

```

    if D2: D2 ⊆ D - D1 and finite D2 for D2
  proof -
    have rgt0: 0 < r i if i ∈ D2 for i
    using ⟨C ⊆ K⟩ D_def ⟨i ∈ D2⟩ D2 r01
    by (simp add: subset_iff)
    then have inj: inj_on (λi. ball (a i) (5 * r i)) D2
    using ⟨C ⊆ K⟩ D2 by (fastforce simp: inj_on_def D_def
ball_eq_ball_iff intro: ar_injective)
    have ?μ (⋃ i∈D2. ball (a i) (5 * r i)) ≤ sum (?μ) ((λi. ball (a i) (5
* r i)) ‘ D2)
    using that by (force intro: measure_Union_le)
    also have ... = (∑ i∈D2. ?μ (ball (a i) (5 * r i)))
    by (simp add: comm_monoid_add_class.sum.reindex [OF inj])
    also have ... = (∑ i∈D2. 5 ^ DIM('n) * ?μ (ball (a i) (r i)))
  proof (rule sum.cong [OF refl])
    fix i assume i ∈ D2
    thus ?μ (ball (a i) (5 * r i)) = 5 ^ DIM('n) * ?μ (ball (a i) (r i))
    using content_ball_conv_unit_ball[of 5 * r i a i]
    content_ball_conv_unit_ball[of r i a i] rgt0[of i] by auto
  qed
  also have ... = (∑ i∈D2. ?μ (ball (a i) (r i))) * 5 ^ DIM('n)
  by (simp add: sum_distrib_left mult.commute)
  finally have ?μ (⋃ i∈D2. ball (a i) (5 * r i)) ≤ (∑ i∈D2. ?μ (ball (a
i) (r i))) * 5 ^ DIM('n) .
  moreover have (∑ i∈D2. ?μ (ball (a i) (r i))) ≤ e / 5 ^ DIM('n)
  proof -
    have D12_dis: ((⋃ x∈D1. cball (a x) (r x)) ∩ (⋃ x∈D2. cball (a x)
(r x))) ≤ {}
    proof clarify
      fix w d1 d2
      assume d1 ∈ D1 w d1 d2 ∈ cball (a d1) (r d1) d2 ∈ D2 w d1 d2
∈ cball (a d2) (r d2)
      then show w d1 d2 ∈ {}
      by (metis DiffE disjnt_iff subsetCE D2 ⟨D1 ⊆ D⟩ ⟨D ⊆ C⟩
pairwiseD [OF pwC, of d1 d2])
    qed
    have inj: inj_on (λi. cball (a i) (r i)) D2
    using rgt0 D2 ⟨D ⊆ C⟩ by (force simp: inj_on_def cball_eq_cball_iff
intro!: ar_injective)
    have ds: disjoint ((λi. cball (a i) (r i)) ‘ D2)
    using D2 ⟨D ⊆ C⟩ by (auto intro: pairwiseI pairwiseD [OF pwC])
    have (∑ i∈D2. ?μ (ball (a i) (r i))) = (∑ i∈D2. ?μ (cball (a i) (r
i)))
    by (simp add: content_cball_conv_ball)
    also have ... = sum ?μ ((λi. cball (a i) (r i)) ‘ D2)
    by (simp add: comm_monoid_add_class.sum.reindex [OF inj])
    also have ... = ?μ (⋃ i∈D2. cball (a i) (r i))
    by (auto intro: measure_Union' [symmetric] ds simp add: ⟨finite
D2⟩)

```

```

    finally have ?μ (⋃ i∈D1. cball (a i) (r i)) + (∑ i∈D2. ?μ (ball (a
i) (r i))) =
      ?μ (⋃ i∈D1. cball (a i) (r i)) + ?μ (⋃ i∈D2. cball (a i)
(r i))
      by simp
    also have ... = ?μ (⋃ i ∈ D1 ∪ D2. cball (a i) (r i))
    using D12_dis by (simp add: measure_Un3 ⟨finite D1⟩ ⟨finite D2⟩
fmeasurable.finite_UN)
    also have ... ≤ ?μ (⋃ i∈D. cball (a i) (r i))
    using D2 ⟨D1 ⊆ D⟩ by (fastforce intro!: measure_mono_fmeasurable
[OF __ UD] ⟨finite D1⟩ ⟨finite D2⟩)
    finally have ?μ (⋃ i∈D1. cball (a i) (r i)) + (∑ i∈D2. ?μ (ball (a
i) (r i))) ≤ ?μ (⋃ i∈D. cball (a i) (r i)) .
    with measD1 show ?thesis
    by simp
  qed
  ultimately show ?thesis
  by (simp add: field_split_simps)
qed
have co: countable (D - D1)
  by (simp add: ⟨countable D⟩)
show (⋃ i∈D - D1. ball (a i) (5 * r i)) ∈ lmeasurable
  using ⟨e > 0⟩ by (auto simp: fmeasurable_UN_bound [OF co _ 3])
show ?μ (⋃ i∈D - D1. ball (a i) (5 * r i)) ≤ e
  using ⟨e > 0⟩ by (auto simp: measure_UN_bound [OF co _ 3])
qed
qed
qed
qed (use C pwC in auto)
qed
define K' where K' ≡ {i ∈ K. r i ≤ 1}
have 1: ⋀ i. i ∈ K' ⟹ 0 < r i ∧ r i ≤ 1
  using K'_def r by auto
have 2: ∃ i. i ∈ K' ∧ x ∈ cball (a i) (r i) ∧ r i < d
  if x ∈ S ∧ 0 < d for x d
  using that by (auto simp: K'_def dest!: S [where d = min d 1])
have K' ⊆ K
  using K'_def by auto
then show thesis
  using * [OF 1 2] that by fastforce
qed

```

theorem Vitali_covering_theorem_balls:

fixes $a :: 'a \Rightarrow 'b::euclidean\_space$

assumes $S: \bigwedge x d. \llbracket x \in S; 0 < d \rrbracket \Longrightarrow \exists i. i \in K \wedge x \in \text{ball } (a \ i) \ (r \ i) \wedge r \ i < d$

obtains C **where** $\text{countable } C \ C \subseteq K$

$\text{pairwise } (\lambda i \ j. \text{disjnt } (\text{ball } (a \ i) \ (r \ i)) \ (\text{ball } (a \ j) \ (r \ j))) \ C$

$\text{negligible}(S - (\bigcup i \in C. \text{ball } (a \ i) \ (r \ i)))$

proof –

have $1: \exists i. i \in \{i \in K. 0 < r\ i\} \wedge x \in cball\ (a\ i)\ (r\ i) \wedge r\ i < d$
if $xd: x \in S\ d > 0$ **for** $x\ d$
by (*metis* (*mono_tags*, *lifting*) *assms* *ball_eq_empty* *less_eq_real_def* *mem_Collect_eq* *mem_ball* *mem_cball* *not_le* *xd(1)* *xd(2)*)
obtain C **where** $C: countable\ C\ C \subseteq K$
and $pw: pairwise\ (\lambda i\ j. disjnt\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)))\ C$
and $neg: negligible(S - (\bigcup i \in C. cball\ (a\ i)\ (r\ i)))$
by (*rule* *Vitali_covering_theorem_cballs* [*of* $\{i \in K. 0 < r\ i\}\ r\ S\ a, OF\_1$])
auto
show *thesis*
proof
show $pairwise\ (\lambda i\ j. disjnt\ (ball\ (a\ i)\ (r\ i))\ (ball\ (a\ j)\ (r\ j)))\ C$
apply (*rule* *pairwise_mono* [*OF* *pw*])
apply (*auto* *simp: disjnt_def*)
by (*meson* *disjoint_iff_not_equal* *less_imp_le* *mem_cball*)
have $negligible\ (\bigcup i \in C. sphere\ (a\ i)\ (r\ i))$
by (*auto* *intro: negligible_sphere* $\langle countable\ C \rangle$)
then have $negligible\ (S - (\bigcup i \in C. cball(a\ i)(r\ i)) \cup (\bigcup i \in C. sphere\ (a\ i)\ (r\ i)))$
by (*rule* *negligible_Un* [*OF* *neg*])
then show $negligible\ (S - (\bigcup i \in C. ball\ (a\ i)\ (r\ i)))$
by (*rule* *negligible_subset*) *force*
qed (*use* C *in* *auto*)
qed

lemma *negligible_eq_zero_density_alt*:

$negligible\ S \longleftrightarrow$
 $(\forall x \in S. \forall e > 0.$
 $\exists d\ U. 0 < d \wedge d \leq e \wedge S \cap ball\ x\ d \subseteq U \wedge$
 $U \in lmeasurable \wedge measure\ lebesgue\ U < e * measure\ lebesgue\ (ball\ x\ d))$
 $(is\_ = (\forall x \in S. \forall e > 0. ?Q\ x\ e))$

proof (*intro* *iffI* *ballI* *allI* *impI*)

fix x **and** $e :: real$

assume $negligible\ S$ **and** $x \in S$ **and** $e > 0$

then

show $\exists d\ U. 0 < d \wedge d \leq e \wedge S \cap ball\ x\ d \subseteq U \wedge U \in lmeasurable \wedge$
 $measure\ lebesgue\ U < e * measure\ lebesgue\ (ball\ x\ d)$

apply (*rule_tac* $x=e$ *in* *exI*)

apply (*rule_tac* $x=S \cap ball\ x\ e$ *in* *exI*)

apply (*auto* *simp: negligible_imp_measurable* *negligible_Int* *negligible_imp_measure0* *zero_less_measure_iff*

intro: mult_pos_pos *content_ball_pos*)

done

next

assume R [*rule_format*]: $\forall x \in S. \forall e > 0. ?Q\ x\ e$

let $? \mu = measure\ lebesgue$

```

have  $\exists U. \text{openin } (\text{top\_of\_set } S) \ U \wedge z \in U \wedge \text{negligible } U$ 
  if  $z \in S$  for  $z$ 
proof (intro exI conjI)
  show  $\text{openin } (\text{top\_of\_set } S) \ (S \cap \text{ball } z \ 1)$ 
    by (simp add: openin_open_Int)
  show  $z \in S \cap \text{ball } z \ 1$ 
    using  $\langle z \in S \rangle$  by auto
  show  $\text{negligible } (S \cap \text{ball } z \ 1)$ 
proof (clarsimp simp: negligible_outer_le)
  fix  $e :: \text{real}$ 
  assume  $e > 0$ 
  let  $?K = \{(x,d). x \in S \wedge 0 < d \wedge \text{ball } x \ d \subseteq \text{ball } z \ 1 \wedge$ 
     $(\exists U. S \cap \text{ball } x \ d \subseteq U \wedge U \in \text{lmeasurable} \wedge$ 
     $? \mu \ U < e / ? \mu (\text{ball } z \ 1) * ? \mu (\text{ball } x \ d))\}$ 
  obtain  $C$  where  $\text{countable } C$  and  $C_{\text{sub}}: C \subseteq ?K$ 
    and  $\text{pwC}: \text{pairwise } (\lambda i \ j. \text{disjnt } (\text{ball } (\text{fst } i) (\text{snd } i)) (\text{ball } (\text{fst } j) (\text{snd } j))) \ C$ 
    and  $\text{negC}: \text{negligible}((S \cap \text{ball } z \ 1) - (\bigcup i \in C. \text{ball } (\text{fst } i) (\text{snd } i)))$ 
  proof (rule Vitali_covering_theorem_balls [of  $S \cap \text{ball } z \ 1 \ ?K \ \text{fst } \text{snd}$ ])
    fix  $x$  and  $d :: \text{real}$ 
    assume  $x: x \in S \cap \text{ball } z \ 1$  and  $d > 0$ 
    obtain  $k$  where  $k > 0$  and  $k: \text{ball } x \ k \subseteq \text{ball } z \ 1$ 
      by (meson Int_iff open_ball openE x)
    let  $? \varepsilon = \min (e / ? \mu (\text{ball } z \ 1) / 2) (\min (d / 2) k)$ 
    obtain  $r \ U$  where  $r: r > 0 \ r \leq ? \varepsilon$  and  $U: S \cap \text{ball } x \ r \subseteq U \ U \in \text{lmeasurable}$ 
      and  $mU: ? \mu \ U < ? \varepsilon * ? \mu (\text{ball } x \ r)$ 
    using  $R$  [of  $x \ ? \varepsilon$ ]  $\langle d > 0 \rangle \langle e > 0 \rangle \langle k > 0 \rangle x$  by (auto simp: content_ball_pos)
    show  $\exists i. i \in ?K \wedge x \in \text{ball } (\text{fst } i) (\text{snd } i) \wedge \text{snd } i < d$ 
  proof (rule exI [of  $\_ (x,r)$ ], simp, intro conjI exI)
    have  $\text{ball } x \ r \subseteq \text{ball } x \ k$ 
      using  $r$  by (simp add: ball_subset_ball_iff)
    also have  $\dots \subseteq \text{ball } z \ 1$ 
      using ball_subset_ball_iff  $k$  by auto
    finally show  $\text{ball } x \ r \subseteq \text{ball } z \ 1$  .
    have  $? \varepsilon * ? \mu (\text{ball } x \ r) \leq e * \text{content } (\text{ball } x \ r) / \text{content } (\text{ball } z \ 1)$ 
      using  $r \ \langle e > 0 \rangle$  by (simp add: ord_class.min_def field_split_simps
content_ball_pos)
    with  $mU$  show  $? \mu \ U < e * \text{content } (\text{ball } x \ r) / \text{content } (\text{ball } z \ 1)$ 
      by auto
    qed (use  $r \ U \ x$  in auto)
  qed
have  $\exists U. \text{case } p \text{ of } (x,d) \Rightarrow S \cap \text{ball } x \ d \subseteq U \wedge$ 
   $U \in \text{lmeasurable} \wedge ? \mu \ U < e / ? \mu (\text{ball } z \ 1) * ? \mu (\text{ball } x \ d)$ 
  if  $p \in C$  for  $p$ 
  using that  $C_{\text{sub}}$  unfolding case_prod_unfold by blast
then obtain  $U$  where  $U:$ 
   $\bigwedge p. p \in C \Rightarrow$ 
   $\text{case } p \text{ of } (x,d) \Rightarrow S \cap \text{ball } x \ d \subseteq U \wedge$ 
   $U \in \text{lmeasurable} \wedge ? \mu (U \ p) < e / ? \mu (\text{ball } z \ 1) * ? \mu (\text{ball } x \ d)$ 
  by (rule that [OF someI_ex])

```

```

let ?T = ((S ∩ ball z 1) - (⋃(x,d)∈C. ball x d)) ∪ ⋃(U ' C)
show ∃ T. S ∩ ball z 1 ⊆ T ∧ T ∈ lmeasurable ∧ ?μ T ≤ e
proof (intro exI conjI)
  show S ∩ ball z 1 ⊆ ?T
    using U by fastforce
  { have Um: U i ∈ lmeasurable if i ∈ C for i
    using that U by blast
    have lee: ?μ (⋃ i ∈ I. U i) ≤ e if I ⊆ C finite I for I
    proof -
      have ?μ (⋃(x,d)∈I. ball x d) ≤ ?μ (ball z 1)
      apply (rule measure_mono_fmeasurable)
      using ⟨I ⊆ C⟩ ⟨finite I⟩ Csub by (force simp: prod.case_eq_if
sets.finite_UN)+
      then have le1: (?μ (⋃(x,d)∈I. ball x d) / ?μ (ball z 1)) ≤ 1
      by (simp add: content_ball_pos)
      have ?μ (⋃ i ∈ I. U i) ≤ (∑ i ∈ I. ?μ (U i))
      using that U by (blast intro: measure_UNION_le)
      also have ... ≤ (∑(x,r)∈I. e / ?μ (ball z 1) * ?μ (ball x r))
      by (rule sum_mono) (use ⟨I ⊆ C⟩ U in force)
      also have ... = (e / ?μ (ball z 1)) * (∑(x,r)∈I. ?μ (ball x r))
      by (simp add: case_prod_app prod.case_distrib sum_distrib_left)
      also have ... = e * (?μ (⋃(x,r)∈I. ball x r) / ?μ (ball z 1))
      apply (subst measure_UNION')
      using that pwC by (auto simp: case_prod_unfold elim: pairwise_mono)
      also have ... ≤ e
      by (metis mult.commute mult.left_neutral mult_le_cancel_iff1 ⟨e >
0⟩ le1)
    finally show ?thesis .
  }
qed
have ⋃(U ' C) ∈ lmeasurable ?μ (⋃(U ' C)) ≤ e
using ⟨e > 0⟩ Um lee
  by(auto intro!: fmeasurable_UN_bound [OF ⟨countable C⟩] mea-
sure_UN_bound [OF ⟨countable C⟩])
}
moreover have ?μ ?T = ?μ (⋃(U ' C))
proof (rule measure_negligible_symdiff [OF ⟨⋃(U ' C) ∈ lmeasurable⟩])
  show negligible((⋃(U ' C) - ?T) ∪ (?T - ⋃(U ' C)))
  by (force intro!: negligible_subset [OF negC])
qed
ultimately show ?T ∈ lmeasurable ?μ ?T ≤ e
by (simp_all add: fmeasurable.Un negC negligible_imp_measurable
split_def)
qed
qed
qed
with locally_negligible_alt show negligible S
by metis
qed

```


proposition *negligible_eq_zero_density*:

```

  negligible  $S \longleftrightarrow$ 
    ( $\forall x \in S. \forall r > 0. \forall e > 0. \exists d. 0 < d \wedge d \leq r \wedge$ 
       $(\exists U. S \cap \text{ball } x \ d \subseteq U \wedge U \in \text{lmeasurable} \wedge \text{measure lebesgue } U$ 
 $< e * \text{measure lebesgue } (\text{ball } x \ d)))$ 
proof –
  let  $?Q = \lambda x \ d \ e. \exists U. S \cap \text{ball } x \ d \subseteq U \wedge U \in \text{lmeasurable} \wedge \text{measure lebesgue } U$ 
 $< e * \text{content } (\text{ball } x \ d)$ 
  have  $(\forall e > 0. \exists d > 0. d \leq e \wedge ?Q \ x \ d \ e) = (\forall r > 0. \forall e > 0. \exists d > 0. d \leq r \wedge ?Q$ 
 $x \ d \ e)$ 
    if  $x \in S$  for  $x$ 
  proof (intro iffI allI impI)
    fix  $r :: \text{real}$  and  $e :: \text{real}$ 
    assume  $L$  [rule_format]:  $\forall e > 0. \exists d > 0. d \leq e \wedge ?Q \ x \ d \ e$  and  $r > 0 \ e > 0$ 
    show  $\exists d > 0. d \leq r \wedge ?Q \ x \ d \ e$ 
      using  $L$  [of min  $r \ e$ ] apply (rule ex_forward)
      using  $\langle r > 0 \rangle \langle e > 0 \rangle$  by (auto intro: less_le_trans elim!: ex_forward simp:
content_ball_pos)
    qed auto
  then show  $?thesis$ 
    by (force simp: negligible_eq_zero_density_alt)
qed

end

```

6.47 Change of Variables Theorems

theory *Change_Of_Vars*

imports *Vitali_Covering_Theorem Determinants*

begin

6.47.1 Measurable Shear and Stretch

proposition

```

  fixes  $a :: \text{real}^n$ 
  assumes  $m \neq n$  and  $ab\_ne: \text{cbox } a \ b \neq \{\}$  and  $an: 0 \leq a\$n$ 
  shows measurable_shear_interval:  $(\lambda x. \chi \ i. \text{if } i = m \text{ then } x\$m + x\$n \text{ else } x\$i)$ 
 $' (\text{cbox } a \ b) \in \text{lmeasurable}$ 
    (is  $?f ' \_ \in \_$ )
    and measure_shear_interval:  $\text{measure lebesgue } ((\lambda x. \chi \ i. \text{if } i = m \text{ then } x\$m +$ 
 $x\$n \text{ else } x\$i) ' \text{cbox } a \ b)$ 
      =  $\text{measure lebesgue } (\text{cbox } a \ b)$  (is  $?Q$ )

```

proof –

```

  have  $\text{lin}: \text{linear } ?f$ 
    by (rule linearI) (auto simp: plus_vec_def scaleR_vec_def algebra_simps)
  show  $\text{fab}: ?f ' \text{cbox } a \ b \in \text{lmeasurable}$ 
    by (simp add: lin measurable_linear_image_interval)
  let  $?c = \chi \ i. \text{if } i = m \text{ then } b\$m + b\$n \text{ else } b\$i$ 

```

```

let ?mn = axis m 1 - axis n (1::real)
have eq1: measure lebesgue (cbox a ?c)
    = measure lebesgue (?f ' cbox a b)
    + measure lebesgue (cbox a ?c ∩ {x. ?mn • x ≤ a$m})
    + measure lebesgue (cbox a ?c ∩ {x. ?mn • x ≥ b$m})
proof (rule measure_Un3_negligible)
  show cbox a ?c ∩ {x. ?mn • x ≤ a$m} ∈ lmeasurable cbox a ?c ∩ {x. ?mn • x
    ≥ b$m} ∈ lmeasurable
  by (auto simp: convex_Int convex_halfspace_le convex_halfspace_ge bounded_Int
    measurable_convex)
  have negligible {x. ?mn • x = a$m}
    by (metis ⟨m ≠ n⟩ axis_index_axis eq_iff_diff_eq_0 negligible_hyperplane)
  moreover have ?f ' cbox a b ∩ (cbox a ?c ∩ {x. ?mn • x ≤ a $ m}) ⊆ {x.
    ?mn • x = a$m}
  using ⟨m ≠ n⟩ antisym_conv by (fastforce simp: algebra_simps mem_box_cart
    inner_axis')
  ultimately show negligible ((?f ' cbox a b) ∩ (cbox a ?c ∩ {x. ?mn • x ≤ a $
    m}))
    by (rule negligible_subset)
  have negligible {x. ?mn • x = b$m}
    by (metis ⟨m ≠ n⟩ axis_index_axis eq_iff_diff_eq_0 negligible_hyperplane)
  moreover have (?f ' cbox a b) ∩ (cbox a ?c ∩ {x. ?mn • x ≥ b$m}) ⊆ {x.
    ?mn • x = b$m}
  using ⟨m ≠ n⟩ antisym_conv by (fastforce simp: algebra_simps mem_box_cart
    inner_axis')
  ultimately show negligible (?f ' cbox a b ∩ (cbox a ?c ∩ {x. ?mn • x ≥ b$m}))
    by (rule negligible_subset)
  have negligible {x. ?mn • x = b$m}
    by (metis ⟨m ≠ n⟩ axis_index_axis eq_iff_diff_eq_0 negligible_hyperplane)
  moreover have (cbox a ?c ∩ {x. ?mn • x ≤ a $ m}) ∩ (cbox a ?c ∩ {x. ?mn •
    x ≥ b$m})) ⊆ {x. ?mn • x = b$m}
  using ⟨m ≠ n⟩ ab_ne
  apply (auto simp: algebra_simps mem_box_cart inner_axis')
  apply (drule_tac x=m in spec)+
  apply simp
  done
  ultimately show negligible (cbox a ?c ∩ {x. ?mn • x ≤ a $ m} ∩ (cbox a ?c
    ∩ {x. ?mn • x ≥ b$m}))
    by (rule negligible_subset)
  show ?f ' cbox a b ∪ cbox a ?c ∩ {x. ?mn • x ≤ a $ m} ∪ cbox a ?c ∩ {x. ?mn
    • x ≥ b$m} = cbox a ?c (is ?lhs = _)
  proof
    show ?lhs ⊆ cbox a ?c
    by (auto simp: mem_box_cart add_mono) (meson add_increasing2 an
      order_trans)
    show cbox a ?c ⊆ ?lhs
    apply (auto simp: algebra_simps image_iff inner_axis' lambda_add_Galois
      [OF ⟨m ≠ n⟩])
    apply (auto simp: mem_box_cart split: if_split_asm)

```

```

done
qed
qed (fact fab)
let ?d =  $\chi$  i. if i = m then a $ m - b $ m else 0
have eq2: measure lebesgue (cbox a ?c  $\cap$  {x. ?mn  $\cdot$  x  $\leq$  a $ m}) + measure
lebesgue (cbox a ?c  $\cap$  {x. ?mn  $\cdot$  x  $\geq$  b $ m})
= measure lebesgue (cbox a ( $\chi$  i. if i = m then a $ m + b $ n else b $ i))
proof (rule measure_translate_add[of cbox a ?c  $\cap$  {x. ?mn  $\cdot$  x  $\leq$  a $ m} cbox a
?c  $\cap$  {x. ?mn  $\cdot$  x  $\geq$  b $ m}
( $\chi$  i. if i = m then a $ m - b $ m else 0) cbox a ( $\chi$  i. if i = m then a $ m + b $ n
else b $ i)])
show (cbox a ?c  $\cap$  {x. ?mn  $\cdot$  x  $\leq$  a $ m})  $\in$  lmeasurable
cbox a ?c  $\cap$  {x. ?mn  $\cdot$  x  $\geq$  b $ m}  $\in$  lmeasurable
by (auto simp: convex_Int convex_halfspace_le convex_halfspace_ge bounded_Int
measurable_convex)
have  $\bigwedge x. \llbracket x$  $ n + a $ m  $\leq$  x $ m  $\rrbracket$ 
 $\implies x \in (+) (\chi$  i. if i = m then a $ m - b $ m else 0) ' {x. x $ n + b $
m  $\leq$  x $ m}
using <m  $\neq$  n>
by (rule_tac x=x - ( $\chi$  i. if i = m then a $ m - b $ m else 0) in image_eqI)
(simp_all add: mem_box_cart)
then have imeq: (+) ?d ' {x. b $ m  $\leq$  ?mn  $\cdot$  x} = {x. a $ m  $\leq$  ?mn  $\cdot$  x}
using <m  $\neq$  n> by (auto simp: mem_box_cart inner_axis' algebra_simps)
have  $\bigwedge x. \llbracket 0 \leq a$  $ n; x $ n + a $ m  $\leq$  x $ m  $\rrbracket$ 
 $\forall i. i \neq m \longrightarrow a$  $ i  $\leq$  x $ i  $\wedge$  x $ i  $\leq$  b $ i  $\rrbracket$ 
 $\implies a$  $ m  $\leq$  x $ m
using <m  $\neq$  n> by force
then have (+) ?d ' (cbox a ?c  $\cap$  {x. b $ m  $\leq$  ?mn  $\cdot$  x})
= cbox a ( $\chi$  i. if i = m then a $ m + b $ n else b $ i)  $\cap$  {x. a $ m  $\leq$ 
?mn  $\cdot$  x}
using an ab_ne
apply (simp add: cbox_translation [symmetric] translation_Int interval_ne_empty_cart
imeq)
apply (auto simp: mem_box_cart inner_axis' algebra_simps if_distrib
all_if_distrib)
by (metis (full_types) add_mono mult_2_right)
then show cbox a ?c  $\cap$  {x. ?mn  $\cdot$  x  $\leq$  a $ m}  $\cup$ 
(+) ?d ' (cbox a ?c  $\cap$  {x. b $ m  $\leq$  ?mn  $\cdot$  x}) =
cbox a ( $\chi$  i. if i = m then a $ m + b $ n else b $ i) (is ?lhs = ?rhs)
using an <m  $\neq$  n>
apply (auto simp: mem_box_cart inner_axis' algebra_simps if_distrib
all_if_distrib, force)
apply (drule_tac x=n in spec)+
by (meson ab_ne add_mono_thms_linordered_semiring(3) dual_order.trans
interval_ne_empty_cart(1))
have negligible{x. ?mn  $\cdot$  x = a $ m}
by (metis <m  $\neq$  n> axis_index_axis_eq_iff_diff_eq_0 negligible_hyperplane)
moreover have (cbox a ?c  $\cap$  {x. ?mn  $\cdot$  x  $\leq$  a $ m}  $\cap$ 
(+) ?d ' (cbox a ?c  $\cap$  {x. b $ m  $\leq$  ?mn  $\cdot$  x}))  $\subseteq$  {x.

```

```

?mn • x = a$m}
  using ⟨m ≠ n⟩ antisym_conv by (fastforce simp: algebra_simps mem_box_cart
inner_axis')
  ultimately show negligible (cbox a ?c ∩ {x. ?mn • x ≤ a $ m} ∩
    (+) ?d ' (cbox a ?c ∩ {x. b $ m ≤ ?mn • x}))
    by (rule negligible_subset)
qed
have ac_ne: cbox a ?c ≠ {}
  using ab_ne an
  by (clarsimp simp: interval_eq_empty_cart) (meson add_less_same_cancel1
le_less_linear less_le_trans)
have ax_ne: cbox a (χ i. if i = m then a $ m + b $ n else b $ i) ≠ {}
  using ab_ne an
  by (clarsimp simp: interval_eq_empty_cart) (meson add_less_same_cancel1
le_less_linear less_le_trans)
have eq3: measure lebesgue (cbox a ?c) = measure lebesgue (cbox a (χ i. if i =
m then a$m + b$n else b$i)) + measure lebesgue (cbox a b)
  by (simp add: content_cbox_if_cart ab_ne ac_ne ax_ne algebra_simps prod.delta_remove
if_distrib [of λu. u - z for z] prod.remove)
show ?Q
  using eq1 eq2 eq3
  by (simp add: algebra_simps)
qed

```

proposition

```

fixes S :: (real^n) set
assumes S ∈ lmeasurable
shows measurable_stretch: ((λx. χ k. m k * x$k) ' S) ∈ lmeasurable (is ?f ' S
∈ _)
  and measure_stretch: measure lebesgue ((λx. χ k. m k * x$k) ' S) = |prod m
UNIV| * measure lebesgue S
  (is ?MEQ)
proof -
  have (?f ' S) ∈ lmeasurable ∧ ?MEQ
  proof (cases ∀ k. m k ≠ 0)
  case True
    have m0: 0 < |prod m UNIV|
    using True by simp
    have (indicat_real (?f ' S) has_integral |prod m UNIV| * measure lebesgue S)
    UNIV
    proof (clarsimp simp add: has_integral_alt [where i=UNIV])
      fix e :: real
      assume e > 0
      have (indicat_real S has_integral (measure lebesgue S)) UNIV
      using assms lmeasurable_iff_has_integral by blast
      then obtain B where B>0
      and B: ⋀ a b. ball 0 B ⊆ cbox a b ⟹
        ∃ z. (indicat_real S has_integral z) (cbox a b) ∧

```

```

      |z - measure lebesgue S| < e / |prod m UNIV|
    by (simp add: has_integral_alt [where i=UNIV]) (metis (full_types)
divide_pos_pos m0 m0 ‹e > 0›)
  show ∃ B > 0. ∀ a b. ball 0 B ⊆ cbox a b ⟶
    (∃ z. (indicat_real (?f ' S) has_integral z) (cbox a b) ∧
      |z - |prod m UNIV| * measure lebesgue S| < e)
  proof (intro exI conjI allI)
    let ?C = Max (range (λk. |m k|)) * B
    show ?C > 0
      using True ‹B > 0› by (simp add: Max_gr_iff)
    show ball 0 ?C ⊆ cbox u v ⟶
      (∃ z. (indicat_real (?f ' S) has_integral z) (cbox u v) ∧
        |z - |prod m UNIV| * measure lebesgue S| < e) for u v
    proof
      assume uv: ball 0 ?C ⊆ cbox u v
      with ‹?C > 0› have cbox_ne: cbox u v ≠ {}
        using centre_in_ball by blast
      let ?α = λk. u $ k / m k
      let ?β = λk. v $ k / m k
      have invm0: ∧k. inverse (m k) ≠ 0
        using True by auto
      have ball 0 B ⊆ (λx. χ k. x $ k / m k) ' ball 0 ?C
      proof clarsimp
        fix x :: real
        assume x: norm x < B
        have [simp]: |Max (range (λk. |m k|))| = Max (range (λk. |m k|))
          by (meson Max_ge abs_ge_zero abs_of_nonneg finite finite_imageI
order_trans rangeI)
        have norm (χ k. m k * x $ k) ≤ norm (Max (range (λk. |m k|)) *R x)
          by (rule norm_le_componentwise_cart) (auto simp: abs_mult intro:
mult_right_mono)
        also have ... < ?C
          using x ‹0 < (MAX k. |m k|) * B› ‹0 < B› zero_less_mult_pos2 by
fastforce
        finally have norm (χ k. m k * x $ k) < ?C .
        then show x ∈ (λx. χ k. x $ k / m k) ' ball 0 ?C
          using stretch_Galois [of inverse ∘ m] True by (auto simp: image_iff
field_simps)
      qed
      then have Bsub: ball 0 B ⊆ cbox (χ k. min (?α k) (?β k)) (χ k. max (?α
k) (?β k))
        using cbox_ne uv image_stretch_interval_cart [of inverse ∘ m u v,
symmetric]
        by (force simp: field_simps)
      obtain z where zint: (indicat_real S has_integral z) (cbox (χ k. min (?α
k) (?β k)) (χ k. max (?α k) (?β k)))
        and zless: |z - measure lebesgue S| < e / |prod m UNIV|
        using B [OF Bsub] by blast
      have ind: indicat_real (?f ' S) = (λx. indicator S (χ k. x $ k / m k))

```

```

    using True stretch_Galois [of m] by (force simp: indicator_def)
  show  $\exists z. (\text{indicat\_real } (?f \text{ ' } S) \text{ has\_integral } z) (cbox \text{ } u \text{ } v) \wedge$ 
     $|z - |\text{prod } m \text{ } UNIV| * \text{measure lebesgue } S| < e$ 
  proof (simp add: ind, intro conjI exI)
    have  $((\lambda x. \text{indicat\_real } S (\chi \text{ } k. x \text{ } \$ \text{ } k / m \text{ } k)) \text{ has\_integral } z *_R |\text{prod } m$ 
       $UNIV|)$ 
       $((\lambda x. \chi \text{ } k. x \text{ } \$ \text{ } k * m \text{ } k) \text{ ' } cbox (\chi \text{ } k. \min (? \alpha \text{ } k) (? \beta \text{ } k)) (\chi \text{ } k. \max$ 
       $(? \alpha \text{ } k) (? \beta \text{ } k)))$ 
      using True has_integral_stretch_cart [OF zint, of inverse  $\circ$  m]
      by (simp add: field_simps prod_dividef)
    moreover have  $((\lambda x. \chi \text{ } k. x \text{ } \$ \text{ } k * m \text{ } k) \text{ ' } cbox (\chi \text{ } k. \min (? \alpha \text{ } k) (? \beta \text{ } k))$ 
       $(\chi \text{ } k. \max (? \alpha \text{ } k) (? \beta \text{ } k))) = cbox \text{ } u \text{ } v$ 
      using True image_stretch_interval_cart [of inverse  $\circ$  m u v, symmetric]
      image_stretch_interval_cart [of  $\lambda k. 1 \text{ } u \text{ } v$ , symmetric]  $\langle cbox \text{ } u \text{ } v \neq$ 
       $\{\} \rangle$ 
      by (simp add: field_simps image_comp o_def)
    ultimately show  $((\lambda x. \text{indicat\_real } S (\chi \text{ } k. x \text{ } \$ \text{ } k / m \text{ } k)) \text{ has\_integral}$ 
       $z *_R |\text{prod } m \text{ } UNIV|) (cbox \text{ } u \text{ } v)$ 
      by simp
    have  $|z *_R |\text{prod } m \text{ } UNIV| - |\text{prod } m \text{ } UNIV| * \text{measure lebesgue } S|$ 
       $= |\text{prod } m \text{ } UNIV| * |z - \text{measure lebesgue } S|$ 
      by (metis (no_types, opaque_lifting) abs_abs abs_scaleR mult.commute
        real_scaleR_def right_diff_distrib)
    also have  $\dots < e$ 
      using zless True by (simp add: field_simps)
    finally show  $|z *_R |\text{prod } m \text{ } UNIV| - |\text{prod } m \text{ } UNIV| * \text{measure lebesgue}$ 
       $S| < e .$ 
    qed
  qed
  qed
  qed
  then show ?thesis
    by (auto simp: has_integral_integrable integral_unique lmeasure_integral_UNIV
      measurable_integrable)
  next
    case False
    then obtain k where  $m \text{ } k = 0$  and  $\text{prm: prod } m \text{ } UNIV = 0$ 
      by auto
    have  $\text{nfS: negligible } (?f \text{ ' } S)$ 
      by (rule negligible_subset [OF negligible_standard_hyperplane_cart]) (use  $\langle m$ 
       $k = 0 \rangle$  in auto)
    then have  $(?f \text{ ' } S) \in \text{lmeasurable}$ 
      by (simp add: negligible_iff_measure)
    with  $\text{nfS}$  show ?thesis
      by (simp add: prm negligible_iff_measure0)
  qed
  then show  $(?f \text{ ' } S) \in \text{lmeasurable ?MEQ}$ 
    by metis+
  qed

```

proposition

```

fixes  $f :: \text{real}^n :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n :: \_$ 
assumes  $\text{linear } f \text{ } S \in \text{lmeasurable}$ 
shows  $\text{measurable\_linear\_image}: (f \text{ } S) \in \text{lmeasurable}$ 
and  $\text{measure\_linear\_image}: \text{measure lebesgue } (f \text{ } S) = |\det (\text{matrix } f)| * \text{measure lebesgue } S$ 
(is ?Q f S)
proof –
  have  $\forall S \in \text{lmeasurable}. (f \text{ } S) \in \text{lmeasurable} \wedge ?Q f S$ 
proof (rule induct_linear_elementary [OF  $\langle \text{linear } f \rangle$ ]; intro ballI)
  fix  $f \text{ } g$  and  $S :: (\text{real}, 'n) \text{ vec set}$ 
  assume  $\text{linear } f$  and  $\text{linear } g$ 
  and  $f$  [rule_format]:  $\forall S \in \text{lmeasurable}. f \text{ } S \in \text{lmeasurable} \wedge ?Q f S$ 
  and  $g$  [rule_format]:  $\forall S \in \text{lmeasurable}. g \text{ } S \in \text{lmeasurable} \wedge ?Q g S$ 
  and  $S: S \in \text{lmeasurable}$ 
  then have  $gS: g \text{ } S \in \text{lmeasurable}$ 
  by blast
  show  $(f \circ g) \text{ } S \in \text{lmeasurable} \wedge ?Q (f \circ g) S$ 
  using  $f$  [OF  $gS$ ]  $g$  [OF  $S$ ] matrix_compose [OF  $\langle \text{linear } g \rangle \langle \text{linear } f \rangle$ ]
  by (simp add: o_def image_comp abs_mult det_mul)
next
  fix  $f :: \text{real}^n :: \_ \Rightarrow \text{real}^n :: \_$  and  $i$  and  $S :: (\text{real}^n :: \_) \text{ set}$ 
  assume  $\text{linear } f$  and  $0: \bigwedge x. f \text{ } x \text{ } i = 0$  and  $S \in \text{lmeasurable}$ 
  then have  $\neg \text{inj } f$ 
  by (metis (full_types) linear_injective_imp_surjective one_neq_zero surjE vec_component)
  have  $\text{det}f: \det (\text{matrix } f) = 0$ 
  using  $\langle \neg \text{inj } f \rangle$  det_nz_iff_inj [OF  $\langle \text{linear } f \rangle$ ] by blast
  show  $f \text{ } S \in \text{lmeasurable} \wedge ?Q f S$ 
  proof
    show  $f \text{ } S \in \text{lmeasurable}$ 
    using lmeasurable_iff_indicator_has_integral  $\langle \text{linear } f \rangle \langle \neg \text{inj } f \rangle$  negligible_UNIV negligible_linear_singular_image by blast
    have  $\text{measure lebesgue } (f \text{ } S) = 0$ 
    by (meson  $\langle \neg \text{inj } f \rangle \langle \text{linear } f \rangle$  negligible_imp_measure0 negligible_linear_singular_image)
    also have  $\dots = |\det (\text{matrix } f)| * \text{measure lebesgue } S$ 
    by (simp add: detf)
    finally show  $?Q f S$  .
  qed
next
  fix  $c$  and  $S :: (\text{real}^n :: \_) \text{ set}$ 
  assume  $S \in \text{lmeasurable}$ 
  show  $(\lambda a. \chi i. c \text{ } i * a \text{ } i) \text{ } S \in \text{lmeasurable} \wedge ?Q (\lambda a. \chi i. c \text{ } i * a \text{ } i) S$ 
  proof
    show  $(\lambda a. \chi i. c \text{ } i * a \text{ } i) \text{ } S \in \text{lmeasurable}$ 
    by (simp add:  $\langle S \in \text{lmeasurable} \rangle$  measurable_stretch)
    show  $?Q (\lambda a. \chi i. c \text{ } i * a \text{ } i) S$ 
    by (simp add: measure_stretch [OF  $\langle S \in \text{lmeasurable} \rangle$ , of  $c$ ] axis_def)
  
```

```

matrix_def det_diagonal)
qed
next
fix m :: 'n and n :: 'n and S :: (real, 'n) vec set
assume m ≠ n and S ∈ lmeasurable
let ?h = λv::(real, 'n) vec. χ i. v $ Transposition.transpose m n i
have lin: linear ?h
  by (rule linearI) (simp_all add: plus_vec_def scaleR_vec_def)
have meq: measure lebesgue ((λv::(real, 'n) vec. χ i. v $ Transposition.transpose
m n i) ' cbox a b)
  = measure lebesgue (cbox a b) for a b
proof (cases cbox a b = {})
case True then show ?thesis
  by simp
next
case False
then have him: ?h ' (cbox a b) ≠ {}
  by blast
have eq: ?h ' (cbox a b) = cbox (?h a) (?h b)
  by (auto simp: image_iff lambda_swap_Galois mem_box_cart) (metis
transpose_involutory)+
show ?thesis
  using him prod.permute [OF permutes_swap_id, where S=UNIV and
g=λi. (b - a)$i, symmetric]
  by (simp add: eq content_cbox_cart False)
qed
have (χ i j. if Transposition.transpose m n i = j then 1 else 0) = (χ i j. if j =
Transposition.transpose m n i then 1 else (0::real))
  by (auto intro!: Cart_lambda_cong)
then have matrix ?h = transpose(χ i j. mat 1 $ i $ Transposition.transpose m
n j)
  by (auto simp: matrix_eq transpose_def axis_def mat_def matrix_def)
then have 1: |det (matrix ?h)| = 1
  by (simp add: det_permute_columns permutes_swap_id sign_swap_id abs_mult)
show ?h ' S ∈ lmeasurable ∧ ?Q ?h S
proof
show ?h ' S ∈ lmeasurable ?Q ?h S
  using measure_linear_sufficient [OF lin ⟨S ∈ lmeasurable⟩] meq 1 by force+
qed
next
fix m n :: 'n and S :: (real, 'n) vec set
assume m ≠ n and S ∈ lmeasurable
let ?h = λv::(real, 'n) vec. χ i. if i = m then v $ m + v $ n else v $ i
have lin: linear ?h
  by (rule linearI) (auto simp: algebra_simps plus_vec_def scaleR_vec_def
vec_eq_iff)
consider m < n | n < m
using ⟨m ≠ n⟩ less_linear by blast
then have 1: det(matrix ?h) = 1

```



```

proof cases
  assume  $m < n$ 
  have *:  $\text{matrix } ?h \ \$ \ i \ \$ \ j = (0::\text{real})$  if  $j < i$  for  $i \ j :: 'n$ 
  proof –
    have  $\text{axis } j \ 1 = (\chi \ n. \text{if } n = j \text{ then } 1 \text{ else } (0::\text{real}))$ 
    using axis_def by blast
    then have  $(\chi \ p \ q. \text{if } p = m \text{ then } \text{axis } q \ 1 \ \$ \ m + \text{axis } q \ 1 \ \$ \ n \text{ else } \text{axis } q \ 1$ 
 $\$ \ p) \ \$ \ i \ \$ \ j = (0::\text{real})$ 
    using  $\langle j < i \rangle$  axis_def  $\langle m < n \rangle$  by auto
    with  $\langle m < n \rangle$  show ?thesis
    by (auto simp: matrix_def axis_def cong: if_cong)
  qed
  show ?thesis
  using  $\langle m \neq n \rangle$  by (subst det_upperdiagonal [OF *]) (auto simp: matrix_def
axis_def cong: if_cong)
next
  assume  $n < m$ 
  have *:  $\text{matrix } ?h \ \$ \ i \ \$ \ j = (0::\text{real})$  if  $j > i$  for  $i \ j :: 'n$ 
  proof –
    have  $\text{axis } j \ 1 = (\chi \ n. \text{if } n = j \text{ then } 1 \text{ else } (0::\text{real}))$ 
    using axis_def by blast
    then have  $(\chi \ p \ q. \text{if } p = m \text{ then } \text{axis } q \ 1 \ \$ \ m + \text{axis } q \ 1 \ \$ \ n \text{ else } \text{axis } q \ 1$ 
 $\$ \ p) \ \$ \ i \ \$ \ j = (0::\text{real})$ 
    using  $\langle j > i \rangle$  axis_def  $\langle m > n \rangle$  by auto
    with  $\langle m > n \rangle$  show ?thesis
    by (auto simp: matrix_def axis_def cong: if_cong)
  qed
  show ?thesis
  using  $\langle m \neq n \rangle$ 
  by (subst det_lowerdiagonal [OF *]) (auto simp: matrix_def axis_def cong:
if_cong)
qed
have meq: measure lebesgue ( $?h \ ' (cbox \ a \ b)) = \text{measure lebesgue } (cbox \ a \ b)$  for
 $a \ b$ 
proof (cases cbox a b = {})
  case True then show ?thesis by simp
next
  case False
  then have ne: (+) ( $\chi \ i. \text{if } i = n \text{ then } - \ a \ \$ \ n \text{ else } 0$ ) ' cbox a b  $\neq \{\}$ 
  by auto
  let  $?v = \chi \ i. \text{if } i = n \text{ then } - \ a \ \$ \ n \text{ else } 0$ 
  have  $?h \ ' \ cbox \ a \ b$ 
     $= (+) (\chi \ i. \text{if } i = m \vee i = n \text{ then } a \ \$ \ n \text{ else } 0) \ ' \ ?h \ ' (+) \ ?v \ ' (cbox \ a \ b)$ 
  using  $\langle m \neq n \rangle$  unfolding image_comp o_def by (force simp: vec_eq_iff)
  then have  $\text{measure lebesgue } ((\lambda v. \chi \ i. \text{if } i = m \text{ then } v \ \$ \ m + v \ \$ \ n \text{ else } v \ \$ \ i) \ ' \$ 
 $(+) \ ?v \ ' \ cbox \ a \ b)$ 
  by (rule ssubst) (rule measure_translation)
  also have  $\dots = \text{measure lebesgue } ((\lambda v. \chi \ i. \text{if } i = m \text{ then } v \ \$ \ m + v \ \$ \ n \text{ else } \$ 

```

3222

```

v $ i) ' cbox (?v + a) (?v + b))
  by (metis (no_types, lifting) cbox_translation)
  also have ... = measure lebesgue ((+) (χ i. if i = n then - a $ n else 0) '
cbox a b)
  apply (subst measure_shear_interval)
  using ⟨m ≠ n⟩ ne apply auto
  apply (simp add: cbox_translation)
  by (metis cbox_borel cbox_translation measure_completion_sets_lborel)
  also have ... = measure lebesgue (cbox a b)
  by (rule measure_translation)
  finally show ?thesis .
qed
show ?h ' S ∈ lmeasurable ∧ ?Q ?h S
  using measure_linear_sufficient [OF lin ⟨S ∈ lmeasurable⟩] meq 1 by force
qed
with assms show (f ' S) ∈ lmeasurable ?Q f S
  by metis+
qed

```

lemma

```

fixes f :: real^'n::{finite,wellorder} ⇒ real^'n::_
assumes f: orthogonal_transformation f and S: S ∈ lmeasurable
shows measurable_orthogonal_image: f ' S ∈ lmeasurable
  and measure_orthogonal_image: measure lebesgue (f ' S) = measure lebesgue
S
proof -
  have linear f
  by (simp add: f orthogonal_transformation_linear)
  then show f ' S ∈ lmeasurable
  by (metis S measurable_linear_image)
  show measure lebesgue (f ' S) = measure lebesgue S
  by (simp add: measure_linear_image ⟨linear f⟩ S f)
qed

```

proposition *measure_semicontinuous_with_hausdist_explicit:*

assumes *bounded S* **and** *neg: negligible(frontier S)* **and** *e > 0*
obtains *d* **where** *d > 0*

$$\bigwedge T. \llbracket T \in \text{lmeasurable}; \bigwedge y. y \in T \implies \exists x. x \in S \wedge \text{dist } x \ y < d \rrbracket \\
\implies \text{measure lebesgue } T < \text{measure lebesgue } S + e$$

proof (cases *S* = {}) **proof**

case *True*

with *that* ⟨*e* > 0⟩ **show** ?thesis **by** force

next

case *False*

then have *frS: frontier S ≠ {}*

using ⟨*bounded S*⟩ *frontier_eq_empty not_bounded_UNIV* **by** blast

have *S ∈ lmeasurable*

by (simp add: ⟨*bounded S*⟩ *measurable_Jordan neg*)

```

have null: (frontier S) ∈ null_sets lebesgue
  by (metis neg negligible_iff_null_sets)
have frontier S ∈ lmeasurable and mS0: measure lebesgue (frontier S) = 0
  using neg negligible_imp_measurable negligible_iff_measure by blast+
with ⟨e > 0⟩ sets_lebesgue_outer_open
obtain U where open U
  and U: frontier S ⊆ U U − frontier S ∈ lmeasurable emeasure lebesgue (U −
frontier S) < e
  by (metis fmeasurableD)
with null have U ∈ lmeasurable
  by (metis borel_open measurable_Diff_null_set sets_completionI_sets_sets_lborel)
have measure lebesgue (U − frontier S) = measure lebesgue U
  using mS0 by (simp add: ⟨U ∈ lmeasurable⟩ fmeasurableD measure_Diff_null_set
null)
with U have mU: measure lebesgue U < e
  by (simp add: emeasure_eq_measure2 ennreal_less_iff)
show ?thesis
proof
  have U ≠ UNIV
    using ⟨U ∈ lmeasurable⟩ by auto
  then have − U ≠ {}
    by blast
  with ⟨open U⟩ ⟨frontier S ⊆ U⟩ show setdist (frontier S) (− U) > 0
    by (auto simp: ⟨bounded S⟩ open_closed compact_frontier_bounded set-
dist_gt_0_compact_closed frS)
  fix T
  assume T ∈ lmeasurable
  and T: ⋀t. t ∈ T ⟹ ∃y. y ∈ S ∧ dist y t < setdist (frontier S) (− U)
  then have measure lebesgue T − measure lebesgue S ≤ measure lebesgue (T
− S)
    by (simp add: ⟨S ∈ lmeasurable⟩ measure_diff_le_measure_setdiff)
  also have ... ≤ measure lebesgue U
  proof −
    have T − S ⊆ U
  proof clarify
    fix x
    assume x ∈ T and x ∉ S
    then obtain y where y ∈ S and y: dist y x < setdist (frontier S) (− U)
      using T by blast
    have closed_segment x y ∩ frontier S ≠ {}
      using connected_Int_frontier ⟨x ∉ S⟩ ⟨y ∈ S⟩ by blast
    then obtain z where z: z ∈ closed_segment x y z ∈ frontier S
      by auto
    with y have dist z x < setdist(frontier S) (− U)
      by (auto simp: dist_commute dest!: dist_in_closed_segment)
    with z have False if x ∈ − U
      using setdist_le_dist [OF ⟨z ∈ frontier S⟩ that] by auto
    then show x ∈ U
      by blast
  end
  end

```

```

    qed
    then show ?thesis
      by (simp add: ⟨S ∈ lmeasurable⟩ ⟨T ∈ lmeasurable⟩ ⟨U ∈ lmeasurable⟩
fmeasurableD measure_mono_fmeasurable sets.Diff)
    qed
    finally have measure lebesgue T - measure lebesgue S ≤ measure lebesgue U .
    with mU show measure lebesgue T < measure lebesgue S + e
      by linarith
    qed
  qed

```

proposition

```

fixes f :: real^'n::{finite,wellorder} ⇒ real^'n::_
assumes S: S ∈ lmeasurable
and deriv: ∧x. x ∈ S ⇒ (f has_derivative f' x) (at x within S)
and int: (λx. |det (matrix (f' x))|) integrable_on S
and bounded: ∧x. x ∈ S ⇒ |det (matrix (f' x))| ≤ B
shows measurable_bounded_differentiable_image:
  f ' S ∈ lmeasurable
and measure_bounded_differentiable_image:
  measure lebesgue (f ' S) ≤ B * measure lebesgue S (is ?M)

```

proof –

```

have f ' S ∈ lmeasurable ∧ measure lebesgue (f ' S) ≤ B * measure lebesgue S
proof (cases B < 0)
  case True
    then have S = {}
    by (meson abs_ge_zero bounded empty_iff equalityI less_le_trans linorder_not_less
subsetI)
    then show ?thesis
      by auto
  next
  case False
    then have B ≥ 0
      by arith
    let ?μ = measure lebesgue
    have f_diff: f differentiable_on S
      using deriv by (auto simp: differentiable_on_def differentiable_def)
    have eps: f ' S ∈ lmeasurable ?μ (f ' S) ≤ (B+e) * ?μ S (is ?ME)
      if e > 0 for e
    proof –
      have eps_d: f ' S ∈ lmeasurable ?μ (f ' S) ≤ (B+e) * (?μ S + d) (is ?MD)
        if d > 0 for d
      proof –
        obtain T where T: open T S ⊆ T and TS: (T-S) ∈ lmeasurable and
emeasure lebesgue (T-S) < ennreal d
        using S ⟨d > 0⟩ sets_lebesgue_outer_open by blast
        then have ?μ (T-S) < d
          by (metis emeasure_eq_measure2 ennreal_leI not_less)
        with S T TS have T ∈ lmeasurable and Tless: ?μ T < ?μ S + d

```

```

    by (auto simp: measurable_measure_Diff dest!: fmeasurable_Diff_D)
  have  $\exists r. 0 < r \wedge r < d \wedge \text{ball } x \ r \subseteq T \wedge f' (S \cap \text{ball } x \ r) \in \text{lmeasurable} \wedge$ 
 $\quad ?\mu (f' (S \cap \text{ball } x \ r)) \leq (B + e) * ?\mu (\text{ball } x \ r)$ 
    if  $x \in S \wedge d > 0$  for  $x \ d$ 
  proof -
    have  $\text{lin}: \text{linear } (f' x)$ 
    and  $\text{lim0}: ((\lambda y. (f y - (f x + f' x (y - x))) /_R \text{norm}(y - x)) \longrightarrow 0)$ 
    (at  $x$  within  $S$ )
    using  $\text{deriv } \langle x \in S \rangle$  by (auto simp: has_derivative_within bounded_linear.linear
    field_simps)
    have  $\text{bo}: \text{bounded } (f' x \text{ ' ball } 0 \ 1)$ 
    by (simp add: bounded_linear_image linear_linear lin)
    have  $\text{neg}: \text{negligible } (\text{frontier } (f' x \text{ ' ball } 0 \ 1))$ 
    using  $\text{deriv has\_derivative\_linear } \langle x \in S \rangle$ 
    by (auto intro!: negligible_convex_frontier [OF convex_linear_image])
    let  $?\text{unit\_vol} = \text{content } (\text{ball } (0 :: \text{real} ^ n :: \{\text{finite}, \text{wellorder}\}) \ 1)$ 
    have  $0: 0 < e * ?\text{unit\_vol}$ 
    using  $\langle e > 0 \rangle$  by (simp add: content_ball_pos)
    obtain  $k$  where  $k > 0$  and  $k:$ 
 $\quad \bigwedge U. [\![U \in \text{lmeasurable}; \bigwedge y. y \in U \implies \exists z. z \in f' x \text{ ' ball } 0 \ 1 \wedge \text{dist}$ 
 $\quad z \ y < k]\!]$ 
 $\quad \implies ?\mu U < ?\mu (f' x \text{ ' ball } 0 \ 1) + e * ?\text{unit\_vol}$ 
    using  $\text{measure\_semicontinuous\_with\_hausdist\_explicit } [OF \ \text{bo} \ \text{neg} \ 0]$ 
  by blast
  obtain  $l$  where  $l > 0$  and  $l: \text{ball } x \ l \subseteq T$ 
    using  $\langle x \in S \rangle \langle \text{open } T \rangle \langle S \subseteq T \rangle \text{openE}$  by blast
  obtain  $\zeta$  where  $0 < \zeta$ 
    and  $\zeta: \bigwedge y. [\![y \in S; y \neq x; \text{dist } y \ x < \zeta]\!]$ 
 $\quad \implies \text{norm } (f y - (f x + f' x (y - x))) / \text{norm } (y - x) < k$ 
    using  $\text{lim0 } \langle k > 0 \rangle$  by (simp add: Lim_within) (auto simp add:
    field_simps)
  define  $r$  where  $r \equiv \min (\min l (\zeta/2)) (\min 1 (d/2))$ 
  show ?thesis
  proof (intro exI conjI)
    show  $r > 0 \wedge r < d$ 
    using  $\langle l > 0 \rangle \langle \zeta > 0 \rangle \langle d > 0 \rangle$  by (auto simp: r_def)
    have  $r \leq l$ 
    by (auto simp: r_def)
    with  $l$  show  $\text{ball } x \ r \subseteq T$ 
    by auto
    have  $\text{ex\_lessK}: \exists x' \in \text{ball } 0 \ 1. \text{dist } (f' x \ x') ((f y - f x) /_R r) < k$ 
    if  $y \in S$  and  $\text{dist } x \ y < r$  for  $y$ 
    proof (cases  $y = x$ )
    case True
    with  $\text{lin linear\_0 } \langle k > 0 \rangle$  that show ?thesis
    by (rule_tac  $x=0$  in  $\text{bexI}$ ) (auto simp: linear_0)
    next
    case False
    then show ?thesis

```

```

proof (rule_tac x=(y - x) /R r in beXI)
  have f' x ((y - x) /R r) = f' x (y - x) /R r
    by (simp add: lin linear_scale)
  then have dist (f' x ((y - x) /R r)) ((f y - f x) /R r) = norm (f'
x (y - x) /R r - (f y - f x) /R r)
    by (simp add: dist_norm)
  also have ... = norm (f' x (y - x) - (f y - f x)) / r
    using ⟨r > 0⟩ by (simp add: divide_simps scale_right_diff_distrib
[symmetric])
  also have ... ≤ norm (f y - (f x + f' x (y - x))) / norm (y - x)
    using that ⟨r > 0⟩ False by (simp add: field_split_simps dist_norm
norm_minus_commute mult_right_mono)
  also have ... < k
    using that ⟨0 < ζ⟩ by (simp add: dist_commute r_def ζ [OF ⟨y
∈ S⟩ False])
  finally show dist (f' x ((y - x) /R r)) ((f y - f x) /R r) < k .
  show (y - x) /R r ∈ ball 0 1
    using that ⟨r > 0⟩ by (simp add: dist_norm divide_simps
norm_minus_commute)
qed
qed
let ?rfs = (λx. x /R r) ' (+) (- f x) ' f ' (S ∩ ball x r)
have rfs_mble: ?rfs ∈ lmeasurable
proof (rule bounded_set_imp_lmeasurable)
  have f differentiable_on S ∩ ball x r
    using f_diff by (auto simp: fmeasurableD differentiable_on_subset)
  with S show ?rfs ∈ sets lebesgue
    by (auto simp: sets.Int intro!: lebesgue_sets_translation differen-
tible_image_in_sets_lebesgue)
  let ?B = (λ(x, y). x + y) ' (f' x ' ball 0 1 × ball 0 k)
  have bounded ?B
    by (simp add: bounded_plus [OF bo])
  moreover have ?rfs ⊆ ?B
    apply (auto simp: dist_norm image_iff dest!: ex_lessK)
    by (metis (no_types, opaque_lifting) add.commute diff_add_cancel
dist_0_norm dist_commute dist_norm mem_ball)
  ultimately show bounded (?rfs)
    by (rule bounded_subset)
qed
then have (λx. r *R x) ' ?rfs ∈ lmeasurable
  by (simp add: measurable_linear_image)
with ⟨r > 0⟩ have (+) (- f x) ' f ' (S ∩ ball x r) ∈ lmeasurable
  by (simp add: image_comp o_def)
then have (+) (f x) ' (+) (- f x) ' f ' (S ∩ ball x r) ∈ lmeasurable
  using measurable_translation by blast
then show fsb: f ' (S ∩ ball x r) ∈ lmeasurable
  by (simp add: image_comp o_def)
have ?μ (f ' (S ∩ ball x r)) = ?μ (?rfs) * r ^ CARD('n)
  using ⟨r > 0⟩ fsb

```

```

    by (simp add: measure_linear_image measure_translation_subtract
measurable_translation_subtract field_simps cong: image_cong_simp)
    also have ... ≤ (|det (matrix (f' x))| * ?unit_vol + e * ?unit_vol) * r
^ CARD('n)
  proof -
    have ?μ (?rfs) < ?μ (f' x ' ball 0 1) + e * ?unit_vol
      using rfs_mble by (force intro: k dest!: ex_lessK)
    then have ?μ (?rfs) < |det (matrix (f' x))| * ?unit_vol + e * ?unit_vol
      by (simp add: lin_measure_linear_image [of f' x])
    with ‹r > 0› show ?thesis
      by auto
    qed
    also have ... ≤ (B + e) * ?μ (ball x r)
      using bounded [OF ‹x ∈ S›] ‹r > 0›
      by (simp add: algebra_simps content_ball_conv_unit_ball[of r]
content_ball_pos)
    finally show ?μ (f ' (S ∩ ball x r)) ≤ (B + e) * ?μ (ball x r) .
  qed
qed
then obtain r where
  r0d: ∧x d. [x ∈ S; d > 0] ⇒ 0 < r x d ∧ r x d < d
  and rT: ∧x d. [x ∈ S; d > 0] ⇒ ball x (r x d) ⊆ T
  and r: ∧x d. [x ∈ S; d > 0] ⇒
    (f ' (S ∩ ball x (r x d))) ∈ lmeasurable ∧
    ?μ (f ' (S ∩ ball x (r x d))) ≤ (B + e) * ?μ (ball x (r x d))
  by metis
obtain C where countable C and Csub: C ⊆ {(x,r x t) | x t. x ∈ S ∧ 0 <
t}
  and pwC: pairwise (λi j. disjnt (ball (fst i) (snd i)) (ball (fst j) (snd j)))
C
  and negC: negligible(S - (⋃ i ∈ C. ball (fst i) (snd i)))
  apply (rule Vitali_covering_theorem_balls [of S {(x,r x t) | x t. x ∈ S ∧
0 < t} fst snd])
  apply auto
  by (metis dist_eq_0_iff r0d)
let ?UB = (⋃ (x,s) ∈ C. ball x s)
have eq: f ' (S ∩ ?UB) = (⋃ (x,s) ∈ C. f ' (S ∩ ball x s))
  by auto
have mle: ?μ (⋃ (x,s) ∈ K. f ' (S ∩ ball x s)) ≤ (B + e) * (?μ S + d) (is
?l ≤ ?r)
  if K ⊆ C and finite K for K
proof -
  have gt0: b > 0 if (a, b) ∈ K for a b
    using Csub that ‹K ⊆ C› r0d by auto
  have inj: inj_on (λ(x, y). ball x y) K
    by (force simp: inj_on_def ball_eq_ball_iff dest: gt0)
  have disjnt: disjoint ((λ(x, y). ball x y) ' K)
    using pwC that
  apply (clarsimp simp: pairwise_def case_prod_unfold ball_eq_ball_iff)

```

```

    by (metis subsetD fst_conv snd_conv)
  have ?l ≤ (∑ i∈K. ?μ (case i of (x, s) ⇒ f ' (S ∩ ball x s)))
  proof (rule measure_UNION_le [OF <finite K>], clarify)
    fix x r
    assume (x,r) ∈ K
    then have x ∈ S
    using Csub <K ⊆ C> by auto
    show f ' (S ∩ ball x r) ∈ sets lebesgue
    by (meson Int_lower1 S differentiable_on_subset f_diff fmeasurableD
lmeasurable_ball order_refl sets.Int differentiable_image_in_sets lebesgue)
  qed
  also have ... ≤ (∑ (x,s) ∈ K. (B + e) * ?μ (ball x s))
  apply (rule sum_mono)
  using Csub r <K ⊆ C> by auto
  also have ... = (B + e) * (∑ (x,s) ∈ K. ?μ (ball x s))
  by (simp add: prod.case_distrib sum_distrib_left)
  also have ... = (B + e) * sum ?μ ((λ(x, y). ball x y) ' K)
  using <B ≥ 0> <e > 0> by (simp add: inj_sum.reindex prod.case_distrib)
  also have ... = (B + e) * ?μ (⋃ (x,s) ∈ K. ball x s)
  using <B ≥ 0> <e > 0> that
  by (subst measure_Union') (auto simp: disjnt measure_Union')
  also have ... ≤ (B + e) * ?μ T
  using <B ≥ 0> <e > 0> that apply simp
  apply (rule measure_mono_fmeasurable [OF _ _ <T ∈ lmeasurable>])
  using Csub r T by force+
  also have ... ≤ (B + e) * (?μ S + d)
  using <B ≥ 0> <e > 0> Tless by simp
  finally show ?thesis .
  qed
  have fSUB_mble: (f ' (S ∩ ?UB)) ∈ lmeasurable
  unfolding eq using Csub r False <e > 0> that
  by (auto simp: intro!: fmeasurable_UN_bound [OF <countable C> _ mle])
  have fSUB_meas: ?μ (f ' (S ∩ ?UB)) ≤ (B + e) * (?μ S + d) (is ?MUB)
  unfolding eq using Csub r False <e > 0> that
  by (auto simp: intro!: measure_UN_bound [OF <countable C> _ mle])
  have neg: negligible ((f ' (S ∩ ?UB)) - f ' S) ∪ (f ' S - f ' (S ∩ ?UB))
  proof (rule negligible_subset [OF negligible_differentiable_image_negligible
[OF order_refl negC, where f=f]])
    show f differentiable_on S - (⋃ i∈C. ball (fst i) (snd i))
    by (meson DiffE differentiable_on_subset subsetI f_diff)
  qed force
  show f ' S ∈ lmeasurable
  by (rule lmeasurable_negligible_symdiff [OF fSUB_mble neg])
  show ?MD
  using fSUB_meas measure_negligible_symdiff [OF fSUB_mble neg] by
simp
  qed
  show f ' S ∈ lmeasurable
  using eps_d [of 1] by simp

```



```

  show ?ME
  proof (rule field_le_epsilon)
    fix  $\delta :: \text{real}$ 
    assume  $0 < \delta$ 
    then show  $?_\mu (f \text{ ' } S) \leq (B + e) * ?_\mu S + \delta$ 
      using eps_d [of  $\delta / (B+e)$ ]  $\langle e > 0 \rangle \langle B \geq 0 \rangle$  by (auto simp: divide_simps
mult_ac)
    qed
  qed
  show ?thesis
  proof (cases  $?_\mu S = 0$ )
    case True
    with eps have  $?_\mu (f \text{ ' } S) = 0$ 
      by (metis mult_zero_right not_le zero_less_measure_iff)
    then show ?thesis
      using eps [of 1] by (simp add: True)
  next
    case False
    have  $?_\mu (f \text{ ' } S) \leq B * ?_\mu S$ 
    proof (rule field_le_epsilon)
      fix  $e :: \text{real}$ 
      assume  $e > 0$ 
      then show  $?_\mu (f \text{ ' } S) \leq B * ?_\mu S + e$ 
        using eps [of  $e / ?_\mu S$ ] False by (auto simp: algebra_simps zero_less_measure_iff)
    qed
    with eps [of 1] show ?thesis by auto
  qed
  qed
  then show  $f \text{ ' } S \in \text{lmeasurable } ?M$  by blast+
qed

lemma m_diff_image_weak:
  fixes  $f :: \text{real}^n :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n :: \_$ 
  assumes  $S: S \in \text{lmeasurable}$ 
    and deriv:  $\bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } S)$ 
    and int:  $(\lambda x. |\det (\text{matrix } (f' x))|) \text{ integrable\_on } S$ 
  shows  $f \text{ ' } S \in \text{lmeasurable} \wedge \text{measure lebesgue } (f \text{ ' } S) \leq \text{integral } S (\lambda x. |\det$ 
 $(\text{matrix } (f' x))|)$ 
  proof -
    let  $?_\mu = \text{measure lebesgue}$ 
    have aint_S:  $(\lambda x. |\det (\text{matrix } (f' x))|) \text{ absolutely\_integrable\_on } S$ 
      using int unfolding absolutely_integrable_on_def by auto
    define m where  $m \equiv \text{integral } S (\lambda x. |\det (\text{matrix } (f' x))|)$ 
    have *:  $f \text{ ' } S \in \text{lmeasurable } ?_\mu (f \text{ ' } S) \leq m + e * ?_\mu S$ 
      if  $e > 0$  for e
    proof -
      define T where  $T \equiv \lambda n. \{x \in S. n * e \leq |\det (\text{matrix } (f' x))| \wedge$ 
 $|\det (\text{matrix } (f' x))| < (\text{Suc } n) * e\}$ 
      have meas_t:  $T n \in \text{lmeasurable}$  for n

```

```

proof –
  have *:  $(\lambda x. |\det (\text{matrix } (f' x))|) \in \text{borel\_measurable } (\text{lebesgue\_on } S)$ 
    using aint_S by (simp add: S borel\_measurable\_restrict\_space\_iff fmeasurableD set\_integrable\_def)
  have [intro]:  $x \in \text{sets } (\text{lebesgue\_on } S) \implies x \in \text{sets lebesgue}$  for  $x$ 
    using S sets\_restrict\_space\_subset by blast
  have  $\{x \in S. \text{real } n * e \leq |\det (\text{matrix } (f' x))|\} \in \text{sets lebesgue}$ 
    using * by (auto simp: borel\_measurable\_iff\_halfspace\_ge space\_restrict\_space)
  then have 1:  $\{x \in S. \text{real } n * e \leq |\det (\text{matrix } (f' x))|\} \in \text{lmeasurable}$ 
    using S by (simp add: fmeasurableI2)
  have  $\{x \in S. |\det (\text{matrix } (f' x))| < (1 + \text{real } n) * e\} \in \text{sets lebesgue}$ 
    using * by (auto simp: borel\_measurable\_iff\_halfspace\_less space\_restrict\_space)
  then have 2:  $\{x \in S. |\det (\text{matrix } (f' x))| < (1 + \text{real } n) * e\} \in \text{lmeasurable}$ 
    using S by (simp add: fmeasurableI2)
  show ?thesis
    using fmeasurable.Int [OF 1 2] by (simp add: T_def Int_def cong: conj\_cong)
  qed
  have aint_T:  $\bigwedge k. (\lambda x. |\det (\text{matrix } (f' x))|) \text{ absolutely\_integrable\_on } T k$ 
    using set\_integrable\_subset [OF aint_S] meas_t T_def by blast
  have Seq:  $S = (\bigcup n. T n)$ 
    apply (auto simp: T_def)
    apply (rule_tac x= $\text{nat}(\text{floor}(\text{abs}(\det(\text{matrix}(f' x))) / e))$  in exI)
    using that apply auto
    using of_int_floor_le pos_le_divide_eq apply blast
    by (metis add.commute pos_divide_less_eq real_of_int_floor_add_one_gt)
  have meas_ft:  $f \restriction T n \in \text{lmeasurable}$  for  $n$ 
  proof (rule measurable_bounded_differentiable_image)
    show  $T n \in \text{lmeasurable}$ 
      by (simp add: meas_t)
  next
    fix  $x :: (\text{real}, 'n) \text{ vec}$ 
    assume  $x \in T n$ 
    show  $(f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } T n)$ 
      by (metis (no_types, lifting) x ∈ T n deriv has\_derivative_subset mem_Collect_eq subsetI T_def)
    show  $|\det (\text{matrix } (f' x))| \leq (\text{Suc } n) * e$ 
      using  $x \in T n$  T_def by auto
  next
    show  $(\lambda x. |\det (\text{matrix } (f' x))|) \text{ integrable\_on } T n$ 
      using aint_T absolutely\_integrable\_on_def by blast
  qed
  have disT: disjoint (range T)
    unfolding disjoint_def
  proof clarsimp
    show  $T m \cap T n = \{\}$  if  $T m \neq T n$  for  $m n$ 
      using that
    proof (induction m n rule: linorder_less_wlog)
      case (less m n)

```

```

with ‹e > 0› show ?case
  unfolding T_def
  proof (clarsimp simp add: Collect_conj_eq [symmetric])
    fix x
    assume e > 0 m < n n * e ≤ |det (matrix (f' x))| |det (matrix (f'
x))| < (1 + real m) * e
    then have n < 1 + real m
      by (metis (no_types, opaque_lifting) less_le_trans mult.commute
not_le mult_le_cancel_iff2)
    then show False
      using less.hyps by linarith
    qed
  qed auto
qed
have injT: inj_on T ({n. T n ≠ {}})
  unfolding inj_on_def
proof clarsimp
  show m = n if T m = T n T n ≠ {} for m n
    using that
  proof (induction m n rule: linorder_less_wlog)
    case (less m n)
    have False if T n ⊆ T m x ∈ T n for x
      using ‹e > 0› ‹m < n› that
    apply (auto simp: T_def mult.commute intro: less_le_trans dest!: subsetD)
    by (metis add.commute less_le_trans nat_less_real_le not_le mult_le_cancel_iff2)
    then show ?case
      using less.premis by blast
    qed auto
  qed
have sum_eq_Tim: (∑ k≤n. f (T k)) = sum f (T ‘ {..n}) if f {} = 0 for f ::
_ ⇒ real and n
proof (subst sum.reindex_nontrivial)
  fix i j assume i ∈ {..n} j ∈ {..n} i ≠ j T i = T j
  with that injT [unfolded inj_on_def] show f (T i) = 0
    by simp metis
qed (use atMost_atLeast0 in auto)
let ?B = m + e * ?μ S
have (∑ k≤n. ?μ (f ‘ T k)) ≤ ?B for n
proof -
  have (∑ k≤n. ?μ (f ‘ T k)) ≤ (∑ k≤n. ((k+1) * e) * ?μ(T k))
  proof (rule sum_mono [OF measure_bounded_differentiable_image])
    show (f has_derivative f' x) (at x within T k) if x ∈ T k for k x
      using that unfolding T_def by (blast intro: deriv_has_derivative_subset)
    show (λx. |det (matrix (f' x))|) integrable_on T k for k
      using absolutely_integrable_on_def aint_T by blast
    show |det (matrix (f' x))| ≤ real (k + 1) * e if x ∈ T k for k x
      using T_def that by auto
    qed (use meas_t in auto)
  also have ... ≤ (∑ k≤n. (k * e) * ?μ(T k)) + (∑ k≤n. e * ?μ(T k))

```

```

    by (simp add: algebra_simps sum.distrib)
  also have ... ≤ ?B
proof (rule add_mono)
  have (∑ k≤n. real k * e * ?μ (T k)) = (∑ k≤n. integral (T k) (λx. k * e))
    by (simp add: lmeasure_integral [OF meas_t]
        flip: integral_mult_right integral_mult_left)
  also have ... ≤ (∑ k≤n. integral (T k) (λx. (abs (det (matrix (f' x))))))
proof (rule sum_mono)
  fix k
  assume k ∈ {..n}
  show integral (T k) (λx. k * e) ≤ integral (T k) (λx. |det (matrix (f' x))|)
  proof (rule integral_le [OF integrable_on_const [OF meas_t]])
    show (λx. |det (matrix (f' x))|) integrable_on T k
      using absolutely_integrable_on_def aint_T by blast
  next
    fix x assume x ∈ T k
    show k * e ≤ |det (matrix (f' x))|
      using ⟨x ∈ T k⟩ T_def by blast
  qed
qed
also have ... = sum (λT. integral T (λx. |det (matrix (f' x))|)) (T ' {..n})
  by (auto intro: sum_eq_Tim)
also have ... = integral (⋃ k≤n. T k) (λx. |det (matrix (f' x))|)
proof (rule integral_unique [OF has_integral_Union, symmetric])
  fix S assume S ∈ T ' {..n}
  then show ((λx. |det (matrix (f' x))|) has_integral integral S (λx. |det
(matrix (f' x))|)) S
    using absolutely_integrable_on_def aint_T by blast
  next
    show pairwise (λS S'. negligible (S ∩ S')) (T ' {..n})
      using disT unfolding disjnt_iff by (auto simp: pairwise_def intro!:
empty_imp_negligible)
  qed auto
also have ... ≤ m
  unfolding m_def
proof (rule integral_subset_le)
  have (λx. |det (matrix (f' x))|) absolutely_integrable_on (⋃ k≤n. T k)
    apply (rule set_integrable_subset [OF aint_S])
    apply (intro measurable_meas_t fmeasurableD)
    apply (force simp: Seq)
  done
  then show (λx. |det (matrix (f' x))|) integrable_on (⋃ k≤n. T k)
    using absolutely_integrable_on_def by blast
qed (use Seq int in auto)
finally show (∑ k≤n. real k * e * ?μ (T k)) ≤ m .
next
  have (∑ k≤n. ?μ (T k)) = sum ?μ (T ' {..n})
    by (auto intro: sum_eq_Tim)
  also have ... = ?μ (⋃ k≤n. T k)

```

```

    using  $S$  disT by (auto simp: pairwise_def meas_t intro: measure_Union'
[symmetric])
    also have  $\dots \leq ?\mu S$ 
    using  $S$  by (auto simp: Seq intro: meas_t fmeasurableD measure_mono_fmeasurable)
    finally have  $(\sum_{k \leq n}. ?\mu (T\ k)) \leq ?\mu S$  .
    then show  $(\sum_{k \leq n}. e * ?\mu (T\ k)) \leq e * ?\mu S$ 
    by (metis less_eq_real_def ordered_comm_semiring_class.comm_mult_left_mono
sum_distrib_left that)
    qed
    finally show  $(\sum_{k \leq n}. ?\mu (f' \circ T\ k)) \leq ?B$  .
  qed
  moreover have measure_lebesgue  $(\bigcup_{k \leq n}. f' \circ T\ k) \leq (\sum_{k \leq n}. ?\mu (f' \circ T\ k))$ 
for  $n$ 
  by (simp add: fmeasurableD meas_ft measure_UNION_le)
ultimately have  $B\_ge\_m$ :  $?\mu (\bigcup_{k \leq n}. (f' \circ T\ k)) \leq ?B$  for  $n$ 
  by (meson order_trans)
have  $(\bigcup_{n}. f' \circ T\ n) \in lmeasurable$ 
  by (rule fmeasurable_countable_Union [OF meas_ft B_ge_m])
moreover have  $?\mu (\bigcup_{n}. f' \circ T\ n) \leq m + e * ?\mu S$ 
  by (rule measure_countable_Union_le [OF meas_ft B_ge_m])
ultimately show  $f' \circ S \in lmeasurable$   $?\mu (f' \circ S) \leq m + e * ?\mu S$ 
  by (auto simp: Seq_image_Union)
qed
show ?thesis
proof
  show  $f' \circ S \in lmeasurable$ 
    using * linordered_field_no_ub by blast
  let  $?x = m - ?\mu (f' \circ S)$ 
  have False if  $?\mu (f' \circ S) > \text{integral } S (\lambda x. |\det (\text{matrix } (f' \ x))|)$ 
  proof -
    have  $ml$ :  $m < ?\mu (f' \circ S)$ 
    using  $m\_def$  that by blast
    then have  $?\mu S \neq 0$ 
    using *(2) bgauge_existence_lemma by fastforce
    with  $ml$  have  $0$ :  $0 < -(m - ?\mu (f' \circ S))/2 / ?\mu S$ 
    using that zero_less_measure_iff by force
    then show ?thesis
    using * [OF 0] that by (auto simp: field_split_simps  $m\_def$  split: if_split_asm)
  qed
  then show  $?\mu (f' \circ S) \leq \text{integral } S (\lambda x. |\det (\text{matrix } (f' \ x))|)$ 
    by fastforce
  qed
qed

```

theorem

fixes $f :: \text{real}^n :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n :: \_$
assumes S : $S \in \text{sets lebesgue}$
and deriv : $\bigwedge x. x \in S \Rightarrow (f \text{ has\_derivative } f' \ x) \text{ (at } x \text{ within } S)$

```

    and int: (λx. |det (matrix (f' x))|) integrable_on S
  shows measurable_differentiable_image: f ' S ∈ lmeasurable
    and measure_differentiable_image:
      measure lebesgue (f ' S) ≤ integral S (λx. |det (matrix (f' x))|) (is ?M)
proof -
  let ?I = λn::nat. cbox (vec (-n)) (vec n) ∩ S
  let ?μ = measure lebesgue
  have x ∈ cbox (vec (- real (nat [norm x]))) (vec (real (nat [norm x]))) for x
  :: real^'n::_
  apply (auto simp: mem_box_cart)
  apply (metis abs_le_iff component_le_norm_cart minus_le_iff of_nat_ceiling
order.trans)
  by (meson abs_le_D1 norm_bound_component_le_cart real_nat_ceiling_ge)
  then have Seq: S = (⋃ n. ?I n)
  by auto
  have fIn: f ' ?I n ∈ lmeasurable
    and mfIn: ?μ (f ' ?I n) ≤ integral S (λx. |det (matrix (f' x))|) (is ?MN) for
n
  proof -
    have In: ?I n ∈ lmeasurable
      by (simp add: S bounded_Int bounded_set_imp_lmeasurable sets.Int)
    moreover have ∧x. x ∈ ?I n ⇒ (f has_derivative f' x) (at x within ?I n)
      by (meson Int_iff deriv_has_derivative_subset subsetI)
    moreover have int_In: (λx. |det (matrix (f' x))|) integrable_on ?I n
    proof -
      have (λx. |det (matrix (f' x))|) absolutely_integrable_on S
        using int absolutely_integrable_integrable_bound by force
      then have (λx. |det (matrix (f' x))|) absolutely_integrable_on ?I n
      by (metis (no_types) Int_lower1 In_fmeasurableD inf_commute set_integrable_subset)
      then show ?thesis
        using absolutely_integrable_on_def by blast
    qed
    ultimately have f ' ?I n ∈ lmeasurable ?μ (f ' ?I n) ≤ integral (?I n) (λx.
|det (matrix (f' x))|)
      using m_diff_image_weak by metis+
    moreover have integral (?I n) (λx. |det (matrix (f' x))|) ≤ integral S (λx.
|det (matrix (f' x))|)
      by (simp add: int_In int_integral_subset_le)
    ultimately show f ' ?I n ∈ lmeasurable ?MN
      by auto
  qed
  have ?I k ⊆ ?I n if k ≤ n for k n
    by (rule Int_mono) (use that in ⟨auto simp: subset_interval_imp_cart⟩)
  then have (⋃ k≤n. f ' ?I k) = f ' ?I n for n
    by (fastforce simp add:)
  with mfIn have ?μ (⋃ k≤n. f ' ?I k) ≤ integral S (λx. |det (matrix (f' x))|) for
n
    by simp
  then have (⋃ n. f ' ?I n) ∈ lmeasurable ?μ (⋃ n. f ' ?I n) ≤ integral S (λx. |det

```

```

(matrix (f' x))|)
  by (rule fmeasurable_countable_Union [OF fIn] measure_countable_Union_le
[OF fIn])+
  then show f ' S ∈ lmeasurable ?M
  by (metis Seq image_UN)+
qed

```

lemma *borel_measurable_simple_function_limit_increasing:*

fixes *f* :: 'a::euclidean_space $\Rightarrow$ real

shows $(f \in \text{borel\_measurable lebesgue} \wedge (\forall x. 0 \leq f x)) \longleftrightarrow$

$(\exists g. (\forall n x. 0 \leq g n x \wedge g n x \leq f x) \wedge (\forall n x. g n x \leq (g(\text{Suc } n) x)) \wedge$
 $(\forall n. g n \in \text{borel\_measurable lebesgue}) \wedge (\forall n. \text{finite}(\text{range } (g n))) \wedge$
 $(\forall x. (\lambda n. g n x) \longrightarrow f x))$

(is ?lhs = ?rhs)

proof

assume *f*: ?lhs

have *leb_f*: $\{x. a \leq f x \wedge f x < b\} \in \text{sets lebesgue}$ **for** *a b*

proof –

have $\{x. a \leq f x \wedge f x < b\} = \{x. f x < b\} - \{x. f x < a\}$

by *auto*

also have $\dots \in \text{sets lebesgue}$

using *borel_measurable_vimage_halfspace_component_lt* [of *f UNIV*] *f* **by**

auto

finally show ?thesis .

qed

have $g n x \leq f x$

if *inc_g*: $\bigwedge n x. 0 \leq g n x \wedge g n x \leq g (\text{Suc } n) x$

and *meas_g*: $\bigwedge n. g n \in \text{borel\_measurable lebesgue}$

and *fin*: $\bigwedge n. \text{finite}(\text{range } (g n))$ **and** *lim*: $\bigwedge x. (\lambda n. g n x) \longrightarrow f x$ **for**

g n x

proof –

have $\exists r > 0. \forall N. \exists n \geq N. \text{dist } (g n x) (f x) \geq r$ **if** $g n x > f x$

proof –

have *g*: $g n x \leq g (N + n) x$ **for** *N*

by (*rule transitive_stepwise_le*) (*use inc_g in auto*)

have $\exists na \geq N. g n x - f x \leq \text{dist } (g na x) (f x)$ **for** *N*

apply (*rule_tac x=N+n in exI*)

using *g* [of *N*] **by** (*auto simp: dist_norm*)

with that show ?thesis

using *diff_gt_0_iff_gt* **by** *blast*

qed

with *lim show* ?thesis

apply (*auto simp: lim_sequentially*)

by (*meson less_le_not_le not_le_imp_less*)

qed

moreover

let $? \Omega = \lambda n k. \text{indicator } \{y. k/2^n \leq f y \wedge f y < (k+1)/2^n\}$

let $?g = \lambda n x. (\sum k::\text{real} \mid k \in \mathbb{Z} \wedge |k| \leq 2^n (2*n). k/2^n * ? \Omega n k x)$

```

have  $\exists g. (\forall n x. 0 \leq g n x \wedge g n x \leq (g(Suc n) x)) \wedge$ 
   $(\forall n. g n \in \text{borel\_measurable\_lebesgue}) \wedge (\forall n. \text{finite}(\text{range } (g n))) \wedge (\forall x.$ 
 $(\lambda n. g n x) \longrightarrow f x)$ 
proof (intro exI allI conjI)
  show  $0 \leq ?g n x$  for  $n x$ 
  proof (clarify intro!: ordered_comm_monoid_add_class.sum_nonneg)
    fix  $k::\text{real}$ 
    assume  $k \in \mathbb{Z}$  and  $k: |k| \leq 2^{\wedge}(2*n)$ 
    show  $0 \leq k/2^{\wedge}n * ?\Omega n k x$ 
    using  $f \langle k \in \mathbb{Z} \rangle$  apply (auto simp: indicator_def field_split_simps Ints_def)
    apply (drule spec [where  $x=x$ ])
    using zero_le_power [of  $2::\text{real } n$ ] mult_nonneg_nonneg [of  $f x 2^{\wedge}n$ ]
    by linarith
  qed
show  $?g n x \leq ?g (Suc n) x$  for  $n x$ 
proof -
  have  $?g n x =$ 
     $(\sum k \mid k \in \mathbb{Z} \wedge |k| \leq 2^{\wedge}(2*n).$ 
     $k/2^{\wedge}n * (\text{indicator } \{y. k/2^{\wedge}n \leq f y \wedge f y < (k+1)/2^{\wedge}n\} x +$ 
     $\text{indicator } \{y. (k+1)/2^{\wedge}n \leq f y \wedge f y < (k+1)/2^{\wedge}n\} x)$ 
    by (rule sum.cong [OF refl]) (simp add: indicator_def field_split_simps)
  also have  $\dots = (\sum k \mid k \in \mathbb{Z} \wedge |k| \leq 2^{\wedge}(2*n). k/2^{\wedge}n * \text{indicator } \{y. k/2^{\wedge}n$ 
 $\leq f y \wedge f y < (k+1)/2^{\wedge}n\} x) +$ 
     $(\sum k \mid k \in \mathbb{Z} \wedge |k| \leq 2^{\wedge}(2*n). k/2^{\wedge}n * \text{indicator } \{y.$ 
 $(k+1)/2^{\wedge}n \leq f y \wedge f y < (k+1)/2^{\wedge}n\} x)$ 
    by (simp add: comm_monoid_add_class.sum.distrib algebra_simps)
  also have  $\dots = (\sum k \mid k \in \mathbb{Z} \wedge |k| \leq 2^{\wedge}(2*n). (2 * k)/2^{\wedge}Suc n * \text{indicator}$ 
 $\{y. (2 * k)/2^{\wedge}Suc n \leq f y \wedge f y < (2 * k+1)/2^{\wedge}Suc n\} x) +$ 
     $(\sum k \mid k \in \mathbb{Z} \wedge |k| \leq 2^{\wedge}(2*n). (2 * k)/2^{\wedge}Suc n * \text{indicator}$ 
 $\{y. (2 * k+1)/2^{\wedge}Suc n \leq f y \wedge f y < ((2 * k+1) + 1)/2^{\wedge}Suc n\} x)$ 
    by (force simp: field_simps indicator_def intro: sum.cong)
  also have  $\dots \leq (\sum k \mid k \in \mathbb{Z} \wedge |k| \leq 2^{\wedge}(2 * Suc n). k/2^{\wedge}Suc n *$ 
 $(\text{indicator } \{y. k/2^{\wedge}Suc n \leq f y \wedge f y < (k+1)/2^{\wedge}Suc n\} x))$ 
    (is  $?a + \_ \leq ?b$ )
  proof -
    have *:  $\llbracket \text{sum } f I \leq \text{sum } h I; a + \text{sum } h I \leq b \rrbracket \implies a + \text{sum } f I \leq b$  for  $I$ 
     $a b f$  and  $h :: \text{real} \Rightarrow \text{real}$ 
    by linarith
    let  $?h = \lambda k. (2*k+1)/2^{\wedge}Suc n *$ 
     $(\text{indicator } \{y. (2 * k+1)/2^{\wedge}Suc n \leq f y \wedge f y < ((2*k+1) +$ 
     $1)/2^{\wedge}Suc n\} x)$ 
    show ?thesis
    proof (rule *)
      show  $(\sum k \mid k \in \mathbb{Z} \wedge |k| \leq 2^{\wedge}(2*n).$ 
     $2 * k/2^{\wedge}Suc n * \text{indicator } \{y. (2 * k+1)/2^{\wedge}Suc n \leq f y \wedge f y$ 
 $< (2 * k+1 + 1)/2^{\wedge}Suc n\} x)$ 
     $\leq \text{sum } ?h \{k \in \mathbb{Z}. |k| \leq 2^{\wedge}(2*n)\}$ 
      by (rule sum_mono) (simp add: indicator_def field_split_simps)
    next

```



```

have  $\alpha$ :  $?a = (\sum k \in (*)2 \text{ ' } \{k \in \mathbb{Z}. |k| \leq 2^{\wedge}(2*n)\}.$ 
 $k/2^{\wedge} \text{Suc } n * \text{indicator } \{y. k/2^{\wedge} \text{Suc } n \leq f y \wedge f y < (k+1)/2$ 
 $^{\wedge} \text{Suc } n\} x)$ 
by (auto simp: inj_on_def field_simps comm_monoid_add_class.sum.reindex)
have  $\beta$ :  $\text{sum } ?h \{k \in \mathbb{Z}. |k| \leq 2^{\wedge}(2*n)\}$ 
 $= (\sum k \in (\lambda x. 2*x + 1) \text{ ' } \{k \in \mathbb{Z}. |k| \leq 2^{\wedge}(2*n)\}.$ 
 $k/2^{\wedge} \text{Suc } n * \text{indicator } \{y. k/2^{\wedge} \text{Suc } n \leq f y \wedge f y < (k+1)/2$ 
 $^{\wedge} \text{Suc } n\} x)$ 
by (auto simp: inj_on_def field_simps comm_monoid_add_class.sum.reindex)
have  $0$ :  $(*) 2 \text{ ' } \{k \in \mathbb{Z}. P k\} \cap (\lambda x. 2 * x + 1) \text{ ' } \{k \in \mathbb{Z}. P k\} = \{\}$  for
 $P :: \text{real} \Rightarrow \text{bool}$ 
proof -
have  $2 * i \neq 2 * j + 1$  for  $i j :: \text{int}$  by arith
thus  $?thesis$ 
unfolding Ints_def by auto (use of int_eq_iff in fastforce)
qed
have  $?a + \text{sum } ?h \{k \in \mathbb{Z}. |k| \leq 2^{\wedge}(2*n)\}$ 
 $= (\sum k \in (*)2 \text{ ' } \{k \in \mathbb{Z}. |k| \leq 2^{\wedge}(2*n)\} \cup (\lambda x. 2*x + 1) \text{ ' } \{k \in \mathbb{Z}.$ 
 $|k| \leq 2^{\wedge}(2*n)\}.$ 
 $k/2^{\wedge} \text{Suc } n * \text{indicator } \{y. k/2^{\wedge} \text{Suc } n \leq f y \wedge f y < (k+1)/2^{\wedge}$ 
 $\text{Suc } n\} x)$ 
unfolding  $\alpha \beta$ 
using finite_abs_int_segment [of  $2^{\wedge}(2*n)$ ]
by (subst sum_Un) (auto simp: 0)
also have  $\dots \leq ?b$ 
proof (rule sum_mono2)
show finite  $\{k :: \text{real}. k \in \mathbb{Z} \wedge |k| \leq 2^{\wedge}(2 * \text{Suc } n)\}$ 
by (rule finite_abs_int_segment)
show  $(*) 2 \text{ ' } \{k :: \text{real}. k \in \mathbb{Z} \wedge |k| \leq 2^{\wedge}(2*n)\} \cup (\lambda x. 2*x + 1) \text{ ' } \{k \in$ 
 $\mathbb{Z}. |k| \leq 2^{\wedge}(2*n)\} \subseteq \{k \in \mathbb{Z}. |k| \leq 2^{\wedge}(2 * \text{Suc } n)\}$ 
apply auto
using one_le_power [of  $2 :: \text{real } 2*n$ ] by linarith
have  $*$ :  $\llbracket x \in (S \cup T) - U; \bigwedge x. x \in S \Rightarrow x \in U; \bigwedge x. x \in T \Rightarrow x \in$ 
 $U \rrbracket \Rightarrow P x$  for  $S T U P$ 
by blast
have  $0 \leq b$  if  $b \in \mathbb{Z} f x * (2 * 2^{\wedge} n) < b + 1$  for  $b$ 
proof -
have  $0 \leq f x * (2 * 2^{\wedge} n)$ 
by (simp add: f)
also have  $\dots < b + 1$ 
by (simp add: that)
finally show  $0 \leq b$ 
using  $\langle b \in \mathbb{Z} \rangle$  by (auto simp: elim!: Ints_cases)
qed
then show  $0 \leq b/2^{\wedge} \text{Suc } n * \text{indicator } \{y. b/2^{\wedge} \text{Suc } n \leq f y \wedge f y <$ 
 $(b + 1)/2^{\wedge} \text{Suc } n\} x$ 
if  $b \in \{k \in \mathbb{Z}. |k| \leq 2^{\wedge}(2 * \text{Suc } n)\} -$ 
 $((*) 2 \text{ ' } \{k \in \mathbb{Z}. |k| \leq 2^{\wedge}(2*n)\} \cup (\lambda x. 2*x + 1) \text{ ' } \{k \in \mathbb{Z}.$ 
 $|k| \leq 2^{\wedge}(2*n)\})$  for  $b$ 

```

```

      using that by (simp add: indicator_def divide_simps)
    qed
    finally show ?a + sum ?h {k ∈ ℤ. |k| ≤ 2 ^ (2*n)} ≤ ?b .
  qed
qed
finally show ?thesis .
qed
show ?g n ∈ borel_measurable_lebesgue for n
apply (intro borel_measurable_indicator borel_measurable_times borel_measurable_sum)
  using leb_f_sets_restrict_UNIV by auto
show finite (range (?g n)) for n
proof -
  have (∑ k | k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n). k/2^n * ?Ω n k x)
    ∈ (λk. k/2^n) ' {k ∈ ℤ. |k| ≤ 2 ^ (2*n)} for x
  proof (cases ∃ k. k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n) ∧ k/2^n ≤ f x ∧ f x < (k+1)/2^n)
    case True
    then show ?thesis
      by (blast intro: indicator_sum_eq)
  next
    case False
    then have (∑ k | k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n). k/2^n * ?Ω n k x) = 0
      by auto
    then show ?thesis by force
  qed
  then have range (?g n) ⊆ ((λk. (k/2^n)) ' {k. k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n)})
    by auto
  moreover have finite ((λk::real. (k/2^n)) ' {k ∈ ℤ. |k| ≤ 2 ^ (2*n)})
    by (intro finite_imageI finite_abs_int_segment)
  ultimately show ?thesis
    by (rule finite_subset)
qed
show (λn. ?g n x) ⟶ f x for x
proof (clarsimp simp add: lim_sequentially)
  fix e::real
  assume e > 0
  obtain N1 where N1: 2 ^ N1 > abs(f x)
    using real_arch_pow by fastforce
  obtain N2 where N2: (1/2) ^ N2 < e
    using real_arch_pow_inv ⟨e > 0⟩ by fastforce
  have dist (∑ k | k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n). k/2^n * ?Ω n k x) (f x) < e if N1
+ N2 ≤ n for n
  proof -
    let ?m = real_of_int [2^n * f x]
    have |?m| ≤ 2^n * 2^N1
      using N1 apply (simp add: f)
    by (meson floor_mono le_floor_iff less_le_not_le mult_le_cancel_left_pos
zero_less_numeral zero_less_power)
    also have ... ≤ 2 ^ (2*n)
      by (metis that add_leD1 add_le_cancel_left mult.commute mult_2_right

```

```

one_less_numeral_iff
  power_add power_increasing_iff semiring_norm(76))
  finally have m_le:  $|?m| \leq 2^{(2*n)}$  .
  have  $?m/2^n \leq f\ x \leq (?m + 1)/2^n$ 
    by (auto simp: mult.commute pos_divide_le_eq mult_imp_less_div_pos)
  then have eq:  $\text{dist}(\sum k \mid k \in \mathbb{Z} \wedge |k| \leq 2^{(2*n)}. k/2^n * ?\Omega\ n\ k\ x)\ (f\ x)$ 
    =  $\text{dist} (?m/2^n)\ (f\ x)$ 
    by (subst indicator_sum_eq [of ?m]) (auto simp: m_le)
  have  $|2^n| * |?m/2^n - f\ x| = |2^n| * (?m/2^n - f\ x)$ 
    by (simp add: abs_mult)
  also have  $\dots < 2^{N2} * e$ 
    using N2 by (simp add: divide_simps mult.commute) linarith
  also have  $\dots \leq |2^n| * e$ 
    using that  $\langle e > 0 \rangle$  by auto
  finally have  $\text{dist} (?m/2^n)\ (f\ x) < e$ 
    by (simp add: dist_norm)
  then show ?thesis
    using eq by linarith
qed
then show  $\exists no. \forall n \geq no. \text{dist}(\sum k \mid k \in \mathbb{Z} \wedge |k| \leq 2^{(2*n)}. k * ?\Omega\ n\ k\ x/2^n)\ (f\ x) < e$ 
  by force
qed
qed
ultimately show ?rhs
  by metis
next
  assume RHS: ?rhs
  with borel_measurable_simple_function_limit [of f UNIV, unfolded lebesgue_on_UNIV_eq]
  show ?lhs
    by (blast intro: order_trans)
qed

```

6.47.2 Borel measurable Jacobian determinant

lemma *lemma_partial_derivatives0:*

fixes $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$

assumes *linear f and lim0:* $((\lambda x. f\ x \ /_R \text{norm}\ x) \longrightarrow 0)$ (at 0 within S)

and lb: $\bigwedge v. v \neq 0 \implies (\exists k > 0. \forall e > 0. \exists x. x \in S - \{0\} \wedge \text{norm}\ x < e \wedge k * \text{norm}\ x \leq |v \cdot x|)$

shows $f\ x = 0$

proof –

interpret *linear f by fact*

have $\dim \{x. f\ x = 0\} \leq \text{DIM}('a)$

by (rule dim_subset_UNIV)

moreover have *False if less:* $\dim \{x. f\ x = 0\} < \text{DIM}('a)$

proof –

obtain d **where** $d \neq 0$ **and** $d: \bigwedge y. f\ y = 0 \implies d \cdot y = 0$

using *orthogonal_to_subspace_exists* [OF less] *orthogonal_def*

```

    by (metis (mono_tags, lifting) mem_Collect_eq span_base)
  then obtain k where k > 0
    and k:  $\bigwedge e. e > 0 \implies \exists y. y \in S - \{0\} \wedge \text{norm } y < e \wedge k * \text{norm } y \leq |d \cdot$ 
y|
    using lb by blast
  have  $\exists h. \forall n. ((h \ n \in S \wedge h \ n \neq 0 \wedge k * \text{norm } (h \ n) \leq |d \cdot h \ n|) \wedge \text{norm } (h$ 
n) < 1 / real (Suc n))  $\wedge$ 
    norm (h (Suc n)) < norm (h n)
  proof (rule dependent_nat_choice)
    show  $\exists y. (y \in S \wedge y \neq 0 \wedge k * \text{norm } y \leq |d \cdot y|) \wedge \text{norm } y < 1 / \text{real } (Suc$ 
0)
    by simp (metis DiffE insertCI k not_less not_one_le_zero)
  qed (use k [of min (norm x) (1/(Suc n + 1))] for x n] in auto)
  then obtain  $\alpha$  where  $\alpha: \bigwedge n. \alpha \ n \in S - \{0\}$  and  $kd: \bigwedge n. k * \text{norm}(\alpha \ n) \leq$ 
 $|d \cdot \alpha \ n|$ 
    and norm_lt:  $\bigwedge n. \text{norm}(\alpha \ n) < 1/(Suc \ n)$ 
  by force
  let  $? \beta = \lambda n. \alpha \ n /_R \text{norm } (\alpha \ n)$ 
  have com:  $\bigwedge g. (\forall n. g \ n \in \text{sphere } (0::'a) \ 1) \implies \exists l \in \text{sphere } 0 \ 1. \exists g::\text{nat} \Rightarrow \text{nat}. \text{strict\_mono } g \wedge (g \circ g) \longrightarrow l$ 
    using compact_sphere compact_def by metis
  moreover have  $\forall n. ? \beta \ n \in \text{sphere } 0 \ 1$ 
    using  $\alpha$  by auto
  ultimately obtain  $l::'a$  and  $g::\text{nat} \Rightarrow \text{nat}$ 
    where  $l: l \in \text{sphere } 0 \ 1$  and  $\text{strict\_mono } g$  and  $\text{to\_l}: (? \beta \circ g) \longrightarrow l$ 
  by meson
  moreover have continuous (at l) ( $\lambda x. (|d \cdot x| - k)$ )
    by (intro continuous_intros)
  ultimately have  $\text{lim\_dl}: ((\lambda x. (|d \cdot x| - k)) \circ (? \beta \circ g)) \longrightarrow (|d \cdot l| - k)$ 
    by (meson continuous_imp_tendsto)
  have  $\forall_F i \text{ in sequentially. } 0 \leq ((\lambda x. |d \cdot x| - k) \circ ((\lambda n. \alpha \ n /_R \text{norm } (\alpha \ n))$ 
 $\circ g)) \ i$ 
    using  $\alpha \ kd$  by (auto simp: field_split_simps)
  then have  $k \leq |d \cdot l|$ 
    using tendsto_lowerbound [OF lim_dl, of 0] by auto
  moreover have  $d \cdot l = 0$ 
  proof (rule d)
    show  $f \ l = 0$ 
  proof (rule LIMSEQ_unique [of f  $\circ ? \beta \circ g$ ])
    have isCont f l
      using  $\langle \text{linear } f \rangle$  linear_continuous_at linear_conv_bounded_linear by
blast
    then show  $(f \circ (\lambda n. \alpha \ n /_R \text{norm } (\alpha \ n)) \circ g) \longrightarrow f \ l$ 
      unfolding comp_assoc
      using to_l continuous_imp_tendsto by blast
    have  $\alpha \longrightarrow 0$ 
      using norm_lt LIMSEQ_norm_0 by metis
    with  $\langle \text{strict\_mono } g \rangle$  have  $(\alpha \circ g) \longrightarrow 0$ 
      by (metis LIMSEQ_subseq_LIMSEQ)

```

```

    with  $\lim 0 \ \alpha$  have  $((\lambda x. f \ x \ /_R \ norm \ x) \circ (\alpha \circ \varrho)) \longrightarrow 0$ 
    by (force simp: tendsto_at_iff_sequentially)
    then show  $(f \circ (\lambda n. \alpha \ n \ /_R \ norm \ (\alpha \ n)) \circ \varrho) \longrightarrow 0$ 
    by (simp add: o_def scale)
  qed
qed
ultimately show False
using  $\langle k > 0 \rangle$  by auto
qed
ultimately have  $\dim: \dim \{x. f \ x = 0\} = DIM('a)$ 
by force
then show ?thesis
using dim_eq_full
by (metis (mono_tags, lifting) eq_0_on_span eucl.span_Basis linear_axioms
linear_eq_stdbasis
mem_Collect_eq module_hom_zero span_base span_raw_def)
qed

lemma lemma_partial_derivatives:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $linear \ f$  and  $\lim: ((\lambda x. f \ (x - a) \ /_R \ norm \ (x - a)) \longrightarrow 0) \ (at \ a \ within \ S)$ 
  and  $lb: \bigwedge v. v \neq 0 \implies (\exists k > 0. \ \forall e > 0. \ \exists x \in S - \{a\}. \ norm(a - x) < e \wedge k * norm(a - x) \leq |v \cdot (x - a)|)$ 
  shows  $f \ x = 0$ 
proof -
  have  $((\lambda x. f \ x \ /_R \ norm \ x) \longrightarrow 0) \ (at \ 0 \ within \ (\lambda x. x - a) \ 'S)$ 
  using  $\lim$  by (simp add: Lim_within_dist_norm)
  then show ?thesis
  proof (rule lemma_partial_derivatives0 [OF  $\langle linear \ f \rangle$ ])
    fix  $v :: 'a$ 
    assume  $v: v \neq 0$ 
    show  $\exists k > 0. \ \forall e > 0. \ \exists x. x \in (\lambda x. x - a) \ 'S - \{0\} \wedge norm \ x < e \wedge k * norm \ x \leq |v \cdot x|$ 
    using  $lb \ [OF \ v]$  by (force simp: norm_minus_commute)
  qed
qed

```

proposition *borel_measurable_partial_derivatives:*

```

  fixes  $f :: real^m::\{finite, wellorder\} \Rightarrow real^n$ 
  assumes  $S: S \in sets \ lebesgue$ 
  and  $f: \bigwedge x. x \in S \implies (f \ has\_derivative \ f' \ x) \ (at \ x \ within \ S)$ 
  shows  $(\lambda x. (matrix(f' \ x) \$m \$n)) \in borel\_measurable \ (lebesgue\_on \ S)$ 
proof -
  have  $contf: continuous\_on \ S \ f$ 
  using  $continuous\_on\_eq\_continuous\_within \ f \ has\_derivative\_continuous$  by blast
  have  $\{x \in S. \ (matrix \ (f' \ x) \$m \$n) \leq b\} \in sets \ lebesgue$  for  $b$ 

```

```

proof (rule sets_negligible_symdiff)
  let ?T = {x ∈ S. ∀ e>0. ∃ d>0. ∃ A. A $ m $ n < b ∧ (∀ i j. A $ i $ j ∈ Q) ∧
    (∀ y ∈ S. norm(y - x) < d → norm(f y - f x - A * v (y -
x)) ≤ e * norm(y - x))}
  let ?U = S ∩
    (⋂ e ∈ {e ∈ Q. e > 0}.
      ⋃ A ∈ {A. A $ m $ n < b ∧ (∀ i j. A $ i $ j ∈ Q)}.
        ⋃ d ∈ {d ∈ Q. 0 < d}.
          S ∩ (⋂ y ∈ S. {x ∈ S. norm(y - x) < d → norm(f y - f x -
A * v (y - x)) ≤ e * norm(y - x)}))
  have ?T = ?U
  proof (intro set_eqI iffI)
    fix x
    assume xT: x ∈ ?T
    then show x ∈ ?U
    proof (clarsimp simp add:)
      fix q :: real
      assume q ∈ Q q > 0
      then obtain d A where d > 0 and A: A $ m $ n < b ∧ i j. A $ i $ j ∈ Q
        ∧ y. [y ∈ S; norm(y - x) < d] ⇒ norm(f y - f x - A * v (y - x)) ≤ q
        * norm(y - x)
      using xT by auto
      then obtain δ where d > δ δ > 0 δ ∈ Q
      using Rats_dense_in_real by blast
      with A show ∃ A. A $ m $ n < b ∧ (∀ i j. A $ i $ j ∈ Q) ∧
        (∃ s. s ∈ Q ∧ 0 < s ∧ (∀ y ∈ S. norm(y - x) < s → norm(f
y - f x - A * v (y - x)) ≤ q * norm(y - x)))
      by force
    qed
  next
    fix x
    assume xU: x ∈ ?U
    then show x ∈ ?T
    proof clarsimp
      fix e :: real
      assume e > 0
      then obtain ε where e > ε ε > 0 ε ∈ Q
      using Rats_dense_in_real by blast
      with xU obtain A r where x ∈ S and Ar: A $ m $ n < b ∧ i j. A $ i $ j
∈ Q r ∈ Q r > 0
      and ∀ y ∈ S. norm(y - x) < r → norm(f y - f x - A * v (y - x)) ≤ ε
        * norm(y - x)
      by (auto simp: split: if_split_asm)
      then have ∀ y ∈ S. norm(y - x) < r → norm(f y - f x - A * v (y - x))
≤ e * norm(y - x)
      by (meson ‹e > ε› less_eq_real_def mult_right_mono norm_ge_zero
order_trans)
      then show ∃ d>0. ∃ A. A $ m $ n < b ∧ (∀ i j. A $ i $ j ∈ Q) ∧ (∀ y ∈ S.
norm(y - x) < d → norm(f y - f x - A * v (y - x)) ≤ e * norm(y - x))

```

```

      using ‹ $x \in S$ › Ar by blast
    qed
  qed
  moreover have "?U ∈ sets lebesgue"
  proof -
    have coQ: countable {e ∈ ℚ. 0 < e}
      using countable_Collect countable_rat by blast
    have ne: {e ∈ ℚ. (0::real) < e} ≠ {}
      using zero_less_one Rats_1 by blast
    have coA: countable {A. A $ m $ n < b ∧ (∀ i j. A $ i $ j ∈ ℚ)}
    proof (rule countable_subset)
      show countable {A. ∀ i j. A $ i $ j ∈ ℚ}
        using countable_vector [OF countable_vector, of λi j. ℚ] by (simp add:
countable_rat)
      qed blast
    have *: ‹U ≠ {} ⟹ closedin (top_of_set S) (S ∩ ⋂ U)›
      ⟹ closedin (top_of_set S) (S ∩ ⋂ U) for U
      by fastforce
    have eq: {x::(real,'m)vec. P x ∧ (Q x ⟶ R x)} = {x. P x ∧ ¬ Q x} ∪ {x.
P x ∧ R x} for P Q R
      by auto
    have sets: S ∩ (⋂ y∈S. {x ∈ S. norm (y - x) < d ⟶ norm (f y - f x - A
*v (y - x)) ≤ e * norm (y - x)})
      ∈ sets lebesgue for e A d
    proof -
      have clo: closedin (top_of_set S)
        {x ∈ S. norm (y - x) < d ⟶ norm (f y - f x - A *v (y - x))
≤ e * norm (y - x)}
        for y
      proof -
        have cont1: continuous_on S (λx. norm (y - x))
          and cont2: continuous_on S (λx. e * norm (y - x) - norm (f y - f x
- (A *v y - A *v x)))
          by (force intro: contf continuous_intros)+
        have clo1: closedin (top_of_set S) {x ∈ S. d ≤ norm (y - x)}
          using continuous_closedin_preimage [OF cont1, of {d..}] by (simp add:
vimage_def Int_def)
        have clo2: closedin (top_of_set S)
          {x ∈ S. norm (f y - f x - (A *v y - A *v x)) ≤ e * norm (y
- x)}
          using continuous_closedin_preimage [OF cont2, of {0..}] by (simp add:
vimage_def Int_def)
        show ?thesis
          by (auto simp: eq not_less matrix_vector_mult_diff_distrib intro: clo1
clo2)
      qed
    qed
    show ?thesis
      by (rule lebesgue_closedin [of S]) (force intro: * S clo)+
  qed

```

```

show ?thesis
by (intro sets.sets.Int S sets.countable_UN'' sets.countable_INT'' coQ coA)
auto
qed
ultimately show ?T ∈ sets.lebesgue
by simp
let ?M = (?T - {x ∈ S. matrix (f' x) $ m $ n ≤ b} ∪ ({x ∈ S. matrix (f' x)
$ m $ n ≤ b} - ?T))
let ?Θ = λx v. ∀ξ>0. ∃e>0. ∀y ∈ S - {x}. norm (x - y) < e ⟶ |v • (y -
x)| < ξ * norm (x - y)
have nN: negligible {x ∈ S. ∃v≠0. ?Θ x v}
unfolding negligible_eq_zero_density
proof clarsimp
fix x v and r e :: real
assume x ∈ S v ≠ 0 r > 0 e > 0
and Theta [rule_format]: ?Θ x v
moreover have (norm v * e / 2) / CARD('m) ^ CARD('m) > 0
by (simp add: ‹v ≠ 0› ‹e > 0›)
ultimately obtain d where d > 0
and dless: ∧y. [y ∈ S - {x}; norm (x - y) < d] ⟹
|v • (y - x)| < ((norm v * e / 2) / CARD('m) ^ CARD('m))
* norm (x - y)
by metis
let ?W = ball x (min d r) ∩ {y. |v • (y - x)| < (norm v * e / 2 * min d r) /
CARD('m) ^ CARD('m)}
have open {x. |v • (x - a)| < b} for a b
by (intro open_Collect_less_continuous_intros)
show ∃d>0. d ≤ r ∧
(∃U. {x' ∈ S. ∃v≠0. ?Θ x' v} ∩ ball x d ⊆ U ∧
U ∈ lmeasurable ∧ measure lebesgue U < e * content (ball x d))
proof (intro exI conjI)
show 0 < min d r min d r ≤ r
using ‹r > 0› ‹d > 0› by auto
show {x' ∈ S. ∃v. v ≠ 0 ∧ (∀ξ>0. ∃e>0. ∀z∈S - {x'}. norm (x' - z)
< e ⟶ |v • (z - x')| < ξ * norm (x' - z))} ∩ ball x (min d r) ⊆ ?W
proof (clarsimp simp: dist_norm norm_minus_commute)
fix y w
assume y ∈ S w ≠ 0
and less [rule_format]:
∀ξ>0. ∃e>0. ∀z∈S - {y}. norm (y - z) < e ⟶ |w • (z - y)|
< ξ * norm (y - z)
and d: norm (y - x) < d and r: norm (y - x) < r
show |v • (y - x)| < norm v * e * min d r / (2 * real CARD('m) ^
CARD('m))
proof (cases y = x)
case True
with ‹r > 0› ‹d > 0› ‹e > 0› ‹v ≠ 0› show ?thesis
by simp
next

```



```

      case False
      have  $|v \cdot (y - x)| < \text{norm } v * e / 2 / \text{real } (\text{CARD}('m) \wedge \text{CARD}('m))$ 
      *  $\text{norm } (x - y)$ 
      apply (rule dless)
      using False  $\langle y \in S \rangle d$  by (auto simp: norm_minus_commute)
      also have  $\dots \leq \text{norm } v * e * \min d r / (2 * \text{real } \text{CARD}('m) \wedge$ 
       $\text{CARD}('m))$ 
      using  $d r \langle e > 0 \rangle$  by (simp add: field_simps norm_minus_commute
      mult_left_mono)
      finally show ?thesis .
    qed
  qed
  show ?W  $\in$  lmeasurable
  by (simp add: fmeasurable_Int_fmeasurable borel_open)
  obtain  $k::'m$  where True
  by metis
  obtain T where T: orthogonal_transformation T and  $v: v = T(\text{norm } v$ 
   $*_R \text{axis } k (1::\text{real}))$ 
  using rotation_rightward_line by metis
  define b where  $b \equiv \text{norm } v$ 
  have  $b > 0$ 
  using  $\langle v \neq 0 \rangle$  by (auto simp: b_def)
  obtain eqb:  $\text{inv } T v = b *_R \text{axis } k (1::\text{real})$  and  $\text{inj } T \text{ bij } T$  and  $\text{inv } T$ :
  orthogonal_transformation (inv T)
  by (metis UNIV_I b_def T v bij_betw_inv_into_left orthogonal_
  transformation_inj orthogonal_transformation_bij orthogonal_transformation_inv)
  let  $?v = \chi \ i. \min d r / \text{CARD}('m)$ 
  let  $?v' = \chi \ i. \text{if } i = k \text{ then } (e/2 * \min d r) / \text{CARD}('m) \wedge \text{CARD}('m)$ 
  else  $\min d r$ 
  let  $?x' = \text{inv } T x$ 
  let  $?W' = (\text{ball } ?x' (\min d r) \cap \{y. |(y - ?x')\$k| < e * \min d r / (2 * \text{CARD}('m) \wedge \text{CARD}('m))\})$ 
  have  $\text{abs: } x - e \leq y \wedge y \leq x + e \longleftrightarrow \text{abs}(y - x) \leq e$  for  $x \ y \ e::\text{real}$ 
  by auto
  have  $?W = T \ ` ?W'$ 
  proof -
    have 1:  $T \ ` (\text{ball } (\text{inv } T x) (\min d r)) = \text{ball } x (\min d r)$ 
    by (simp add: T image_orthogonal_transformation_ball orthogonal_
    transformation_surj_surj_f_inv_f)
    have 2:  $\{y. |v \cdot (y - x)| < b * e * \min d r / (2 * \text{real } \text{CARD}('m) \wedge \text{CARD}('m))\} =$ 
     $T \ ` \{y. |y \$ k - ?x' \$ k| < e * \min d r / (2 * \text{real } \text{CARD}('m) \wedge \text{CARD}('m))\}$ 
  proof -
    have *:  $|T (b *_R \text{axis } k 1) \cdot (y - x)| = b * |\text{inv } T y \$ k - ?x' \$ k|$  for
    y
  proof -
    have  $|T (b *_R \text{axis } k 1) \cdot (y - x)| = |(b *_R \text{axis } k 1) \cdot \text{inv } T (y - x)|$ 
    by (metis (no_types, opaque_lifting) b_def eqb invT orthogo-
```

```

nal_transformation_def v)
  also have ... = b * |(axis k 1) • inv T (y - x)|
  using ⟨b > 0⟩ by (simp add: abs_mult)
  also have ... = b * |inv T y $ k - ?x' $ k|
  using orthogonal_transformation_linear [OF invT]
  by (simp add: inner_axis' linear_diff)
  finally show ?thesis
  by simp
qed
show ?thesis
  using v_b_def [symmetric]
  using ⟨b > 0⟩ by (simp add: * bij_image_Collect_eq [OF ⟨bij T⟩]
mult_less_cancel_left_pos times_divide_eq_right [symmetric] del: times_divide_eq_right)
  qed
  show ?thesis
  using ⟨b > 0⟩ by (simp add: image_Int ⟨inj T⟩ 1 2 b_def [symmetric])
  qed
  moreover have ?W' ∈ lmeasurable
  by (auto intro: fmeasurable_Int_fmeasurable)
  ultimately have measure lebesgue ?W = measure lebesgue ?W'
  by (metis measure_orthogonal_image T)
  also have ... ≤ measure lebesgue (cbox (?x' - ?v') (?x' + ?v'))
  proof (rule measure_mono_fmeasurable)
    show ?W' ⊆ cbox (?x' - ?v') (?x' + ?v')
    apply (clarsimp simp add: mem_box_cart abs_dist_norm norm_minus_commute
simp del: min_less_iff_conj min.bounded_iff)
    by (metis component_le_norm_cart less_eq_real_def le_less_trans
vector_minus_component)
  qed auto
  also have ... ≤ e/2 * measure lebesgue (cbox (?x' - ?v) (?x' + ?v))
  proof -
    have cbox (?x' - ?v) (?x' + ?v) ≠ {}
    using ⟨r > 0⟩ ⟨d > 0⟩ by (auto simp: interval_eq_empty_cart
divide_less_0_iff)
    with ⟨r > 0⟩ ⟨d > 0⟩ ⟨e > 0⟩ show ?thesis
    apply (simp add: content_cbox_if_cart mem_box_cart)
    apply (auto simp: prod_nonneg)
    apply (simp add: abs_if_distrib prod.delta_remove field_simps power_diff
split: if_split_asm)
  done
  qed
  also have ... ≤ e/2 * measure lebesgue (cball ?x' (min d r))
  proof (rule mult_left_mono [OF measure_mono_fmeasurable])
    have *: norm (?x' - y) ≤ min d r
    if y: ∧ i. |?x' $ i - y $ i| ≤ min d r / real CARD('m) for y
    proof -
      have norm (?x' - y) ≤ (∑ i ∈ UNIV. |(?x' - y) $ i|)
      by (rule norm_le_l1_cart)
      also have ... ≤ real CARD('m) * (min d r / real CARD('m))

```

```

      by (rule sum_bounded_above) (use y in auto)
    finally show ?thesis
      by simp
  qed
  show cbox (?x' - ?v) (?x' + ?v)  $\subseteq$  cball ?x' (min d r)
    apply (clarsimp simp only: mem_box_cart dist_norm mem_cball
intro!: *)
      by (simp add: abs_diff_le_iff abs_minus_commute)
  qed (use <e > 0> in auto)
  also have ... < e * content (cball ?x' (min d r))
    using <r > 0> <d > 0> <e > 0> by (auto intro: content_cball_pos)
  also have ... = e * content (ball x (min d r))
    using <r > 0> <d > 0> content_ball_conv_unit_ball[of min d r inv T x]
      content_ball_conv_unit_ball[of min d r x]
    by (simp add: content_cball_conv_ball)
  finally show measure lebesgue ?W < e * content (ball x (min d r)) .
  qed
qed
have *: ( $\bigwedge x. (x \notin S) \implies (x \in T \longleftrightarrow x \in U)$ )  $\implies (T - U) \cup (U - T) \subseteq S$ 
for S T U :: (real,'m) vec set
  by blast
have MN: ?M  $\subseteq \{x \in S. \exists v \neq 0. ?\Theta x v\}$ 
proof (rule *)
  fix x
  assume x:  $x \notin \{x \in S. \exists v \neq 0. ?\Theta x v\}$ 
  show  $(x \in ?T) \longleftrightarrow (x \in \{x \in S. \text{matrix } (f' x) \$ m \$ n \leq b\})$ 
  proof (cases x  $\in S$ )
    case True
    then have x:  $\neg ?\Theta x v$  if  $v \neq 0$  for v
      using x that by force
    show ?thesis
  proof (rule iffI; clarsimp)
    assume b:  $\forall e > 0. \exists d > 0. \exists A. A \$ m \$ n < b \wedge (\forall i j. A \$ i \$ j \in \mathbb{Q}) \wedge$ 
       $(\forall y \in S. \text{norm } (y - x) < d \longrightarrow \text{norm } (f y - f x - A$ 
 $* v (y - x)) \leq e * \text{norm } (y - x))$ 
      (is  $\forall e > 0. \exists d > 0. \exists A. ?\Phi e d A$ )
    then have  $\forall k. \exists d > 0. \exists A. ?\Phi (1 / \text{Suc } k) d A$ 
    by (metis (no_types, opaque_lifting) less_Suc_eq_0_disj of_nat_0_less_iff
zero_less_divide_1_iff)
    then obtain  $\delta A$  where  $\delta: \bigwedge k. \delta k > 0$ 
      and Ab:  $\bigwedge k. A k \$ m \$ n < b$ 
      and A:  $\bigwedge k y. \llbracket y \in S; \text{norm } (y - x) < \delta k \rrbracket \implies$ 
 $\text{norm } (f y - f x - A k * v (y - x)) \leq 1 / (\text{Suc } k)$ 
 $* \text{norm } (y - x)$ 
    by metis
    have  $\forall i j. \exists a. (\lambda n. A n \$ i \$ j) \longrightarrow a$ 
    proof (intro allI)
      fix i j
      have vax:  $(A n * v \text{ axis } j 1) \$ i = A n \$ i \$ j$  for n

```

```

    by (metis cart_eq_inner_axis_matrix_vector_mul_component)
  let ?CA = {x. Cauchy (λn. (A n) *v x)}
  have subspace ?CA
    unfolding subspace_def convergent_eq_Cauchy [symmetric]
    by (force simp: algebra_simps intro: tendsto_intros)
  then have CA_eq: ?CA = span ?CA
    by (metis span_eq_iff)
  also have ... = UNIV
  proof -
    have dim ?CA ≤ CARD('m)
      using dim_subset_UNIV[of ?CA]
      by auto
    moreover have False if less: dim ?CA < CARD('m)
    proof -
      obtain d where d ≠ 0 and d: ⋀y. y ∈ span ?CA ⟹ orthogonal d y
        using less by (force intro: orthogonal_to_subspace_exists [of ?CA])
      with x [OF 'd ≠ 0] obtain ξ where ξ > 0
        and ξ: ⋀e. e > 0 ⟹ ∃y ∈ S - {x}. norm (x - y) < e ∧ ξ * norm
(x - y) ≤ |d • (y - x)|
        by (fastforce simp: not_le Bex_def)
      obtain γ z where γSx: ⋀i. γ i ∈ S - {x}
        and γle: ⋀i. ξ * norm(γ i - x) ≤ |d • (γ i - x)|
        and γx: γ ⟶ x
        and z: (λn. (γ n - x) /R norm (γ n - x)) ⟶ z
      proof -
        have ∃γ. (∀i. (γ i ∈ S - {x} ∧
          ξ * norm(γ i - x) ≤ |d • (γ i - x)| ∧ norm(γ i - x)
< 1 / Suc i) ∧
          norm(γ (Suc i) - x) < norm(γ i - x))
        proof (rule dependent_nat_choice)
          show ∃y. y ∈ S - {x} ∧ ξ * norm (y - x) ≤ |d • (y - x)| ∧
norm (y - x) < 1 / Suc 0
            using ξ [of 1] by (auto simp: dist_norm norm_minus_commute)
          next
            fix y i
            assume y ∈ S - {x} ∧ ξ * norm (y - x) ≤ |d • (y - x)| ∧ norm
(y - x) < 1 / Suc i
            then have min (norm(y - x)) (1 / ((Suc i) + 1)) > 0
              by auto
            then obtain y' where y' ∈ S - {x} and y': norm (x - y') <
min (norm (y - x)) (1 / ((Suc i) + 1))
              using ξ by metis
            with ξ show ∃y'. (y' ∈ S - {x} ∧ ξ * norm (y' - x) ≤ |d • (y'
- x)| ∧
              norm (y' - x) < 1 / (Suc (Suc i))) ∧ norm (y' - x) <
norm (y - x)
              by (auto simp: dist_norm norm_minus_commute)
          qed
      qed
    qed
  qed

```

```

    then obtain  $\gamma$  where
       $\gamma Sx$ :  $\bigwedge i. \gamma i \in S - \{x\}$ 
      and  $\gamma le$ :  $\bigwedge i. \xi * \text{norm}(\gamma i - x) \leq |d \cdot (\gamma i - x)|$ 
      and  $\gamma conv$ :  $\bigwedge i. \text{norm}(\gamma i - x) < 1/(\text{Suc } i)$ 
    by blast
    let  $?f = \lambda i. (\gamma i - x) /_R \text{norm}(\gamma i - x)$ 
    have  $?f i \in \text{sphere } 0 \ 1$  for  $i$ 
      using  $\gamma Sx$  by auto
    then obtain  $l$   $\varrho$  where  $l \in \text{sphere } 0 \ 1$  strict_mono  $\varrho$  and  $l$ :  $(?f \circ$ 
 $\varrho) \longrightarrow l$ 
    using compact_sphere [of  $0::(\text{real}, 'm)$  vec 1] unfolding compact_def
  by meson

  show thesis
  proof
    show  $(\gamma \circ \varrho) i \in S - \{x\} \ \xi * \text{norm}((\gamma \circ \varrho) i - x) \leq |d \cdot ((\gamma \circ$ 
 $\varrho) i - x)|$  for  $i$ 
      using  $\gamma Sx \ \gamma le$  by auto
      have  $\gamma \longrightarrow x$ 
      proof (clarsimp simp add: LIMSEQ_def dist_norm)
        fix  $r :: \text{real}$ 
        assume  $r > 0$ 
        with real_arch_invD obtain  $no$  where  $no \neq 0$  real  $no > 1/r$ 
        by (metis divide_less_0_1_iff not_less_iff_gr_or_eq
of_nat_0_eq_iff reals_Archimedean2)
        with  $\gamma conv$  show  $\exists no. \forall n \geq no. \text{norm}(\gamma n - x) < r$ 
        by (metis  $\langle r > 0 \rangle$  add.commute divide_inverse inverse_inverse_eq
inverse_less_imp_less less_trans mult.left_neutral nat_le_real_less of_nat_Suc)
      qed
      with  $\langle \text{strict\_mono } \varrho \rangle$  show  $(\gamma \circ \varrho) \longrightarrow x$ 
      by (metis LIMSEQ_subseq LIMSEQ)
      show  $(\lambda n. ((\gamma \circ \varrho) n - x) /_R \text{norm}((\gamma \circ \varrho) n - x)) \longrightarrow l$ 
      using  $l$  by (auto simp: o_def)
    qed
  qed
  have isCont  $(\lambda x. (|d \cdot x| - \xi)) \ z$ 
    by (intro continuous_intros)
  from isCont_tendsto_compose [OF this z]
  have lim:  $(\lambda y. |d \cdot ((\gamma y - x) /_R \text{norm}(\gamma y - x))| - \xi) \longrightarrow |d \cdot$ 
 $z| - \xi$ 
    by auto
  moreover have  $\forall_F i$  in sequentially.  $0 \leq |d \cdot ((\gamma i - x) /_R \text{norm}$ 
 $(\gamma i - x))| - \xi$ 
  proof (rule eventuallyI)
    fix  $n$ 
    show  $0 \leq |d \cdot ((\gamma n - x) /_R \text{norm}(\gamma n - x))| - \xi$ 
    using  $\gamma le$  [of  $n$ ]  $\gamma Sx$  by (auto simp: abs_mult divide_simps)
  qed
  ultimately have  $\xi \leq |d \cdot z|$ 
    using tendsto_lowerbound [where  $a=0$ ] by fastforce

```

```

have Cauchy (λn. (A n) *v z)
proof (clarsimp simp add: Cauchy_def)
  fix ε :: real
  assume 0 < ε
  then obtain N::nat where N > 0 and N: ε/2 > 1/N
by (metis half_gt_zero inverse_eq_divide neq0_conv real_arch_inverse)
show ∃ M. ∀ m ≥ M. ∀ n ≥ M. dist (A m *v z) (A n *v z) < ε
proof (intro exI allI impI)
  fix i j
  assume ij: N ≤ i N ≤ j
  let ?V = λi k. A i *v ((γ k - x) /R norm (γ k - x))
  have ∀_F k in sequentially. dist (γ k) x < min (δ i) (δ j)
  using γx [unfolded tendsto_iff] by (meson min_less_iff_conj δ)
  then have even: ∀_F k in sequentially. norm (?V i k - ?V j k) -
2 / N ≤ 0
proof (rule eventually_mono, clarsimp)
  fix p
  assume p: dist (γ p) x < δ i dist (γ p) x < δ j
  let ?C = λk. f (γ p) - f x - A k *v (γ p - x)
  have norm ((A i - A j) *v (γ p - x)) = norm (?C j - ?C i)
  by (simp add: algebra_simps)
  also have ... ≤ norm (?C j) + norm (?C i)
  using norm_triangle_ineq4 by blast
  also have ... ≤ 1/(Suc j) * norm (γ p - x) + 1/(Suc i) *
norm (γ p - x)
  by (metis A Diff_iff γSx dist_norm p add_mono)
  also have ... ≤ 1/N * norm (γ p - x) + 1/N * norm (γ p -
x)

  apply (intro add_mono mult_right_mono)
  using ij ⟨N > 0⟩ by (auto simp: field_simps)
  also have ... = 2 / N * norm (γ p - x)
  by simp
  finally have no_le: norm ((A i - A j) *v (γ p - x)) ≤ 2 / N
* norm (γ p - x) .
  have norm (?V i p - ?V j p) =
    norm ((A i - A j) *v ((γ p - x) /R norm (γ p - x)))
  by (simp add: algebra_simps)
  also have ... = norm ((A i - A j) *v (γ p - x)) / norm (γ p
- x)

  by (simp add: divide_inverse matrix_vector_mult_scaleR)
  also have ... ≤ 2 / N
  using no_le by (auto simp: field_split_simps)
  finally show norm (?V i p - ?V j p) ≤ 2 / N .
qed
have isCont (λw. (norm(A i *v w - A j *v w) - 2 / N)) z
  by (intro continuous_intros)
from isCont_tendsto_compose [OF this z]
have lim: (λw. norm (A i *v ((γ w - x) /R norm (γ w - x)) -
A j *v ((γ w - x) /R norm (γ w - x))) - 2 / N)

```

```

       $\longrightarrow \text{norm } (A \ i \ *v \ z - A \ j \ *v \ z) - 2 / N$ 
    by auto
  have  $\text{dist } (A \ i \ *v \ z) (A \ j \ *v \ z) \leq 2 / N$ 
    using tendsto_upperbound [OF lim even] by (auto simp:
dist_norm)
  with N show  $\text{dist } (A \ i \ *v \ z) (A \ j \ *v \ z) < \varepsilon$ 
    by linarith
  qed
  qed
  then have  $d \cdot z = 0$ 
    using CA_eq d orthogonal_def by auto
  then show False
    using  $\langle 0 < \xi \rangle \langle \xi \leq |d \cdot z| \rangle$  by auto
  qed
  ultimately show ?thesis
    using dim_eq_full by fastforce
  qed
  finally have ?CA = UNIV .
  then have Cauchy  $(\lambda n. (A \ n) \ *v \ \text{axis } j \ 1)$ 
    by auto
  then obtain L where  $(\lambda n. A \ n \ *v \ \text{axis } j \ 1) \longrightarrow L$ 
    by (auto simp: Cauchy_convergent_iff convergent_def)
  then have  $(\lambda x. (A \ x \ *v \ \text{axis } j \ 1) \ \$ \ i) \longrightarrow L \ \$ \ i$ 
    by (rule tendsto_vec_nth)
  then show  $\exists a. (\lambda n. A \ n \ \$ \ i \ \$ \ j) \longrightarrow a$ 
    by (force simp: va x)
  qed
  then obtain B where  $B: \bigwedge i \ j. (\lambda n. A \ n \ \$ \ i \ \$ \ j) \longrightarrow B \ \$ \ i \ \$ \ j$ 
    by (auto simp: lambda_skolem)
  have lin_df: linear  $(f' \ x)$ 
    and lim_df:  $((\lambda y. (1 / \text{norm } (y - x)) *_{\mathbb{R}} (f \ y - (f \ x + f' \ x \ (y - x)))) \longrightarrow 0) \text{ (at } x \text{ within } S)$ 
    using  $\langle x \in S \rangle$  assms by (auto simp: has_derivative_within linear_linear)
  moreover
  interpret linear  $f' \ x$  by fact
  have  $(\text{matrix } (f' \ x) - B) *v \ w = 0 \text{ for } w$ 
  proof (rule lemma_partial_derivatives [of  $(*v) (\text{matrix } (f' \ x) - B)$ ])
    show linear  $((*v) (\text{matrix } (f' \ x) - B))$ 
      by (rule matrix_vector_mul_linear)
    have  $((\lambda y. ((f \ x + f' \ x \ (y - x)) - f \ y) /_{\mathbb{R}} \text{norm } (y - x)) \longrightarrow 0) \text{ (at } x \text{ within } S)$ 
      using tendsto_minus [OF lim_df] by (simp add: field_split_simps)
    then show  $((\lambda y. (\text{matrix } (f' \ x) - B) *v (y - x) /_{\mathbb{R}} \text{norm } (y - x)) \longrightarrow 0) \text{ (at } x \text{ within } S)$ 
  proof (rule Lim_transform)
    have  $((\lambda y. ((f \ y + B *v x - (f \ x + B *v y)) /_{\mathbb{R}} \text{norm } (y - x))) \longrightarrow 0) \text{ (at } x \text{ within } S)$ 
  proof (clarsimp simp add: Lim_within dist_norm)
    fix e :: real

```

```

    assume e > 0
    then obtain q::nat where q ≠ 0 and qe2: 1/q < e/2
      by (metis divide_pos_pos inverse_eq_divide real_arch_inverse
zero_less_numeral)
    let ?g = λp. sum (λi. sum (λj. abs((A p - B)$i$j)) UNIV) UNIV
    have (λk. onorm (λy. (A k - B) *v y)) → 0
    proof (rule Lim_null_comparison)
      show ∀F k in sequentially. norm (onorm (λy. (A k - B) *v y)) ≤
?g k

    proof (rule eventually_sequentiallyI)
      fix k :: nat
      assume 0 ≤ k
      have 0 ≤ onorm ((*v) (A k - B))
      using matrix_vector_mul_bounded_linear
      by (rule onorm_pos_le)
      then show norm (onorm ((*v) (A k - B))) ≤ (∑ i∈UNIV.
∑ j∈UNIV. |(A k - B) $ i $ j|)
      by (simp add: onorm_le_matrix_component_sum del:
vector_minus_component)
    qed
  next
    show ?g → 0
      using B Lim_null tendsto_rabs_zero_iff by (fastforce intro!:
tendsto_null_sum)
    qed
    with ⟨e > 0⟩ obtain p where ∧n. n ≥ p ⇒ |onorm ((*v) (A n -
B))| < e/2
      unfolding lim_sequentially by (metis diff_zero dist_real_def
divide_pos_pos zero_less_numeral)
    then have pqe2: |onorm ((*v) (A (p + q) - B))| < e/2
      using le_add1 by blast
    show ∃ d>0. ∀ y∈S. y ≠ x ∧ norm (y - x) < d →
      inverse (norm (y - x)) * norm (f y + B *v x - (f x + B *v
y)) < e
    proof (intro exI, safe)
      show 0 < δ(p + q)
      by (simp add: δ)
    next
      fix y
      assume y: y ∈ S norm (y - x) < δ(p + q) and y ≠ x
      have *: [|norm(b - c) < e - d; norm(y - x - b) ≤ d|] ⇒ norm(y
- x - c) < e
      for b c d e x and y:: realn
      using norm_triangle_ineq2 [of y - x - c y - x - b] by simp
      have norm (f y - f x - B *v (y - x)) < e * norm (y - x)
      proof (rule *)
        show norm (f y - f x - A (p + q) *v (y - x)) ≤ norm (y - x)
/ (Suc (p + q))
        using A [OF y] by simp

```



```

      have norm (A (p + q) *v (y - x) - B *v (y - x)) ≤ onorm(λx.
(A(p + q) - B) *v x) * norm(y - x)
      by (metis linear_linear matrix_vector_mul_linear ma-
trix_vector_mult_diff_rdistrib onorm)
      also have ... < (e/2) * norm (y - x)
      using ‹y ≠ x› pge2 by auto
      also have ... ≤ (e - 1 / (Suc (p + q))) * norm (y - x)
      proof (rule mult_right_mono)
        have 1 / Suc (p + q) ≤ 1 / q
        using ‹q ≠ 0› by (auto simp: field_split_simps)
        also have ... < e/2
        using qe2 by auto
        finally show e / 2 ≤ e - 1 / real (Suc (p + q))
        by linarith
      qed auto
      finally show norm (A (p + q) *v (y - x) - B *v (y - x)) < e *
norm (y - x) - norm (y - x) / real (Suc (p + q))
      by (simp add: algebra_simps)
    qed
    then show inverse (norm (y - x)) * norm (f y + B *v x - (f x +
B *v y)) < e
      using ‹y ≠ x› by (simp add: field_split_simps algebra_simps)
    qed
  qed
  then show ((λy. (matrix (f' x) - B) *v (y - x) /R
norm (y - x) - (f x + f' x (y - x) - f y) /R norm (y -
x)) → 0)
    (at x within S)
    by (simp add: algebra_simps diff lin_df scalar_mult_eq_scaleR)
  qed
qed (use x in ‹simp; auto simp: not_less›)
ultimately have f' x = (*v) B
  by (force simp: algebra_simps scalar_mult_eq_scaleR)
show matrix (f' x) $ m $ n ≤ b
proof (rule tendsto_upperbound [of λi. (A i $ m $ n) _ sequentially])
  show (λi. A i $ m $ n) → matrix (f' x) $ m $ n
  by (simp add: B ‹f' x = (*v) B›)
  show ∀F i in sequentially. A i $ m $ n ≤ b
  by (simp add: Ab less_eq_real_def)
qed auto
next
fix e :: real
assume x ∈ S and b: matrix (f' x) $ m $ n ≤ b and e > 0
then obtain d where d>0
  and d: ∧y. y∈S ⇒ 0 < dist y x ∧ dist y x < d → norm (f y - f x
- f' x (y - x)) / (norm (y - x))
    < e/2
  using f [OF ‹x ∈ S›]
  by (simp add: Deriv.has_derivative_at_within Lim_within)

```

```

      (auto simp add: field_simps dest: spec [of _ e/2])
    let ?A = matrix(f' x) - (χ i j. if i = m ∧ j = n then e / 4 else 0)
    obtain B where BRats: ∧ i j. B$ i $ j ∈ ℚ and Bo_e6: onorm((∗v) (?A -
B)) < e/6
      using matrix_rational_approximation ⟨e > 0⟩
      by (metis zero_less_divide_iff zero_less_numeral)
    show ∃ d > 0. ∃ A. A $ m $ n < b ∧ (∀ i j. A $ i $ j ∈ ℚ) ∧
      (∀ y ∈ S. norm (y - x) < d ⟶ norm (f y - f x - A ∗ v (y - x)) ≤ e
* norm (y - x))
    proof (intro exI conjI ballI allI impI)
      show d > 0
        by (rule ⟨d > 0⟩)
      show B $ m $ n < b
      proof -
        have |matrix ((∗v) (?A - B)) $ m $ n| ≤ onorm ((∗v) (?A - B))
          using component_le_onorm [OF matrix_vector_mul_linear, of _ m
n] by metis
        then show ?thesis
          using b Bo_e6 by simp
      qed
      show B $ i $ j ∈ ℚ for i j
        using BRats by auto
      show norm (f y - f x - B ∗ v (y - x)) ≤ e * norm (y - x)
        if y ∈ S and y: norm (y - x) < d for y
      proof (cases y = x)
        case True then show ?thesis
          by simp
      next
        case False
          have *: norm(d' - d) ≤ e/2 ⟹ norm(y - (x + d')) < e/2 ⟹
norm(y - x - d) ≤ e for d d' e and x y :: real^n
            using norm_triangle_le [of d' - d y - (x + d')] by simp
          show ?thesis
            proof (rule *)
              have split246: [norm y ≤ e / 6; norm(x - y) ≤ e / 4] ⟹ norm x
≤ e/2 if e > 0 for e and x y :: real^n
                using norm_triangle_le [of y x - y e/2] ⟨e > 0⟩ by simp
              have linear (f' x)
                using True f has_derivative_linear by blast
              then have norm (f' x (y - x) - B ∗ v (y - x)) = norm ((matrix (f'
x) - B) ∗ v (y - x))
                by (simp add: matrix_vector_mult_diff_rdistrib)
              also have ... ≤ (e * norm (y - x)) / 2
                proof (rule split246)
                  have norm ((?A - B) ∗ v (y - x)) / norm (y - x) ≤ onorm(λx.
(?A - B) ∗ v x)
                    by (rule le_onorm) auto
                  also have ... < e/6
                    by (rule Bo_e6)

```

```

finally have norm ((?A - B) *v (y - x)) / norm (y - x) < e / 6 .
then show norm ((?A - B) *v (y - x)) ≤ e * norm (y - x) / 6
  by (simp add: field_split_simps False)
have norm ((matrix (f' x) - B) *v (y - x) - ((?A - B) *v (y -
x))) = norm ((χ i j. if i = m ∧ j = n then e / 4 else 0) *v (y - x))
  by (simp add: algebra_simps)
also have ... = norm((e/4) *R (y - x)$n *R axis m (1::real))
proof -
  have (∑ j∈UNIV. (if i = m ∧ j = n then e / 4 else 0) * (y $ j
- x $ j)) * 4 = e * (y $ n - x $ n) * axis m 1 $ i for i
  proof (cases i=m)
    case True then show ?thesis
      by (auto simp: if_distrib [of λz. z * _] cong: if_cong)
    next
      case False then show ?thesis
        by (simp add: axis_def)
  qed
then have (χ i j. if i = m ∧ j = n then e / 4 else 0) *v (y - x)
= (e/4) *R (y - x)$n *R axis m (1::real)
  by (auto simp: vec_eq_iff matrix_vector_mult_def)
then show ?thesis
  by metis
qed
also have ... ≤ e * norm (y - x) / 4
  using ⟨e > 0⟩ apply (simp add: norm_mult abs_mult)
  by (metis component_le_norm_cart vector_minus_component)
finally show norm ((matrix (f' x) - B) *v (y - x) - ((?A - B)
*v (y - x))) ≤ e * norm (y - x) / 4 .
  show 0 < e * norm (y - x)
    by (simp add: False ⟨e > 0⟩)
  qed
finally show norm (f' x (y - x) - B *v (y - x)) ≤ (e * norm (y -
x)) / 2 .
  show norm (f y - (f x + f' x (y - x))) < (e * norm (y - x)) / 2
    using False d [OF ⟨y ∈ S⟩] y by (simp add: dist_norm field_simps)
  qed
qed
qed
qed
qed
qed auto
qed
show negligible ?M
  using negligible_subset [OF nN MN] .
qed
then show ?thesis
  by (simp add: borel_measurable_vimage_halfspace_component_le sets_restrict_space_iff
assms)
qed

```

theorem *borel_measurable_det_Jacobian*:
fixes $f :: \text{real}^n :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n :: \_$
assumes $S: S \in \text{sets lebesgue}$ **and** $f: \bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } S)$
shows $(\lambda x. \det(\text{matrix}(f' x))) \in \text{borel\_measurable } (\text{lebesgue\_on } S)$
unfolding *det_def*
by (*intro measurable*) (*auto intro: f borel_measurable_partial_derivatives [OF S]*)

The localisation wrt S uses the same argument for many similar results.

theorem *borel_measurable_lebesgue_on_preimage_borel*:
fixes $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$
assumes $S \in \text{sets lebesgue}$
shows $f \in \text{borel\_measurable } (\text{lebesgue\_on } S) \longleftrightarrow$
 $(\forall T. T \in \text{sets borel} \longrightarrow \{x \in S. f x \in T\} \in \text{sets lebesgue})$
proof –
have $\{x. (\text{if } x \in S \text{ then } f x \text{ else } 0) \in T\} \in \text{sets lebesgue} \longleftrightarrow \{x \in S. f x \in T\} \in \text{sets lebesgue}$
if $T \in \text{sets borel}$ **for** T
proof (*cases* $0 \in T$)
case *True*
then have $\{x \in S. f x \in T\} = \{x. (\text{if } x \in S \text{ then } f x \text{ else } 0) \in T\} \cap S$
 $\{x. (\text{if } x \in S \text{ then } f x \text{ else } 0) \in T\} = \{x \in S. f x \in T\} \cup -S$
by *auto*
then show *?thesis*
by (*metis (no_types, lifting) Compl_in_sets_lebesgue assms sets.Int sets.Un*)
next
case *False*
then have $\{x. (\text{if } x \in S \text{ then } f x \text{ else } 0) \in T\} = \{x \in S. f x \in T\}$
by *auto*
then show *?thesis*
by *auto*
qed
then show *?thesis*
unfolding *borel_measurable_lebesgue_preimage_borel borel_measurable_if*
 $[OF \text{ assms, symmetric}]$
by *blast*
qed

lemma *sets_lebesgue_almost_borel*:
assumes $S \in \text{sets lebesgue}$
obtains $B N$ **where** $B \in \text{sets borel negligible}$ $N B \cup N = S$
proof –
obtain $T N N'$ **where** $S = T \cup N$ $N \subseteq N'$ $N' \in \text{null\_sets lborel}$ $T \in \text{sets borel}$
using *sets_completionE* $[OF \text{ assms}]$ **by** *auto*
then show *thesis*
by (*metis negligible_iff_null_sets negligible_subset null_sets_completionI* *that*)
qed

```

lemma double_lebesgue_sets:
  assumes  $S: S \in \text{sets lebesgue}$  and  $T: T \in \text{sets lebesgue}$  and  $fim: f \text{ ' } S \subseteq T$ 
  shows  $(\forall U. U \in \text{sets lebesgue} \wedge U \subseteq T \longrightarrow \{x \in S. f x \in U\} \in \text{sets lebesgue})$ 
 $\longleftrightarrow$ 
     $f \in \text{borel\_measurable} (\text{lebesgue\_on } S) \wedge$ 
     $(\forall U. \text{negligible } U \wedge U \subseteq T \longrightarrow \{x \in S. f x \in U\} \in \text{sets lebesgue})$ 
    (is ?lhs  $\longleftrightarrow$  _  $\wedge$  ?rhs)
  unfolding borel_measurable_lebesgue_on_preimage_borel [OF S]
proof (intro iffI allI conjI impI, safe)
  fix  $V :: 'b \text{ set}$ 
  assume *:  $\forall U. U \in \text{sets lebesgue} \wedge U \subseteq T \longrightarrow \{x \in S. f x \in U\} \in \text{sets lebesgue}$ 
    and  $V \in \text{sets borel}$ 
  then have  $V: V \in \text{sets lebesgue}$ 
    by simp
  have  $\{x \in S. f x \in V\} = \{x \in S. f x \in T \cap V\}$ 
    using fim by blast
  also have  $\{x \in S. f x \in T \cap V\} \in \text{sets lebesgue}$ 
    using T V * le_inf_iff by blast
  finally show  $\{x \in S. f x \in V\} \in \text{sets lebesgue} .$ 
next
  fix  $U :: 'b \text{ set}$ 
  assume  $\forall U. U \in \text{sets lebesgue} \wedge U \subseteq T \longrightarrow \{x \in S. f x \in U\} \in \text{sets lebesgue}$ 
     $\text{negligible } U \wedge U \subseteq T$ 
  then show  $\{x \in S. f x \in U\} \in \text{sets lebesgue}$ 
    using negligible_imp_sets by blast
next
  fix  $U :: 'b \text{ set}$ 
  assume 1 [rule_format]:  $(\forall T. T \in \text{sets borel} \longrightarrow \{x \in S. f x \in T\} \in \text{sets lebesgue})$ 
    and 2 [rule_format]:  $\forall U. \text{negligible } U \wedge U \subseteq T \longrightarrow \{x \in S. f x \in U\} \in \text{sets lebesgue}$ 
    and  $U \in \text{sets lebesgue} \wedge U \subseteq T$ 
  then obtain C N where  $C: C \in \text{sets borel} \wedge \text{negligible } N \wedge C \cup N = U$ 
    using sets_lebesgue_almost_borel
    bymetis
  then have  $\{x \in S. f x \in C\} \in \text{sets lebesgue}$ 
    by (blast intro: 1)
  moreover have  $\{x \in S. f x \in N\} \in \text{sets lebesgue}$ 
    using C  $\langle U \subseteq T \rangle$  by (blast intro: 2)
  moreover have  $\{x \in S. f x \in C \cup N\} = \{x \in S. f x \in C\} \cup \{x \in S. f x \in N\}$ 
    by auto
  ultimately show  $\{x \in S. f x \in U\} \in \text{sets lebesgue}$ 
    using C by auto
qed

```

6.47.3 Simplest case of Sard's theorem (we don't need continuity of derivative)

```

lemma Sard_lemma00:
  fixes P :: 'b::euclidean_space set
  assumes a ≥ 0 and a: a *R i ≠ 0 and i: i ∈ Basis
    and P: P ⊆ {x. a *R i • x = 0}
    and 0 ≤ m 0 ≤ e
  obtains S where S ∈ lmeasurable
    and {z. norm z ≤ m ∧ (∃ t ∈ P. norm(z - t) ≤ e)} ⊆ S
    and measure lebesgue S ≤ (2 * e) * (2 * m) ^ (DIM('b) - 1)
proof -
  have a > 0
    using assms by simp
  let ?v = (∑ j ∈ Basis. (if j = i then e else m) *R j)
  show thesis
  proof
    have - e ≤ x • i x • i ≤ e
      if t ∈ P norm (x - t) ≤ e for x t
      using ⟨a > 0⟩ that Basis_le_norm [of i x-t] P i
      by (auto simp: inner_commute algebra_simps)
    moreover have - m ≤ x • j x • j ≤ m
      if norm x ≤ m t ∈ P norm (x - t) ≤ e j ∈ Basis and j ≠ i
      for x t j
      using that Basis_le_norm [of j x] by auto
    ultimately
    show {z. norm z ≤ m ∧ (∃ t ∈ P. norm (z - t) ≤ e)} ⊆ cbox (-?v) ?v
      by (auto simp: mem_box)
    have *: ∀ k ∈ Basis. - ?v • k ≤ ?v • k
      using ⟨0 ≤ m⟩ ⟨0 ≤ e⟩ by (auto simp: inner_Basis)
    have 2: 2 ^ DIM('b) = 2 * 2 ^ (DIM('b) - Suc 0)
      by (metis DIM_positive Suc_pred power_Suc)
    show measure lebesgue (cbox (-?v) ?v) ≤ 2 * e * (2 * m) ^ (DIM('b) - 1)
      using ⟨i ∈ Basis⟩
      by (simp add: content_cbox [OF *] prod.distrib prod.If_cases Diff_eq [symmetric]
        2)
  qed blast
qed

```

As above, but reorienting the vector (HOL Light's @textGEOM_BASIS_MULTIPLE_TAC)

```

lemma Sard_lemma0:
  fixes P :: (real^'n::{finite,wellorder}) set
  assumes a ≠ 0
    and P: P ⊆ {x. a • x = 0} and 0 ≤ m 0 ≤ e
  obtains S where S ∈ lmeasurable
    and {z. norm z ≤ m ∧ (∃ t ∈ P. norm(z - t) ≤ e)} ⊆ S
    and measure lebesgue S ≤ (2 * e) * (2 * m) ^ (CARD('n) - 1)
proof -

```

```

  obtain T and k::'n where T: orthogonal_transformation T and a: a = T
    (norm a *R axis k (1::real))
  using rotation_rightward_line by metis
  have Tinv [simp]: T (inv T x) = x for x
    by (simp add: T orthogonal_transformation_surj surj_f_inv_f)
  obtain S where S: S ∈ lmeasurable
    and subS: {z. norm z ≤ m ∧ (∃ t ∈ T - 'P. norm (z - t) ≤ e)} ⊆ S
    and mS: measure lebesgue S ≤ (2 * e) * (2 * m) ^ (CARD('n) - 1)
  proof (rule Sard_lemma00 [of norm a axis k (1::real) T - 'P m e])
    have norm a *R axis k 1 · x = 0 if T x ∈ P for x
    proof -
      have a · T x = 0
      using P that by blast
    then show ?thesis
      by (metis (no_types, lifting) T a orthogonal_orthogonal_transformation
        orthogonal_def)
    qed
    then show T - 'P ⊆ {x. norm a *R axis k 1 · x = 0}
      by auto
    qed (use assms T in auto)
  show thesis
  proof
    show T - 'S ∈ lmeasurable
      using S measurable_orthogonal_image T by blast
    have {z. norm z ≤ m ∧ (∃ t ∈ P. norm (z - t) ≤ e)} ⊆ T - ' {z. norm z ≤ m ∧
      (∃ t ∈ T - 'P. norm (z - t) ≤ e)}
    proof clarsimp
      fix x t
      assume norm x ≤ m t ∈ P norm (x - t) ≤ e
      then have norm (inv T x) ≤ m
      using orthogonal_transformation_inv [OF T] by (simp add: orthogonal_
        transformation_norm)
      moreover have ∃ t ∈ T - 'P. norm (inv T x - t) ≤ e
      proof
        have T (inv T x - inv T t) = x - t
        using T linear_diff orthogonal_transformation_def
        by (metis (no_types, opaque_lifting) Tinv)
        then have norm (inv T x - inv T t) = norm (x - t)
        by (metis T orthogonal_transformation_norm)
        then show norm (inv T x - inv T t) ≤ e
        using <norm (x - t) ≤ e> by linarith
      next
        show inv T t ∈ T - 'P
        using <t ∈ P> by force
      qed
    qed
    ultimately show x ∈ T - ' {z. norm z ≤ m ∧ (∃ t ∈ T - 'P. norm (z - t) ≤
      e)}
    by force
  qed

```

```

then show  $\{z. \text{norm } z \leq m \wedge (\exists t \in P. \text{norm } (z - t) \leq e)\} \subseteq T \text{ ' } S$ 
  using image_mono [OF subS] by (rule order_trans)
show measure lebesgue  $(T \text{ ' } S) \leq 2 * e * (2 * m) ^ (CARD('n) - 1)$ 
  using mS T by (simp add: S measure_orthogonal_image)
qed
qed

```

As above, but translating the sets (HOL Light's @textGEN_GEOM_ORIGIN_TAC)

```

lemma Sard_lemma1:
  fixes P ::  $(\text{real}^n::\{\text{finite}, \text{wellorder}\}) \text{ set}$ 
  assumes P:  $\dim P < CARD('n)$  and  $0 \leq m$   $0 \leq e$ 
  obtains S where  $S \in \text{lmeasurable}$ 
    and  $\{z. \text{norm}(z - w) \leq m \wedge (\exists t \in P. \text{norm}(z - w - t) \leq e)\} \subseteq S$ 
    and  $\text{measure lebesgue } S \leq (2 * e) * (2 * m) ^ (CARD('n) - 1)$ 
proof -
  obtain a where  $a \neq 0$   $P \subseteq \{x. a \cdot x = 0\}$ 
    using lowdim_subset_hyperplane [of P] P span_base by auto
  then obtain S where  $S: S \in \text{lmeasurable}$ 
    and subS:  $\{z. \text{norm } z \leq m \wedge (\exists t \in P. \text{norm}(z - t) \leq e)\} \subseteq S$ 
    and mS:  $\text{measure lebesgue } S \leq (2 * e) * (2 * m) ^ (CARD('n) - 1)$ 
    by (rule Sard_lemma0 [OF _ _  $\langle 0 \leq m \rangle \langle 0 \leq e \rangle$ ])
  show thesis
proof
  show  $(+)w \text{ ' } S \in \text{lmeasurable}$ 
    by (metis measurable_translation S)
  show  $\{z. \text{norm } (z - w) \leq m \wedge (\exists t \in P. \text{norm } (z - w - t) \leq e)\} \subseteq (+)w \text{ ' } S$ 
    using subS by force
  show  $\text{measure lebesgue } ((+)w \text{ ' } S) \leq 2 * e * (2 * m) ^ (CARD('n) - 1)$ 
    by (metis measure_translation mS)
qed
qed

```

```

lemma Sard_lemma2:
  fixes f ::  $\text{real}^m::\{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n::\{\text{finite}, \text{wellorder}\}$ 
  assumes mlen:  $CARD('m) \leq CARD('n)$  (is ?m  $\leq$  ?n)
    and B > 0 bounded S
    and derS:  $\bigwedge x. x \in S \Rightarrow (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } S)$ 
    and rank:  $\bigwedge x. x \in S \Rightarrow \text{rank}(\text{matrix}(f' x)) < CARD('n)$ 
    and B:  $\bigwedge x. x \in S \Rightarrow \text{onorm}(f' x) \leq B$ 
  shows negligible(f ' S)
proof -
  have lin_f':  $\bigwedge x. x \in S \Rightarrow \text{linear}(f' x)$ 
    using derS has_derivative_linear by blast
  show ?thesis
proof (clarsimp simp add: negligible_outer_le)
  fix e :: real
  assume e > 0
  obtain c where csub:  $S \subseteq \text{cbox } (- (\text{vec } c)) (\text{vec } c)$  and c > 0

```



```

proof -
  obtain b where b:  $\bigwedge x. x \in S \implies \text{norm } x \leq b$ 
    using  $\langle \text{bounded } S \rangle$  by (auto simp: bounded_iff)
  show thesis
  proof
    have  $-|b| - 1 \leq x \ \$ \ i \wedge x \ \$ \ i \leq |b| + 1$  if  $x \in S$  for  $x \ i$ 
      using component_le_norm_cart [of  $x \ i$ ] b [OF that] by auto
    then show  $S \subseteq \text{cbox } (- \text{vec } (|b| + 1)) (\text{vec } (|b| + 1))$ 
      by (auto simp: mem_box_cart)
    qed auto
  qed
  then have box_cc:  $\text{box } (- (\text{vec } c)) (\text{vec } c) \neq \{\}$  and cbox_cc:  $\text{cbox } (- (\text{vec } c)) (\text{vec } c) \neq \{\}$ 
    by (auto simp: interval_eq_empty_cart)
  obtain d where  $d > 0 \ d \leq B$ 
    and  $d: (d * 2) * (4 * B) \wedge (?n - 1) \leq e / (2 * c) \wedge ?m / ?m \wedge ?m$ 
    apply (rule that [of min B (e / (2 * c)  $\wedge ?m / ?m \wedge ?m / (4 * B) \wedge (?n - 1) / 2$ )]))
    using  $\langle B > 0 \rangle \langle c > 0 \rangle \langle e > 0 \rangle$ 
    by (simp_all add: divide_simps min_mult_distrib_right)
  have  $\exists r. 0 < r \wedge r \leq 1/2 \wedge$ 
    ( $x \in S$ 
       $\longrightarrow (\forall y. y \in S \wedge \text{norm}(y - x) < r$ 
         $\longrightarrow \text{norm}(f y - f x - f' x (y - x)) \leq d * \text{norm}(y - x))$ ) for  $x$ 
  proof (cases  $x \in S$ )
  case True
    then obtain r where  $r > 0$ 
      and  $\bigwedge y. \llbracket y \in S; \text{norm } (y - x) < r \rrbracket$ 
         $\implies \text{norm } (f y - f x - f' x (y - x)) \leq d * \text{norm } (y - x)$ 
      using derS  $\langle d > 0 \rangle$  by (force simp: has_derivative_within_alt)
    then show ?thesis
      by (rule_tac  $x = \min r (1/2)$  in exI) simp
  next
  case False
    then show ?thesis
      by (rule_tac  $x = 1/2$  in exI) simp
  qed
  then obtain r where r12:  $\bigwedge x. 0 < r \wedge r \leq 1/2$ 
    and  $r: \bigwedge x y. \llbracket x \in S; y \in S; \text{norm}(y - x) < r \rrbracket$ 
       $\implies \text{norm}(f y - f x - f' x (y - x)) \leq d * \text{norm}(y - x)$ 
    by metis
  then have ga: gauge ( $\lambda x. \text{ball } x (r x)$ )
    by (auto simp: gauge_def)
  obtain  $\mathcal{D}$  where  $\mathcal{D}$ : countable  $\mathcal{D}$  and sub_cc:  $\bigcup \mathcal{D} \subseteq \text{cbox } (- \text{vec } c) (\text{vec } c)$ 
    and cbox:  $\bigwedge K. K \in \mathcal{D} \implies \text{interior } K \neq \{\}$   $\wedge (\exists u v. K = \text{cbox } u v)$ 
    and disjointish: pairwise ( $\lambda A B. \text{interior } A \cap \text{interior } B = \{\}$ )  $\mathcal{D}$ 
    and covered:  $\bigwedge K. K \in \mathcal{D} \implies \exists x \in S \cap K. K \subseteq \text{ball } x (r x)$ 
    and close:  $\bigwedge u v. \text{cbox } u v \in \mathcal{D} \implies \exists n. \forall i::'m. v \ \$ \ i - u \ \$ \ i = 2 * c / 2^n$ 
    and covers:  $S \subseteq \bigcup \mathcal{D}$ 

```

```

    apply (rule covering_lemma [OF csub box_cc ga])
    apply (auto simp: Basis_vec_def cart_eq_inner_axis [symmetric])
  done
let ?μ = measure lebesgue
have ∃ T. T ∈ lmeasurable ∧ f' (K ∩ S) ⊆ T ∧ ?μ T ≤ e / (2*c) ^ ?m * ?μ
K
  if K ∈ D for K
proof -
  obtain u v where uv: K = cbox u v
  using cbox ⟨K ∈ D⟩ by blast
  then have uv_ne: cbox u v ≠ {}
  using cbox that by fastforce
  obtain x where x: x ∈ S ∩ cbox u v cbox u v ⊆ ball x (r x)
  using ⟨K ∈ D⟩ covered uv by blast
  then have dim (range (f' x)) < ?n
  using rank_dim_range [of matrix (f' x)] x rank[of x]
  by (auto simp: matrix_works scalar_mult_eq_scaleR lin_f')
  then obtain T where T: T ∈ lmeasurable
    and subT: {z. norm(z - f x) ≤ (2 * B) * norm(v - u) ∧ (∃ t ∈ range
(f' x). norm(z - f x - t) ≤ d * norm(v - u))} ⊆ T
    and measT: ?μ T ≤ (2 * (d * norm(v - u))) * (2 * ((2 * B) * norm(v
- u))) ^ (?n - 1)
    (is _ ≤ ?DVU)
  apply (rule Sard_lemma1 [of range (f' x) (2 * B) * norm(v - u) d *
norm(v - u) f x])
  using ⟨B > 0⟩ ⟨d > 0⟩ by simp_all
  show ?thesis
  proof (intro exI conjI)
    have f' (K ∩ S) ⊆ {z. norm(z - f x) ≤ (2 * B) * norm(v - u) ∧ (∃ t ∈
range (f' x). norm(z - f x - t) ≤ d * norm(v - u))}
    unfolding uv
  proof (clarsimp simp: mult.assoc, intro conjI)
    fix y
    assume y: y ∈ cbox u v and y ∈ S
    then have norm (y - x) < r x
    by (metis dist_norm mem_ball norm_minus_commute subsetCE x(2))
    then have le_dy: norm (f y - f x - f' x (y - x)) ≤ d * norm (y - x)
    using r [of x y] x ⟨y ∈ S⟩ by blast
    have yx_le: norm (y - x) ≤ norm (v - u)
  proof (rule norm_le_componentwise_cart)
    show norm ((y - x) $ i) ≤ norm ((v - u) $ i) for i
    using x y by (force simp: mem_box_cart dest!: spec [where x=i])
  qed
  have *: [norm(y - x - z) ≤ d; norm z ≤ B; d ≤ B] ⇒ norm(y - x)
≤ 2 * B
    for x y z :: real^n::_ and d B
    using norm_triangle_ineq2 [of y - x z] by auto
  show norm (f y - f x) ≤ 2 * (B * norm (v - u))
  proof (rule * [OF le_dy])

```

```

    have norm (f' x (y - x)) ≤ onorm (f' x) * norm (y - x)
      using onorm [of f' x y-x] by (meson IntE lin_f' linear_linear x(1))
    also have ... ≤ B * norm (v - u)
    proof (rule mult_mono)
      show onorm (f' x) ≤ B
        using B x by blast
      qed (use ⟨B > 0⟩ yx_le in auto)
    finally show norm (f' x (y - x)) ≤ B * norm (v - u) .
    show d * norm (y - x) ≤ B * norm (v - u)
      using ⟨B > 0⟩ by (auto intro: mult_mono [OF ⟨d ≤ B⟩ yx_le])
    qed
  show ∃ t. norm (f y - f x - f' x t) ≤ d * norm (v - u)
    apply (rule_tac x=y-x in exI)
    using ⟨d > 0⟩ yx_le le_dy mult_left_mono [where c=d]
    by (meson order_trans mult_le_cancel_iff2)
  qed
with subT show f ' (K ∩ S) ⊆ T by blast
show ?μ T ≤ e / (2*c) ^ ?m * ?μ K
proof (rule order_trans [OF measT])
  have ?DVU = (d * 2 * (4 * B) ^ (?n - 1)) * norm (v - u) ^ ?n
    using ⟨c > 0⟩
    apply (simp add: algebra_simps)
    by (metis Suc_pred power_Suc zero_less_card_finite)
  also have ... ≤ (e / (2*c) ^ ?m / (?m ^ ?m)) * norm(v - u) ^ ?n
    by (rule mult_right_mono [OF d]) auto
  also have ... ≤ e / (2*c) ^ ?m * ?μ K
  proof -
    have u ∈ ball (x) (r x) v ∈ ball x (r x)
      using box_ne_empty(1) contra_subsetD [OF x(2)] mem_box(2) uv_ne
    by fastforce+
    moreover have r x ≤ 1/2
      using r12 by auto
    ultimately have norm (v - u) ≤ 1
      using norm_triangle_half_r [of x u 1 v]
      by (metis (no_types, opaque_lifting) dist_commute dist_norm
less_eq_real_def less_le_trans mem_ball)
    then have norm (v - u) ^ ?n ≤ norm (v - u) ^ ?m
      by (simp add: power_decreasing [OF mlen])
    also have ... ≤ ?μ K * real (?m ^ ?m)
    proof -
      obtain n where n: ∧ i. v $ i - u $ i = 2 * c / 2^n
        using close [of u v] ⟨K ∈ D⟩ uv by blast
      have norm (v - u) ^ ?m ≤ (∑ i ∈ UNIV. |(v - u) $ i|) ^ ?m
        by (intro norm_le_l1_cart power_mono) auto
      also have ... ≤ (∏ i ∈ UNIV. v $ i - u $ i) * real CARD('m) ^
CARD('m)
        by (simp add: n field_simps ⟨c > 0⟩ less_eq_real_def)
      also have ... = ?μ K * real (?m ^ ?m)
        by (simp add: uv uv_ne content_cbox_cart)
    qed
  qed

```

```

    finally show ?thesis .
  qed
  finally have *: 1 / real (?m ^ ?m) * norm (v - u) ^ ?n ≤ ?μ K
    by (simp add: field_split_simps)
  show ?thesis
    using mult_left_mono [OF *, of e / (2*c) ^ ?m] ⟨c > 0⟩ ⟨e > 0⟩ by
auto
  qed
  finally show ?DVU ≤ e / (2*c) ^ ?m * ?μ K .
  qed
  qed (use T in auto)
  qed
  then obtain g where meas_g:  $\bigwedge K. K \in \mathcal{D} \implies g \, K \in \text{lmeasurable}$ 
    and sub_g:  $\bigwedge K. K \in \mathcal{D} \implies f \, (K \cap S) \subseteq g \, K$ 
    and le_g:  $\bigwedge K. K \in \mathcal{D} \implies ?\mu (g \, K) \leq e / (2*c) ^ ?m * ?\mu K$ 
    by metis
  have le_e:  $?\mu (\bigcup_{i \in \mathcal{F}} g \, i) \leq e$ 
    if  $\mathcal{F} \subseteq \mathcal{D}$  finite  $\mathcal{F}$  for  $\mathcal{F}$ 
  proof -
    have  $?\mu (\bigcup_{i \in \mathcal{F}} g \, i) \leq (\sum_{i \in \mathcal{F}} ?\mu (g \, i))$ 
      using meas_g ⟨ $\mathcal{F} \subseteq \mathcal{D}$ ⟩ by (auto intro: measure_UNION_le [OF ⟨finite
 $\mathcal{F}$ ⟩])
    also have  $\dots \leq (\sum_{K \in \mathcal{F}} e / (2*c) ^ ?m * ?\mu K)$ 
      using  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$  sum_mono [OF le_g] by (meson le_g subsetCE sum_mono)
    also have  $\dots = e / (2*c) ^ ?m * (\sum_{K \in \mathcal{F}} ?\mu K)$ 
      by (simp add: sum_distrib_left)
    also have  $\dots \leq e$ 
  proof -
    have  $\mathcal{F}$  division_of  $\bigcup \mathcal{F}$ 
    proof (rule division_ofI)
      show  $K \subseteq \bigcup \mathcal{F} \, K \neq \{\}$   $\exists a \, b. K = \text{cbox } a \, b$  if  $K \in \mathcal{F}$  for  $K$ 
        using  $\langle K \in \mathcal{F} \rangle$  covered_cbox  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$  by (auto simp: Union_upper)
      show  $\text{interior } K \cap \text{interior } L = \{\}$  if  $K \in \mathcal{F}$  and  $L \in \mathcal{F}$  and  $K \neq L$  for
 $K \, L$ 
    by (metis (mono_tags, lifting)  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$  pairwiseD disjointish pairwise_subset
that)
    qed (use that in auto)
  then have  $\text{sum } ?\mu \, \mathcal{F} \leq ?\mu (\bigcup \mathcal{F})$ 
    by (simp add: content_division)
  also have  $\dots \leq ?\mu (\text{cbox } (- \text{vec } c) (\text{vec } c) :: (\text{real}, 'm) \text{ vec set})$ 
  proof (rule measure_mono_fmeasurable)
    show  $\bigcup \mathcal{F} \subseteq \text{cbox } (- \text{vec } c) (\text{vec } c)$ 
      by (meson Sup_subset_mono sub_cc order_trans  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$ )
    qed (use  $\langle \mathcal{F} \text{ division\_of } \bigcup \mathcal{F} \rangle$  lmeasurable_division in auto)
  also have  $\dots = \text{content } (\text{cbox } (- \text{vec } c) (\text{vec } c) :: (\text{real}, 'm) \text{ vec set})$ 
    by simp
  also have  $\dots \leq (2 ^ ?m * c ^ ?m)$ 
    using  $\langle c > 0 \rangle$  by (simp add: content_cbox_if_cart)
  finally have  $\text{sum } ?\mu \, \mathcal{F} \leq (2 ^ ?m * c ^ ?m)$  .

```

```

    then show ?thesis
      using ‹e > 0› ‹c > 0› by (auto simp: field_split_simps)
    qed
    finally show ?thesis .
  qed
show  $\exists T. f \restriction S \subseteq T \wedge T \in \text{lmeasurable} \wedge ?\mu T \leq e$ 
proof (intro exI conjI)
  show  $f \restriction S \subseteq \bigcup (g \restriction \mathcal{D})$ 
    using covers_sub_g by force
  show  $\bigcup (g \restriction \mathcal{D}) \in \text{lmeasurable}$ 
    by (rule fmeasurable_UN_bound [OF ‹countable  $\mathcal{D}$ › meas_g le_e])
  show  $?\mu (\bigcup (g \restriction \mathcal{D})) \leq e$ 
    by (rule measure_UN_bound [OF ‹countable  $\mathcal{D}$ › meas_g le_e])
  qed
qed
qed
qed

```

```

theorem baby_Sard:
  fixes f :: real^'m::{finite,wellorder}  $\Rightarrow$  real^'n::{finite,wellorder}
  assumes mlen:  $\text{CARD}('m) \leq \text{CARD}('n)$ 
    and der:  $\bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } S)$ 
    and rank:  $\bigwedge x. x \in S \implies \text{rank}(\text{matrix}(f' x)) < \text{CARD}('n)$ 
  shows negligible(f  $\restriction S$ )
proof -
  let ?U =  $\lambda n. \{x \in S. \text{norm}(x) \leq n \wedge \text{onorm}(f' x) \leq \text{real } n\}$ 
  have  $\bigwedge x. x \in S \implies \exists n. \text{norm } x \leq \text{real } n \wedge \text{onorm } (f' x) \leq \text{real } n$ 
    by (meson linear_order_trans real_arch_simple)
  then have eq:  $S = (\bigcup n. ?U n)$ 
    by auto
  have negligible (f  $\restriction ?U n$ ) for n
  proof (rule Sard_lemma2 [OF mlen])
    show  $0 < \text{real } n + 1$ 
      by auto
    show bounded (?U n)
      using bounded_iff by blast
    show (f has_derivative f' x) (at x within ?U n) if x  $\in ?U n$  for x
      using der that by (force intro: has_derivative_subset)
    qed (use rank in auto)
  then show ?thesis
    by (subst eq) (simp add: image_Union negligible_Union_nat)
  qed

```

6.47.4 A one-way version of change-of-variables not assuming injectivity.

```

lemma integral_on_image_ubound_weak:
  fixes f :: real^'n::{finite,wellorder}  $\Rightarrow$  real
  assumes S:  $S \in \text{sets lebesgue}$ 

```

```

    and f: f ∈ borel_measurable (lebesgue_on (g ' S))
    and nonneg_fg:  $\bigwedge x. x \in S \implies 0 \leq f(g\ x)$ 
    and der_g:  $\bigwedge x. x \in S \implies (g \text{ has\_derivative } g'\ x) \text{ (at } x \text{ within } S)$ 
    and det_int_fg:  $(\lambda x. |\det (\text{matrix } (g'\ x))| * f(g\ x)) \text{ integrable\_on } S$ 
    and meas_gim:  $\bigwedge T. \llbracket T \subseteq g\ ' S; T \in \text{sets lebesgue} \rrbracket \implies \{x \in S. g\ x \in T\}$ 
  ∈ sets lebesgue
  shows f integrable_on (g ' S) ∧
    integral (g ' S) f ≤ integral S (λx. |det (matrix (g' x))| * f(g x))
    (is _ ∧ _ ≤ ?b)
proof -
  let ?D = λx. |det (matrix (g' x))|
  have cont_g: continuous_on S g
    using der_g has_derivative_continuous_on by blast
  have [simp]: space (lebesgue_on S) = S
    by (simp add: S)
  have gS_in_sets_leb: g ' S ∈ sets lebesgue
    apply (rule differentiable_image_in_sets_lebesgue)
    using der_g by (auto simp: S differentiable_def differentiable_on_def)
  obtain h where nonneg_h:  $\bigwedge n\ x. 0 \leq h\ n\ x$ 
    and h_le_f:  $\bigwedge n\ x. x \in S \implies h\ n\ (g\ x) \leq f\ (g\ x)$ 
    and h_inc:  $\bigwedge n\ x. h\ n\ x \leq h\ (\text{Suc } n)\ x$ 
    and h_meas:  $\bigwedge n. h\ n \in \text{borel\_measurable lebesgue}$ 
    and fin_R:  $\bigwedge n. \text{finite}(\text{range } (h\ n))$ 
    and lim:  $\bigwedge x. x \in g\ ' S \implies (\lambda n. h\ n\ x) \longrightarrow f\ x$ 
  proof -
    let ?f = λx. if x ∈ g ' S then f x else 0
    have ?f ∈ borel_measurable lebesgue ∧  $(\forall x. 0 \leq ?f\ x)$ 
      by (auto simp: gS_in_sets_leb f nonneg_fg measurable_restrict_space_iff
        [symmetric])
    then show ?thesis
      apply (clarify simp add: borel_measurable_simple_function_limit_increasing)
      apply (rename_tac h)
      by (rule_tac h=h in that) (auto split: if_split_asm)
    qed
  have h_lmeas:  $\{t. h\ n\ (g\ t) = y\} \cap S \in \text{sets lebesgue}$  for y n
  proof -
    have space (lebesgue_on (UNIV::(real,'n) vec set)) = UNIV
      by simp
    then have  $((h\ n) - \{y\} \cap g\ ' S) \in \text{sets (lebesgue\_on (g\ ' S))}$ 
      by (metis Int_commute borel_measurable_vimage h_meas image_eqI inf_top.right_neutral
        sets_restrict_space space_borel space_completion space_lborel)
    then have  $(\{u. h\ n\ u = y\} \cap g\ ' S) \in \text{sets lebesgue}$ 
      using gS_in_sets_leb
      by (simp add: integral_indicator_fmeasurableI2 sets_restrict_space_iff vimage_def)
    then have  $\{x \in S. g\ x \in (\{u. h\ n\ u = y\} \cap g\ ' S)\} \in \text{sets lebesgue}$ 
      using meas_gim[of  $(\{u. h\ n\ u = y\} \cap g\ ' S)$ ] by force
    moreover have  $\{t. h\ n\ (g\ t) = y\} \cap S = \{x \in S. g\ x \in (\{u. h\ n\ u = y\} \cap g\ ' S)\}$ 

```

```

    by blast
  ultimately show ?thesis
    by auto
qed
have hint:  $h \ n \text{ integrable\_on } g \text{ ' } S \wedge \text{integral } (g \text{ ' } S) (h \ n) \leq \text{integral } S (\lambda x. ?D \ x * h \ n \ (g \ x))$ 
  (is ?INT  $\wedge$  ?lhs  $\leq$  ?rhs) for n
proof -
  let ?R = range (h n)
  have hn_eq:  $h \ n = (\lambda x. \sum_{y \in ?R} y * \text{indicat\_real } \{x. h \ n \ x = y\} \ x)$ 
    by (simp add: indicator_def if_distrib fin_R cong: if_cong)
  have yind:  $(\lambda t. y * \text{indicator} \{x. h \ n \ x = y\} \ t) \text{ integrable\_on } (g \text{ ' } S) \wedge$ 
     $(\text{integral } (g \text{ ' } S) (\lambda t. y * \text{indicator} \{x. h \ n \ x = y\} \ t))$ 
     $\leq \text{integral } S (\lambda t. |\det (\text{matrix } (g' \ t))| * y * \text{indicator} \{x. h \ n \ x = y\}$ 
     $(g \ t))$ 
    if y:  $y \in ?R$  for y::real
  proof (cases y=0)
    case True
    then show ?thesis using gS_in_sets_leb integrable_0 by force
  next
    case False
    with that have y > 0
      using less_eq_real_def nonneg_h by fastforce
    have  $(\lambda x. \text{if } x \in \{t. h \ n \ (g \ t) = y\} \text{ then } ?D \ x \text{ else } 0) \text{ integrable\_on } S$ 
      proof (rule measurable_bounded_by_integrable_imp_integrable)
        have  $(\lambda x. ?D \ x) \in \text{borel\_measurable } (\text{lebesgue\_on } (\{t. h \ n \ (g \ t) = y\} \cap S))$ 
          apply (intro borel_measurable_abs borel_measurable_det_Jacobian [OF
            h_lmeas, where f=g])
          by (meson der_g IntD2 has_derivative_subset inf_le2)
        then have  $(\lambda x. \text{if } x \in \{t. h \ n \ (g \ t) = y\} \cap S \text{ then } ?D \ x \text{ else } 0) \in$ 
           $\text{borel\_measurable } \text{lebesgue}$ 
          by (rule borel_measurable_if_I [OF _ h_lmeas])
        then show  $(\lambda x. \text{if } x \in \{t. h \ n \ (g \ t) = y\} \text{ then } ?D \ x \text{ else } 0) \in \text{borel\_measurable}$ 
           $(\text{lebesgue\_on } S)$ 
          by (simp add: if_if_eq_conj Int_commute borel_measurable_if [OF S,
            symmetric])
        show  $(\lambda x. ?D \ x *_{\mathbb{R}} f \ (g \ x) /_{\mathbb{R}} y) \text{ integrable\_on } S$ 
          by (rule integrable_cmul) (use det_int_fg in auto)
        show  $\text{norm } (\text{if } x \in \{t. h \ n \ (g \ t) = y\} \text{ then } ?D \ x \text{ else } 0) \leq ?D \ x *_{\mathbb{R}} f \ (g \ x)$ 
          / $\mathbb{R}$  y
          if  $x \in S$  for x
          using nonneg_h [of n x]  $\langle y > 0 \rangle$  nonneg_fg [of x] h_le_f [of x n] that
          by (auto simp: divide_simps mult_left_mono)
      qed (use S in auto)
    then have int_det:  $(\lambda t. |\det (\text{matrix } (g' \ t))|) \text{ integrable\_on } (\{t. h \ n \ (g \ t) =$ 
     $y\} \cap S)$ 
      using integrable_restrict_Int by force
    have  $(g \text{ ' } (\{t. h \ n \ (g \ t) = y\} \cap S)) \in \text{lmeasurable}$ 
      apply (rule measurable_differentiable_image [OF h_lmeas])

```

```

    apply (blast intro: has_derivative_subset [OF der_g])
  apply (rule int_det)
done
moreover have g ' ( $\{t. h \ n \ (g \ t) = y\} \cap S) = \{x. h \ n \ x = y\} \cap g \ ' \ S$ 
  by blast
moreover have measure_lebesgue (g ' ( $\{t. h \ n \ (g \ t) = y\} \cap S)$ )
   $\leq \text{integral } (\{t. h \ n \ (g \ t) = y\} \cap S) \ (\lambda t. |\det (\text{matrix } (g' \ t))|)$ 
  apply (rule measure_differentiable_image [OF h_lmeas _ int_det])
  apply (blast intro: has_derivative_subset [OF der_g])
done
ultimately show ?thesis
  using  $\langle y > 0 \rangle$  integral_restrict_Int [of S  $\{t. h \ n \ (g \ t) = y\} \lambda t. |\det (\text{matrix } (g' \ t))| * y]$ 
  apply (simp add: integrable_on_indicator integral_indicator)
  apply (simp add: indicator_def of_bool_def if_distrib cong: if_cong)
done
qed
have hn_int: h n integrable_on g ' S
  apply (subst hn_eq)
  using yind by (force intro: integrable_sum [OF fin_R])
then show ?thesis
proof
  have ?lhs = integral (g ' S) ( $\lambda x. \sum_{y \in \text{range } (h \ n)}. y * \text{indicat\_real } \{x. h \ n \ x = y\} \ x$ )
    by (metis hn_eq)
  also have ... = ( $\sum_{y \in \text{range } (h \ n)}. \text{integral } (g \ ' \ S) \ (\lambda x. y * \text{indicat\_real } \{x. h \ n \ x = y\} \ x)$ )
    by (rule integral_sum [OF fin_R]) (use yind in blast)
  also have ...  $\leq (\sum_{y \in \text{range } (h \ n)}. \text{integral } S \ (\lambda u. |\det (\text{matrix } (g' \ u))| * y * \text{indicat\_real } \{x. h \ n \ x = y\} \ (g \ u)))$ 
    using yind by (force intro: sum_mono)
  also have ... = integral S ( $\lambda u. \sum_{y \in \text{range } (h \ n)}. |\det (\text{matrix } (g' \ u))| * y * \text{indicat\_real } \{x. h \ n \ x = y\} \ (g \ u)$ )
    proof (rule integral_sum [OF fin_R, symmetric])
      fix y assume y:  $y \in ?R$ 
      with nonneg_h have  $y \geq 0$ 
      by auto
      show ( $\lambda u. |\det (\text{matrix } (g' \ u))| * y * \text{indicat\_real } \{x. h \ n \ x = y\} \ (g \ u)$ )
        integrable_on S
    proof (rule measurable_bounded_by_integrable_imp_integrable)
      have ( $\lambda x. \text{indicat\_real } \{x. h \ n \ x = y\} \ (g \ x) \in \text{borel\_measurable } (\text{lebesgue\_on } S)$ )
        using h_lmeas S
      by (auto simp: indicator_vimage [symmetric] borel_measurable_indicator_iff
        sets_restrict_space_iff)
      then show ( $\lambda u. |\det (\text{matrix } (g' \ u))| * y * \text{indicat\_real } \{x. h \ n \ x = y\} \ (g \ u) \in \text{borel\_measurable } (\text{lebesgue\_on } S)$ )
        by (intro borel_measurable_times borel_measurable_abs borel_measurable_const
          borel_measurable_det_Jacobian [OF S der_g])
    end
    end

```



```

next
  fix x
  assume  $x \in S$ 
  have  $y * \text{indicat\_real } \{x. h\ n\ x = y\} (g\ x) \leq f (g\ x)$ 
    by (metis  $\langle x \in S \rangle h\_le\_f$  indicator_simps(1) indicator_simps(2))
  mem_Collect_eq mult.commute mult.left_neutral mult.zero_left nonneg_fg
  with  $\langle y \geq 0 \rangle$  show  $\text{norm } (?D\ x * y * \text{indicat\_real } \{x. h\ n\ x = y\} (g\ x))$ 
 $\leq ?D\ x * f(g\ x)$ 
    by (simp add: abs_mult mult.assoc mult_left_mono)
  qed (use S det_int_fg in auto)
qed
also have  $\dots = \text{integral } S (\lambda T. |\det (\text{matrix } (g'\ T))| * (\sum_{y \in \text{range } (h\ n)}. y * \text{indicat\_real } \{x. h\ n\ x = y\} (g\ T)))$ 
    by (simp add: sum_distrib_left mult.assoc)
  also have  $\dots = ?rhs$ 
    by (metis hn_eq)
  finally show  $\text{integral } (g\ ' S) (h\ n) \leq ?rhs$  .
qed
qed
have le:  $\text{integral } S (\lambda T. |\det (\text{matrix } (g'\ T))| * h\ n\ (g\ T)) \leq ?b$  for n
proof (rule integral_le)
  show  $(\lambda T. |\det (\text{matrix } (g'\ T))| * h\ n\ (g\ T)) \text{ integrable\_on } S$ 
  proof (rule measurable_bounded_by_integrable_imp_integrable)
    have  $(\lambda T. |\det (\text{matrix } (g'\ T))| *_R h\ n\ (g\ T)) \in \text{borel\_measurable } (\text{lebesgue\_on } S)$ 
    proof (intro borel_measurable_scaleR borel_measurable_abs borel_measurable_det_Jacobian
 $\langle S \in \text{sets lebesgue} \rangle$ )
      have eq:  $\{x \in S. f\ x \leq a\} = (\bigcup b \in (f\ ' S) \cap \text{atMost } a. \{x. f\ x = b\} \cap S)$ 
    for f and a::real
      by auto
    have finite  $((\lambda x. h\ n\ (g\ x))\ ' S \cap \{..a\})$  for a
      by (force intro: finite_subset [OF fin_R])
    with h_lmeas [of n] show  $(\lambda x. h\ n\ (g\ x)) \in \text{borel\_measurable } (\text{lebesgue\_on } S)$ 
    apply (simp add: borel_measurable_vimage_halfspace_component_le  $\langle S \in \text{sets lebesgue} \rangle$  sets_restrict_space_iff eq)
      by (metis (mono_tags) SUP_inf sets.finite_UN)
    qed (use der_g in blast)
    then show  $(\lambda T. |\det (\text{matrix } (g'\ T))| * h\ n\ (g\ T)) \in \text{borel\_measurable } (\text{lebesgue\_on } S)$ 
      by simp
    show  $\text{norm } (?D\ x * h\ n\ (g\ x)) \leq ?D\ x *_R f (g\ x)$ 
      if  $x \in S$  for x
      by (simp add: h_le_f mult_left_mono nonneg_h that)
  qed (use S det_int_fg in auto)
show  $?D\ x * h\ n\ (g\ x) \leq ?D\ x * f (g\ x)$  if  $x \in S$  for x
  by (simp add:  $\langle x \in S \rangle h\_le\_f$  mult_left_mono)
show  $(\lambda x. ?D\ x * f (g\ x)) \text{ integrable\_on } S$ 

```

```

    using det_int_fg by blast
  qed
  have f integrable_on g ' S  $\wedge$  ( $\lambda k.$  integral (g ' S) (h k))  $\longrightarrow$  integral (g ' S) f
  proof (rule monotone_convergence_increasing)
    have |integral (g ' S) (h n)|  $\leq$  integral S ( $\lambda x.$  ?D x * f (g x)) for n
    proof -
      have |integral (g ' S) (h n)| = integral (g ' S) (h n)
      using hint by (simp add: integral_nonneg nonneg_h)
      also have ...  $\leq$  integral S ( $\lambda x.$  ?D x * f (g x))
      using hint le by (meson order_trans)
      finally show ?thesis .
    qed
    then show bounded (range ( $\lambda k.$  integral (g ' S) (h k)))
    by (force simp: bounded_iff)
  qed (use h_inc lim hint in auto)
  moreover have integral (g ' S) (h n)  $\leq$  integral S ( $\lambda x.$  ?D x * f (g x)) for n
  using hint by (blast intro: le order_trans)
  ultimately show ?thesis
  by (auto intro: Lim_bounded)
qed

```

```

lemma integral_on_image_ubound_nonneg:
  fixes f :: realn::{finite,wellorder}  $\Rightarrow$  real
  assumes nonneg_fg:  $\bigwedge x. x \in S \implies 0 \leq f(g\ x)$ 
    and der_g:  $\bigwedge x. x \in S \implies (g \text{ has\_derivative } g' \ x) \text{ (at } x \text{ within } S)$ 
    and intS: ( $\lambda x.$  |det (matrix (g' x))| * f(g x)) integrable_on S
  shows f integrable_on (g ' S)  $\wedge$  integral (g ' S) f  $\leq$  integral S ( $\lambda x.$  |det (matrix
(g' x))| * f(g x))
    (is _  $\wedge$  _  $\leq$  ?b)
  proof -
    let ?D =  $\lambda x.$  det (matrix (g' x))
    define S' where S'  $\equiv$  {x  $\in$  S. ?D x * f(g x)  $\neq$  0}
    then have der_gS':  $\bigwedge x. x \in S' \implies (g \text{ has\_derivative } g' \ x) \text{ (at } x \text{ within } S')$ 
    by (metis (mono_tags, lifting) der_g has_derivative_subset mem_Collect_eq
subset_iff)
    have ( $\lambda x.$  if x  $\in$  S then |?D x| * f (g x) else 0) integrable_on UNIV
    by (simp add: integrable_restrict_UNIV intS)
    then have Df_borel: ( $\lambda x.$  if x  $\in$  S then |?D x| * f (g x) else 0)  $\in$  borel_measurable
lebesgue
    using integrable_imp_measurable lebesgue_on_UNIV_eq by force
    have S': S'  $\in$  sets lebesgue
    proof -
      from Df_borel borel_measurable_vimage_open [of _ UNIV]
      have {x. (if x  $\in$  S then |?D x| * f (g x) else 0)  $\in$  T}  $\in$  sets lebesgue
      if open T for T
      using that unfolding lebesgue_on_UNIV_eq
      by (fastforce simp add: dest!: spec)
      then have {x. (if x  $\in$  S then |?D x| * f (g x) else 0)  $\in$  -{0}}  $\in$  sets lebesgue
    qed
  qed

```

```

    using open_Compl by blast
  then show ?thesis
    by (simp add: S'_def conj_ac split: if_split_asm cong: conj_cong)
qed
then have gS':  $g \restriction S' \in \text{sets lebesgue}$ 
proof (rule differentiable_image_in_sets_lebesgue)
  show  $g \text{ differentiable\_on } S'$ 
    using der_g unfolding S'_def differentiable_def differentiable_on_def
    by (blast intro: has_derivative_subset)
qed auto
have f:  $f \in \text{borel\_measurable (lebesgue\_on (g \restriction S'))}$ 
proof (clarsimp simp add: borel_measurable_vimage_open)
  fix T :: real set
  assume open T
  have  $\{x \in g \restriction S'. f x \in T\} = g \restriction \{x \in S'. f(g x) \in T\}$ 
    by blast
  moreover have  $g \restriction \{x \in S'. f(g x) \in T\} \in \text{sets lebesgue}$ 
  proof (rule differentiable_image_in_sets_lebesgue)
    let ?h =  $\lambda x. |?D x| * f(g x) /_R |?D x|$ 
    have  $(\lambda x. \text{if } x \in S' \text{ then } |?D x| * f(g x) \text{ else } 0) = (\lambda x. \text{if } x \in S \text{ then } |?D x|$ 
    *  $f(g x) \text{ else } 0)$ 
      by (auto simp: S'_def)
    also have  $\dots \in \text{borel\_measurable lebesgue}$ 
      by (rule Df_borel)
    finally have *:  $(\lambda x. |?D x| * f(g x)) \in \text{borel\_measurable (lebesgue\_on } S')$ 
      by (simp add: borel_measurable_if_D)
    have ?h  $\in \text{borel\_measurable (lebesgue\_on } S')$ 
      by (intro * S' der_gS' borel_measurable_det_Jacobian measurable) (blast
intro: der_gS')
    moreover have ?h  $x = f(g x)$  if  $x \in S'$  for  $x$ 
      using that by (auto simp: S'_def)
    ultimately have  $(\lambda x. f(g x)) \in \text{borel\_measurable (lebesgue\_on } S')$ 
      by (metis (no_types, lifting) measurable_lebesgue_cong)
    then show  $\{x \in S'. f(g x) \in T\} \in \text{sets lebesgue}$ 
      by (simp add: S'  $\in \text{sets lebesgue}$  open T borel_measurable_vimage_open
sets_restrict_space_iff)
    show  $g \text{ differentiable\_on } \{x \in S'. f(g x) \in T\}$ 
      using der_g unfolding S'_def differentiable_def differentiable_on_def
      by (blast intro: has_derivative_subset)
  qed auto
  ultimately have  $\{x \in g \restriction S'. f x \in T\} \in \text{sets lebesgue}$ 
    by metis
  then show  $\{x \in g \restriction S'. f x \in T\} \in \text{sets (lebesgue\_on (g \restriction S'))}$ 
    by (simp add: g  $\restriction S' \in \text{sets lebesgue}$  sets_restrict_space_iff)
qed
have intS':  $(\lambda x. |?D x| * f(g x)) \text{ integrable\_on } S'$ 
  using intS
  by (rule integrable_spike_set) (auto simp: S'_def intro: empty_imp_negligible)
have lebS':  $\{x \in S'. g x \in T\} \in \text{sets lebesgue}$  if  $T \subseteq g \restriction S' T \in \text{sets lebesgue}$ 

```

```

for T
proof -
  have  $g \in \text{borel\_measurable } (\text{lebesgue\_on } S')$ 
  using  $\text{der\_gS' has\_derivative\_continuous\_on } S'$ 
  by (blast intro:  $\text{continuous\_imp\_measurable\_on\_sets\_lebesgue}$ )
  moreover have  $\{x \in S'. g \ x \in U\} \in \text{sets lebesgue}$  if negligible  $U \subseteq g \ ' S'$ 
for U
proof (intro negligible_imp_sets negligible_differentiable_vimage that)
  fix x
  assume  $x: x \in S'$ 
  then have  $\text{linear } (g' \ x)$ 
  using  $\text{der\_gS' has\_derivative\_linear}$  by blast
  with x show  $\text{inj } (g' \ x)$ 
  by (auto simp:  $S'_{\text{def}} \text{det\_nz\_iff\_inj}$ )
qed (use  $\text{der\_gS'}$  in auto)
ultimately show ?thesis
  using  $\text{double\_lebesgue\_sets } [OF \ S' \ gS' \ \text{order\_refl}]$  that by blast
qed
have  $\text{int\_gS': } f \text{ integrable\_on } g \ ' S' \wedge \text{integral } (g \ ' S') \ f \leq \text{integral } S' \ (\lambda x. |?D \ x| * f(g \ x))$ 
  using  $\text{integral\_on\_image\_ubound\_weak } [OF \ S' \ f \ \text{nonneg\_fg } \text{der\_gS' } \text{intS' } \text{lebS'}]$   $S'_{\text{def}}$  by blast
  have  $\text{negligible } (g \ ' \{x \in S. \text{det}(\text{matrix}(g' \ x)) = 0\})$ 
  proof (rule  $\text{baby\_Sard, simp\_all}$ )
    fix x
    assume  $x: x \in S \wedge \text{det}(\text{matrix}(g' \ x)) = 0$ 
    then show  $(g \text{ has\_derivative } g' \ x) \text{ (at } x \text{ within } \{x \in S. \text{det}(\text{matrix}(g' \ x)) = 0\})$ 
    by (metis (no_types, lifting)  $\text{der\_g has\_derivative\_subset mem\_Collect\_eq subsetI}$ )
    then show  $\text{rank}(\text{matrix}(g' \ x)) < \text{CARD}('n)$ 
    using  $\text{det\_nz\_iff\_inj matrix\_vector\_mul\_linear } x$ 
    by (fastforce simp add:  $\text{less\_rank\_noninjective}$ )
  qed
then have  $\text{negg: negligible } (g \ ' S - g \ ' \{x \in S. ?D \ x \neq 0\})$ 
  by (rule  $\text{negligible\_subset}$ ) (auto simp:  $S'_{\text{def}}$ )
have  $\text{null: } g \ ' \{x \in S. ?D \ x \neq 0\} - g \ ' S = \{\}$ 
  by (auto simp:  $S'_{\text{def}}$ )
let  $?F = \{x \in S. f(g \ x) \neq 0\}$ 
have  $\text{eq: } g \ ' S' = g \ ' ?F \cap g \ ' \{x \in S. ?D \ x \neq 0\}$ 
  by (auto simp:  $S'_{\text{def}} \text{image\_iff}$ )
show ?thesis
proof
  have  $((\lambda x. \text{if } x \in g \ ' ?F \text{ then } f \ x \text{ else } 0) \text{ integrable\_on } g \ ' \{x \in S. ?D \ x \neq 0\})$ 
  using  $\text{int\_gS' eq integrable\_restrict\_Int } [\text{where } f=f]$ 
  by simp
  then have  $f \text{ integrable\_on } g \ ' \{x \in S. ?D \ x \neq 0\}$ 
  by (auto simp:  $\text{image\_iff elim!:: integrable\_eq}$ )
  then show  $f \text{ integrable\_on } g \ ' S$ 

```

```

apply (rule integrable_spike_set [OF _ empty_imp_negligible negligible_subset])
using negg null by auto
have integral (g ' S) f = integral (g ' {x ∈ S. ?D x ≠ 0}) f
using negg by (auto intro: negligible_subset integral_spike_set)
also have ... = integral (g ' {x ∈ S. ?D x ≠ 0}) (λx. if x ∈ g ' ?F then f x
else 0)
by (auto simp: image_iff intro!: integral_cong)
also have ... = integral (g ' S') f
using eq integral_restrict_Int by simp
also have ... ≤ integral S' (λx. |?D x| * f(g x))
by (metis int_gS')
also have ... ≤ ?b
by (rule integral_subset_le [OF _ intS' intS]) (use nonneg_fg S'_def in
auto)
finally show integral (g ' S) f ≤ ?b .
qed
qed

```

lemma absolutely_integrable_on_image_real:

```

fixes f :: real^'n::{finite,wellorder} ⇒ real and g :: real^'n::_ ⇒ real^'n::_
assumes der_g: ∧x. x ∈ S ⇒ (g has_derivative g' x) (at x within S)
and intS: (λx. |det (matrix (g' x))| * f(g x)) absolutely_integrable_on S
shows f absolutely_integrable_on (g ' S)
proof -
let ?D = λx. |det (matrix (g' x))| * f (g x)
let ?N = {x ∈ S. f (g x) < 0} and ?P = {x ∈ S. f (g x) > 0}
have eq: {x. (if x ∈ S then ?D x else 0) > 0} = {x ∈ S. ?D x > 0}
      {x. (if x ∈ S then ?D x else 0) < 0} = {x ∈ S. ?D x < 0}
by auto
have ?D integrable_on S
using intS absolutely_integrable_on_def by blast
then have (λx. if x ∈ S then ?D x else 0) integrable_on UNIV
by (simp add: integrable_restrict_UNIV)
then have D_borel: (λx. if x ∈ S then ?D x else 0) ∈ borel_measurable (lebesgue_on
UNIV)
using integrable_imp_measurable lebesgue_on_UNIV_eq by blast
then have Dlt: {x ∈ S. ?D x < 0} ∈ sets lebesgue
unfolding borel_measurable_vimage_halfspace_component_lt
by (drule_tac x=0 in spec) (auto simp: eq)
from D_borel have Dgt: {x ∈ S. ?D x > 0} ∈ sets lebesgue
unfolding borel_measurable_vimage_halfspace_component_gt
by (drule_tac x=0 in spec) (auto simp: eq)

have dfgbm: ?D ∈ borel_measurable (lebesgue_on S)
using intS absolutely_integrable_on_def integrable_imp_measurable by blast
have der_gN: (g has_derivative g' x) (at x within ?N) if x ∈ ?N for x
using der_g has_derivative_subset that by force
have (λx. - f x) integrable_on g ' ?N ∧

```

```

      integral (g ' ?N) (λx. - f x) ≤ integral ?N (λx. |det (matrix (g' x))| * - f
(g x))
    proof (rule integral_on_image_ubound_nonneg [OF _ der_gN])
      have 1: ?D integrable_on {x ∈ S. ?D x < 0}
        using Dlt
      by (auto intro: set_lebesgue_integral_eq_integral [OF set_integrable_subset]
intS)
      have uminus ∘ (λx. |det (matrix (g' x))| * - f (g x)) integrable_on ?N
      by (simp add: o_def mult_less_0_iff empty_imp_negligible integrable_spike_set
[OF 1])
      then show (λx. |det (matrix (g' x))| * - f (g x)) integrable_on ?N
        by (simp add: integrable_neg_iff o_def)
    qed auto
    then have f integrable_on g ' ?N
      by (simp add: integrable_neg_iff)
    moreover have g ' ?N = {y ∈ g ' S. f y < 0}
      by auto
    ultimately have f integrable_on {y ∈ g ' S. f y < 0}
      by simp
    then have N: f absolutely_integrable_on {y ∈ g ' S. f y < 0}
      by (rule absolutely_integrable_absolutely_integrable_ubound) auto

    have der_gP: (g has_derivative g' x) (at x within ?P) if x ∈ ?P for x
      using der_g has_derivative_subset that by force
    have f integrable_on g ' ?P ∧ integral (g ' ?P) f ≤ integral ?P ?D
    proof (rule integral_on_image_ubound_nonneg [OF _ der_gP])
      have ?D integrable_on {x ∈ S. 0 < ?D x}
        using Dgt
      by (auto intro: set_lebesgue_integral_eq_integral [OF set_integrable_subset]
intS)
      then show ?D integrable_on ?P
        apply (rule integrable_spike_set)
        by (auto simp: zero_less_mult_iff empty_imp_negligible)
    qed auto
    then have f integrable_on g ' ?P
      bymetis
    moreover have g ' ?P = {y ∈ g ' S. f y > 0}
      by auto
    ultimately have f integrable_on {y ∈ g ' S. f y > 0}
      by simp
    then have P: f absolutely_integrable_on {y ∈ g ' S. f y > 0}
      by (rule absolutely_integrable_absolutely_integrable_lbound) auto
    have (λx. if x ∈ g ' S ∧ f x < 0 ∨ x ∈ g ' S ∧ 0 < f x then f x else 0) = (λx.
if x ∈ g ' S then f x else 0)
      by auto
    then show ?thesis
      using absolutely_integrable_Un [OF N P] absolutely_integrable_restrict_UNIV
[symmetric, where f=f]
      by simp

```

qed

proposition *absolutely_integrable_on_image*:

fixes $f :: \text{real}^m :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n$ **and** $g :: \text{real}^m :: \_ \Rightarrow \text{real}^m :: \_$
assumes $\text{der\_}g: \bigwedge x. x \in S \implies (g \text{ has\_derivative } g' \ x) \text{ (at } x \text{ within } S)$
and $\text{int}S: (\lambda x. |\det (\text{matrix } (g' \ x))| *_{\mathbb{R}} f(g \ x)) \text{ absolutely\_integrable\_on } S$
shows $f \text{ absolutely\_integrable\_on } (g \text{ ` } S)$
apply (rule *absolutely_integrable_componentwise* [OF *absolutely_integrable_on_image_real* [OF *der_g*]])
using *absolutely_integrable_component* [OF *intS*] **by** auto

proposition *integral_on_image_ubound*:

fixes $f :: \text{real}^n :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}$ **and** $g :: \text{real}^n :: \_ \Rightarrow \text{real}^n :: \_$
assumes $\bigwedge x. x \in S \implies 0 \leq f(g \ x)$
and $\bigwedge x. x \in S \implies (g \text{ has\_derivative } g' \ x) \text{ (at } x \text{ within } S)$
and $(\lambda x. |\det (\text{matrix } (g' \ x))| * f(g \ x)) \text{ integrable\_on } S$
shows $\text{integral } (g \text{ ` } S) f \leq \text{integral } S (\lambda x. |\det (\text{matrix } (g' \ x))| * f(g \ x))$
using *integral_on_image_ubound_nonneg* [OF *assms*] **by** simp

6.47.5 Change-of-variables theorem

The classic change-of-variables theorem. We have two versions with quite general hypotheses, the first that the transforming function has a continuous inverse, the second that the base set is Lebesgue measurable.

lemma *cov_invertible_nonneg_le*:

fixes $f :: \text{real}^n :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}$ **and** $g :: \text{real}^n :: \_ \Rightarrow \text{real}^n :: \_$
assumes $\text{der\_}g: \bigwedge x. x \in S \implies (g \text{ has\_derivative } g' \ x) \text{ (at } x \text{ within } S)$
and $\text{der\_}h: \bigwedge y. y \in T \implies (h \text{ has\_derivative } h' \ y) \text{ (at } y \text{ within } T)$
and $f0: \bigwedge y. y \in T \implies 0 \leq f \ y$
and $hg: \bigwedge x. x \in S \implies g \ x \in T \wedge h(g \ x) = x$
and $gh: \bigwedge y. y \in T \implies h \ y \in S \wedge g(h \ y) = y$
and $id: \bigwedge y. y \in T \implies h' \ y \circ g'(h \ y) = id$
shows $f \text{ integrable\_on } T \wedge (\text{integral } T f) \leq b \iff$
 $(\lambda x. |\det (\text{matrix } (g' \ x))| * f(g \ x)) \text{ integrable\_on } S \wedge$
 $\text{integral } S (\lambda x. |\det (\text{matrix } (g' \ x))| * f(g \ x)) \leq b$
(is ?lhs = ?rhs)

proof –

have $Tg: T = g \text{ ` } S$  **and**  $Seq: S = h \text{ ` } T$
using *hg gh image_iff* **by** fastforce+
have $gS: g \text{ differentiable\_on } S$
by (meson *der_g differentiable_def differentiable_on_def*)
let $?D = \lambda x. |\det (\text{matrix } (g' \ x))| * f \ (g \ x)$
show *?thesis*
proof
assume *?lhs*
then have $fT: f \text{ integrable\_on } T$ **and** $intf: \text{integral } T f \leq b$
by blast+
show *?rhs*

```

proof
  let ?fgh =  $\lambda x. |\det (\text{matrix } (h' x))| * (|\det (\text{matrix } (g' (h x)))| * f (g (h x)))$ 
  have ddf: ?fgh  $x = f x$ 
  if  $x \in T$  for  $x$ 
  proof -
    have  $\text{matrix } (h' x) ** \text{matrix } (g' (h x)) = \text{mat } 1$ 
    using that id[OF that] der_g[of  $h x$ ] gh[OF that] left_inverse_linear
has_derivative_linear
    by (subst matrix_compose[symmetric]) (force simp: matrix_id_mat_1
has_derivative_linear)+
    then have  $|\det (\text{matrix } (h' x))| * |\det (\text{matrix } (g' (h x)))| = 1$ 
    by (metis abs_1 abs_mult det_I det_mul)
    then show ?thesis
    by (simp add: gh that)
  qed
  have ?D integrable_on ( $h \text{ ' } T$ )
proof (intro set_lebesgue_integral_eq_integral absolutely_integrable_on_image_real)
  show ( $\lambda x. ?fgh x$ ) absolutely_integrable_on  $T$ 
  proof (subst absolutely_integrable_on_iff_nonneg)
  show ( $\lambda x. ?fgh x$ ) integrable_on  $T$ 
  using ddf fT integrable_eq by force
  qed (simp add: zero_le_mult_iff f0 gh)
qed (use der_h in auto)
with Seq show ( $\lambda x. ?D x$ ) integrable_on  $S$ 
by simp
have  $\text{integral } S (\lambda x. ?D x) \leq \text{integral } T (\lambda x. ?fgh x)$ 
unfolding Seq
proof (rule integral_on_image_ubound)
  show ( $\lambda x. ?fgh x$ ) integrable_on  $T$ 
  using ddf fT integrable_eq by force
qed (use f0 gh der_h in auto)
also have  $\dots = \text{integral } T f$ 
by (force simp: ddf intro: integral_cong)
also have  $\dots \leq b$ 
by (rule intf)
finally show  $\text{integral } S (\lambda x. ?D x) \leq b$  .
qed
next
assume  $R: ?rhs$ 
then have  $f$  integrable_on  $g \text{ ' } S$ 
using der_g f0 hg integral_on_image_ubound_nonneg by blast
moreover have  $\text{integral } (g \text{ ' } S) f \leq \text{integral } S (\lambda x. ?D x)$ 
by (rule integral_on_image_ubound [OF f0 der_g]) (use R Teq in auto)
ultimately show ?lhs
using R by (simp add: Teq)
qed
qed

```


lemma *cov_invertible_nonneg_eq*:

fixes $f :: \text{real}^n::\{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}$ **and** $g :: \text{real}^n::\_ \Rightarrow \text{real}^n::\_$
assumes $\bigwedge x. x \in S \Longrightarrow (g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } S)$
and $\bigwedge y. y \in T \Longrightarrow (h \text{ has\_derivative } h' y) \text{ (at } y \text{ within } T)$
and $\bigwedge y. y \in T \Longrightarrow 0 \leq f y$
and $\bigwedge x. x \in S \Longrightarrow g x \in T \wedge h(g x) = x$
and $\bigwedge y. y \in T \Longrightarrow h y \in S \wedge g(h y) = y$
and $\bigwedge y. y \in T \Longrightarrow h' y \circ g'(h y) = \text{id}$
shows $((\lambda x. |\det(\text{matrix}(g' x))| * f(g x)) \text{ has\_integral } b) S \longleftrightarrow (f \text{ has\_integral } b) T$
using *cov_invertible_nonneg_le* [OF *assms*]
by (*simp add: has_integral_iff*) (*meson eq_iff*)

lemma *cov_invertible_real*:

fixes $f :: \text{real}^n::\{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}$ **and** $g :: \text{real}^n::\_ \Rightarrow \text{real}^n::\_$
assumes *der_g*: $\bigwedge x. x \in S \Longrightarrow (g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } S)$
and *der_h*: $\bigwedge y. y \in T \Longrightarrow (h \text{ has\_derivative } h' y) \text{ (at } y \text{ within } T)$
and *hg*: $\bigwedge x. x \in S \Longrightarrow g x \in T \wedge h(g x) = x$
and *gh*: $\bigwedge y. y \in T \Longrightarrow h y \in S \wedge g(h y) = y$
and *id*: $\bigwedge y. y \in T \Longrightarrow h' y \circ g'(h y) = \text{id}$
shows $(\lambda x. |\det(\text{matrix}(g' x))| * f(g x)) \text{ absolutely\_integrable\_on } S \wedge$
 $\text{integral } S (\lambda x. |\det(\text{matrix}(g' x))| * f(g x)) = b \longleftrightarrow$
 $f \text{ absolutely\_integrable\_on } T \wedge \text{integral } T f = b$
(is ?lhs = ?rhs)

proof –

have *Teq*: $T = g'S$ **and** *Seq*: $S = h'T$
using *hg gh image_iff* **by** *fastforce*
let *?DP* = $\lambda x. |\det(\text{matrix}(g' x))| * f(g x)$ **and** *?DN* = $\lambda x. |\det(\text{matrix}(g' x))| * -f(g x)$
have $+$: $(?DP \text{ has\_integral } b) \{x \in S. f(g x) > 0\} \longleftrightarrow (f \text{ has\_integral } b) \{y \in T. f y > 0\}$ **for** b
proof (*rule cov_invertible_nonneg_eq*)
have $*$: $(\lambda x. f(g x)) - ' Y \cap \{x \in S. f(g x) > 0\}$
 $= ((\lambda x. f(g x)) - ' Y \cap S) \cap \{x \in S. f(g x) > 0\}$ **for** Y
by *auto*
show $(g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } \{x \in S. f(g x) > 0\})$ **if** $x \in \{x \in S. f(g x) > 0\}$ **for** x
using *that der_g has_derivative_subset* **by** *fastforce*
show $(h \text{ has\_derivative } h' y) \text{ (at } y \text{ within } \{y \in T. f y > 0\})$ **if** $y \in \{y \in T. f y > 0\}$ **for** y
using *that der_h has_derivative_subset* **by** *fastforce*
qed (*use gh hg id in auto*)
have $-$: $(?DN \text{ has\_integral } b) \{x \in S. f(g x) < 0\} \longleftrightarrow ((\lambda x. -f x) \text{ has\_integral } b) \{y \in T. f y < 0\}$ **for** b
proof (*rule cov_invertible_nonneg_eq*)
have $*$: $(\lambda x. -f(g x)) - ' y \cap \{x \in S. f(g x) < 0\}$
 $= ((\lambda x. f(g x)) - ' \text{uminus } y \cap S) \cap \{x \in S. f(g x) < 0\}$ **for** y
using *image_iff* **by** *fastforce*

```

show ( $g$  has_derivative  $g'$   $x$ ) (at  $x$  within  $\{x \in S. f(g\ x) < 0\}$ ) if  $x \in \{x \in S. f(g\ x) < 0\}$  for  $x$ 
  using that der_g has_derivative_subset by fastforce
  show ( $h$  has_derivative  $h'$   $y$ ) (at  $y$  within  $\{y \in T. f\ y < 0\}$ ) if  $y \in \{y \in T. f\ y < 0\}$  for  $y$ 
    using that der_h has_derivative_subset by fastforce
qed (use gh hg id in auto)
show ?thesis
proof
  assume LHS: ?lhs
  have eq:  $\{x. (if\ x \in S\ then\ ?DP\ x\ else\ 0) > 0\} = \{x \in S. ?DP\ x > 0\}$ 
     $\{x. (if\ x \in S\ then\ ?DP\ x\ else\ 0) < 0\} = \{x \in S. ?DP\ x < 0\}$ 
    by auto
  have ?DP integrable_on  $S$ 
    using LHS absolutely_integrable_on_def by blast
  then have ( $\lambda x. if\ x \in S\ then\ ?DP\ x\ else\ 0$ ) integrable_on UNIV
    by (simp add: integrable_restrict_UNIV)
  then have D_borel: ( $\lambda x. if\ x \in S\ then\ ?DP\ x\ else\ 0$ )  $\in$  borel_measurable
    (lebesgue_on UNIV)
    using integrable_imp_measurable lebesgue_on_UNIV_eq by blast
  then have SN:  $\{x \in S. ?DP\ x < 0\} \in$  sets lebesgue
    unfolding borel_measurable_vimage_halfspace_component_lt
    by (drule_tac  $x=0$  in spec) (auto simp: eq)
  from D_borel have SP:  $\{x \in S. ?DP\ x > 0\} \in$  sets lebesgue
    unfolding borel_measurable_vimage_halfspace_component_gt
    by (drule_tac  $x=0$  in spec) (auto simp: eq)
  have ?DP absolutely_integrable_on  $\{x \in S. ?DP\ x > 0\}$ 
    using LHS by (fast intro!: set_integrable_subset [OF_, of_  $S$ ] SP)
  then have aP: ?DP absolutely_integrable_on  $\{x \in S. f(g\ x) > 0\}$ 
    by (rule absolutely_integrable_spike_set) (auto simp: zero_less_mult_iff
empty_imp_negligible)
  have ?DP absolutely_integrable_on  $\{x \in S. ?DP\ x < 0\}$ 
    using LHS by (fast intro!: set_integrable_subset [OF_, of_  $S$ ] SN)
  then have aN: ?DP absolutely_integrable_on  $\{x \in S. f(g\ x) < 0\}$ 
    by (rule absolutely_integrable_spike_set) (auto simp: mult_less_0_iff empty_imp_negligible)
  have fN:  $f$  integrable_on  $\{y \in T. f\ y < 0\}$ 
     $integral\ \{y \in T. f\ y < 0\}\ f = integral\ \{x \in S. f(g\ x) < 0\}\ ?DP$ 
    using - [of  $integral\ \{x \in S. f(g\ x) < 0\}\ ?DN$ ] aN
    by (auto simp: set_lebesgue_integral_eq_integral_has_integral_iff integrable_neg_iff)
  have faN:  $f$  absolutely_integrable_on  $\{y \in T. f\ y < 0\}$ 
    apply (rule absolutely_integrable_integrable_bound [where  $g = \lambda x. -f\ x$ ])
    using fN by (auto simp: integrable_neg_iff)
  have fP:  $f$  integrable_on  $\{y \in T. f\ y > 0\}$ 
     $integral\ \{y \in T. f\ y > 0\}\ f = integral\ \{x \in S. f(g\ x) > 0\}\ ?DP$ 
    using + [of  $integral\ \{x \in S. f(g\ x) > 0\}\ ?DP$ ] aP
    by (auto simp: set_lebesgue_integral_eq_integral_has_integral_iff integrable_neg_iff)
  have faP:  $f$  absolutely_integrable_on  $\{y \in T. f\ y > 0\}$ 
    apply (rule absolutely_integrable_integrable_bound [where  $g = f$ ])
    using fP by auto

```

```

have fa: f absolutely_integrable_on ( $\{y \in T. f\ y < 0\} \cup \{y \in T. f\ y > 0\}$ )
  by (rule absolutely_integrable_Un [OF faN faP])
show ?rhs
proof
  have eq:  $((\text{if } x \in T \wedge f\ x < 0 \vee x \in T \wedge 0 < f\ x \text{ then } 1 \text{ else } 0) * f\ x)$ 
    =  $(\text{if } x \in T \text{ then } 1 \text{ else } 0) * f\ x$  for x
  by auto
  show f absolutely_integrable_on T
    using fa by (simp add: indicator_def of_bool_def set_integrable_def eq)
  have [simp]:  $\{y \in T. f\ y < 0\} \cap \{y \in T. 0 < f\ y\} = \{\}$  for T and f ::
    ( $\text{real}^n$ )  $\Rightarrow$  real
  by auto
  have integral T f = integral ( $\{y \in T. f\ y < 0\} \cup \{y \in T. f\ y > 0\}$ ) f
    by (intro empty_imp_negligible integral_spike_set) (auto simp: eq)
  also have ... = integral  $\{y \in T. f\ y < 0\}$  f + integral  $\{y \in T. f\ y > 0\}$  f
    using fN fP by simp
  also have ... = integral  $\{x \in S. f\ (g\ x) < 0\}$  ?DP + integral  $\{x \in S. 0 <$ 
    f (g x)  $\}$  ?DP
    by (simp add: fN fP)
  also have ... = integral  $(\{x \in S. f\ (g\ x) < 0\} \cup \{x \in S. 0 < f\ (g\ x)\})$  ?DP
    using aP aN by (simp add: set_lebesgue_integral_eq_integral)
  also have ... = integral S ?DP
    by (intro empty_imp_negligible integral_spike_set) auto
  also have ... = b
    using LHS by simp
  finally show integral T f = b .
qed
next
assume RHS: ?rhs
have eq:  $\{x. (\text{if } x \in T \text{ then } f\ x \text{ else } 0) > 0\} = \{x \in T. f\ x > 0\}$ 
   $\{x. (\text{if } x \in T \text{ then } f\ x \text{ else } 0) < 0\} = \{x \in T. f\ x < 0\}$ 
  by auto
have f integrable_on T
  using RHS absolutely_integrable_on_def by blast
then have  $(\lambda x. \text{if } x \in T \text{ then } f\ x \text{ else } 0)$  integrable_on UNIV
  by (simp add: integrable_restrict_UNIV)
  then have D_borel:  $(\lambda x. \text{if } x \in T \text{ then } f\ x \text{ else } 0) \in \text{borel\_measurable}$ 
    (lebesgue_on UNIV)
  using integrable_imp_measurable lebesgue_on_UNIV_eq by blast
then have TN:  $\{x \in T. f\ x < 0\} \in \text{sets lebesgue}$ 
  unfolding borel_measurable_vimage_halfspace_component_lt
  by (drule_tac x=0 in spec) (auto simp: eq)
from D_borel have TP:  $\{x \in T. f\ x > 0\} \in \text{sets lebesgue}$ 
  unfolding borel_measurable_vimage_halfspace_component_gt
  by (drule_tac x=0 in spec) (auto simp: eq)
have aint: f absolutely_integrable_on  $\{y. y \in T \wedge 0 < (f\ y)\}$ 
  f absolutely_integrable_on  $\{y. y \in T \wedge (f\ y) < 0\}$ 
  and intT: integral T f = b
  using set_integrable_subset [of _ T] TP TN RHS

```

```

    by blast+
  show ?lhs
  proof
    have fN:  $f \text{ integrable\_on } \{v \in T. f v < 0\}$ 
      using absolutely_integrable_on_def aint by blast
    then have DN:  $(?DN \text{ has\_integral integral } \{y \in T. f y < 0\} (\lambda x. - f x)) \{x \in S. f (g x) < 0\}$ 
      using - [of integral  $\{y \in T. f y < 0\} (\lambda x. - f x)$ ]
      by (simp add: has_integral_neg_iff integrable_integral)
    have aDN:  $?DP \text{ absolutely\_integrable\_on } \{x \in S. f (g x) < 0\}$ 
      apply (rule absolutely_integrable_integrable_bound [where  $g = ?DN$ ])
      using DN hg by (fastforce simp: abs_mult integrable_neg_iff)+
    have fP:  $f \text{ integrable\_on } \{v \in T. f v > 0\}$ 
      using absolutely_integrable_on_def aint by blast
    then have DP:  $(?DP \text{ has\_integral integral } \{y \in T. f y > 0\} f) \{x \in S. f (g x) > 0\}$ 
      using + [of integral  $\{y \in T. f y > 0\} f$ ]
      by (simp add: has_integral_neg_iff integrable_integral)
    have aDP:  $?DP \text{ absolutely\_integrable\_on } \{x \in S. f (g x) > 0\}$ 
      apply (rule absolutely_integrable_integrable_bound [where  $g = ?DP$ ])
      using DP hg by (fastforce simp: integrable_neg_iff)+
    have eq:  $(\text{if } x \in S \text{ then } 1 \text{ else } 0) * ?DP x = (\text{if } x \in S \wedge f (g x) < 0 \vee x \in S \wedge f (g x) > 0 \text{ then } 1 \text{ else } 0) * ?DP x \text{ for } x$ 
      by force
    have  $?DP \text{ absolutely\_integrable\_on } (\{x \in S. f (g x) < 0\} \cup \{x \in S. f (g x) > 0\})$ 
      by (rule absolutely_integrable_Un [OF aDN aDP])
    then show I:  $?DP \text{ absolutely\_integrable\_on } S$ 
      by (simp add: indicator_def of_bool_def eq set_integrable_def)
    have [simp]:  $\{y \in S. f y < 0\} \cap \{y \in S. 0 < f y\} = \{\}$  for  $S$  and  $f :: (real^n::_) \Rightarrow real$ 
      by auto
    have  $\text{integral } S ?DP = \text{integral } (\{x \in S. f (g x) < 0\} \cup \{x \in S. f (g x) > 0\}) ?DP$ 
      by (intro empty_imp_negligible integral_spike_set) auto
    also have  $\dots = \text{integral } \{x \in S. f (g x) < 0\} ?DP + \text{integral } \{x \in S. 0 < f (g x)\} ?DP$ 
      using aDN aDP by (simp add: set_lebesgue_integral_eq_integral)
    also have  $\dots = - \text{integral } \{y \in T. f y < 0\} (\lambda x. - f x) + \text{integral } \{y \in T. f y > 0\} f$ 
      using DN DP by (auto simp: has_integral_iff)
    also have  $\dots = \text{integral } (\{x \in T. f x < 0\} \cup \{x \in T. 0 < f x\}) f$ 
      by (simp add: fN fP)
    also have  $\dots = \text{integral } T f$ 
      by (intro empty_imp_negligible integral_spike_set) auto
    also have  $\dots = b$ 
      using intT by simp
    finally show  $\text{integral } S ?DP = b$  .
  qed

```

qed
qed

lemma *cv_inv_version3*:

fixes $f :: \text{real}^m :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n$ **and** $g :: \text{real}^m :: \_ \Rightarrow \text{real}^m :: \_$
assumes $\text{der\_}g: \bigwedge x. x \in S \implies (g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } S)$
and $\text{der\_}h: \bigwedge y. y \in T \implies (h \text{ has\_derivative } h' y) \text{ (at } y \text{ within } T)$
and $hg: \bigwedge x. x \in S \implies g x \in T \wedge h(g x) = x$
and $gh: \bigwedge y. y \in T \implies h y \in S \wedge g(h y) = y$
and $\text{id}: \bigwedge y. y \in T \implies h' y \circ g'(h y) = \text{id}$
shows $(\lambda x. |\det(\text{matrix}(g' x))| *_R f(g x)) \text{ absolutely\_integrable\_on } S \wedge$
 $\text{integral } S (\lambda x. |\det(\text{matrix}(g' x))| *_R f(g x)) = b$
 $\longleftrightarrow f \text{ absolutely\_integrable\_on } T \wedge \text{integral } T f = b$
proof –
let $?D = \lambda x. |\det(\text{matrix}(g' x))| *_R f(g x)$
have $((\lambda x. |\det(\text{matrix}(g' x))| * f(g x) \$ i) \text{ absolutely\_integrable\_on } S \wedge \text{integral}$
 $S (\lambda x. |\det(\text{matrix}(g' x))| * (f(g x) \$ i)) = b \$ i) \longleftrightarrow$
 $((\lambda x. f x \$ i) \text{ absolutely\_integrable\_on } T \wedge \text{integral } T (\lambda x. f x \$ i) = b \$$
i) **for** *i*
by (*rule cov_invertible_real [OF der_g der_h hg gh id]*)
then have $?D \text{ absolutely\_integrable\_on } S \wedge (?D \text{ has\_integral } b) S \longleftrightarrow$
 $f \text{ absolutely\_integrable\_on } T \wedge (f \text{ has\_integral } b) T$
unfolding *absolutely_integrable_componentwise_iff [where f=f] has_integral_componentwise_iff*
[of f]
 $\text{absolutely\_integrable\_componentwise\_iff [where f=?D] has\_integral\_componentwise\_iff}$
[of ?D]
by (*auto simp: all_conj_distrib Basis_vec_def cart_eq_inner_axis [symmetric]*)
 $\text{has\_integral\_iff set\_lebesgue\_integral\_eq\_integral}$
then show *?thesis*
using *absolutely_integrable_on_def* **by** *blast*
qed

lemma *cv_inv_version4*:

fixes $f :: \text{real}^m :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n$ **and** $g :: \text{real}^m :: \_ \Rightarrow \text{real}^m :: \_$
assumes $\text{der\_}g: \bigwedge x. x \in S \implies (g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } S) \wedge \text{invert-}$
 $\text{ible}(\text{matrix}(g' x))$
and $hg: \bigwedge x. x \in S \implies \text{continuous\_on } (g \text{ ' } S) h \wedge h(g x) = x$
shows $(\lambda x. |\det(\text{matrix}(g' x))| *_R f(g x)) \text{ absolutely\_integrable\_on } S \wedge$
 $\text{integral } S (\lambda x. |\det(\text{matrix}(g' x))| *_R f(g x)) = b$
 $\longleftrightarrow f \text{ absolutely\_integrable\_on } (g \text{ ' } S) \wedge \text{integral } (g \text{ ' } S) f = b$
proof –
have $\forall x. \exists h'. x \in S$
 $\longrightarrow (g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } S) \wedge \text{linear } h' \wedge g' x \circ h' = \text{id} \wedge$
 $h' \circ g' x = \text{id}$
using *der_g matrix_invertible has_derivative_linear* **by** *blast*
then obtain h' **where** $h':$
 $\bigwedge x. x \in S$

```

    => (g has_derivative g' x) (at x within S) ∧
      linear (h' x) ∧ g' x ∘ (h' x) = id ∧ (h' x) ∘ g' x = id
  by metis
show ?thesis
proof (rule cv_inv_version3)
  show  $\bigwedge y. y \in g \text{ ' } S \implies (h \text{ has\_derivative } h' (h y)) \text{ (at } y \text{ within } g \text{ ' } S)$ 
    using h' hg
  by (force simp: continuous_on_eq_continuous_within intro!: has_derivative_inverse_within)
qed (use h' hg in auto)
qed

```

```

theorem has_absolute_integral_change_of_variables_invertible:
  fixes f :: real^m::{finite,wellorder} => real^n and g :: real^m::_ => real^m::_
  assumes der_g:  $\bigwedge x. x \in S \implies (g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } S)$ 
    and hg:  $\bigwedge x. x \in S \implies h(g x) = x$ 
    and conth: continuous_on (g ' S) h
  shows ( $\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)$ ) absolutely_integrable_on S ∧ integral
    S ( $\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)$ ) = b  $\longleftrightarrow$ 
      f absolutely_integrable_on (g ' S) ∧ integral (g ' S) f = b
    (is ?lhs = ?rhs)
proof -
  let ?S = {x ∈ S. invertible (matrix (g' x))} and ?D =  $\lambda x. |\det (\text{matrix } (g' x))|$ 
  *_R f(g x)
  have *: ?D absolutely_integrable_on ?S ∧ integral ?S ?D = b
     $\longleftrightarrow$  f absolutely_integrable_on (g ' ?S) ∧ integral (g ' ?S) f = b
  proof (rule cv_inv_version4)
    show (g has_derivative g' x) (at x within ?S) ∧ invertible (matrix (g' x))
      if x ∈ ?S for x
      using der_g that has_derivative_subset that by fastforce
    show continuous_on (g ' ?S) h ∧ h (g x) = x
      if x ∈ ?S for x
      using that continuous_on_subset [OF conth] by (simp add: hg image_mono)
    qed
    have (g has_derivative g' x) (at x within {x ∈ S. rank (matrix (g' x)) <
      CARD('m)}) if x ∈ S for x
      by (metis (no_types, lifting) der_g has_derivative_subset mem_Collect_eq
        subsetI that)
    then have negligible (g ' {x ∈ S.  $\neg$  invertible (matrix (g' x))})
      by (auto simp: invertible_det_nz det_eq_0_rank intro: baby_Sard)
    then have neg: negligible {x ∈ g ' S. x  $\notin$  g ' ?S ∧ f x  $\neq$  0}
      by (auto intro: negligible_subset)
    have [simp]: {x ∈ g ' ?S. x  $\notin$  g ' S ∧ f x  $\neq$  0} = {}
      by auto
    have ?D absolutely_integrable_on ?S ∧ integral ?S ?D = b
       $\longleftrightarrow$  ?D absolutely_integrable_on S ∧ integral S ?D = b
    apply (intro conj_cong absolutely_integrable_spike_set_eq)
    apply (auto simp: integral_spike_set invertible_det_nz empty_imp_negligible
      neg)

```

```

done
moreover
have  $f$  absolutely_integrable_on  $(g \text{ ' } ?S) \wedge \text{integral } (g \text{ ' } ?S) f = b$ 
 $\longleftrightarrow f$  absolutely_integrable_on  $(g \text{ ' } S) \wedge \text{integral } (g \text{ ' } S) f = b$ 
by (auto intro!: conj_cong absolutely_integrable_spike_set_eq integral_spike_set
neg)
ultimately
show ?thesis
using * by blast
qed

```

```

theorem has_absolute_integral_change_of_variables_compact:
fixes  $f :: \text{real}^m :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n$  and  $g :: \text{real}^m :: \_ \Rightarrow \text{real}^m :: \_$ 
assumes compact  $S$ 
and  $\text{der\_}g: \bigwedge x. x \in S \implies (g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } S)$ 
and  $\text{inj}: \text{inj\_on } g \ S$ 
shows  $((\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)) \text{ absolutely\_integrable\_on } S \wedge$ 
 $\text{integral } S (\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)) = b$ 
 $\longleftrightarrow f \text{ absolutely\_integrable\_on } (g \text{ ' } S) \wedge \text{integral } (g \text{ ' } S) f = b)$ 
proof -
obtain  $h$  where  $hg: \bigwedge x. x \in S \implies h(g x) = x$ 
using  $\text{inj}$  by (metis the_inv_into_f_f)
have  $\text{conth}: \text{continuous\_on } (g \text{ ' } S) \ h$ 
by (metis <compact  $S$ > continuous_on_inv der_g has_derivative_continuous_on
hg)
show ?thesis
by (rule has_absolute_integral_change_of_variables_invertible [OF der_g hg
conth])
qed

```

```

lemma has_absolute_integral_change_of_variables_compact_family:
fixes  $f :: \text{real}^m :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n$  and  $g :: \text{real}^m :: \_ \Rightarrow \text{real}^m :: \_$ 
assumes compact:  $\bigwedge n :: \text{nat}. \text{compact } (F n)$ 
and  $\text{der\_}g: \bigwedge x. x \in (\bigcup n. F n) \implies (g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } (\bigcup n. F n))$ 
and  $\text{inj}: \text{inj\_on } g (\bigcup n. F n)$ 
shows  $((\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)) \text{ absolutely\_integrable\_on } (\bigcup n. F n)$ 
 $\wedge$ 
 $\text{integral } (\bigcup n. F n) (\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)) = b$ 
 $\longleftrightarrow f \text{ absolutely\_integrable\_on } (g \text{ ' } (\bigcup n. F n)) \wedge \text{integral } (g \text{ ' } (\bigcup n. F n)) f$ 
 $= b)$ 
proof -
let ?D =  $\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)$ 
let ?U =  $\lambda n. \bigcup m \leq n. F m$ 
let ?lift =  $\text{vec} :: \text{real} \Rightarrow \text{real}^1$ 
have  $F\_leb: F m \in \text{sets lebesgue}$  for  $m$ 

```

```

    by (simp add: compact borel_compact)
  have iff:  $(\lambda x. |\det (\text{matrix } (g' x))| *_R f (g x)) \text{ absolutely\_integrable\_on } (?U n)$ 
 $\wedge$ 
 $\int_{(?U n)} (\lambda x. |\det (\text{matrix } (g' x))| *_R f (g x)) = b$ 
 $\longleftrightarrow f \text{ absolutely\_integrable\_on } (g ' (?U n)) \wedge \int_{(g ' (?U n))} f = b$ 
for n b and f ::  $\text{real}^m \Rightarrow \text{real}^k$ 
  proof (rule has_absolute_integral_change_of_variables_compact)
    show compact (?U n)
      by (simp add: compact_compact_UN)
    show (g has_derivative g' x) (at x within (?U n))
      if x  $\in$  ?U n for x
      using that by (blast intro!: has_derivative_subset [OF der_g])
    show inj_on g (?U n)
      using inj by (auto simp: inj_on_def)
  qed
  show ?thesis
    unfolding image_UN
  proof safe
    assume DS: ?D absolutely_integrable_on  $(\bigcup n. F n)$ 
    and b:  $b = \int_{(\bigcup n. F n)} ?D$ 
    have DU:  $\bigwedge n. ?D \text{ absolutely\_integrable\_on } (?U n)$ 
       $(\lambda n. \int_{(?U n)} ?D) \longrightarrow \int_{(\bigcup n. F n)} ?D$ 
    using integral_countable_UN [OF DS F_leb] by auto
    with iff have fag:  $f \text{ absolutely\_integrable\_on } g ' (?U n)$ 
    and fg_int:  $\int_{(\bigcup m \leq n. g ' F m)} f = \int_{(?U n)} ?D$  for n
    by (auto simp: image_UN)
    let ?h =  $\lambda x. \text{if } x \in (\bigcup m. g ' F m) \text{ then norm}(f x) \text{ else } 0$ 
    have  $(\lambda x. \text{if } x \in (\bigcup m. g ' F m) \text{ then } f x \text{ else } 0) \text{ absolutely\_integrable\_on UNIV}$ 
    proof (rule dominated_convergence_absolutely_integrable)
      show  $(\lambda x. \text{if } x \in (\bigcup m \leq k. g ' F m) \text{ then } f x \text{ else } 0) \text{ absolutely\_integrable\_on UNIV}$  for k
        unfolding absolutely_integrable_restrict_UNIV
        using fag by (simp add: image_UN)
      let ?nf =  $\lambda n x. \text{if } x \in (\bigcup m \leq n. g ' F m) \text{ then norm}(f x) \text{ else } 0$ 
      show ?h integrable_on UNIV
        proof (rule monotone_convergence_increasing [THEN conjunct1])
          show ?nf k integrable_on UNIV for k
            using fag
            unfolding integrable_restrict_UNIV absolutely_integrable_on_def by
(simp add: image_UN)
          { fix n
            have  $(\text{norm} \circ ?D) \text{ absolutely\_integrable\_on } ?U n$ 
              by (intro absolutely_integrable_norm DU)
            then have  $\int_{(g ' ?U n)} (\text{norm} \circ f) = \int_{(?U n)} (\text{norm} \circ ?D)$ 
              using iff [of n vec  $\circ \text{norm} \circ f$  integral (?U n)  $(\lambda x. |\det (\text{matrix } (g' x))|$ 
 $*_R (\text{?lift} \circ \text{norm} \circ f) (g x))]$ 
            unfolding absolutely_integrable_on_1_iff integral_on_1_eq by (auto
simp: o_def)
          }
        }
      }
    end
  end

```



```

moreover have bounded (range ( $\lambda k. \text{integral } (?U\ k) (norm \circ ?D)$ ))
  unfolding bounded_iff
proof (rule exI, clarify)
  fix k
  show norm (integral ( $?U\ k$ ) (norm  $\circ ?D$ ))  $\leq$  integral ( $\bigcup n. F\ n$ ) (norm  $\circ$ 
?D)
    unfolding integral_restrict_UNIV [of _ norm  $\circ ?D$ , symmetric]
    proof (rule integral_norm_bound_integral)
    show ( $\lambda x. \text{if } x \in \bigcup (F\ \{..k\}) \text{ then } (norm \circ ?D)\ x \text{ else } 0$ ) integrable_on
UNIV
      ( $\lambda x. \text{if } x \in (\bigcup n. F\ n) \text{ then } (norm \circ ?D)\ x \text{ else } 0$ ) integrable_on UNIV
      using DU(1) DS
    unfolding absolutely_integrable_on_def o_def integrable_restrict_UNIV
by auto
    qed auto
  qed
ultimately show bounded (range ( $\lambda k. \text{integral UNIV } (?nf\ k)$ ))
  by (simp add: integral_restrict_UNIV image_UN [symmetric] o_def)
next
  show ( $\lambda k. \text{if } x \in (\bigcup m \leq k. g\ \{..F\ m\}) \text{ then } norm\ (f\ x) \text{ else } 0$ )
     $\longrightarrow$  ( $\text{if } x \in (\bigcup m. g\ \{..F\ m\}) \text{ then } norm\ (f\ x) \text{ else } 0$ ) for x
    by (force intro: tendsto_eventually_eventually_sequentiallyI)
  qed auto
next
  show ( $\lambda k. \text{if } x \in (\bigcup m \leq k. g\ \{..F\ m\}) \text{ then } f\ x \text{ else } 0$ )
     $\longrightarrow$  ( $\text{if } x \in (\bigcup m. g\ \{..F\ m\}) \text{ then } f\ x \text{ else } 0$ ) for x
  proof clarsimp
    fix m y
    assume  $y \in F\ m$ 
    show ( $\lambda k. \text{if } \exists x \in \{..k\}. g\ y \in g\ \{..F\ x\} \text{ then } f\ (g\ y) \text{ else } 0$ )  $\longrightarrow$   $f\ (g\ y)$ 
    using  $\langle y \in F\ m \rangle$  by (force intro: tendsto_eventually_eventually_sequentiallyI
[of m])
    qed
  qed auto
then show fai:  $f$  absolutely_integrable_on ( $\bigcup m. g\ \{..F\ m\}$ )
  using absolutely_integrable_restrict_UNIV by blast
show integral ( $(\bigcup x. g\ \{..F\ x\})\ f$ ) = integral ( $\bigcup n. F\ n$ ) ?D
proof (rule LIMSEQ_unique)
  show ( $\lambda n. \text{integral } (?U\ n)\ ?D$ )  $\longrightarrow$  integral ( $\bigcup x. g\ \{..F\ x\}\ f$ )
    unfolding fg_int [symmetric]
  proof (rule integral_countable_UN [OF fai])
    show  $g\ \{..F\ m\} \in \text{sets lebesgue for } m$ 
    proof (rule differentiable_image_in_sets_lebesgue [OF F_leb])
      show  $g$  differentiable_on  $F\ m$ 
      by (meson der_g differentiableI UnionI differentiable_on_def differen-
tible_on_subset rangeI subsetI)
    qed auto
  qed
next

```

```

    show ( $\lambda n. \text{integral } (?U\ n) \ ?D$ )  $\longrightarrow$   $\text{integral } (\bigcup n. F\ n) \ ?D$ 
    by (rule DU)
  qed
next
  assume fs:  $f \text{ absolutely\_integrable\_on } (\bigcup x. g \ ' F\ x)$ 
  and b:  $b = \text{integral } ((\bigcup x. g \ ' F\ x))\ f$ 
  have gF_leb:  $g \ ' F\ m \in \text{sets lebesgue for } m$ 
  proof (rule differentiable_image_in_sets_lebesgue [OF F_leb])
    show  $g \text{ differentiable\_on } F\ m$ 
    using der_g unfolding differentiable_def differentiable_on_def
    by (meson Sup_upper UNIV_I UnionI has_derivative_subset image_eqI)
  qed auto
  have fgU:  $\bigwedge n. f \text{ absolutely\_integrable\_on } (\bigcup m \leq n. g \ ' F\ m)$ 
  ( $\lambda n. \text{integral } (\bigcup m \leq n. g \ ' F\ m)\ f$ )  $\longrightarrow$   $\text{integral } (\bigcup m. g \ ' F\ m)\ f$ 
  using integral_countable_UN [OF fs gF_leb] by auto
  with iff have DUn:  $?D \text{ absolutely\_integrable\_on } ?U\ n$ 
  and D_int:  $\text{integral } (?U\ n) \ ?D = \text{integral } (\bigcup m \leq n. g \ ' F\ m)\ f$  for  $n$ 
  by (auto simp: image_UN)
  let ?h =  $\lambda x. \text{if } x \in (\bigcup n. F\ n) \text{ then } \text{norm}(?D\ x) \text{ else } 0$ 
  have ( $\lambda x. \text{if } x \in (\bigcup n. F\ n) \text{ then } ?D\ x \text{ else } 0$ )  $\text{absolutely\_integrable\_on UNIV}$ 
  proof (rule dominated_convergence_absolutely_integrable)
    show ( $\lambda x. \text{if } x \in ?U\ k \text{ then } ?D\ x \text{ else } 0$ )  $\text{absolutely\_integrable\_on UNIV for } k$ 
    k
    unfolding absolutely_integrable_restrict_UNIV using DUn by simp
    let ?nD =  $\lambda n\ x. \text{if } x \in ?U\ n \text{ then } \text{norm}(?D\ x) \text{ else } 0$ 
    show ?h  $\text{integrable\_on UNIV}$ 
    proof (rule monotone_convergence_increasing [THEN conjunct1])
      show ?nD  $k \text{ integrable\_on UNIV for } k$ 
      using DUn
      unfolding integrable_restrict_UNIV absolutely_integrable_on_def by
    (simp add: image_UN)
    { fix n::nat
      have ( $\text{norm} \circ f$ )  $\text{absolutely\_integrable\_on } (\bigcup m \leq n. g \ ' F\ m)$ 
      apply (rule absolutely_integrable_norm)
      using fgU by blast
      then have  $\text{integral } (?U\ n) (\text{norm} \circ ?D) = \text{integral } (g \ ' ?U\ n) (\text{norm} \circ f)$ 
      using iff [of n ?lift  $\circ \text{norm} \circ f$  integral ( $g \ ' ?U\ n$ ) (?lift  $\circ \text{norm} \circ f$ )]
      unfolding absolutely_integrable_on_1_iff integral_on_1_eq image_UN
    by (auto simp: o_def)
    }
    moreover have bounded ( $\text{range } (\lambda k. \text{integral } (g \ ' ?U\ k) (\text{norm} \circ f))$ )
    unfolding bounded_iff
    proof (rule exI, clarify)
      fix k
      show  $\text{norm } (\text{integral } (g \ ' ?U\ k) (\text{norm} \circ f)) \leq \text{integral } (g \ ' (\bigcup n. F\ n))$ 
      (norm  $\circ f$ )
      unfolding integral_restrict_UNIV [of _ norm  $\circ f$ , symmetric]
      proof (rule integral_norm_bound_integral)
        show ( $\lambda x. \text{if } x \in g \ ' ?U\ k \text{ then } (\text{norm} \circ f)\ x \text{ else } 0$ )  $\text{integrable\_on UNIV}$ 

```

```

      (λx. if x ∈ g ‘ (⋃ n. F n) then (norm ∘ f) x else 0) integrable_on
UNIV
  using fgU fs
  unfolding absolutely_integrable_on_def o_def integrable_restrict_UNIV
  by (auto simp: image_UN)
qed auto
qed
ultimately show bounded (range (λk. integral UNIV (?nD k)))
  unfolding integral_restrict_UNIV image_UN [symmetric] o_def by simp
next
  show (λk. if x ∈ ?U k then norm (?D x) else 0) ⟶ (if x ∈ (⋃ n. F n)
then norm (?D x) else 0) for x
    by (force intro: tendsto_eventually eventually_sequentiallyI)
  qed auto
next
  show (λk. if x ∈ ?U k then ?D x else 0) ⟶ (if x ∈ (⋃ n. F n) then ?D x
else 0) for x
    proof clarsimp
      fix n
      assume x ∈ F n
      show (λm. if ∃ j ∈ {..m}. x ∈ F j then ?D x else 0) ⟶ ?D x
        using ⟨x ∈ F n⟩ by (auto intro!: tendsto_eventually eventually_sequentiallyI
[of n])
      qed
    qed auto
  then show Dai: ?D absolutely_integrable_on (⋃ n. F n)
    unfolding absolutely_integrable_restrict_UNIV by simp
  show integral (⋃ n. F n) ?D = integral ((⋃ x. g ‘ F x) f)
  proof (rule LIMSEQ_unique)
    show (λn. integral (⋃ m ≤ n. g ‘ F m) f) ⟶ integral (⋃ x. g ‘ F x) f
      by (rule fgU)
    show (λn. integral (⋃ m ≤ n. g ‘ F m) f) ⟶ integral (⋃ n. F n) ?D
      unfolding D_int [symmetric] by (rule integral_countable_UN [OF Dai
F_leb])
    qed
  qed
qed

```

theorem has_absolute_integral_change_of_variables:

```

  fixes f :: real^'m::{finite,wellorder} ⇒ real^'n and g :: real^'m::_ ⇒ real^'n::_
  assumes S: S ∈ sets lebesgue
  and der_g: ∧x. x ∈ S ⟹ (g has_derivative g' x) (at x within S)
  and inj: inj_on g S
  shows (λx. |det (matrix (g' x))| *R f(g x)) absolutely_integrable_on S ∧
        integral S (λx. |det (matrix (g' x))| *R f(g x)) = b
    ⟷ f absolutely_integrable_on (g ‘ S) ∧ integral (g ‘ S) f = b
  proof –
    obtain C N where fsigma C and N: N ∈ null_sets lebesgue and CNS: C ∪ N

```

```

= S and disjoint C N
  using lebesgue_set_almost_fsigma [OF S] .
then obtain F :: nat  $\Rightarrow$  (realm::_) set
  where F: range F  $\subseteq$  Collect compact and Ceq: C = Union(range F)
  using fsigma_Union_compact by metis
have negligible N
  using N by (simp add: negligible_iff_null_sets)
let ?D =  $\lambda x. |\det(\text{matrix } (g' x))| *_R f (g x)$ 
have ?D absolutely_integrable_on C  $\wedge$  integral C ?D = b
 $\longleftrightarrow$  f absolutely_integrable_on (g ' C)  $\wedge$  integral (g ' C) f = b
  unfolding Ceq
proof (rule has_absolute_integral_change_of_variables_compact_family)
  fix n x
  assume x  $\in \bigcup(F ' UNIV)$ 
  then show (g has_derivative g' x) (at x within  $\bigcup(F ' UNIV)$ )
    using Ceq  $\langle C \cup N = S \rangle$  der_g has_derivative_subset by blast
next
  have  $\bigcup(F ' UNIV) \subseteq S$ 
    using Ceq  $\langle C \cup N = S \rangle$  by blast
  then show inj_on g ( $\bigcup(F ' UNIV)$ )
    using inj by (meson inj_on_subset)
qed (use F in auto)
moreover
have ?D absolutely_integrable_on C  $\wedge$  integral C ?D = b
 $\longleftrightarrow$  ?D absolutely_integrable_on S  $\wedge$  integral S ?D = b
proof (rule conj_cong)
  have neg: negligible  $\{x \in C - S. ?D x \neq 0\}$  negligible  $\{x \in S - C. ?D x \neq 0\}$ 
  using CNS by (blast intro: negligible_subset [OF  $\langle$ negligible N $\rangle$ ])+
  then show (?D absolutely_integrable_on C) = (?D absolutely_integrable_on
S)
    by (rule absolutely_integrable_spike_set_eq)
  show (integral C ?D = b)  $\longleftrightarrow$  (integral S ?D = b)
    using integral_spike_set [OF neg] by simp
qed
moreover
have f absolutely_integrable_on (g ' C)  $\wedge$  integral (g ' C) f = b
 $\longleftrightarrow$  f absolutely_integrable_on (g ' S)  $\wedge$  integral (g ' S) f = b
proof (rule conj_cong)
  have g differentiable_on N
    by (metis CNS der_g differentiable_def differentiable_on_def differentiable_on_subset
sup.cobounded2)
  with  $\langle$ negligible N $\rangle$ 
  have neg_gN: negligible (g ' N)
    by (blast intro: negligible_differentiable_image_negligible)
  have neg: negligible  $\{x \in g ' C - g ' S. f x \neq 0\}$ 
    negligible  $\{x \in g ' S - g ' C. f x \neq 0\}$ 
    using CNS by (blast intro: negligible_subset [OF neg_gN])+
  then show (f absolutely_integrable_on g ' C) = (f absolutely_integrable_on g

```

```

‘ S)
  by (rule absolutely_integrable_spike_set_eq)
  show (integral (g ‘ C) f = b)  $\longleftrightarrow$  (integral (g ‘ S) f = b)
  using integral_spike_set [OF neg] by simp
qed
ultimately show ?thesis
  by simp
qed

```

corollary *absolutely_integrable_change_of_variables:*

```

fixes f :: real^m::{finite,wellorder}  $\Rightarrow$  real^n and g :: real^m::_  $\Rightarrow$  real^m::_
assumes S  $\in$  sets lebesgue
  and  $\bigwedge x. x \in S \implies (g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } S)$ 
  and inj_on g S
shows f absolutely_integrable_on (g ‘ S)
   $\longleftrightarrow (\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)) \text{ absolutely\_integrable\_on } S$ 
using assms has_absolute_integral_change_of_variables by blast

```

corollary *integral_change_of_variables:*

```

fixes f :: real^m::{finite,wellorder}  $\Rightarrow$  real^n and g :: real^m::_  $\Rightarrow$  real^m::_
assumes S: S  $\in$  sets lebesgue
  and der_g:  $\bigwedge x. x \in S \implies (g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } S)$ 
  and inj: inj_on g S
  and disj: (f absolutely_integrable_on (g ‘ S)  $\vee$ 
     $(\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)) \text{ absolutely\_integrable\_on } S)$ )
shows integral (g ‘ S) f = integral S  $(\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x))$ 
using has_absolute_integral_change_of_variables [OF S der_g inj] disj
by blast

```

lemma *has_absolute_integral_change_of_variables_1:*

```

fixes f :: real  $\Rightarrow$  real^n::{finite,wellorder} and g :: real  $\Rightarrow$  real
assumes S: S  $\in$  sets lebesgue
  and der_g:  $\bigwedge x. x \in S \implies (g \text{ has\_vector\_derivative } g' x) \text{ (at } x \text{ within } S)$ 
  and inj: inj_on g S
shows  $(\lambda x. |g' x| *_R f(g x)) \text{ absolutely\_integrable\_on } S \wedge$ 
  integral S  $(\lambda x. |g' x| *_R f(g x)) = b$ 
 $\longleftrightarrow f \text{ absolutely\_integrable\_on } (g ‘ S) \wedge \text{integral } (g ‘ S) f = b$ 
proof -
  let ?lift = vec :: real  $\Rightarrow$  real^1
  let ?drop =  $(\lambda x::\text{real}^1. x \$ 1)$ 
  have S': ?lift ‘ S  $\in$  sets lebesgue
  by (auto intro: differentiable_image_in_sets_lebesgue [OF S] differentiable_vec)
  have  $((\lambda x. \text{vec } (g (x \$ 1))) \text{ has\_derivative } (*_R) (g' z)) \text{ (at } (\text{vec } z) \text{ within } ?\text{lift ‘ } S)$ 
  if z  $\in$  S for z
  using der_g [OF that]
  by (simp add: has_vector_derivative_def has_derivative_vector_1)
  then have der':  $\bigwedge x. x \in ?\text{lift ‘ } S \implies$ 

```

```

      (?lift ∘ g ∘ ?drop has_derivative (*R) (g' (?drop x))) (at x within ?lift ' S)
    by (auto simp: o_def)
  have inj': inj_on (vec ∘ g ∘ ?drop) (vec ' S)
    using inj by (simp add: inj_on_def)
  let ?fg = λx. |g' x| *R f(g x)
  have ((λx. ?fg x $ i) absolutely_integrable_on S ∧ ((λx. ?fg x $ i) has_integral
b $ i) S
    ↔ (λx. f x $ i) absolutely_integrable_on g ' S ∧ ((λx. f x $ i) has_integral
b $ i) (g ' S)) for i
    using has_absolute_integral_change_of_variables [OF S' der' inj', of λx.
?lift(f (?drop x) $ i) ?lift (b $ i)]
  unfolding integrable_on_1_iff integral_on_1_eq absolutely_integrable_on_1_iff
absolutely_integrable_drop absolutely_integrable_on_def
  by (auto simp: image_comp o_def integral_vec1_eq has_integral_iff)
  then have ?fg absolutely_integrable_on S ∧ (?fg has_integral b) S
    ↔ f absolutely_integrable_on (g ' S) ∧ (f has_integral b) (g ' S)
  unfolding has_integral_componentwise_iff [where y=b]
    absolutely_integrable_componentwise_iff [where f=f]
    absolutely_integrable_componentwise_iff [where f = ?fg]
  by (force simp: Basis_vec_def cart_eq_inner_axis)
  then show ?thesis
    using absolutely_integrable_on_def by blast
qed

```

corollary *absolutely_integrable_change_of_variables_1*:

```

  fixes f :: real ⇒ realn::finite,wellorder and g :: real ⇒ real
  assumes S: S ∈ sets lebesgue
    and der_g: ∧x. x ∈ S ⇒ (g has_vector_derivative g' x) (at x within S)
    and inj: inj_on g S
  shows (f absolutely_integrable_on g ' S ↔
    (λx. |g' x| *R f(g x)) absolutely_integrable_on S)
  using has_absolute_integral_change_of_variables_1 [OF assms] by auto

```

when $n = (1::'a)$

lemma *has_absolute_integral_change_of_variables_1'*:

```

  fixes f :: real ⇒ real and g :: real ⇒ real
  assumes S: S ∈ sets lebesgue
    and der_g: ∧x. x ∈ S ⇒ (g has_field_derivative g' x) (at x within S)
    and inj: inj_on g S
  shows (λx. |g' x| *R f(g x)) absolutely_integrable_on S ∧
    integral S (λx. |g' x| *R f(g x)) = b
    ↔ f absolutely_integrable_on (g ' S) ∧ integral (g ' S) f = b
  proof -
    have (λx. |g' x| *R vec (f(g x)) :: real ^ 1) absolutely_integrable_on S ∧
      integral S (λx. |g' x| *R vec (f(g x))) = (vec b :: real ^ 1)
      ↔ (λx. vec (f x) :: real ^ 1) absolutely_integrable_on (g ' S) ∧
        integral (g ' S) (λx. vec (f x)) = (vec b :: real ^ 1)
      using assms unfolding has_field_derivative_iff_has_vector_derivative

```

```

  by (intro has_absolute_integral_change_of_variables_1 assms) auto
  thus ?thesis
  by (simp add: absolutely_integrable_on_1_iff integral_on_1_eq)
qed

```

6.47.6 Change of variables for integrals: special case of linear function

```

lemma has_absolute_integral_change_of_variables_linear:
  fixes f :: real^'m::{finite,wellorder}  $\Rightarrow$  real^'n and g :: real^'m::_  $\Rightarrow$  real^'m::_
  assumes linear g
  shows ( $\lambda x. |det (matrix g)| *_{\mathbb{R}} f(g x)$ ) absolutely_integrable_on S  $\wedge$ 
    integral S ( $\lambda x. |det (matrix g)| *_{\mathbb{R}} f(g x)$ ) = b
     $\longleftrightarrow$  f absolutely_integrable_on (g ' S)  $\wedge$  integral (g ' S) f = b
proof (cases det(matrix g) = 0)
  case True
  then have negligible(g ' S)
    using assms det_nz_iff_inj negligible_linear_singular_image by blast
  with True show ?thesis
  by (auto simp: absolutely_integrable_on_def integrable_negligible integral_negligible)
next
  case False
  then obtain h where h:  $\bigwedge x. x \in S \implies h (g x) = x$  linear h
    using assms det_nz_iff_inj linear_injective_isomorphism by metis
  show ?thesis
  proof (rule has_absolute_integral_change_of_variables_invertible)
    show (g has_derivative g) (at x within S) for x
    by (simp add: assms linear_imp_has_derivative)
    show continuous_on (g ' S) h
    using continuous_on_eq continuous_within has_derivative_continuous linear_imp_has_derivative h by blast
  qed (use h in auto)
qed

```

```

lemma absolutely_integrable_change_of_variables_linear:
  fixes f :: real^'m::{finite,wellorder}  $\Rightarrow$  real^'n and g :: real^'m::_  $\Rightarrow$  real^'m::_
  assumes linear g
  shows ( $\lambda x. |det (matrix g)| *_{\mathbb{R}} f(g x)$ ) absolutely_integrable_on S
     $\longleftrightarrow$  f absolutely_integrable_on (g ' S)
  using assms has_absolute_integral_change_of_variables_linear by blast

```

```

lemma absolutely_integrable_on_linear_image:
  fixes f :: real^'m::{finite,wellorder}  $\Rightarrow$  real^'n and g :: real^'m::_  $\Rightarrow$  real^'m::_
  assumes linear g
  shows f absolutely_integrable_on (g ' S)
     $\longleftrightarrow$  (f  $\circ$  g) absolutely_integrable_on S  $\vee$  det(matrix g) = 0
  unfolding assms absolutely_integrable_change_of_variables_linear [OF assms,
    symmetric] absolutely_integrable_on_scaleR_iff
  by (auto simp: set_integrable_def)

```

lemma *integral_change_of_variables_linear*:
fixes $f :: \text{real}^m :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n$ **and** $g :: \text{real}^m :: \_ \Rightarrow \text{real}^n :: \_$
assumes *linear g*
and $f \text{ absolutely\_integrable\_on } (g \text{ ` } S) \vee (f \circ g) \text{ absolutely\_integrable\_on } S$
shows $\text{integral } (g \text{ ` } S) f = |\det (\text{matrix } g)| *_{\mathbb{R}} \text{integral } S (f \circ g)$
proof –
have $((\lambda x. |\det (\text{matrix } g)| *_{\mathbb{R}} f (g x)) \text{ absolutely\_integrable\_on } S) \vee (f \text{ absolutely\_integrable\_on } g \text{ ` } S)$
using *absolutely_integrable_on_linear_image assms* **by** *blast*
moreover
have *?thesis* **if** $((\lambda x. |\det (\text{matrix } g)| *_{\mathbb{R}} f (g x)) \text{ absolutely\_integrable\_on } S) (f \text{ absolutely\_integrable\_on } g \text{ ` } S)$
using *has_absolute_integral_change_of_variables_linear [OF <linear g>]* *that*
by *(auto simp: o_def)*
ultimately show *?thesis*
using *absolutely_integrable_change_of_variables_linear [OF <linear g>]*
by *blast*
qed

6.47.7 Change of variable for measure

lemma *has_measure_differentiable_image*:
fixes $f :: \text{real}^n :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n :: \_$
assumes $S \in \text{sets lebesgue}$
and $\bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } S)$
and *inj_on f S*
shows $f \text{ ` } S \in \text{lmeasurable} \wedge \text{measure lebesgue } (f \text{ ` } S) = m$
 $\longleftrightarrow ((\lambda x. |\det (\text{matrix } (f' x))|) \text{ has\_integral } m) S$
using *has_absolute_integral_change_of_variables [OF assms, of \lambda x. (1::real^1) vec m]*
unfolding *absolutely_integrable_on_1_iff integral_on_1_eq integrable_on_1_iff absolutely_integrable_on_def*
by *(auto simp: has_integral_iff lmeasurable_iff integrable_on lmeasure_integral)*

lemma *measurable_differentiable_image_eq*:
fixes $f :: \text{real}^n :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n :: \_$
assumes $S \in \text{sets lebesgue}$
and $\bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } S)$
and *inj_on f S*
shows $f \text{ ` } S \in \text{lmeasurable} \longleftrightarrow (\lambda x. |\det (\text{matrix } (f' x))|) \text{ integrable\_on } S$
using *has_measure_differentiable_image [OF assms]*
by *blast*

lemma *measurable_differentiable_image_alt*:
fixes $f :: \text{real}^n :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n :: \_$
assumes $S \in \text{sets lebesgue}$
and $\bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } S)$
and *inj_on f S*


```

shows  $f' S \in \text{lmeasurable} \longleftrightarrow (\lambda x. |\det (\text{matrix } (f' x))|) \text{ absolutely\_integrable\_on } S$ 
using measurable_differentiable_image_eq [OF assms]
by (simp only: absolutely_integrable_on_iff_nonneg)

lemma measure_differentiable_image_eq:
  fixes  $f :: \text{real}^n :: \{\text{finite, wellorder}\} \Rightarrow \text{real}^n :: \_$ 
  assumes  $S: S \in \text{sets lebesgue}$ 
    and  $\text{der\_f}: \bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } S)$ 
    and  $\text{inj}: \text{inj\_on } f S$ 
    and  $\text{intS}: (\lambda x. |\det (\text{matrix } (f' x))|) \text{ integrable\_on } S$ 
  shows  $\text{measure lebesgue } (f' S) = \text{integral } S (\lambda x. |\det (\text{matrix } (f' x))|)$ 
  using measurable_differentiable_image_eq [OF S der_f inj]
    assms has_measure_differentiable_image by blast

```

end

6.48 Lipschitz Continuity

theory Lipschitz

imports

Derivative

begin

definition lipschitz_on

where $\text{lipschitz\_on } C U f \longleftrightarrow (0 \leq C \wedge (\forall x \in U. \forall y \in U. \text{dist } (f x) (f y) \leq C * \text{dist } x y))$

bundle lipschitz_syntax **begin**

notation lipschitz_on ($\_ - \text{lipschitz\_on}$ [1000])

end

bundle no_lipschitz_syntax **begin**

no_notation lipschitz_on ($\_ - \text{lipschitz\_on}$ [1000])

end

unbundle lipschitz_syntax

lemma lipschitz_onI: $L - \text{lipschitz\_on } X f$

if $\bigwedge x y. x \in X \implies y \in X \implies \text{dist } (f x) (f y) \leq L * \text{dist } x y$ $0 \leq L$

using that **by** (auto simp: lipschitz_on_def)

lemma lipschitz_onD:

$\text{dist } (f x) (f y) \leq L * \text{dist } x y$

if $L - \text{lipschitz\_on } X f$ $x \in X$ $y \in X$

using that **by** (auto simp: lipschitz_on_def)

lemma lipschitz_on_nonneg:

$0 \leq L$ **if** $L - \text{lipschitz\_on } X f$

using that **by** (auto simp: lipschitz_on_def)

```

lemma lipschitz_on_normD:
   $\text{norm } (f\ x - f\ y) \leq L * \text{norm } (x - y)$ 
  if lipschitz_on  $L\ X\ f\ x \in X\ y \in X$ 
  using lipschitz_onD[OF that]
  by (simp add: dist_norm)

lemma lipschitz_on_mono:  $L\text{-lipschitz\_on } D\ f\ \text{if } M\text{-lipschitz\_on } E\ f\ D \subseteq E\ M$ 
 $\leq L$ 
  using that
  by (force simp: lipschitz_on_def intro: order_trans[OF _ mult_right_mono])

lemmas lipschitz_on_subset = lipschitz_on_mono[OF _ _ order_refl]
  and lipschitz_on_le = lipschitz_on_mono[OF _ order_refl]

lemma lipschitz_on_leI:
   $L\text{-lipschitz\_on } X\ f$ 
  if  $\bigwedge x\ y. x \in X \implies y \in X \implies x \leq y \implies \text{dist } (f\ x)\ (f\ y) \leq L * \text{dist } x\ y$ 
   $0 \leq L$ 
  for  $f::'a::\{\text{linorder\_topology, ordered\_real\_vector, metric\_space}\} \Rightarrow 'b::\text{metric\_space}$ 
proof (rule lipschitz_onI)
  fix  $x\ y$  assume  $xy: x \in X\ y \in X$ 
  consider  $y \leq x \mid x \leq y$ 
  by (rule le_cases)
  then show  $\text{dist } (f\ x)\ (f\ y) \leq L * \text{dist } x\ y$ 
proof cases
  case 1
  then have  $\text{dist } (f\ y)\ (f\ x) \leq L * \text{dist } y\ x$ 
  by (auto intro!: that xy)
  then show ?thesis
  by (simp add: dist_commute)
  qed (auto intro!: that xy)
qed fact

lemma lipschitz_on_concat:
  fixes  $a\ b\ c::\text{real}$ 
  assumes  $f: L\text{-lipschitz\_on } \{a .. b\}\ f$ 
  assumes  $g: L\text{-lipschitz\_on } \{b .. c\}\ g$ 
  assumes  $fg: f\ b = g\ b$ 
  shows  $\text{lipschitz\_on } L\ \{a .. c\}\ (\lambda x. \text{if } x \leq b \text{ then } f\ x \text{ else } g\ x)$ 
  (is lipschitz_on _ _ ?f)
proof (rule lipschitz_on_leI)
  fix  $x\ y$ 
  assume  $x: x \in \{a..c\}$  and  $y: y \in \{a..c\}$  and  $xy: x \leq y$ 
  consider  $x \leq b \wedge b < y \mid x \geq b \vee y \leq b$  by arith
  then show  $\text{dist } (?f\ x)\ (?f\ y) \leq L * \text{dist } x\ y$ 
proof cases
  case 1
  have  $\text{dist } (f\ x)\ (g\ y) \leq \text{dist } (f\ x)\ (f\ b) + \text{dist } (g\ b)\ (g\ y)$ 

```

```

    unfolding fg by (rule dist_triangle)
  also have  $\text{dist } (f\ x) (f\ b) \leq L * \text{dist } x\ b$ 
    using 1 x
    by (auto intro!: lipschitz_onD[OF f])
  also have  $\text{dist } (g\ b) (g\ y) \leq L * \text{dist } b\ y$ 
    using 1 x y
    by (auto intro!: lipschitz_onD[OF g] lipschitz_onD[OF f])
  finally have  $\text{dist } (f\ x) (g\ y) \leq L * \text{dist } x\ b + L * \text{dist } b\ y$ 
    by simp
  also have  $\dots = L * (\text{dist } x\ b + \text{dist } b\ y)$ 
    by (simp add: algebra_simps)
  also have  $\text{dist } x\ b + \text{dist } b\ y = \text{dist } x\ y$ 
    using 1 x y
    by (auto simp: dist_real_def abs_real_def)
  finally show ?thesis
    using 1 by simp
next
case 2
with lipschitz_onD[OF f, of x y] lipschitz_onD[OF g, of x y] x y xy
show ?thesis
  by (auto simp: fg)
qed
qed (rule lipschitz_on_nonneg[OF f])

lemma lipschitz_on_concat_max:
  fixes a b c::real
  assumes f:  $L\text{-lipschitz\_on } \{a .. b\} f$ 
  assumes g:  $M\text{-lipschitz\_on } \{b .. c\} g$ 
  assumes fg:  $f\ b = g\ b$ 
  shows  $(\max L\ M)\text{-lipschitz\_on } \{a .. c\} (\lambda x. \text{if } x \leq b \text{ then } f\ x \text{ else } g\ x)$ 
proof -
  have lipschitz_on  $(\max L\ M) \{a .. b\} f$  lipschitz_on  $(\max L\ M) \{b .. c\} g$ 
    by (auto intro!: lipschitz_on_mono[OF f order_refl] lipschitz_on_mono[OF g
order_refl])
  from lipschitz_on_concat[OF this fg] show ?thesis .
qed

```

Continuity

proposition *lipschitz_on_uniformly_continuous:*

assumes $L\text{-lipschitz\_on } X\ f$

shows *uniformly_continuous_on* $X\ f$

unfolding *uniformly_continuous_on_def*

proof *safe*

fix $e::\text{real}$

assume $0 < e$

from *assms* have $l: (L+1)\text{-lipschitz\_on } X\ f$

by (rule lipschitz_on_mono) auto

show $\exists d > 0. \forall x \in X. \forall x' \in X. \text{dist } x'\ x < d \longrightarrow \text{dist } (f\ x') (f\ x) < e$

using *lipschitz_onD*[*OF l*] *lipschitz_on_nonneg*[*OF assms*] $\langle 0 < e \rangle$
 by (force intro!: *exI*[**where** $x=e/(L+1)$] *simp*: *field_simps*)
 qed

proposition *lipschitz_on_continuous_on*:
continuous_on X f **if** *L-lipschitz_on X f*
 by (rule *uniformly_continuous_imp_continuous*[*OF lipschitz_on_uniformly_continuous*[*OF that*]])

lemma *lipschitz_on_continuous_within*:
continuous (at x within X) f **if** *L-lipschitz_on X f* $x \in X$
 using *lipschitz_on_continuous_on*[*OF that*(1)] *that*(2)
 by (auto *simp*: *continuous_on_eq_continuous_within*)

Differentiable functions

proposition *bounded_derivative_imp_lipschitz*:
assumes $\bigwedge x. x \in X \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } X)$
assumes *convex*: *convex X*
assumes $\bigwedge x. x \in X \implies \text{onorm } (f' x) \leq C$ $0 \leq C$
shows *C-lipschitz_on X f*
proof (rule *lipschitz_onI*)
 show $\bigwedge x y. x \in X \implies y \in X \implies \text{dist } (f x) (f y) \leq C * \text{dist } x y$
 by (auto intro!: *assms differentiable_bound*[*unfolded dist_norm[symmetric]*, *OF convex*])
 qed *fact*

Structural introduction rules

named_theorems *lipschitz_intros* structural introduction rules for Lipschitz controls

lemma *lipschitz_on_compose* [*lipschitz_intros*]:
 $(D * C)\text{-lipschitz\_on } U (g \circ f)$
if $f: C\text{-lipschitz\_on } U f$ **and** $g: D\text{-lipschitz\_on } (f'U) g$
proof (rule *lipschitz_onI*)
 show $D * C \geq 0$ **using** *lipschitz_on_nonneg*[*OF f*] *lipschitz_on_nonneg*[*OF g*]
 by *auto*
 fix $x y$ **assume** $H: x \in U y \in U$
 have $\text{dist } (g (f x)) (g (f y)) \leq D * \text{dist } (f x) (f y)$
 apply (rule *lipschitz_onD*[*OF g*]) **using** H **by** *auto*
 also have $\dots \leq D * C * \text{dist } x y$
 using *mult_left_mono*[*OF lipschitz_onD*(1)[*OF f H*] *lipschitz_on_nonneg*[*OF g*]] **by** *auto*
 finally **show** $\text{dist } ((g \circ f) x) ((g \circ f) y) \leq D * C * \text{dist } x y$
 unfolding *comp_def* **by** (auto *simp* add: *mult.commute*)
 qed

lemma *lipschitz_on_compose2*:
 $(D * C)\text{-lipschitz\_on } U (\lambda x. g (f x))$

if $C\text{-lipschitz\_on } U f D\text{-lipschitz\_on } (f \cdot U) g$
using $\text{lipschitz\_on\_compose}[OF \text{ that}]$ **by** (simp add: o_def)

lemma $\text{lipschitz\_on\_cong}[\text{cong}]$:
 $C\text{-lipschitz\_on } U g \longleftrightarrow D\text{-lipschitz\_on } V f$
if $C = D \ U = V \ \bigwedge x. x \in V \implies g x = f x$
using that **by** ($\text{auto simp: lipschitz\_on\_def}$)

lemma $\text{lipschitz\_on\_transform}$:

$C\text{-lipschitz\_on } U g$
if $C\text{-lipschitz\_on } U f$
 $\bigwedge x. x \in U \implies g x = f x$
using that
by simp

lemma $\text{lipschitz\_on\_empty\_iff}[\text{simp}]$: $C\text{-lipschitz\_on } \{ \} f \longleftrightarrow C \geq 0$
by ($\text{auto simp: lipschitz\_on\_def}$)

lemma $\text{lipschitz\_on\_insert\_iff}[\text{simp}]$:
 $C\text{-lipschitz\_on } (\text{insert } y X) f \longleftrightarrow$
 $C\text{-lipschitz\_on } X f \wedge (\forall x \in X. \text{dist } (f x) (f y) \leq C * \text{dist } x y)$
by ($\text{auto simp: lipschitz\_on\_def dist\_commute}$)

lemma $\text{lipschitz\_on\_singleton } [\text{lipschitz\_intros}]$: $C \geq 0 \implies C\text{-lipschitz\_on } \{x\}$
 f
and $\text{lipschitz\_on\_empty } [\text{lipschitz\_intros}]$: $C \geq 0 \implies C\text{-lipschitz\_on } \{ \} f$
by simp_all

lemma $\text{lipschitz\_on\_id } [\text{lipschitz\_intros}]$: $1\text{-lipschitz\_on } U (\lambda x. x)$
by ($\text{auto simp: lipschitz\_on\_def}$)

lemma $\text{lipschitz\_on\_constant } [\text{lipschitz\_intros}]$: $0\text{-lipschitz\_on } U (\lambda x. c)$
by ($\text{auto simp: lipschitz\_on\_def}$)

lemma $\text{lipschitz\_on\_add } [\text{lipschitz\_intros}]$:
fixes $f::'a::\text{metric\_space} \Rightarrow 'b::\text{real\_normed\_vector}$
assumes $C\text{-lipschitz\_on } U f$
 $D\text{-lipschitz\_on } U g$
shows $(C+D)\text{-lipschitz\_on } U (\lambda x. f x + g x)$
proof ($\text{rule lipschitz\_onI}$)
show $C + D \geq 0$
using $\text{lipschitz\_on\_nonneg}[OF \text{ assms}(1)]$ $\text{lipschitz\_on\_nonneg}[OF \text{ assms}(2)]$
by auto
fix $x y$ **assume** $H: x \in U y \in U$
have $\text{dist } (f x + g x) (f y + g y) \leq \text{dist } (f x) (f y) + \text{dist } (g x) (g y)$
by ($\text{simp add: dist\_triangle\_add}$)
also have $\dots \leq C * \text{dist } x y + D * \text{dist } x y$
using $\text{lipschitz\_onD}(1)[OF \text{ assms}(1) H]$ $\text{lipschitz\_onD}(1)[OF \text{ assms}(2) H]$ **by**
 auto

finally show $\text{dist } (f x + g x) (f y + g y) \leq (C+D) * \text{dist } x y$ **by** $(\text{auto simp add: algebra_simps})$
qed

lemma *lipschitz_on_cmult* [*lipschitz_intros*]:
fixes $f::'a::\text{metric\_space} \Rightarrow 'b::\text{real\_normed\_vector}$
assumes $C\text{-lipschitz\_on } U f$
shows $(\text{abs}(a) * C)\text{-lipschitz\_on } U (\lambda x. a *_R f x)$
proof (*rule lipschitz_onI*)
show $\text{abs}(a) * C \geq 0$ **using** *lipschitz_on_nonneg*[*OF assms(1)*] **by auto**
fix $x y$ **assume** $H: x \in U y \in U$
have $\text{dist } (a *_R f x) (a *_R f y) = \text{abs}(a) * \text{dist } (f x) (f y)$
by (*metis dist_norm norm_scaleR real_vector.scale_right_diff_distrib*)
also have $\dots \leq \text{abs}(a) * C * \text{dist } x y$
using *lipschitz_onD(1)*[*OF assms(1)*] H **by** (*simp add: Groups.mult_ac(1) mult_left_mono*)
finally show $\text{dist } (a *_R f x) (a *_R f y) \leq |a| * C * \text{dist } x y$ **by auto**
qed

lemma *lipschitz_on_cmult_real* [*lipschitz_intros*]:
fixes $f::'a::\text{metric\_space} \Rightarrow \text{real}$
assumes $C\text{-lipschitz\_on } U f$
shows $(\text{abs}(a) * C)\text{-lipschitz\_on } U (\lambda x. a * f x)$
using *lipschitz_on_cmult*[*OF assms*] **by auto**

lemma *lipschitz_on_cmult_nonneg* [*lipschitz_intros*]:
fixes $f::'a::\text{metric\_space} \Rightarrow 'b::\text{real\_normed\_vector}$
assumes $C\text{-lipschitz\_on } U f$
 $a \geq 0$
shows $(a * C)\text{-lipschitz\_on } U (\lambda x. a *_R f x)$
using *lipschitz_on_cmult*[*OF assms(1)*, *of a*] *assms(2)* **by auto**

lemma *lipschitz_on_cmult_real_nonneg* [*lipschitz_intros*]:
fixes $f::'a::\text{metric\_space} \Rightarrow \text{real}$
assumes $C\text{-lipschitz\_on } U f$
 $a \geq 0$
shows $(a * C)\text{-lipschitz\_on } U (\lambda x. a * f x)$
using *lipschitz_on_cmult_nonneg*[*OF assms*] **by auto**

lemma *lipschitz_on_cmult_upper* [*lipschitz_intros*]:
fixes $f::'a::\text{metric\_space} \Rightarrow 'b::\text{real\_normed\_vector}$
assumes $C\text{-lipschitz\_on } U f$
 $\text{abs}(a) \leq D$
shows $(D * C)\text{-lipschitz\_on } U (\lambda x. a *_R f x)$
apply (*rule lipschitz_on_mono*[*OF lipschitz_on_cmult*[*OF assms(1)*, *of a*], *of* $D * C$])
using *assms(2)* *lipschitz_on_nonneg*[*OF assms(1)*] *mult_right_mono* **by auto**

lemma *lipschitz_on_cmult_real_upper* [*lipschitz_intros*]:

```

fixes f::'a::metric_space  $\Rightarrow$  real
assumes C-lipschitz_on U f
  abs(a)  $\leq$  D
shows (D * C)-lipschitz_on U ( $\lambda x. a * f x$ )
using lipschitz_on_cmult_upper[OF assms] by auto

lemma lipschitz_on_minus[lipschitz_intros]:
  fixes f::'a::metric_space  $\Rightarrow$  'b::real_normed_vector
  assumes C-lipschitz_on U f
  shows C-lipschitz_on U ( $\lambda x. - f x$ )
  by (metis (mono_tags, lifting) assms dist_minus lipschitz_on_def)

lemma lipschitz_on_minus_iff[simp]:
  L-lipschitz_on X ( $\lambda x. - f x$ )  $\longleftrightarrow$  L-lipschitz_on X f
  L-lipschitz_on X (- f)  $\longleftrightarrow$  L-lipschitz_on X f
  for f::'a::metric_space  $\Rightarrow$  'b::real_normed_vector
  using lipschitz_on_minus[of L X f] lipschitz_on_minus[of L X -f]
  by auto

lemma lipschitz_on_diff[lipschitz_intros]:
  fixes f::'a::metric_space  $\Rightarrow$  'b::real_normed_vector
  assumes C-lipschitz_on U f D-lipschitz_on U g
  shows (C + D)-lipschitz_on U ( $\lambda x. f x - g x$ )
  using lipschitz_on_add[OF assms(1) lipschitz_on_minus[OF assms(2)]] by
  auto

lemma lipschitz_on_closure [lipschitz_intros]:
  assumes C-lipschitz_on U f continuous_on (closure U) f
  shows C-lipschitz_on (closure U) f
proof (rule lipschitz_onI)
  show C  $\geq$  0 using lipschitz_on_nonneg[OF assms(1)] by simp
  fix x y assume x  $\in$  closure U y  $\in$  closure U
  obtain u v::nat  $\Rightarrow$  'a where *:  $\bigwedge n. u n \in U \ u \longrightarrow x$ 
   $\bigwedge n. v n \in U \ v \longrightarrow y$ 
  using  $\langle x \in \text{closure } U \rangle \langle y \in \text{closure } U \rangle$  unfolding closure_sequential by blast
  have a:  $(\lambda n. f (u n)) \longrightarrow f x$ 
  using *(1) *(2)  $\langle x \in \text{closure } U \rangle \langle \text{continuous\_on } (\text{closure } U) f \rangle$ 
  unfolding comp_def continuous_on_closure_sequentially[of U f] by auto
  have b:  $(\lambda n. f (v n)) \longrightarrow f y$ 
  using *(3) *(4)  $\langle y \in \text{closure } U \rangle \langle \text{continuous\_on } (\text{closure } U) f \rangle$ 
  unfolding comp_def continuous_on_closure_sequentially[of U f] by auto
  have l:  $(\lambda n. C * \text{dist } (u n) (v n) - \text{dist } (f (u n)) (f (v n))) \longrightarrow C * \text{dist } x y$ 
  -  $\text{dist } (f x) (f y)$ 
  by (intro tendsto_intros * a b)
  have C * dist (u n) (v n) - dist (f (u n)) (f (v n))  $\geq$  0 for n
  using lipschitz_onD(1)[OF assms(1)  $\langle u n \in U \rangle \langle v n \in U \rangle$ ] by simp
  then have C * dist x y - dist (f x) (f y)  $\geq$  0 using LIMSEQ_le_const[OF l,
  of 0] by auto
  then show dist (f x) (f y)  $\leq$  C * dist x y by auto

```

3300

qed

```

lemma lipschitz_on_Pair[lipschitz_intros]:
  assumes f: L-lipschitz_on A f
  assumes g: M-lipschitz_on A g
  shows (sqrt (L2 + M2))-lipschitz_on A (λa. (f a, g a))
proof (rule lipschitz_onI, goal_cases)
  case (1 x y)
  have dist (f x, g x) (f y, g y) = sqrt ((dist (f x) (f y))2 + (dist (g x) (g y))2)
    by (auto simp add: dist_Pair_Pair real_le_sqrt)
  also have ... ≤ sqrt ((L * dist x y)2 + (M * dist x y)2)
    by (auto intro!: real_sqrt_le_mono add_mono power_mono 1 lipschitz_onD f)
  g)
  also have ... ≤ sqrt (L2 + M2) * dist x y
    by (auto simp: power_mult_distrib ring_distrib[symmetric] real_sqrt_mult)
  finally show ?case .
qed simp

```

```

lemma lipschitz_extend_closure:
  fixes f::('a::metric_space) ⇒ ('b::complete_space)
  assumes C-lipschitz_on U f
  shows ∃ g. C-lipschitz_on (closure U) g ∧ (∀ x ∈ U. g x = f x)
proof -
  obtain g where g: ∧x. x ∈ U ⇒ g x = f x uniformly_continuous_on (closure U) g
  using uniformly_continuous_on_extension_on_closure[OF lipschitz_on_uniformly_continuous[OF assms]] by metis
  have C-lipschitz_on (closure U) g
    apply (rule lipschitz_on_closure, rule lipschitz_on_transform[OF assms])
    using g uniformly_continuous_imp_continuous[OF g(2)] by auto
  then show ?thesis using g(1) by auto
qed

```

```

lemma (in bounded_linear) lipschitz_boundE:
  obtains B where B-lipschitz_on A f
proof -
  from nonneg_bounded
  obtain B where B: B ≥ 0 ∧x. norm (f x) ≤ B * norm x
    by (auto simp: ac_simps)
  have B-lipschitz_on A f
    by (auto intro!: lipschitz_onI B simp: dist_norm diff[symmetric])
  thus ?thesis ..
qed

```

6.48.1 Local Lipschitz continuity

Given a function defined on a real interval, it is Lipschitz-continuous if and only if it is locally so, as proved in the following lemmas. It is useful especially for piecewise-defined functions: if each piece is Lipschitz, then so is

the whole function. The same goes for functions defined on geodesic spaces, or more generally on geodesic subsets in a metric space (for instance convex subsets in a real vector space), and this follows readily from the real case, but we will not prove it explicitly.

We give several variations around this statement. This is essentially a connectedness argument.

```

lemma locally_lipschitz_imp_lipschitz_aux:
  fixes  $f::\text{real} \Rightarrow ('a::\text{metric\_space})$ 
  assumes  $a \leq b$ 
     $\text{continuous\_on } \{a..b\} f$ 
     $\bigwedge x. x \in \{a..<b\} \implies \exists y \in \{x<..b\}. \text{dist } (f y) (f x) \leq M * (y-x)$ 
  shows  $\text{dist } (f b) (f a) \leq M * (b-a)$ 
proof -
  define  $A$  where  $A = \{x \in \{a..b\}. \text{dist } (f x) (f a) \leq M * (x-a)\}$ 
  have  $*$ :  $A = (\lambda x. M * (x-a) - \text{dist } (f x) (f a))^{-1}\{0..\} \cap \{a..b\}$ 
    unfolding  $A\_def$  by auto
  have  $a \in A$  unfolding  $A\_def$  using  $\langle a \leq b \rangle$  by auto
  then have  $A \neq \{\}$  by auto
  moreover have  $\text{bdd\_above } A$  unfolding  $A\_def$  by auto
  moreover have  $\text{closed } A$  unfolding  $*$  by (rule closed_vimage_Int, auto intro!:
continuous_intros assms)
  ultimately have  $\text{Sup } A \in A$  by (rule closed_contains_Sup)
  have  $\text{Sup } A = b$ 
  proof (rule ccontr)
    assume  $\text{Sup } A \neq b$ 
    define  $x$  where  $x = \text{Sup } A$ 
    have  $I$ :  $\text{dist } (f x) (f a) \leq M * (x-a)$  using  $\langle \text{Sup } A \in A \rangle$   $x\_def$   $A\_def$  by auto
    have  $x \in \{a..<b\}$  unfolding  $x\_def$  using  $\langle \text{Sup } A \in A \rangle$   $\langle \text{Sup } A \neq b \rangle$   $A\_def$ 
by auto
    then obtain  $y$  where  $J$ :  $y \in \{x<..b\}$   $\text{dist } (f y) (f x) \leq M * (y-x)$  using
assms(3) by blast
    have  $\text{dist } (f y) (f a) \leq \text{dist } (f y) (f x) + \text{dist } (f x) (f a)$  by (rule dist_triangle)
    also have  $\dots \leq M * (y-x) + M * (x-a)$  using  $I J(2)$  by auto
    finally have  $\text{dist } (f y) (f a) \leq M * (y-a)$  by (auto simp add: algebra_simps)
    then have  $y \in A$  unfolding  $A\_def$  using  $\langle y \in \{x<..b\} \rangle$   $\langle x \in \{a..<b\} \rangle$  by
auto
    then have  $y \leq \text{Sup } A$  by (rule cSup_upper, auto simp: A_def)
    then show False using  $\langle y \in \{x<..b\} \rangle$   $x\_def$  by auto
  qed
  then show ?thesis using  $\langle \text{Sup } A \in A \rangle$   $A\_def$  by auto
qed

lemma locally_lipschitz_imp_lipschitz:
  fixes  $f::\text{real} \Rightarrow ('a::\text{metric\_space})$ 
  assumes  $\text{continuous\_on } \{a..b\} f$ 
     $\bigwedge x y. x \in \{a..<b\} \implies y > x \implies \exists z \in \{x<..y\}. \text{dist } (f z) (f x) \leq M * (z-x)$ 
   $M \geq 0$ 

```

```

shows lipschitz_on M {a..b} f
proof (rule lipschitz_onI[OF _ ‹M ≥ 0›])
  have *: dist (f t) (f s) ≤ M * (t-s) if s ≤ t s ∈ {a..b} t ∈ {a..b} for s t
  proof (rule locally_lipschitz_imp_lipschitz_aux, simp add: ‹s ≤ t›)
    show continuous_on {s..t} f using continuous_on_subset[OF assms(1)] that
  by auto
  fix x assume x ∈ {s..<t}
  then have x ∈ {a..<b} using that by auto
  show ∃ z ∈ {x<..t}. dist (f z) (f x) ≤ M * (z - x)
    using assms(2)[OF ‹x ∈ {a..<b}›, of t] ‹x ∈ {s..<t}› by auto
qed
fix x y assume x ∈ {a..b} y ∈ {a..b}
consider x ≤ y | y ≤ x by linarith
then show dist (f x) (f y) ≤ M * dist x y
  apply (cases)
  using *[OF _ ‹x ∈ {a..b}› ‹y ∈ {a..b}›] *[OF _ ‹y ∈ {a..b}› ‹x ∈ {a..b}›]
  by (auto simp add: dist_commute dist_real_def)
qed

```

We deduce that if a function is Lipschitz on finitely many closed sets on the real line, then it is Lipschitz on any interval contained in their union. The difficulty in the proof is to show that any point z in this interval (except the maximum) has a point arbitrarily close to it on its right which is contained in a common initial closed set. Otherwise, we show that there is a small interval (z, T) which does not intersect any of the initial closed sets, a contradiction.

```

proposition lipschitz_on_closed_Union:
  assumes ‹∧ i. i ∈ I ⇒ lipschitz_on M (U i) f›
    ‹∧ i. i ∈ I ⇒ closed (U i)›
    ‹finite I›
    ‹M ≥ 0›
    ‹{u..(v::real)} ⊆ (⋃ i ∈ I. U i)›
  shows lipschitz_on M {u..v} f
proof (rule locally_lipschitz_imp_lipschitz[OF _ _ ‹M ≥ 0›])
  have *: continuous_on (U i) f if i ∈ I for i
    by (rule lipschitz_on_continuous_on[OF assms(1)][OF ‹i ∈ I›])
  have continuous_on (⋃ i ∈ I. U i) f
    apply (rule continuous_on_closed_Union) using ‹finite I› * assms(2) by auto
  then show continuous_on {u..v} f
    using ‹{u..(v::real)} ⊆ (⋃ i ∈ I. U i)› continuous_on_subset by auto

  fix z Z assume z: z ∈ {u..<v} z < Z
  then have u ≤ v by auto
  define T where T = min Z v
  then have T: T > z T ≤ v T ≥ u T ≤ Z using z by auto
  define A where A = (⋃ i ∈ I ∩ {i. U i ∩ {z<..T} ≠ {}}. U i ∩ {z..T})
  have a: closed A
    unfolding A_def apply (rule closed_UN) using ‹finite I› ‹∧ i. i ∈ I ⇒

```

```

closed (U i) by auto
have b: bdd_below A unfolding A_def using ⟨finite I⟩ by auto
have ∃ i ∈ I. T ∈ U i using ⟨{u..v} ⊆ (⋃ i ∈ I. U i)⟩ T by auto
then have c: T ∈ A unfolding A_def using T by (auto, fastforce)
have Inf A ≥ z
  apply (rule cInf_greatest, auto) using c unfolding A_def by auto
moreover have Inf A ≤ z
proof (rule ccontr)
  assume ¬(Inf A ≤ z)
  then obtain w where w: w > z w < Inf A by (meson dense not_le_imp_less)
  have Inf A ≤ T using a b c by (simp add: cInf_lower)
  then have w ≤ T using w by auto
  then have w ∈ {u..v} using w ⟨z ∈ {u..<v}⟩ T by auto
  then obtain j where j: j ∈ I w ∈ U j using ⟨{u..v} ⊆ (⋃ i ∈ I. U i)⟩ by
fastforce
  then have w ∈ U j ∩ {z..T} U j ∩ {z<..T} ≠ {} using j T w ⟨w ≤ T⟩ by
auto
  then have w ∈ A unfolding A_def using ⟨j ∈ I⟩ by auto
  then have Inf A ≤ w using a b by (simp add: cInf_lower)
  then show False using w by auto
qed
ultimately have Inf A = z by simp
moreover have Inf A ∈ A
  apply (rule closed_contains_Inf) using a b c by auto
ultimately have z ∈ A by simp
then obtain i where i: i ∈ I U i ∩ {z<..T} ≠ {} z ∈ U i unfolding A_def
by auto
then obtain t where t ∈ U i ∩ {z<..T} by blast
then have dist (f t) (f z) ≤ M * (t - z)
  using lipschitz_onD(1)[OF assms(1)[of i], of t z] i dist_real_def by auto
then show ∃ t ∈ {z<..Z}. dist (f t) (f z) ≤ M * (t - z) using ⟨T ≤ Z⟩ ⟨t ∈ U i
∩ {z<..T}⟩ by auto
qed

```

6.48.2 Local Lipschitz continuity (uniform for a family of functions)

definition *local_lipschitz::*

'a::metric_space set ⇒ 'b::metric_space set ⇒ ('a ⇒ 'b ⇒ 'c::metric_space) ⇒ bool

where

local_lipschitz T X f ≡ ∀ x ∈ X. ∀ t ∈ T.

∃ u > 0. ∃ L. ∀ t ∈ cball t u ∩ T. L-lipschitz_on (cball x u ∩ X) (f t)

lemma *local_lipschitzI:*

assumes $\bigwedge t x. t \in T \implies x \in X \implies \exists u > 0. \exists L. \forall t \in \text{cball } t \ u \cap T. L\text{-lipschitz\_on } (\text{cball } x \ u \cap X) (f \ t)$

shows *local_lipschitz T X f*

using *assms*

unfolding *local_lipschitz_def*
by *auto*

lemma *local_lipschitzE*:
assumes *local_lipschitz*: *local_lipschitz* *T* *X* *f*
assumes $t \in T$ $x \in X$
obtains u L **where** $u > 0 \wedge s. s \in \text{cball } t \ u \cap T \implies L\text{-lipschitz\_on } (\text{cball } x \ u \cap X) (f \ s)$
using *assms local_lipschitz_def*
by *metis*

lemma *local_lipschitz_continuous_on*:
assumes *local_lipschitz*: *local_lipschitz* *T* *X* *f*
assumes $t \in T$
shows *continuous_on* *X* (*f* *t*)
unfolding *continuous_on_def*
proof *safe*
fix x **assume** $x \in X$
from *local_lipschitzE*[*OF* *local_lipschitz* $\langle t \in T \rangle \langle x \in X \rangle$] **obtain** u L
where $0 < u$
and L : $\bigwedge s. s \in \text{cball } t \ u \cap T \implies L\text{-lipschitz\_on } (\text{cball } x \ u \cap X) (f \ s)$
by *metis*
have $x \in \text{ball } x \ u$ **using** $\langle 0 < u \rangle$ **by** *simp*
from *lipschitz_on_continuous_on*[*OF* L]
have *tendsto*: $(f \ t \longrightarrow f \ t \ x)$ (*at* x *within* $\text{cball } x \ u \cap X$)
using $\langle 0 < u \rangle \langle x \in X \rangle \langle t \in T \rangle$
by (*auto simp: continuous_on_def*)
moreover **have** $\forall_F \ x a \text{ in } \text{at } x. (x a \in \text{cball } x \ u \cap X) = (x a \in X)$
using *eventually_at_ball*[*OF* $\langle 0 < u \rangle$, *of* x *UNIV*]
by *eventually_elim auto*
ultimately show $(f \ t \longrightarrow f \ t \ x)$ (*at* x *within* X)
by (*rule Lim_transform_within_set*)
qed

lemma
local_lipschitz_compose1:
assumes *ll*: *local_lipschitz* ($g \text{ ' } T$) *X* ($\lambda t. f \ t$)
assumes *g*: *continuous_on* *T* *g*
shows *local_lipschitz* *T* *X* ($\lambda t. f \ (g \ t)$)
proof (*rule local_lipschitzI*)
fix $t \ x$
assume $t \in T$ $x \in X$
then have $g \ t \in g \text{ ' } T$ **by** *simp*
from *local_lipschitzE*[*OF* *assms*(1) *this* $\langle x \in X \rangle$]
obtain u L **where** $0 < u$ **and** l : $(\bigwedge s. s \in \text{cball } (g \ t) \ u \cap g \text{ ' } T \implies L\text{-lipschitz\_on } (\text{cball } x \ u \cap X) (f \ s))$
by *auto*
from *g*[*unfolded continuous_on_eq_continuous_within*, *rule_format*, *OF* $\langle t \in T \rangle$,

```

  unfolded continuous_within_eps_delta, rule_format, OF <0 < u>]
obtain d where d: d>0  $\wedge x'. x' \in T \implies \text{dist } x' t < d \implies \text{dist } (g x') (g t) < u$ 
  by (auto)
show  $\exists u>0. \exists L. \forall t \in \text{cball } t u \cap T. L\text{-lipschitz\_on } (\text{cball } x u \cap X) (f (g t))$ 
  using d <0 < u>
  by (fastforce intro: exI[where x=(min d u)/2] exI[where x=L]
    intro!: less_imp_le[OF d(2)] lipschitz_on_subset[OF l] simp: dist_commute)
qed

```

context

```

  fixes T::'a::metric_space set and X f
  assumes local_lipschitz: local_lipschitz T X f
begin

```

lemma continuous_on_TimesI:

```

  assumes y:  $\bigwedge x. x \in X \implies \text{continuous\_on } T (\lambda t. f t x)$ 
  shows continuous_on (T  $\times$  X) ( $\lambda(t, x). f t x$ )
  unfolding continuous_on_iff
proof (safe, simp)
  fix a b and e::real
  assume H: a  $\in$  T b  $\in$  X 0 < e
  hence 0 < e/2 by simp
  from y[unfolded continuous_on_iff, OF <b  $\in$  X>, rule_format, OF <a  $\in$  T> <0 < e/2>]
  obtain d where d: d > 0  $\wedge t. t \in T \implies \text{dist } t a < d \implies \text{dist } (f t b) (f a b) < e/2$ 
  by auto

```

from <a : T> <b $\in$ X>

obtain u L **where** u: 0 < u

and L: $\bigwedge t. t \in \text{cball } a u \cap T \implies L\text{-lipschitz\_on } (\text{cball } b u \cap X) (f t)$

by (erule local_lipschitzE[OF local_lipschitz])

have a $\in$ cball a u $\cap$ T **by** (auto simp: <0 < u> <a $\in$ T> less_imp_le)

from lipschitz_on_nonneg[OF L[OF <a $\in$ cball __ $\cap$ __>]] **have** 0 $\leq$ L .

let ?d = Min {d, u, (e/2/(L + 1))}

show $\exists d>0. \forall x \in T. \forall y \in X. \text{dist } (x, y) (a, b) < d \longrightarrow \text{dist } (f x y) (f a b) < e$

proof (rule exI[**where** x = ?d], safe)

show 0 < ?d

using <0 $\leq$ L> <0 < u> <0 < e> <0 < d>

by (auto intro!: divide_pos_pos)

fix x y

assume x $\in$ T y $\in$ X

assume dist_less: dist (x, y) (a, b) < ?d

have dist y b $\leq$ dist (x, y) (a, b)

using dist_snd_le[of (x, y) (a, b)]

by auto

also

```

note dist_less
also
{
  note calculation
  also have ?d ≤ u by simp
  finally have dist y b < u .
}
have ?d ≤ e/2/(L + 1) by simp
also have (L + 1) * ... ≤ e / 2
  using ⟨0 < e⟩ ⟨L ≥ 0⟩
  by (auto simp: field_split_simps)
finally have le1: (L + 1) * dist y b < e / 2 using ⟨L ≥ 0⟩ by simp

have dist x a ≤ dist (x, y) (a, b)
  using distfst_le[of (x, y) (a, b)]
  by auto
also note dist_less
finally have dist x a < ?d .
also have ?d ≤ d by simp
finally have dist x a < d .
note ⟨dist x a < ?d⟩
also have ?d ≤ u by simp
finally have dist x a < u .
then have x ∈ cball a u ∩ T
  using ⟨x ∈ T⟩
  by (auto simp: dist_commute)
have dist (f x y) (f a b) ≤ dist (f x y) (f x b) + dist (f x b) (f a b)
  by (rule dist_triangle)
also have (L + 1)-lipschitz_on (cball b u ∩ X) (f x)
  using L[OF ⟨x ∈ cball a u ∩ T⟩]
  by (rule lipschitz_on_le) simp
then have dist (f x y) (f x b) ≤ (L + 1) * dist y b
  apply (rule lipschitz_onD)
  subgoal
    using ⟨y ∈ X⟩ ⟨dist y b < u⟩
    by (simp add: dist_commute)
  subgoal
    using ⟨0 < u⟩ ⟨b ∈ X⟩
    by (simp add: )
  done
also have (L + 1) * dist y b ≤ e / 2
  using le1 ⟨0 ≤ L⟩ by simp
also have dist (f x b) (f a b) < e / 2
  by (rule d; fact)
also have e / 2 + e / 2 = e by simp
finally show dist (f x y) (f a b) < e by simp
qed
qed

```

```

lemma local_lipschitz_compact_implies_lipschitz:
  assumes compact X compact T
  assumes cont:  $\bigwedge x. x \in X \implies \text{continuous\_on } T (\lambda t. f\ t\ x)$ 
  obtains L where  $\bigwedge t. t \in T \implies L\text{-lipschitz\_on } X (f\ t)$ 
proof -
  {
    assume *:  $\bigwedge n::\text{nat}. \neg(\forall t \in T. n\text{-lipschitz\_on } X (f\ t))$ 
    {
      fix n::nat
      from *[of n] have  $\exists x\ y\ t. t \in T \wedge x \in X \wedge y \in X \wedge \text{dist } (f\ t\ y) (f\ t\ x) > n$ 
      * dist y x
      by (force simp: lipschitz_on_def)
    } then obtain t and x y::nat  $\Rightarrow$  'b where xy:  $\bigwedge n. x\ n \in X \wedge n. y\ n \in X$ 
      and t:  $\bigwedge n. t\ n \in T$ 
      and d:  $\bigwedge n. \text{dist } (f\ (t\ n) (y\ n)) (f\ (t\ n) (x\ n)) > n * \text{dist } (y\ n) (x\ n)$ 
      by metis
      from xy assms obtain lx rx where lx':  $lx \in X \text{ strict\_mono } (rx :: \text{nat} \Rightarrow \text{nat})$ 
      (x o rx)  $\longrightarrow$  lx
      by (metis compact_def)
      with xy have  $\bigwedge n. (y\ o\ rx)\ n \in X$  by auto
      with assms obtain ly ry where ly':  $ly \in X \text{ strict\_mono } (ry :: \text{nat} \Rightarrow \text{nat})$  ((y
      o rx) o ry)  $\longrightarrow$  ly
      by (metis compact_def)
      with t have  $\bigwedge n. ((t\ o\ rx)\ o\ ry)\ n \in T$  by simp
      with assms obtain lt rt where lt':  $lt \in T \text{ strict\_mono } (rt :: \text{nat} \Rightarrow \text{nat})$  (((t
      o rx) o ry) o rt)  $\longrightarrow$  lt
      by (metis compact_def)
      from lx' ly'
      have lx:  $(x\ o\ (rx\ o\ ry\ o\ rt)) \longrightarrow lx$  (is ?x  $\longrightarrow$  _)
      and ly:  $(y\ o\ (rx\ o\ ry\ o\ rt)) \longrightarrow ly$  (is ?y  $\longrightarrow$  _)
      and lt:  $(t\ o\ (rx\ o\ ry\ o\ rt)) \longrightarrow lt$  (is ?t  $\longrightarrow$  _)
      subgoal by (simp add: LIMSEQ_subseq_LIMSEQ o_assoc lt'(2))
      subgoal by (simp add: LIMSEQ_subseq_LIMSEQ ly'(3) o_assoc lt'(2))
      subgoal by (simp add: o_assoc lt'(3))
      done
      hence  $(\lambda n. \text{dist } (?y\ n) (?x\ n)) \longrightarrow \text{dist } ly\ lx$ 
      by (metis tendsto_dist)
      moreover
      let ?S =  $(\lambda(t, x). f\ t\ x) \text{ ` } (T \times X)$ 
      have eventually  $(\lambda n::\text{nat}. n > 0)$  sequentially
      by (metis eventually_at_top_dense)
      hence eventually  $(\lambda n. \text{norm } (\text{dist } (?y\ n) (?x\ n)) \leq \text{norm } (|\text{diameter } ?S| / n))$ 
      * 1) sequentially
      proof eventually_elim
      case (elim n)
      have  $0 < rx\ (ry\ (rt\ n))$  using <0 < n>
      by (metis dual_order.strict_trans1 lt'(2) lx'(2) ly'(2) seq_suble)
      have compact: compact ?S
      by (auto intro!: compact_continuous_image continuous_on_subset[OF

```

```

continuous_on TimesI]
  compact_Times ⟨compact X⟩ ⟨compact T⟩ cont)
  have norm (dist (?y n) (?x n)) = dist (?y n) (?x n) by simp
  also
  from this elim d[of rx (ry (rt n))]
  have ... < dist (f (?t n) (?y n)) (f (?t n) (?x n)) / rx (ry (rt (n)))
    using lx'(2) ly'(2) lt'(2) ⟨0 < rx _⟩
    by (auto simp add: field_split_simps strict_mono_def)
  also have ... ≤ diameter ?S / n
  proof (rule frac_le)
    show diameter ?S ≥ 0
      using compact compact_imp_bounded diameter_ge_0 by blast
    show dist (f (?t n) (?y n)) (f (?t n) (?x n)) ≤ diameter ((λ(t,x). f t x) ‘
(T × X))
      by (metis (no_types) compact compact_imp_bounded diameter_bounded_bound
image_eqI mem_Sigma_iff o_apply split_conv t xy(1) xy(2))
    show real n ≤ real (rx (ry (rt n)))
      by (meson le_trans lt'(2) lx'(2) ly'(2) of_nat_mono strict_mono_imp_increasing)
    qed (use ⟨n > 0⟩ in auto)
    also have ... ≤ abs (diameter ?S) / n
      by (auto intro!: divide_right_mono)
    finally show ?case by simp
  qed
  with _ have (λn. dist (?y n) (?x n)) ⟶ 0
    by (rule tendsto_0_le)
    (metis tendsto_divide_0[OF tendsto_const] filterlim_at_top_imp_at_infinity
filterlim_real_sequentially)
  ultimately have lx = ly
    using LIMSEQ_unique by fastforce
  with assms lx' have lx ∈ X by auto
  from ⟨lt ∈ T⟩ this obtain u L where L: u > 0 ∧ t. t ∈ cball lt u ∩ T ⟹
L-lipschitz_on (cball lx u ∩ X) (f t)
    by (erule local_lipschitzE[OF local_lipschitz])
  hence L ≥ 0 by (force intro!: lipschitz_on_nonneg ⟨lt ∈ T⟩)

  from L lt ly lx ⟨lx = ly⟩
  have
    eventually (λn. ?t n ∈ ball lt u) sequentially
    eventually (λn. ?y n ∈ ball lx u) sequentially
    eventually (λn. ?x n ∈ ball lx u) sequentially
    by (auto simp: dist_commute Lim)
  moreover have eventually (λn. n > L) sequentially
    by (metis filterlim_at_top_dense filterlim_real_sequentially)
  ultimately
  have eventually (λ_. False) sequentially
  proof eventually_elim
    case (elim n)
    hence dist (f (?t n) (?y n)) (f (?t n) (?x n)) ≤ L * dist (?y n) (?x n)
      using assms xy t

```



```

    unfolding dist_norm[symmetric]
    by (intro lipschitz_onD[OF L(2)]) (auto)
  also have ... ≤ n * dist (?y n) (?x n)
    using elim by (intro mult_right_mono) auto
  also have ... ≤ rx (ry (rt n)) * dist (?y n) (?x n)
    by (intro mult_right_mono[OF _ zero_le_dist])
      (meson lt'(2) lx'(2) ly'(2) of_nat_le_iff order_trans seq_suble)
  also have ... < dist (f (?t n) (?y n)) (f (?t n) (?x n))
    by (auto intro!: d)
  finally show ?case by simp
qed
hence False
  by simp
} then obtain L where  $\bigwedge t. t \in T \implies L\text{-lipschitz\_on } X (f t)$ 
  by metis
thus ?thesis ..
qed

lemma local_lipschitz_subset:
  assumes  $S \subseteq T \ Y \subseteq X$ 
  shows local_lipschitz S Y f
proof (rule local_lipschitzI)
  fix t x assume  $t \in S \ x \in Y$ 
  then have  $t \in T \ x \in X$  using assms by auto
  from local_lipschitzE[OF local_lipschitz, OF this]
  obtain u L where  $u: 0 < u$  and  $L: \bigwedge s. s \in cball\ t\ u \cap T \implies L\text{-lipschitz\_on}$ 
    ( $cball\ x\ u \cap X$ ) (f s)
  by blast
  show  $\exists u > 0. \exists L. \forall t \in cball\ t\ u \cap S. L\text{-lipschitz\_on } (cball\ x\ u \cap Y) (f t)$ 
    using assms
    by (auto intro: exI[where x=u] exI[where x=L]
      intro!: u_lipschitz_on_subset[OF _ Int_mono[OF order_refl  $\langle Y \subseteq X \rangle$ ]] L)
qed

end

lemma local_lipschitz_minus:
  fixes  $f::'a::metric\_space \Rightarrow 'b::metric\_space \Rightarrow 'c::real\_normed\_vector$ 
  shows local_lipschitz T X ( $\lambda t x. - f\ t\ x$ ) = local_lipschitz T X f
  by (auto simp: local_lipschitz_def lipschitz_on_minus)

lemma local_lipschitz_PairI:
  assumes  $f: local\_lipschitz\ A\ B (\lambda a\ b. f\ a\ b)$ 
  assumes  $g: local\_lipschitz\ A\ B (\lambda a\ b. g\ a\ b)$ 
  shows local_lipschitz A B ( $\lambda a\ b. (f\ a\ b, g\ a\ b)$ )
proof (rule local_lipschitzI)
  fix t x assume  $t \in A \ x \in B$ 
  from local_lipschitzE[OF f this] local_lipschitzE[OF g this]
  obtain u L v M where  $0 < u (\bigwedge s. s \in cball\ t\ u \cap A \implies L\text{-lipschitz\_on } (cball$ 

```

```

x u ∩ B) (f s))
  0 < v (∧ s. s ∈ cball t v ∩ A ⇒ M-lipschitz_on (cball x v ∩ B) (g s))
  by metis
  then show ∃ u > 0. ∃ L. ∀ t ∈ cball t u ∩ A. L-lipschitz_on (cball x u ∩ B) (λ b.
(f t b, g t b))
  by (intro exI[where x=min u v])
    (force intro: lipschitz_on_subset intro!: lipschitz_on_Pair)
qed

```

```

lemma local_lipschitz_constI: local_lipschitz S T (λ t x. f t)
  by (auto simp: intro!: local_lipschitzI lipschitz_on_constant intro: exI[where
x=1])

```

```

lemma (in bounded_linear) local_lipschitzI:
  shows local_lipschitz A B (λ_. f)
proof (rule local_lipschitzI, goal_cases)
  case (1 t x)
  from lipschitz_boundE[of (cball x 1 ∩ B)] obtain C where C-lipschitz_on
(cball x 1 ∩ B) f by auto
  then show ?case
  by (auto intro: exI[where x=1])
qed

```

```

proposition c1_implies_local_lipschitz:
  fixes T::real set and X::'a::{banach,heine_borel} set
  and f::real ⇒ 'a ⇒ 'a
  assumes f': ∧ t x. t ∈ T ⇒ x ∈ X ⇒ (f t has_derivative blinfun_apply (f'
(t, x))) (at x)
  assumes cont_f': continuous_on (T × X) f'
  assumes open T
  assumes open X
  shows local_lipschitz T X f
proof (rule local_lipschitzI)
  fix t x
  assume t ∈ T x ∈ X
  from open_contains_cball[THEN iffD1, OF ⟨open X⟩, rule_format, OF ⟨x ∈
X⟩]
  obtain u where u: u > 0 cball x u ⊆ X by auto
  moreover
  from open_contains_cball[THEN iffD1, OF ⟨open T⟩, rule_format, OF ⟨t ∈
T⟩]
  obtain v where v: v > 0 cball t v ⊆ T by auto
  ultimately
  have compact (cball t v × cball x u) cball t v × cball x u ⊆ T × X
  by (auto intro!: compact_Times)
  then have compact (f' ` (cball t v × cball x u))
  by (auto intro!: compact_continuous_image continuous_on_subset[OF cont_f'])
  then obtain B where B: B > 0 ∧ s y. s ∈ cball t v ⇒ y ∈ cball x u ⇒ norm
(f' (s, y)) ≤ B

```

```

    by (auto dest!: compact_imp_bounded simp: bounded_pos)

  have lipschitz: B-lipschitz_on (cball x (min u v) ∩ X) (f s) if s: s ∈ cball t v
for s
proof -
  note s
  also note ⟨cball t v ⊆ T⟩
  finally
  have deriv:  $\bigwedge y. y \in \text{cball } x \ u \implies (f \ s \text{ has\_derivative } \text{blinfun\_apply } (f' \ (s, y)))$ 
(at y within cball x u)
  using ⟨_ ⊆ X⟩
  by (auto intro!: has_derivative_at_withinI[OF f'])
  have norm (f s y - f s z) ≤ B * norm (y - z)
  if y ∈ cball x u z ∈ cball x u
  for y z
  using s that
  by (intro differentiable_bound[OF convex_cball deriv])
  (auto intro!: B simp: norm_blinfun.rep_eq[symmetric])
  then show ?thesis
  using ⟨0 < B⟩
  by (auto intro!: lipschitz_onI simp: dist_norm)
qed
show  $\exists u > 0. \exists L. \forall t \in \text{cball } t \ u \cap T. L\text{-lipschitz\_on } (\text{cball } x \ u \cap X) (f \ t)$ 
by (force intro: exI[where x=min u v] exI[where x=B] intro!: lipschitz simp:
u v)
qed

end
theory
  Multivariate_Analysis
imports
  Ordered_Euclidean_Space
  Determinants
  Cross3
  Lipschitz
  Starlike
begin

Entry point excluding integration and complex analysis.
end

```

6.49 Volume of a Simplex

```

theory Simplex_Content
imports Change_Of_Vars
begin

lemma fact_neq_top_enreal [simp]: fact n ≠ (top :: enreal)
  by (induction n) (auto simp: enreal_mult_eq_top_iff)

```

lemma *ennreal_fact*: $\text{ennreal } (\text{fact } n) = \text{fact } n$
by (*induction* n) (*auto simp: ennreal_mult algebra_simps ennreal_of_nat_eq_real_of_nat*)

context

fixes $S :: 'a \text{ set} \Rightarrow \text{real} \Rightarrow ('a \Rightarrow \text{real}) \text{ set}$

defines $S \equiv (\lambda A \ t. \{x. (\forall i \in A. 0 \leq x \ i) \wedge \text{sum } x \ A \leq t\})$

begin

lemma *emeasure_std_simplex_aux_step*:

assumes $b \notin A$ *finite* A

shows $x(b := y) \in S (\text{insert } b \ A) \ t \longleftrightarrow y \in \{0..t\} \wedge x \in S \ A \ (t - y)$

using *assms sum_nonneg[of A x]* **unfolding** S_def

by (*force simp: sum_delta_notmem algebra_simps*)

lemma *emeasure_std_simplex_aux*:

fixes $t :: \text{real}$

assumes *finite* $(A :: 'a \text{ set})$ $t \geq 0$

shows $\text{emeasure } (Pi_M \ A \ (\lambda_. \text{lborel}))$

$(S \ A \ t \cap \text{space } (Pi_M \ A \ (\lambda_. \text{lborel}))) = t \wedge \text{card } A / \text{fact } (\text{card } A)$

using *assms(1,2)*

proof (*induction arbitrary: t rule: finite_induct*)

case (*empty* t)

thus *?case* **by** (*simp add: PiM_empty S_def*)

next

case (*insert* $b \ A \ t$)

define n **where** $n = \text{Suc } (\text{card } A)$

have $n\_pos: n > 0$ **by** (*simp add: n_def*)

let $?M = \lambda A. (Pi_M \ A \ (\lambda_. \text{lborel}))$

{

fix $A :: 'a \text{ set}$ **and** $t :: \text{real}$ **assume** *finite* A

have $S \ A \ t \cap \text{space } (Pi_M \ A \ (\lambda_. \text{lborel})) =$

$Pi_E \ A \ (\lambda_. \{0..t\}) \cap (\lambda x. \text{sum } x \ A) - \{..t\} \cap \text{space } (Pi_M \ A \ (\lambda_. \text{lborel}))$

by (*auto simp: S_def space_PiM*)

also have $\dots \in \text{sets } (Pi_M \ A \ (\lambda_. \text{lborel}))$

using $\langle \text{finite } A \rangle$ **by** *measurable*

finally have $S \ A \ t \cap \text{space } (Pi_M \ A \ (\lambda_. \text{lborel})) \in \text{sets } (Pi_M \ A \ (\lambda_. \text{lborel}))$.

} note *meas [measurable] = this*

interpret *product_sigma_finite* $\lambda_. \text{lborel}$

by *standard*

have $\text{emeasure } (?M \ (\text{insert } b \ A)) \ (S \ (\text{insert } b \ A) \ t \cap \text{space } (?M \ (\text{insert } b \ A)))$

=

$\text{nn\_integral } (?M \ (\text{insert } b \ A))$

$(\lambda x. \text{indicator } (S \ (\text{insert } b \ A) \ t \cap \text{space } (?M \ (\text{insert } b \ A))) \ x)$

using *insert.hyps* **by** (*subst nn_integral_indicator*) *auto*

also have $\dots = (\int^+ y. \int^+ x. \text{indicator } (S \ (\text{insert } b \ A) \ t \cap \text{space } (?M \ (\text{insert } b \ A)))$

$(x(b := y)) \ \partial ?M \ A \ \partial \text{lborel})$

```

using insert.premis insert.hyps by (intro product nn_integral_insert_rev) auto
also have ... =  $(\int^+ y. \int^+ x. \text{indicator } \{0..t\} y * \text{indicator } (S A (t - y) \cap$ 
space ( $?M A$ ))  $x$ 
 $\partial ?M A \partial \text{lborel}$ )
using insert.hyps insert.premis emeasure_std_simplex_aux_step[of b A]
by (intro nn_integral_cong)
(auto simp: fun_eq_iff indicator_def space_PiM PiE_def extensional_def)
also have ... =  $(\int^+ y. \text{indicator } \{0..t\} y * (\int^+ x. \text{indicator } (S A (t - y) \cap$ 
space ( $?M A$ ))  $x$ 
 $\partial ?M A \partial \text{lborel}$ ) using  $\langle \text{finite } A \rangle$ 
by (subst nn_integral_cmult) auto
also have ... =  $(\int^+ y. \text{indicator } \{0..t\} y * \text{emeasure } (?M A) (S A (t - y) \cap$ 
space ( $?M A$ ))  $\partial \text{lborel}$ )
using  $\langle \text{finite } A \rangle$  by (subst nn_integral_indicator) auto
also have ... =  $(\int^+ y. \text{indicator } \{0..t\} y * (t - y) ^ \text{card } A / \text{ennreal } (\text{fact}$ 
(card A))  $\partial \text{lborel}$ )
using insert.IH by (intro nn_integral_cong) (auto simp: indicator_def di-
vide_ennreal)
also have ... =  $(\int^+ y. \text{indicator } \{0..t\} y * (t - y) ^ \text{card } A \partial \text{lborel}) / \text{ennreal}$ 
(fact (card A))
using  $\langle \text{finite } A \rangle$  by (subst nn_integral_divide) auto
also have  $(\int^+ y \in \{0..t\}. \text{ennreal } ((t - y) ^ (n - 1)) \partial \text{lborel}) =$ 
 $(\int^+ y \in \{0..t\}. \text{ennreal } ((t - y) ^ (n - 1)) \partial \text{lborel})$ 
by (intro nn_integral_cong) (auto simp: indicator_def n_def)
also have  $((\lambda x. - ((t - x) ^ n / n)) \text{has\_real\_derivative } (t - x) ^ (n - 1))$ 
(at x)
if  $x \in \{0..t\}$  for x by (rule derivative_eq_intros refl | simp add: n_pos)+
hence  $(\int^+ y \in \{0..t\}. \text{ennreal } ((t - y) ^ (n - 1)) \partial \text{lborel}) =$ 
 $\text{ennreal } (-((t - t) ^ n / n) - (-((t - 0) ^ n / n)))$ 
using insert.premis insert.hyps by (intro nn_integral_FTC_Icc) auto
also have ... =  $\text{ennreal } (t ^ n / n)$  using n_pos by (simp add: zero_power)
also have ... /  $\text{ennreal } (\text{fact } (\text{card } A)) = \text{ennreal } (t ^ n / n / \text{fact } (\text{card } A))$ 
using n_pos  $\langle t \geq 0 \rangle$  by (subst divide_ennreal) auto
also have  $t ^ n / n / \text{fact } (\text{card } A) = t ^ n / \text{fact } n$ 
by (simp add: n_def)
also have  $n = \text{card } (\text{insert } b A)$ 
using insert.hyps by (subst card.insert_remove) (auto simp: n_def)
finally show ?case .
qed

end

lemma emeasure_std_simplex:
emeasure lborel (convex hull (insert 0 Basis :: 'a :: euclidean_space set)) =
ennreal (1 / fact DIM('a))
proof -
have emeasure lborel  $\{x :: 'a. (\forall i \in \text{Basis}. 0 \leq x \cdot i) \wedge \text{sum } ((\cdot) x) \text{Basis} \leq 1\} =$ 
emeasure (distr (PiM Basis ( $\lambda b. \text{lborel}$ )) borel ( $\lambda f. \sum_{b \in \text{Basis}. f b *_{\mathbb{R}}$ 
b))

```

```

      {x::'a. (∀ i∈Basis. 0 ≤ x · i) ∧ sum ((·) x) Basis ≤ 1}
    by (subst lborel_eq) simp
  also have ... = emeasure (Pi_M Basis (λb. lborel))
    ({y::'a ⇒ real. (∀ i∈Basis. 0 ≤ y i) ∧ sum y Basis ≤ 1} ∩
     space (Pi_M Basis (λb. lborel)))
  by (subst emeasure_distr) auto
  also have ... = ennreal (1 / fact DIM('a))
  by (subst emeasure_std_simplex_aux) auto
  finally show ?thesis by (simp only: std_simplex)
qed

```

theorem *content_std_simplex*:

```

  measure lborel (convex hull (insert 0 Basis :: 'a :: euclidean_space set)) =
    1 / fact DIM('a)
  by (simp add: measure_def emeasure_std_simplex)

```

proposition *measure_lebesgue_linear_transformation*:

```

  fixes A :: (real ^ 'n :: {finite, wellorder}) set
  fixes f :: _ ⇒ real ^ 'n :: {finite, wellorder}
  assumes bounded A A ∈ sets lebesgue linear f
  shows measure lebesgue (f ' A) = |det (matrix f)| * measure lebesgue A
proof -
  from assms have [intro]: A ∈ lmeasurable
  by (intro bounded_set_imp_lmeasurable) auto
  hence [intro]: f ' A ∈ lmeasurable
  by (intro lmeasure_integral measurable_linear_image assms)
  have measure lebesgue (f ' A) = integral (f ' A) (λ_. 1)
  by (intro lmeasure_integral measurable_linear_image assms) auto
  also have ... = integral (f ' A) (λ_. 1 :: real ^ 1) $ 0
  by (subst integral_component_eq_cart [symmetric]) (auto intro: integrable_on_const)
  also have ... = |det (matrix f)| * integral A (λx. 1 :: real ^ 1) $ 0
  using assms
  by (subst integral_change_of_variables_linear)
    (auto simp: o_def absolutely_integrable_on_def intro: integrable_on_const)
  also have integral A (λx. 1 :: real ^ 1) $ 0 = integral A (λx. 1)
  by (subst integral_component_eq_cart [symmetric]) (auto intro: integrable_on_const)
  also have ... = measure lebesgue A
  by (intro lmeasure_integral [symmetric]) auto
  finally show ?thesis .
qed

```

theorem *content_simplex*:

```

  fixes X :: (real ^ 'n :: {finite, wellorder}) set and f :: 'n :: _ ⇒ real ^ ('n :: _)
  assumes finite X card X = Suc CARD('n) and x0: x0 ∈ X and bij: bij_betw f
    UNIV (X - {x0})
  defines M ≡ (χ i. χ j. f j $ i - x0 $ i)
  shows content (convex hull X) = |det M| / fact (CARD('n))
proof -

```

```

define  $g$  where  $g = (\lambda x. M * v x)$ 
have [simp]:  $M * v \text{ axis } i \ 1 = f \ i - x0$  for  $i :: 'n$ 
  by (simp add:  $M\_def$   $matrix\_vector\_mult\_basis$   $column\_def$   $vec\_eq\_iff$ )
define  $std$  where  $std = (\text{convex hull insert } 0 \text{ Basis} :: (\text{real} ^ 'n :: \_) \text{ set})$ 
have  $compact$ :  $compact \ std$  unfolding  $std\_def$ 
  by (intro  $finite\_imp\_compact\_convex\_hull$ ) auto

have  $measure \text{ lebesgue } (\text{convex hull } X) = measure \text{ lebesgue } (((+) (-x0)) ' (\text{convex hull } X))$ 
  by (rule  $measure\_translation$  [symmetric])
also have  $((+) (-x0)) ' (\text{convex hull } X) = \text{convex hull } (((+) (-x0)) ' X)$ 
  by (rule  $convex\_hull\_translation$  [symmetric])
also have  $((+) (-x0)) ' X = \text{insert } 0 ((\lambda x. x - x0) ' (X - \{x0\}))$ 
  using  $x0$  by (auto simp:  $image\_iff$ )
finally have  $eq$ :  $measure \text{ lebesgue } (\text{convex hull } X) = measure \text{ lebesgue } (\text{convex hull } \dots)$  .

from  $compact$  have  $measure \text{ lebesgue } (g ' std) = |\det M| * measure \text{ lebesgue } std$ 
  by (subst  $measure\_lebesgue\_linear\_transformation$ )
  (auto intro:  $finite\_imp\_bounded\_convex\_hull$   $dest$ :  $compact\_imp\_closed$ 
 $simp$ :  $g\_def \ std\_def$ )
also have  $measure \text{ lebesgue } std = content \ std$  using  $compact$ 
  by (intro  $measure\_completion$ ) (auto  $dest$ :  $compact\_imp\_closed$ )
also have  $content \ std = 1 / \text{fact } CARD('n)$  unfolding  $std\_def$ 
  by (simp add:  $content\_std\_simplex$ )
also have  $g ' std = \text{convex hull } (g ' \text{insert } 0 \text{ Basis})$  unfolding  $std\_def$ 
  by (rule  $convex\_hull\_linear\_image$ ) (auto simp:  $g\_def$ )
also have  $g ' \text{insert } 0 \text{ Basis} = \text{insert } 0 (g ' \text{Basis})$ 
  by (auto simp:  $g\_def$ )
also have  $g ' \text{Basis} = (\lambda x. x - x0) ' \text{range } f$ 
  by (auto simp:  $g\_def \ Basis\_vec\_def \ image\_iff$ )
also have  $\text{range } f = X - \{x0\}$  using  $bij$ 
  using  $bij\_betw\_imp\_surj\_on$  by blast
also note  $eq$  [symmetric]
finally show  $?thesis$ 
  using  $finite\_imp\_compact\_convex\_hull[OF \langle finite \ X \rangle]$  by (auto  $dest$ :  $compact\_imp\_closed$ )
qed

theorem  $content\_triangle$ :
  fixes  $A \ B \ C :: \text{real} ^ 2$ 
  shows  $content (\text{convex hull } \{A, B, C\}) =$ 
     $| (C \$ 1 - A \$ 1) * (B \$ 2 - A \$ 2) - (B \$ 1 - A \$ 1) * (C \$ 2 - A \$ 2) | / 2$ 
proof -
  define  $M :: \text{real} ^ 2 ^ 2$  where  $M \equiv (\chi \ i. \chi \ j. (\text{if } j = 1 \text{ then } B \text{ else } C) \$ i - A \$ i)$ 
  define  $g$  where  $g = (\lambda x. M * v x)$ 
  define  $std$  where  $std = (\text{convex hull insert } 0 \text{ Basis} :: (\text{real} ^ 2) \text{ set})$ 

```

```

have [simp]:  $M * v \text{ axis } i \ 1 = (\text{if } i = 1 \text{ then } B - A \text{ else } C - A)$  for  $i$ 
  by (auto simp:  $M\_def$   $matrix\_vector\_mult\_basis$   $column\_def$   $vec\_eq\_iff$ )
have compact: compact std unfolding std_def
  by (intro finite_imp_compact_convex_hull) auto

have measure_lebesgue (convex_hull {A, B, C}) =
  measure_lebesgue (((+) (-A)) ' (convex_hull {A, B, C}))
  by (rule measure_translation [symmetric])
also have ((+) (-A)) ' (convex_hull {A, B, C}) = convex_hull (((+) (-A)) '
{A, B, C})
  by (rule convex_hull_translation [symmetric])
also have ((+) (-A)) ' {A, B, C} = {0, B - A, C - A}
  by (auto simp: image_iff)
finally have eq: measure_lebesgue (convex_hull {A, B, C}) =
  measure_lebesgue (convex_hull {0, B - A, C - A}) .

from compact have measure_lebesgue (g ' std) = |det M| * measure_lebesgue std
  by (subst measure_lebesgue_linear_transformation)
  (auto intro: finite_imp_bounded_convex_hull dest: compact_imp_closed
simp: g_def std_def)
also have measure_lebesgue std = content std using compact
  by (intro measure_completion) (auto dest: compact_imp_closed)
also have content std = 1 / 2 unfolding std_def
  by (simp add: content_std_simplex)
also have g ' std = convex_hull (g ' insert 0 Basis) unfolding std_def
  by (rule convex_hull_linear_image) (auto simp: g_def)
also have g ' insert 0 Basis = insert 0 (g ' Basis)
  by (auto simp: g_def)
also have (2 :: 2)  $\neq$  1 by auto
hence  $\neg(\forall y::2. y = 1)$  by blast
hence g ' Basis = {B - A, C - A}
  by (auto simp: g_def Basis_vec_def image_iff)
also note eq [symmetric]
finally show ?thesis
  using finite_imp_compact_convex_hull[of {A, B, C}]
  by (auto dest!: compact_imp_closed simp: det_2 M_def)
qed

```

theorem heron:

```

fixes A B C :: real ^ 2
defines a  $\equiv$  dist B C and b  $\equiv$  dist A C and c  $\equiv$  dist A B
defines s  $\equiv$  (a + b + c) / 2
shows content (convex_hull {A, B, C}) = sqrt (s * (s - a) * (s - b) * (s -
c))
proof -
  have [simp]: (UNIV :: 2 set) = {1, 2}
    using exhaust_2 by auto
  have dist_eq: dist (A :: real ^ 2) B ^ 2 = (A $ 1 - B $ 1) ^ 2 + (A $ 2 - B
$ 2) ^ 2

```



```

for A B by (simp add: dist_vec_def dist_real_def)
have nonneg:  $s * (s - a) * (s - b) * (s - c) \geq 0$ 
using dist_triangle[of A B C] dist_triangle[of A C B] dist_triangle[of B C A]
by (intro mult_nonneg_nonneg) (auto simp: s_def a_def b_def c_def dist_commute)

have 16 * content (convex hull {A, B, C}) ^ 2 =
  4 * ((C $ 1 - A $ 1) * (B $ 2 - A $ 2) - (B $ 1 - A $ 1) * (C $ 2 -
A $ 2)) ^ 2
by (subst content_triangle) (simp add: power_divide)
also have ... = (2 * (dist A B ^ 2 * dist A C ^ 2 + dist A B ^ 2 * dist B C
^ 2 +
  dist A C ^ 2 * dist B C ^ 2) - (dist A B ^ 2) ^ 2 - (dist A C ^ 2) ^ 2 -
(dist B C ^ 2) ^ 2)
unfolding dist_eq unfolding power2_eq_square by algebra
also have ... = (a + b + c) * ((a + b + c) - 2 * a) * ((a + b + c) - 2 * b) *
((a + b + c) - 2 * c)
unfolding power2_eq_square by (simp add: s_def a_def b_def c_def alge-
bra_simps)
also have ... = 16 * s * (s - a) * (s - b) * (s - c)
by (simp add: s_def field_split_simps)
finally have content (convex hull {A, B, C}) ^ 2 = s * (s - a) * (s - b) * (s
- c)
by simp
also have ... = sqrt (s * (s - a) * (s - b) * (s - c)) ^ 2
by (intro real_sqrt_pow2 [symmetric] nonneg)
finally show ?thesis using nonneg
by (subst (asm) power2_eq_iff_nonneg) auto
qed

end

```

6.50 Convergence of Formal Power Series

```

theory FPS_Convergence
imports
  Generalised_Binomial_Theorem
  HOL-Computational_Algebra.Formal_Power_Series
begin

```

In this theory, we will connect formal power series (which are algebraic objects) with analytic functions. This will become more important in complex analysis, and indeed some of the less trivial results will only be proven there.

6.50.1 Balls with extended real radius

The following is a variant of *ball* that also allows an infinite radius.

```

definition eball :: 'a :: metric_space  $\Rightarrow$  ereal  $\Rightarrow$  'a set where
  eball z r = {z'. ereal (dist z z') < r}

```

lemma *in_eball_iff* [*simp*]: $z \in \text{eball } z0 \ r \longleftrightarrow \text{ereal } (\text{dist } z0 \ z) < r$
by (*simp add: eball_def*)

lemma *eball_ereal* [*simp*]: $\text{eball } z \ (\text{ereal } r) = \text{ball } z \ r$
by *auto*

lemma *eball_inf* [*simp*]: $\text{eball } z \ \infty = \text{UNIV}$
by *auto*

lemma *eball_empty* [*simp*]: $r \leq 0 \implies \text{eball } z \ r = \{\}$
proof *safe*
fix *x* **assume** $r \leq 0 \ x \in \text{eball } z \ r$
hence $\text{dist } z \ x < r$ **by** *simp*
also have $\dots \leq \text{ereal } 0$ **using** $\langle r \leq 0 \rangle$ **by** (*simp add: zero_ereal_def*)
finally show $x \in \{\}$ **by** *simp*
qed

lemma *eball_conv_UNION_balls*:
 $\text{eball } z \ r = (\bigcup_{r' \in \{r'. \text{ereal } r' < r\}} \text{ball } z \ r')$
by (*cases r*) (*use dense_gt_ex in force*)**+**

lemma *eball_mono*: $r \leq r' \implies \text{eball } z \ r \leq \text{eball } z \ r'$
by *auto*

lemma *ball_eball_mono*: $\text{ereal } r \leq r' \implies \text{ball } z \ r \leq \text{eball } z \ r'$
using *eball_mono*[*of ereal r r'*] **by** *simp*

lemma *open_eball* [*simp, intro*]: $\text{open } (\text{eball } z \ r)$
by (*cases r*) *auto*

6.50.2 Basic properties of convergent power series

definition *fps_conv_radius* :: $'a :: \{\text{banach}, \text{real\_normed\_div\_algebra}\}$ *fps* $\Rightarrow$ *ereal* **where**
 $\text{fps\_conv\_radius } f = \text{conv\_radius } (\text{fps\_nth } f)$

definition *eval_fps* :: $'a :: \{\text{banach}, \text{real\_normed\_div\_algebra}\}$ *fps* $\Rightarrow 'a \Rightarrow 'a$ **where**
 $\text{eval\_fps } f \ z = (\sum n. \text{fps\_nth } f \ n * z ^ n)$

lemma *norm_summable_fps*:
fixes $f :: 'a :: \{\text{banach}, \text{real\_normed\_div\_algebra}\}$ *fps*
shows $\text{norm } z < \text{fps\_conv\_radius } f \implies \text{summable } (\lambda n. \text{norm } (\text{fps\_nth } f \ n * z ^ n))$
by (*rule abs_summable_in_conv_radius*) (*simp_all add: fps_conv_radius_def*)

lemma *summable_fps*:
fixes $f :: 'a :: \{\text{banach}, \text{real\_normed\_div\_algebra}\}$ *fps*

shows $\text{norm } z < \text{fps\_conv\_radius } f \implies \text{summable } (\lambda n. \text{fps\_nth } f \ n * z^{\wedge} n)$
by (rule *summable_in_conv_radius*) (simp_all add: *fps_conv_radius_def*)

theorem *sums_eval_fps*:

fixes $f :: 'a :: \{\text{banach}, \text{real\_normed\_div\_algebra}\}$ *fps*
assumes $\text{norm } z < \text{fps\_conv\_radius } f$
shows $(\lambda n. \text{fps\_nth } f \ n * z^{\wedge} n) \text{ sums eval\_fps } f \ z$
using *assms* **unfolding** *eval_fps_def fps_conv_radius_def*
by (intro *summable_sums summable_in_conv_radius*) *simp_all*

lemma *continuous_on_eval_fps*:

fixes $f :: 'a :: \{\text{banach}, \text{real\_normed\_div\_algebra}\}$ *fps*
shows *continuous_on* (eball 0 (*fps_conv_radius* *f*)) (*eval_fps* *f*)
proof (subst *continuous_on_eq_continuous_at* [OF *open_eball*], safe)
fix $x :: 'a$ **assume** $x \in \text{eball } 0 \ (\text{fps\_conv\_radius } f)$
define r **where** $r = (\text{if } \text{fps\_conv\_radius } f = \infty \text{ then } \text{norm } x + 1 \text{ else } (\text{norm } x + \text{real\_of\_ereal } (\text{fps\_conv\_radius } f)) / 2)$
have $r: \text{norm } x < r \wedge \text{ereal } r < \text{fps\_conv\_radius } f$
using x **by** (cases *fps_conv_radius* *f*)
(auto simp: *r_def eball_def split: if_splits*)
have *continuous_on* (cball 0 r) $(\lambda x. \sum i. \text{fps\_nth } f \ i * (x - 0)^{\wedge} i)$
by (rule *power_continuous_suminf*) (insert r , auto simp: *fps_conv_radius_def*)
hence *continuous_on* (cball 0 r) (*eval_fps* *f*)
by (simp add: *eval_fps_def*)
thus *isCont* (*eval_fps* *f*) x
by (rule *continuous_on_interior*) (use r in auto)
qed

lemma *continuous_on_eval_fps'* [*continuous_intros*]:

assumes *continuous_on* $A \ g$
assumes $g \ 'A \subseteq \text{eball } 0 \ (\text{fps\_conv\_radius } f)$
shows *continuous_on* $A \ (\lambda x. \text{eval\_fps } f \ (g \ x))$
using *continuous_on_compose2* [OF *continuous_on_eval_fps assms*] .

lemma *has_field_derivative_powser*:

fixes $z :: 'a :: \{\text{banach}, \text{real\_normed\_field}\}$
assumes $\text{ereal } (\text{norm } z) < \text{conv\_radius } f$
shows $((\lambda z. \sum n. f \ n * z^{\wedge} n) \text{ has\_field\_derivative } (\sum n. \text{diffs } f \ n * z^{\wedge} n))$
(at z within A)
proof –
define K **where** $K = (\text{if } \text{conv\_radius } f = \infty \text{ then } \text{norm } z + 1 \text{ else } (\text{norm } z + \text{real\_of\_ereal } (\text{conv\_radius } f)) / 2)$
have $K: \text{norm } z < K \wedge \text{ereal } K < \text{conv\_radius } f$
using *assms* **by** (cases *conv_radius* *f*) (auto simp: *K_def*)
have $0 \leq \text{norm } z$ **by** *simp*
also from K **have** $\dots < K$ **by** *simp*
finally have $K_{\text{pos}}: K > 0$ **by** *simp*

3320

```

have summable ( $\lambda n. f\ n * \text{of\_real } K \wedge n$ )
  using K and K_pos by (intro summable_in_conv_radius) auto
moreover from K and K_pos have norm z < norm (of_real K :: 'a) by auto
ultimately show ?thesis
  by (rule has_field_derivative_at_within [OF termdiffs_strong])
qed

```

```

lemma has_field_derivative_eval_fps:
  fixes z :: 'a :: {banach, real_normed_field}
  assumes norm z < fps_conv_radius f
  shows (eval_fps f has_field_derivative eval_fps (fps_deriv f) z) (at z within A)
proof -
  have (eval_fps f has_field_derivative eval_fps (Abs_fps (diffs (fps_nth f))) z)
    (at z within A)
  using assms unfolding eval_fps_def fps_nth_Abs_fps fps_conv_radius_def
  by (intro has_field_derivative_powser) auto
  also have Abs_fps (diffs (fps_nth f)) = fps_deriv f
  by (simp add: fps_eq_iff diffs_def)
  finally show ?thesis .
qed

```

```

lemma holomorphic_on_eval_fps [holomorphic_intros]:
  fixes z :: 'a :: {banach, real_normed_field}
  assumes A  $\subseteq$  eball 0 (fps_conv_radius f)
  shows eval_fps f holomorphic_on A
proof (rule holomorphic_on_subset [OF __ assms])
  show eval_fps f holomorphic_on eball 0 (fps_conv_radius f)
  proof (subst holomorphic_on_open [OF open_eball], safe, goal_cases)
    case (1 x)
    thus ?case
    by (intro exI [of __ eval_fps (fps_deriv f) x]) (auto intro: has_field_derivative_eval_fps)
  qed
qed

```

```

lemma analytic_on_eval_fps:
  fixes z :: 'a :: {banach, real_normed_field}
  assumes A  $\subseteq$  eball 0 (fps_conv_radius f)
  shows eval_fps f analytic_on A
proof (rule analytic_on_subset [OF __ assms])
  show eval_fps f analytic_on eball 0 (fps_conv_radius f)
  using holomorphic_on_eval_fps [of eball 0 (fps_conv_radius f)]
  by (subst analytic_on_open) auto
qed

```

```

lemma continuous_eval_fps [continuous_intros]:
  fixes z :: 'a :: {real_normed_field, banach}
  assumes norm z < fps_conv_radius F
  shows continuous (at z within A) (eval_fps F)

```

```

proof –
  from ereal_dense2[OF assms] obtain  $K :: \text{real}$  where  $K: \text{norm } z < K \ K < \text{fps\_conv\_radius } F$ 
  by auto
  have  $0 \leq \text{norm } z$  by simp
  also have  $\text{norm } z < K$  by fact
  finally have  $K > 0$  .
  from  $K$  and  $\langle K > 0 \rangle$  have summable  $(\lambda n. \text{fps\_nth } F \ n * \text{of\_real } K ^ n)$ 
  by (intro summable_fps) auto
  from this have isCont (eval_fps F)  $z$  unfolding eval_fps_def
  by (rule isCont_powser) (use  $K$  in auto)
  thus continuous (at  $z$  within  $A$ ) (eval_fps F)
  by (simp add: continuous_at_imp_continuous_within)
qed

```

6.50.3 Lower bounds on radius of convergence

```

lemma fps_conv_radius_deriv:
  fixes  $f :: 'a :: \{\text{banach}, \text{real\_normed\_field}\}$   $\text{fps}$ 
  shows  $\text{fps\_conv\_radius } (\text{fps\_deriv } f) \geq \text{fps\_conv\_radius } f$ 
  unfolding fps_conv_radius_def
proof (rule conv_radius_geI_ex)
  fix  $r :: \text{real}$  assume  $r: r > 0$  ereal  $r < \text{conv\_radius } (\text{fps\_nth } f)$ 
  define  $K$  where  $K = (\text{if } \text{conv\_radius } (\text{fps\_nth } f) = \infty \text{ then } r + 1$ 
     $\text{else } (\text{real\_of\_ereal } (\text{conv\_radius } (\text{fps\_nth } f)) + r) / 2)$ 
  have  $K: r < K \wedge \text{ereal } K < \text{conv\_radius } (\text{fps\_nth } f)$ 
  using  $r$  by (cases conv_radius (fps_nth f)) (auto simp: K_def)
  have summable  $(\lambda n. \text{diffs } (\text{fps\_nth } f) \ n * \text{of\_real } r ^ n)$ 
proof (rule termdiff_converges)
  fix  $x :: 'a$  assume  $\text{norm } x < K$ 
  hence  $\text{ereal } (\text{norm } x) < \text{ereal } K$  by simp
  also have  $\dots < \text{conv\_radius } (\text{fps\_nth } f)$  using  $K$  by simp
  finally show summable  $(\lambda n. \text{fps\_nth } f \ n * x ^ n)$ 
  by (intro summable_in_conv_radius) auto
qed (insert  $K \ r$ , auto)
  also have  $\dots = (\lambda n. \text{fps\_nth } (\text{fps\_deriv } f) \ n * \text{of\_real } r ^ n)$ 
  by (simp add: fps_deriv_def diffs_def)
  finally show  $\exists z::'a. \text{norm } z = r \wedge \text{summable } (\lambda n. \text{fps\_nth } (\text{fps\_deriv } f) \ n * z ^ n)$ 
  using  $r$  by (intro exI[of _ of_real r]) auto
qed

```

```

lemma eval_fps_at_0:  $\text{eval\_fps } f \ 0 = \text{fps\_nth } f \ 0$ 
  by (simp add: eval_fps_def)

```

```

lemma fps_conv_radius_norm [simp]:
   $\text{fps\_conv\_radius } (\text{Abs\_fps } (\lambda n. \text{norm } (\text{fps\_nth } f \ n))) = \text{fps\_conv\_radius } f$ 
  by (simp add: fps_conv_radius_def)

```

```

lemma fps_conv_radius_const [simp]: fps_conv_radius (fps_const c) =  $\infty$ 
proof –
  have fps_conv_radius (fps_const c) = conv_radius ( $\lambda\_ . 0 :: 'a$ )
    unfolding fps_conv_radius_def
    by (intro conv_radius_cong eventually_mono[OF eventually_gt_at_top[of 0]])
  auto
  thus ?thesis by simp
qed

lemma fps_conv_radius_0 [simp]: fps_conv_radius 0 =  $\infty$ 
  by (simp only: fps_const_0_eq_0 [symmetric] fps_conv_radius_const)

lemma fps_conv_radius_1 [simp]: fps_conv_radius 1 =  $\infty$ 
  by (simp only: fps_const_1_eq_1 [symmetric] fps_conv_radius_const)

lemma fps_conv_radius_numeral [simp]: fps_conv_radius (numeral n) =  $\infty$ 
  by (simp add: numeral_fps_const)

lemma fps_conv_radius_fps_X_power [simp]: fps_conv_radius (fps_X ^ n) =
 $\infty$ 
proof –
  have fps_conv_radius (fps_X ^ n :: 'a fps) = conv_radius ( $\lambda\_ . 0 :: 'a$ )
    unfolding fps_conv_radius_def
    by (intro conv_radius_cong eventually_mono[OF eventually_gt_at_top[of n]])

    (auto simp: fps_X_power_iff)
  thus ?thesis by simp
qed

lemma fps_conv_radius_fps_X [simp]: fps_conv_radius fps_X =  $\infty$ 
  using fps_conv_radius_fps_X_power[of 1] by (simp only: power_one_right)

lemma fps_conv_radius_shift [simp]:
  fps_conv_radius (fps_shift n f) = fps_conv_radius f
  by (simp add: fps_conv_radius_def fps_shift_def conv_radius_shift)

lemma fps_conv_radius_cmult_left:
  c  $\neq 0 \implies$  fps_conv_radius (fps_const c * f) = fps_conv_radius f
  unfolding fps_conv_radius_def by (simp add: conv_radius_cmult_left)

lemma fps_conv_radius_cmult_right:
  c  $\neq 0 \implies$  fps_conv_radius (f * fps_const c) = fps_conv_radius f
  unfolding fps_conv_radius_def by (simp add: conv_radius_cmult_right)

lemma fps_conv_radius_uminus [simp]:
  fps_conv_radius (–f) = fps_conv_radius f
  using fps_conv_radius_cmult_left[of –1 f]
  by (simp flip: fps_const_neg)

```

lemma *fps_conv_radius_add*: $\text{fps\_conv\_radius } (f + g) \geq \min (\text{fps\_conv\_radius } f) (\text{fps\_conv\_radius } g)$
unfolding *fps_conv_radius_def* **using** *conv_radius_add_ge* [of *fps_nth f fps_nth g*]
by *simp*

lemma *fps_conv_radius_diff*: $\text{fps\_conv\_radius } (f - g) \geq \min (\text{fps\_conv\_radius } f) (\text{fps\_conv\_radius } g)$
using *fps_conv_radius_add* [of *f -g*] **by** *simp*

lemma *fps_conv_radius_mult*: $\text{fps\_conv\_radius } (f * g) \geq \min (\text{fps\_conv\_radius } f) (\text{fps\_conv\_radius } g)$
using *conv_radius_mult_ge* [of *fps_nth f fps_nth g*]
by (*simp add: fps_mult_nth fps_conv_radius_def atLeast0AtMost*)

lemma *fps_conv_radius_power*: $\text{fps\_conv\_radius } (f ^ n) \geq \text{fps\_conv\_radius } f$
proof (*induction n*)
case (*Suc n*)
hence $\text{fps\_conv\_radius } f \leq \min (\text{fps\_conv\_radius } f) (\text{fps\_conv\_radius } (f ^ n))$
by *simp*
also have $\dots \leq \text{fps\_conv\_radius } (f * f ^ n)$
by (*rule fps_conv_radius_mult*)
finally show ?*case* **by** *simp*
qed *simp_all*

context
begin

lemma *natfun_inverse_bound*:
fixes *f* :: 'a :: {*real_normed_field*} *fps*
assumes *fps_nth f 0 = 1* **and** $\delta > 0$
and *summable*: *summable* ($\lambda n. \text{norm } (\text{fps\_nth } f (\text{Suc } n)) * \delta ^ \text{Suc } n$)
and *le*: $(\sum n. \text{norm } (\text{fps\_nth } f (\text{Suc } n)) * \delta ^ \text{Suc } n) \leq 1$
shows $\text{norm } (\text{natfun\_inverse } f n) \leq \text{inverse } (\delta ^ n)$
proof (*induction n rule: less_induct*)
case (*less m*)
show ?*case*
proof (*cases m*)
case 0
thus ?*thesis* **using** *assms* **by** (*simp add: field_split_simps norm_inverse norm_divide*)
next
case [*simp*]: (*Suc n*)
have $\text{norm } (\text{natfun\_inverse } f (\text{Suc } n)) =$
 $\text{norm } (\sum i = \text{Suc } 0 .. \text{Suc } n. \text{fps\_nth } f i * \text{natfun\_inverse } f (\text{Suc } n - i))$
(is $\_ = \text{norm } ?S$) **using** *assms*
by (*simp add: field_simps norm_mult norm_divide del: sum.cl_ivl_Suc*)
also have $\text{norm } ?S \leq (\sum i = \text{Suc } 0 .. \text{Suc } n. \text{norm } (\text{fps\_nth } f i * \text{natfun\_inverse } f (\text{Suc } n - i)))$

```

    by (rule norm_sum)
  also have ... ≤ (∑ i = Suc 0..Suc n. norm (fps_nth f i) / δ ^ (Suc n - i))
  proof (intro sum_mono, goal_cases)
    case (1 i)
    have norm (fps_nth f i * natfun_inverse f (Suc n - i)) =
      norm (fps_nth f i) * norm (natfun_inverse f (Suc n - i))
    by (simp add: norm_mult)
    also have ... ≤ norm (fps_nth f i) * inverse (δ ^ (Suc n - i))
    using 1 by (intro mult_left_mono less.IH) auto
    also have ... = norm (fps_nth f i) / δ ^ (Suc n - i)
    by (simp add: field_split_simps)
    finally show ?case .
  qed
  also have ... = (∑ i = Suc 0..Suc n. norm (fps_nth f i) * δ ^ i) / δ ^ Suc n
  by (subst sum_divide_distrib, rule sum.cong)
    (insert ⟨δ > 0⟩, auto simp: field_simps power_diff)
  also have (∑ i = Suc 0..Suc n. norm (fps_nth f i) * δ ^ i) =
    (∑ i=0..n. norm (fps_nth f (Suc i)) * δ ^ (Suc i))
  by (subst sum.atLeast_Suc_atMost_Suc_shift) simp_all
  also have {0..n} = {..

```



```

define  $\delta$  where  $\delta = \text{real\_of\_ereal } (\min (\text{ereal } \varepsilon / 2) (?R / 2))$ 
have  $\delta$ :  $0 < \delta \wedge \delta < \varepsilon \wedge \text{ereal } \delta < ?R$ 
  using  $\langle \varepsilon > 0 \rangle$  and assms by (cases ?R) (auto simp:  $\delta\_def$  min_def)

have summable: summable  $(\lambda n. \text{norm } (\text{fps\_nth } f \ n) * \delta ^ n)$ 
  using  $\delta$  by (intro summable_in_conv_radius) (simp_all add:  $\text{fps\_conv\_radius\_def}$ )
hence  $(\lambda n. \text{norm } (\text{fps\_nth } f \ n) * \delta ^ n)$  sums  $\text{eval\_fps } h \ \delta$ 
  by (simp add:  $\text{eval\_fps\_def}$  summable_sums  $h\_def$ )
hence  $(\lambda n. \text{norm } (\text{fps\_nth } f \ (\text{Suc } n)) * \delta ^ {\text{Suc } n})$  sums  $(\text{eval\_fps } h \ \delta - 1)$ 
  by (subst sums_Suc_iff) (auto simp: assms)
moreover {
  from  $\delta$  have  $\delta \in \text{ball } 0 \ \varepsilon$  by auto
  also have  $\dots \subseteq \text{eval\_fps } h - \{..<2\} \cap \text{eball } 0 \ ?R$  by fact
  finally have  $\text{eval\_fps } h \ \delta < 2$  by simp
}
ultimately have  $le$ :  $(\sum n. \text{norm } (\text{fps\_nth } f \ (\text{Suc } n)) * \delta ^ {\text{Suc } n}) \leq 1$ 
  by (simp add: sums_iff)
from summable have summable: summable  $(\lambda n. \text{norm } (\text{fps\_nth } f \ (\text{Suc } n)) * \delta ^ {\text{Suc } n})$ 
  by (subst summable_Suc_iff)

have  $0 < \delta$  using  $\delta$  by blast
also have  $\delta = \text{inverse } (\text{limsup } (\lambda n. \text{ereal } (\text{inverse } \delta)))$ 
  using  $\delta$  by (subst Limsup_const) auto
also have  $\dots \leq \text{conv\_radius } (\text{natfun\_inverse } f)$ 
  unfolding conv_radius_def
proof (intro ereal_inverse_antimono Limsup_mono
  eventually_mono[OF eventually_gt_at_top[of 0]])
  fix  $n :: \text{nat}$  assume  $n$ :  $n > 0$ 
  have  $\text{root } n \ (\text{norm } (\text{natfun\_inverse } f \ n)) \leq \text{root } n \ (\text{inverse } (\delta ^ n))$ 
    using  $n$  assms  $\delta$  le summable
    by (intro real_root_le_mono natfun_inverse_bound) auto
  also have  $\dots = \text{inverse } \delta$ 
    using  $n$   $\delta$  by (simp add: power_inverse [symmetric] real_root_pos2)
  finally show  $\text{ereal } (\text{inverse } \delta) \geq \text{ereal } (\text{root } n \ (\text{norm } (\text{natfun\_inverse } f \ n)))$ 
    by (subst ereal_less_eq)
next
  have  $0 = \text{limsup } (\lambda n. 0 :: \text{ereal})$ 
    by (rule Limsup_const [symmetric]) auto
  also have  $\dots \leq \text{limsup } (\lambda n. \text{ereal } (\text{root } n \ (\text{norm } (\text{natfun\_inverse } f \ n))))$ 
    by (intro Limsup_mono) (auto simp: real_root_ge_zero)
  finally show  $0 \leq \dots$  by simp
qed
also have  $\dots = \text{fps\_conv\_radius } (\text{inverse } f)$ 
  using assms by (simp add:  $\text{fps\_conv\_radius\_def}$   $\text{fps\_inverse\_def}$ )
finally show ?thesis by (simp add: zero_ereal_def)
qed

```

```

lemma fps_conv_radius_inverse_pos:
  fixes f :: 'a :: {banach, real_normed_field} fps
  assumes fps_nth f 0  $\neq$  0 and fps_conv_radius f > 0
  shows fps_conv_radius (inverse f) > 0
proof -
  let ?c = fps_nth f 0
  have fps_conv_radius (inverse f) = fps_conv_radius (fps_const ?c * inverse f)
    using assms by (subst fps_conv_radius_cmult_left) auto
  also have fps_const ?c * inverse f = inverse (fps_const (inverse ?c) * f)
    using assms by (simp add: fps_inverse_mult fps_const_inverse)
  also have fps_conv_radius ... > 0 using assms
    by (intro fps_conv_radius_inverse_pos_aux)
    (auto simp: fps_conv_radius_cmult_left)
  finally show ?thesis .
qed

```

end

```

lemma fps_conv_radius_exp [simp]:
  fixes c :: 'a :: {banach, real_normed_field}
  shows fps_conv_radius (fps_exp c) =  $\infty$ 
  unfolding fps_conv_radius_def
proof (rule conv_radius_inftyI'')
  fix z :: 'a
  have ( $\lambda n.$  norm (c * z) ^ n /  $_R$  fact n) sums exp (norm (c * z))
    by (rule exp_converges)
  also have ( $\lambda n.$  norm (c * z) ^ n /  $_R$  fact n) = ( $\lambda n.$  norm (fps_nth (fps_exp c)
n * z ^ n))
    by (rule ext) (simp add: norm_divide norm_mult norm_power field_split_simps)
  finally have summable ... by (simp add: sums_iff)
  thus summable ( $\lambda n.$  fps_nth (fps_exp c) n * z ^ n)
    by (rule summable_norm_cancel)
qed

```

6.50.4 Evaluating power series

```

theorem eval_fps_deriv:
  assumes norm z < fps_conv_radius f
  shows eval_fps (fps_deriv f) z = deriv (eval_fps f) z
  by (intro DERIV_imp_deriv [symmetric] has_field_derivative_eval_fps assms)

```

```

theorem fps_nth_conv_deriv:
  fixes f :: complex fps
  assumes fps_conv_radius f > 0
  shows fps_nth f n = (deriv  $\widehat{\phantom{x}}$  n) (eval_fps f) 0 / fact n
  using assms
proof (induction n arbitrary: f)
  case 0
  thus ?case by (simp add: eval_fps_def)

```

```

next
  case (Suc n f)
  have (deriv  $\hat{\sim}$  Suc n) (eval_fps f) 0 = (deriv  $\hat{\sim}$  n) (deriv (eval_fps f)) 0
    unfolding funpow_Suc_right o_def ..
  also have eventually ( $\lambda z::\text{complex}.$   $z \in \text{eball } 0 \text{ (fps_conv_radius f)}$ ) (nhds 0)
    using Suc.premis by (intro eventually_nhds_in_open) (auto simp: zero_ereal_def)
  hence eventually ( $\lambda z.$  deriv (eval_fps f) z = eval_fps (fps_deriv f) z) (nhds 0)
    by eventually_elim (simp add: eval_fps_deriv)
  hence (deriv  $\hat{\sim}$  n) (deriv (eval_fps f)) 0 = (deriv  $\hat{\sim}$  n) (eval_fps (fps_deriv
f)) 0
    by (intro higher_deriv_cong_ev refl)
  also have ... / fact n = fps_nth (fps_deriv f) n
    using Suc.premis fps_conv_radius_deriv[of f]
    by (intro Suc.IH [symmetric]) auto
  also have ... / of_nat (Suc n) = fps_nth f (Suc n)
    by (simp add: fps_deriv_def del: of_nat_Suc)
  finally show ?case by (simp add: field_split_simps)
qed

```

```

theorem eval_fps_eqD:
  fixes f g :: complex fps
  assumes fps_conv_radius f > 0 fps_conv_radius g > 0
  assumes eventually ( $\lambda z.$  eval_fps f z = eval_fps g z) (nhds 0)
  shows f = g
proof (rule fps_ext)
  fix n :: nat
  have fps_nth f n = (deriv  $\hat{\sim}$  n) (eval_fps f) 0 / fact n
    using assms by (intro fps_nth_conv_deriv)
  also have (deriv  $\hat{\sim}$  n) (eval_fps f) 0 = (deriv  $\hat{\sim}$  n) (eval_fps g) 0
    by (intro higher_deriv_cong_ev refl assms)
  also have ... / fact n = fps_nth g n
    using assms by (intro fps_nth_conv_deriv [symmetric])
  finally show fps_nth f n = fps_nth g n .
qed

```

```

lemma eval_fps_const [simp]:
  fixes c :: 'a :: {banach, real_normed_div_algebra}
  shows eval_fps (fps_const c) z = c
proof -
  have ( $\lambda n::\text{nat}.$  if  $n \in \{0\}$  then c else 0) sums ( $\sum n \in \{0::\text{nat}\}.$  c)
    by (rule sums_If_finite_set) auto
  also have ?this  $\longleftrightarrow$  ( $\lambda n::\text{nat}.$  fps_nth (fps_const c) n * z  $^{\wedge}$  n) sums ( $\sum n \in \{0::\text{nat}\}.$ 
c)
    by (intro sums_cong) auto
  also have ( $\sum n \in \{0::\text{nat}\}.$  c) = c
    by simp
  finally show ?thesis
    by (simp add: eval_fps_def sums_iff)
qed

```

lemma *eval_fps_0 [simp]:*

eval_fps (0 :: 'a :: {banach, real_normed_div_algebra} fps) z = 0
by (*simp only: fps_const_0_eq_0 [symmetric] eval_fps_const*)

lemma *eval_fps_1 [simp]:*

eval_fps (1 :: 'a :: {banach, real_normed_div_algebra} fps) z = 1
by (*simp only: fps_const_1_eq_1 [symmetric] eval_fps_const*)

lemma *eval_fps_numeral [simp]:*

eval_fps (numeral n :: 'a :: {banach, real_normed_div_algebra} fps) z = numeral n
by (*simp only: numeral_fps_const eval_fps_const*)

lemma *eval_fps_X_power [simp]:*

eval_fps (fps_X ^ m :: 'a :: {banach, real_normed_div_algebra} fps) z = z ^ m
proof –
have ($\lambda n::nat. \text{if } n \in \{m\} \text{ then } z ^ n \text{ else } 0 :: 'a$) *sums* ($\sum n \in \{m::nat\}. z ^ n$)
by (*rule sums_If_finite_set*) *auto*
also have $?this \longleftrightarrow (\lambda n::nat. \text{fps\_nth } (fps\_X ^ m) n * z ^ n)$ *sums* ($\sum n \in \{m::nat\}. z ^ n$)
by (*intro sums_cong*) (*auto simp: fps_X_power_iff*)
also have ($\sum n \in \{m::nat\}. z ^ n$) = $z ^ m$
by *simp*
finally show *?thesis*
by (*simp add: eval_fps_def sums_iff*)
qed

lemma *eval_fps_X [simp]:*

eval_fps (fps_X :: 'a :: {banach, real_normed_div_algebra} fps) z = z
using *eval_fps_X_power[of 1 z]* **by** (*simp only: power_one_right*)

lemma *eval_fps_minus:*

fixes $f :: 'a :: \{banach, real\_normed\_div\_algebra\}$ *fps*
assumes $norm\ z < fps\_conv\_radius\ f$
shows $eval\_fps\ (-f)\ z = -eval\_fps\ f\ z$
using *assms unfolding eval_fps_def*
by (*subst suminf_minus [symmetric]*) (*auto intro!: summable_fps*)

lemma *eval_fps_add:*

fixes $f\ g :: 'a :: \{banach, real\_normed\_div\_algebra\}$ *fps*
assumes $norm\ z < fps\_conv\_radius\ f$ $norm\ z < fps\_conv\_radius\ g$
shows $eval\_fps\ (f + g)\ z = eval\_fps\ f\ z + eval\_fps\ g\ z$
using *assms unfolding eval_fps_def*
by (*subst suminf_add*) (*auto simp: ring_distrib intro!: summable_fps*)

lemma *eval_fps_diff:*

fixes $f\ g :: 'a :: \{banach, real\_normed\_div\_algebra\}$ *fps*
assumes $norm\ z < fps\_conv\_radius\ f$ $norm\ z < fps\_conv\_radius\ g$

```

shows eval_fps (f - g) z = eval_fps f z - eval_fps g z
using assms unfolding eval_fps_def
by (subst suminf_diff) (auto simp: ring_distrib intro!: summable_fps)

```

lemma eval_fps_mult:

```

fixes f g :: 'a :: {banach, real_normed_div_algebra, comm_ring_1} fps
assumes norm z < fps_conv_radius f norm z < fps_conv_radius g
shows eval_fps (f * g) z = eval_fps f z * eval_fps g z
proof -
  have eval_fps f z * eval_fps g z =
    (∑ k. ∑ i ≤ k. fps_nth f i * fps_nth g (k - i) * (z ^ i * z ^ (k - i)))
  unfolding eval_fps_def
  proof (subst Cauchy_product)
    show summable (λk. norm (fps_nth f k * z ^ k)) summable (λk. norm (fps_nth
g k * z ^ k))
    by (rule norm_summable_fps assms)+
  qed (simp_all add: algebra_simps)
  also have (λk. ∑ i ≤ k. fps_nth f i * fps_nth g (k - i) * (z ^ i * z ^ (k - i))) =
    (λk. ∑ i ≤ k. fps_nth f i * fps_nth g (k - i) * z ^ k)
  by (intro ext sum.cong refl) (simp add: power_add [symmetric])
  also have suminf ... = eval_fps (f * g) z
  by (simp add: eval_fps_def fps_mult_nth atLeast0AtMost sum_distrib_right)
  finally show ?thesis ..
qed

```

lemma eval_fps_shift:

```

fixes f :: 'a :: {banach, real_normed_div_algebra, comm_ring_1} fps
assumes n ≤ subdegree f norm z < fps_conv_radius f
shows eval_fps (fps_shift n f) z = (if z = 0 then fps_nth f n else eval_fps f z
/ z ^ n)
proof (cases z = 0)
case False
  have eval_fps (fps_shift n f * fps_X ^ n) z = eval_fps (fps_shift n f) z * z ^ n
  using assms by (subst eval_fps_mult) simp_all
  also from assms have fps_shift n f * fps_X ^ n = f
  by (simp add: fps_shift_times_fps_X_power)
  finally show ?thesis using False by (simp add: field_simps)
qed (simp_all add: eval_fps_at_0)

```

lemma eval_fps_exp [simp]:

```

fixes c :: 'a :: {banach, real_normed_field}
shows eval_fps (fps_exp c) z = exp (c * z) unfolding eval_fps_def exp_def
by (simp add: eval_fps_def exp_def scaleR_conv_of_real field_split_simps)

```

The case of division is more complicated and will therefore not be handled here. Handling division becomes much more easy using complex analysis, and we will do so once that is available.

6.50.5 Power series expansions of analytic functions

This predicate contains the notion that the given formal power series converges in some disc of positive radius around the origin and is equal to the given complex function there.

This relationship is unique in the sense that no complex function can have more than one formal power series to which it expands, and if two holomorphic functions that are holomorphic on a connected open set around the origin and have the same power series expansion, they must be equal on that set.

More concrete statements about the radius of convergence can usually be made, but for many purposes, the statement that the series converges to the function in some neighbourhood of the origin is enough, and that can be shown almost fully automatically in most cases, as there are straightforward introduction rules to show this.

In particular, when one wants to relate the coefficients of the power series to the values of the derivatives of the function at the origin, or if one wants to approximate the coefficients of the series with the singularities of the function, this predicate is enough.

definition

```
has_fps_expansion :: ('a :: {banach, real_normed_div_algebra} => 'a) => 'a fps
=> bool
(infixl has'_fps'_expansion 60)
where (f has_fps_expansion F) <=>
  fps_conv_radius F > 0 & eventually (λz. eval_fps F z = f z) (nhds 0)
```

named_theorems fps_expansion_intros

lemma fps_nth_fps_expansion:

```
fixes f :: complex => complex
assumes f has_fps_expansion F
shows fps_nth F n = (deriv ^^ n) f 0 / fact n
proof -
  have fps_nth F n = (deriv ^^ n) (eval_fps F) 0 / fact n
  using assms by (intro fps_nth_conv_deriv) (auto simp: has_fps_expansion_def)
  also have (deriv ^^ n) (eval_fps F) 0 = (deriv ^^ n) f 0
  using assms by (intro higher_deriv_cong_ev) (auto simp: has_fps_expansion_def)
  finally show ?thesis .
qed
```

lemma has_fps_expansion_imp_continuous:

```
fixes F :: 'a::{real_normed_field, banach} fps
assumes f has_fps_expansion F
shows continuous (at 0 within A) f
proof -
  from assms have isCont (eval_fps F) 0
```

```

    by (intro continuous_eval_fps) (auto simp: has_fps_expansion_def zero_ereal_def)
  also have ?this  $\longleftrightarrow$  isCont f 0 using assms
    by (intro isCont_cong) (auto simp: has_fps_expansion_def)
  finally have isCont f 0 .
  thus continuous (at 0 within A) f
    by (simp add: continuous_at_imp_continuous_within)
qed

lemma has_fps_expansion_const [simp, intro, fps_expansion_intros]:
  ( $\lambda$ _. c) has_fps_expansion fps_const c
  by (auto simp: has_fps_expansion_def)

lemma has_fps_expansion_0 [simp, intro, fps_expansion_intros]:
  ( $\lambda$ _. 0) has_fps_expansion 0
  by (auto simp: has_fps_expansion_def)

lemma has_fps_expansion_1 [simp, intro, fps_expansion_intros]:
  ( $\lambda$ _. 1) has_fps_expansion 1
  by (auto simp: has_fps_expansion_def)

lemma has_fps_expansion_numeral [simp, intro, fps_expansion_intros]:
  ( $\lambda$ _. numeral n) has_fps_expansion numeral n
  by (auto simp: has_fps_expansion_def)

lemma has_fps_expansion_fps_X_power [fps_expansion_intros]:
  ( $\lambda$ x. x ^ n) has_fps_expansion (fps_X ^ n)
  by (auto simp: has_fps_expansion_def)

lemma has_fps_expansion_fps_X [fps_expansion_intros]:
  ( $\lambda$ x. x) has_fps_expansion fps_X
  by (auto simp: has_fps_expansion_def)

lemma has_fps_expansion_cmult_left [fps_expansion_intros]:
  fixes c :: 'a :: {banach, real_normed_div_algebra, comm_ring_1}
  assumes f has_fps_expansion F
  shows ( $\lambda$ x. c * f x) has_fps_expansion fps_const c * F
proof (cases c = 0)
case False
  from assms have eventually ( $\lambda$ z. z  $\in$  eball 0 (fps_conv_radius F)) (nhds 0)
  by (intro eventually_nhds_in_open) (auto simp: has_fps_expansion_def zero_ereal_def)
  moreover from assms have eventually ( $\lambda$ z. eval_fps F z = f z) (nhds 0)
  by (auto simp: has_fps_expansion_def)
  ultimately have eventually ( $\lambda$ z. eval_fps (fps_const c * F) z = c * f z) (nhds
0)
  by eventually_elim (simp_all add: eval_fps_mult)
  with assms and False show ?thesis
  by (auto simp: has_fps_expansion_def fps_conv_radius_cmult_left)
qed auto

```

```

lemma has_fps_expansion_cmult_right [fps_expansion_intros]:
  fixes c :: 'a :: {banach, real_normed_div_algebra, comm_ring_1}
  assumes f has_fps_expansion F
  shows (λx. f x * c) has_fps_expansion F * fps_const c
proof -
  have F * fps_const c = fps_const c * F
  by (intro fps_ext) (auto simp: mult.commute)
  with has_fps_expansion_cmult_left [OF assms] show ?thesis
  by (simp add: mult.commute)
qed

```

```

lemma has_fps_expansion_minus [fps_expansion_intros]:
  assumes f has_fps_expansion F
  shows (λx. - f x) has_fps_expansion -F
proof -
  from assms have eventually (λx. x ∈ eball 0 (fps_conv_radius F)) (nhds 0)
  by (intro eventually_nhds_in_open) (auto simp: has_fps_expansion_def zero_ereal_def)
  moreover from assms have eventually (λx. eval_fps F x = f x) (nhds 0)
  by (auto simp: has_fps_expansion_def)
  ultimately have eventually (λx. eval_fps (-F) x = -f x) (nhds 0)
  by eventually_elim (auto simp: eval_fps_minus)
  thus ?thesis using assms by (auto simp: has_fps_expansion_def)
qed

```

```

lemma has_fps_expansion_add [fps_expansion_intros]:
  assumes f has_fps_expansion F g has_fps_expansion G
  shows (λx. f x + g x) has_fps_expansion F + G
proof -
  from assms have 0 < min (fps_conv_radius F) (fps_conv_radius G)
  by (auto simp: has_fps_expansion_def)
  also have ... ≤ fps_conv_radius (F + G)
  by (rule fps_conv_radius_add)
  finally have radius: ... > 0 .

  from assms have eventually (λx. x ∈ eball 0 (fps_conv_radius F)) (nhds 0)
  eventually (λx. x ∈ eball 0 (fps_conv_radius G)) (nhds 0)
  by (intro eventually_nhds_in_open; force simp: has_fps_expansion_def zero_ereal_def)
  moreover have eventually (λx. eval_fps F x = f x) (nhds 0)
  and eventually (λx. eval_fps G x = g x) (nhds 0)
  using assms by (auto simp: has_fps_expansion_def)
  ultimately have eventually (λx. eval_fps (F + G) x = f x + g x) (nhds 0)
  by eventually_elim (auto simp: eval_fps_add)
  with radius show ?thesis by (auto simp: has_fps_expansion_def)
qed

```

```

lemma has_fps_expansion_diff [fps_expansion_intros]:
  assumes f has_fps_expansion F g has_fps_expansion G
  shows (λx. f x - g x) has_fps_expansion F - G
  using has_fps_expansion_add[of f F λx. - g x - G] assms

```


by (simp add: has_fps_expansion_minus)

lemma has_fps_expansion_mult [fps_expansion_intros]:

fixes $F G :: 'a :: \{\text{banach}, \text{real\_normed\_div\_algebra}, \text{comm\_ring\_1}\}$ fps

assumes f has_fps_expansion F g has_fps_expansion G

shows $(\lambda x. f\ x * g\ x)$ has_fps_expansion $F * G$

proof –

from assms have $0 < \min(\text{fps\_conv\_radius } F) (\text{fps\_conv\_radius } G)$

by (auto simp: has_fps_expansion_def)

also have $\dots \leq \text{fps\_conv\_radius } (F * G)$

by (rule fps_conv_radius_mult)

finally have radius: $\dots > 0$.

from assms have eventually $(\lambda x. x \in \text{eball } 0 (\text{fps\_conv\_radius } F))$ (nhds 0)

eventually $(\lambda x. x \in \text{eball } 0 (\text{fps\_conv\_radius } G))$ (nhds 0)

by (intro eventually_nhds_in_open; force simp: has_fps_expansion_def zero_ereal_def)+

moreover have eventually $(\lambda x. \text{eval\_fps } F\ x = f\ x)$ (nhds 0)

and eventually $(\lambda x. \text{eval\_fps } G\ x = g\ x)$ (nhds 0)

using assms by (auto simp: has_fps_expansion_def)

ultimately have eventually $(\lambda x. \text{eval\_fps } (F * G)\ x = f\ x * g\ x)$ (nhds 0)

by eventually_elim (auto simp: eval_fps_mult)

with radius show ?thesis by (auto simp: has_fps_expansion_def)

qed

lemma has_fps_expansion_inverse [fps_expansion_intros]:

fixes $F :: 'a :: \{\text{banach}, \text{real\_normed\_field}\}$ fps

assumes f has_fps_expansion F

assumes $\text{fps\_nth } F\ 0 \neq 0$

shows $(\lambda x. \text{inverse } (f\ x))$ has_fps_expansion inverse F

proof –

have radius: $\text{fps\_conv\_radius } (\text{inverse } F) > 0$

using assms unfolding has_fps_expansion_def

by (intro fps_conv_radius_inverse_pos) auto

let $?R = \min(\text{fps\_conv\_radius } F) (\text{fps\_conv\_radius } (\text{inverse } F))$

from assms radius

have eventually $(\lambda x. x \in \text{eball } 0 (\text{fps\_conv\_radius } F))$ (nhds 0)

eventually $(\lambda x. x \in \text{eball } 0 (\text{fps\_conv\_radius } (\text{inverse } F)))$ (nhds 0)

by (intro eventually_nhds_in_open; force simp: has_fps_expansion_def zero_ereal_def)+

moreover have eventually $(\lambda z. \text{eval\_fps } F\ z = f\ z)$ (nhds 0)

using assms by (auto simp: has_fps_expansion_def)

ultimately have eventually $(\lambda z. \text{eval\_fps } (\text{inverse } F)\ z = \text{inverse } (f\ z))$ (nhds

0)

proof eventually_elim

case (elim z)

hence $\text{eval\_fps } (\text{inverse } F * F)\ z = \text{eval\_fps } (\text{inverse } F)\ z * f\ z$

by (subst eval_fps_mult) auto

also have $\text{eval\_fps } (\text{inverse } F * F)\ z = 1$

using assms by (simp add: inverse_mult_eq_1)

finally show ?case by (auto simp: field_split_simps)

3334

```

qed
with radius show ?thesis by (auto simp: has_fps_expansion_def)
qed

```

```

lemma has_fps_expansion_exp [fps_expansion_intros]:
  fixes c :: 'a :: {banach, real_normed_field}
  shows (λx. exp (c * x)) has_fps_expansion fps_exp c
  by (auto simp: has_fps_expansion_def)

```

```

lemma has_fps_expansion_exp1 [fps_expansion_intros]:
  (λx::'a :: {banach, real_normed_field}. exp x) has_fps_expansion fps_exp 1
  using has_fps_expansion_exp[of 1] by simp

```

```

lemma has_fps_expansion_exp_neg1 [fps_expansion_intros]:
  (λx::'a :: {banach, real_normed_field}. exp (-x)) has_fps_expansion fps_exp
  (-1)
  using has_fps_expansion_exp[of -1] by simp

```

```

lemma has_fps_expansion_deriv [fps_expansion_intros]:
  assumes f has_fps_expansion F
  shows deriv f has_fps_expansion fps_deriv F
proof -
  have eventually (λz. z ∈ eball 0 (fps_conv_radius F)) (nhds 0)
    using assms by (intro eventually_nhds_in_open)
    (auto simp: has_fps_expansion_def zero_ereal_def)
  moreover from assms have eventually (λz. eval_fps F z = f z) (nhds 0)
    by (auto simp: has_fps_expansion_def)
  then obtain s where open s 0 ∈ s and s: ⋀w. w ∈ s ⟹ eval_fps F w = f w
    by (auto simp: eventually_nhds)
  hence eventually (λw. w ∈ s) (nhds 0)
    by (intro eventually_nhds_in_open) auto
  ultimately have eventually (λz. eval_fps (fps_deriv F) z = deriv f z) (nhds 0)
  proof eventually_elim
    case (elim z)
    hence eval_fps (fps_deriv F) z = deriv (eval_fps F) z
      by (simp add: eval_fps_deriv)
    also have eventually (λw. w ∈ s) (nhds z)
      using elim and ⟨open s⟩ by (intro eventually_nhds_in_open) auto
    hence eventually (λw. eval_fps F w = f w) (nhds z)
      by eventually_elim (simp add: s)
    hence deriv (eval_fps F) z = deriv f z
      by (intro deriv_cong_ev refl)
    finally show ?case .
  qed
with assms and fps_conv_radius_deriv[of F] show ?thesis
  by (auto simp: has_fps_expansion_def)
qed

```

```

lemma fps_conv_radius_binomial:

```

```

fixes  $c :: 'a :: \{\text{real\_normed\_field}, \text{banach}\}$ 
shows  $\text{fps\_conv\_radius} (\text{fps\_binomial } c) = (\text{if } c \in \mathbb{N} \text{ then } \infty \text{ else } 1)$ 
unfolding  $\text{fps\_conv\_radius\_def}$  by ( $\text{simp add: conv\_radius\_gchoose}$ )

lemma  $\text{fps\_conv\_radius\_ln}$ :
  fixes  $c :: 'a :: \{\text{banach}, \text{real\_normed\_field}, \text{field\_char\_0}\}$ 
  shows  $\text{fps\_conv\_radius} (\text{fps\_ln } c) = (\text{if } c = 0 \text{ then } \infty \text{ else } 1)$ 
proof ( $\text{cases } c = 0$ )
  case False
  have  $\text{conv\_radius } (\lambda n. 1 / \text{of\_nat } n :: 'a) = 1$ 
  proof ( $\text{rule conv\_radius\_ratio\_limit\_nonzero}$ )
    show  $(\lambda n. \text{norm } (1 / \text{of\_nat } n :: 'a) / \text{norm } (1 / \text{of\_nat } (\text{Suc } n) :: 'a)) \longrightarrow$ 
    1
    using  $\text{LIMSEQ\_Suc\_n\_over\_n}$  by ( $\text{simp add: norm\_divide del: of\_nat\_Suc}$ )
  qed auto
  also have  $\text{conv\_radius } (\lambda n. 1 / \text{of\_nat } n :: 'a) =$ 
     $\text{conv\_radius } (\lambda n. \text{if } n = 0 \text{ then } 0 \text{ else } (-1) ^ (n - 1) / \text{of\_nat } n ::$ 
    'a)
    by ( $\text{intro conv\_radius\_cong eventually\_mono[OF eventually\_gt\_at\_top[of 0]]}$ )
    ( $\text{simp add: norm\_mult norm\_divide norm\_power}$ )
  finally show ?thesis using False unfolding  $\text{fps\_ln\_def}$ 
    by ( $\text{subst fps\_conv\_radius\_cmult\_left}$ ) ( $\text{simp\_all add: fps\_conv\_radius\_def}$ )
qed ( $\text{auto simp: fps\_ln\_def}$ )

lemma  $\text{fps\_conv\_radius\_ln\_nonzero}$  [simp]:
  assumes  $c \neq (0 :: 'a :: \{\text{banach}, \text{real\_normed\_field}, \text{field\_char\_0}\})$ 
  shows  $\text{fps\_conv\_radius} (\text{fps\_ln } c) = 1$ 
  using assms by ( $\text{simp add: fps\_conv\_radius\_ln}$ )

lemma  $\text{fps\_conv\_radius\_sin}$  [simp]:
  fixes  $c :: 'a :: \{\text{banach}, \text{real\_normed\_field}, \text{field\_char\_0}\}$ 
  shows  $\text{fps\_conv\_radius} (\text{fps\_sin } c) = \infty$ 
proof ( $\text{cases } c = 0$ )
  case False
  have  $\infty = \text{conv\_radius } (\lambda n. \text{of\_real } (\text{sin\_coeff } n) :: 'a)$ 
  proof ( $\text{rule sym, rule conv\_radius\_infyI'', rule summable\_norm\_cancel, goal\_cases}$ )
    case (1 z)
    show ?case using  $\text{summable\_norm\_sin[of } z]$  by ( $\text{simp add: norm\_mult}$ )
  qed
  also have  $\dots / \text{norm } c = \text{conv\_radius } (\lambda n. c ^ n * \text{of\_real } (\text{sin\_coeff } n) :: 'a)$ 
    using False by ( $\text{subst conv\_radius\_mult\_power}$ ) auto
  also have  $\dots = \text{fps\_conv\_radius} (\text{fps\_sin } c)$  unfolding  $\text{fps\_conv\_radius\_def}$ 
    by ( $\text{rule conv\_radius\_cong\_weak}$ ) ( $\text{auto simp add: fps\_sin\_def sin\_coeff\_def}$ )
  finally show ?thesis by simp
qed simp\_all

lemma  $\text{fps\_conv\_radius\_cos}$  [simp]:
  fixes  $c :: 'a :: \{\text{banach}, \text{real\_normed\_field}, \text{field\_char\_0}\}$ 
  shows  $\text{fps\_conv\_radius} (\text{fps\_cos } c) = \infty$ 

```

```

proof (cases c = 0)
  case False
  have  $\infty = \text{conv\_radius } (\lambda n. \text{of\_real } (\text{cos\_coeff } n) :: 'a)$ 
  proof (rule sym, rule conv_radius_inftyI'', rule summable_norm_cancel, goal_cases)
    case (1 z)
    show ?case using summable_norm_cos[of z] by (simp add: norm_mult)
  qed
  also have  $\dots / \text{norm } c = \text{conv\_radius } (\lambda n. c ^ n * \text{of\_real } (\text{cos\_coeff } n) :: 'a)$ 
  using False by (subst conv_radius_mult_power) auto
  also have  $\dots = \text{fps\_conv\_radius } (\text{fps\_cos } c)$  unfolding fps_conv_radius_def
  by (rule conv_radius_cong_weak) (auto simp add: fps_cos_def cos_coeff_def)
  finally show ?thesis by simp
qed simp_all

```

```

lemma eval_fps_sin [simp]:
  fixes  $z :: 'a :: \{\text{banach}, \text{real\_normed\_field}, \text{field\_char\_0}\}$ 
  shows  $\text{eval\_fps } (\text{fps\_sin } c) z = \text{sin } (c * z)$ 
proof -
  have  $(\lambda n. \text{sin\_coeff } n *_R (c * z) ^ n) \text{ sums sin } (c * z)$  by (rule sin_converges)
  also have  $(\lambda n. \text{sin\_coeff } n *_R (c * z) ^ n) = (\lambda n. \text{fps\_nth } (\text{fps\_sin } c) n * z ^ n)$ 
  by (rule ext) (auto simp: sin_coeff_def fps_sin_def power_mult_distrib scaleR_conv_of_real)
  finally show ?thesis by (simp add: sums_iff eval_fps_def)
qed

```

```

lemma eval_fps_cos [simp]:
  fixes  $z :: 'a :: \{\text{banach}, \text{real\_normed\_field}, \text{field\_char\_0}\}$ 
  shows  $\text{eval\_fps } (\text{fps\_cos } c) z = \text{cos } (c * z)$ 
proof -
  have  $(\lambda n. \text{cos\_coeff } n *_R (c * z) ^ n) \text{ sums cos } (c * z)$  by (rule cos_converges)
  also have  $(\lambda n. \text{cos\_coeff } n *_R (c * z) ^ n) = (\lambda n. \text{fps\_nth } (\text{fps\_cos } c) n * z ^ n)$ 
  by (rule ext) (auto simp: cos_coeff_def fps_cos_def power_mult_distrib scaleR_conv_of_real)
  finally show ?thesis by (simp add: sums_iff eval_fps_def)
qed

```

```

lemma cos_eq_zero_imp_norm_ge:
  assumes  $\text{cos } (z :: \text{complex}) = 0$ 
  shows  $\text{norm } z \geq \pi / 2$ 
proof -
  from assms obtain n where  $z = \text{complex\_of\_real } ((\text{of\_int } n + 1 / 2) * \pi i)$ 
  by (auto simp: cos_eq_0 algebra_simps)
  also have  $\text{norm } \dots = |\text{real\_of\_int } n + 1 / 2| * \pi$ 
  by (subst norm_of_real) (simp_all add: abs_mult)
  also have  $\text{real\_of\_int } n + 1 / 2 = \text{of\_int } (2 * n + 1) / 2$  by simp
  also have  $|\dots| = \text{of\_int } |2 * n + 1| / 2$  by (subst abs_divide) simp_all
  also have  $\dots * \pi = \text{of\_int } |2 * n + 1| * (\pi / 2)$  by simp
  also have  $\dots \geq \text{of\_int } 1 * (\pi / 2)$ 
  by (intro mult_right_mono, subst of_int_le_iff) (auto simp: abs_if)

```

finally show *?thesis* by simp
qed

lemma eval_fps_binomial:
fixes $c :: \text{complex}$
assumes $\text{norm } z < 1$
shows $\text{eval\_fps } (\text{fps\_binomial } c) \ z = (1 + z) \text{ powr } c$
using gen_binomial_complex[OF assms] by (simp add: sums_iff eval_fps_def)

lemma has_fps_expansion_binomial_complex [fps_expansion_intros]:
fixes $c :: \text{complex}$
shows $(\lambda x. (1 + x) \text{ powr } c) \text{ has\_fps\_expansion fps\_binomial } c$
proof –
have *: eventually $(\lambda z :: \text{complex}. z \in \text{eball } 0 \ 1) \ (\text{nhds } 0)$
by (intro eventually_nhds_in_open) auto
thus ?thesis
by (auto simp: has_fps_expansion_def eval_fps_binomial fps_conv_radius_binomial
intro!: eventually_mono [OF *])
qed

lemma has_fps_expansion_sin [fps_expansion_intros]:
fixes $c :: 'a :: \{\text{banach}, \text{real\_normed\_field}, \text{field\_char\_0}\}$
shows $(\lambda x. \sin (c * x)) \text{ has\_fps\_expansion fps\_sin } c$
by (auto simp: has_fps_expansion_def)

lemma has_fps_expansion_sin' [fps_expansion_intros]:
 $(\lambda x :: 'a :: \{\text{banach}, \text{real\_normed\_field}\}. \sin x) \text{ has\_fps\_expansion fps\_sin } 1$
using has_fps_expansion_sin[of 1] by simp

lemma has_fps_expansion_cos [fps_expansion_intros]:
fixes $c :: 'a :: \{\text{banach}, \text{real\_normed\_field}, \text{field\_char\_0}\}$
shows $(\lambda x. \cos (c * x)) \text{ has\_fps\_expansion fps\_cos } c$
by (auto simp: has_fps_expansion_def)

lemma has_fps_expansion_cos' [fps_expansion_intros]:
 $(\lambda x :: 'a :: \{\text{banach}, \text{real\_normed\_field}\}. \cos x) \text{ has\_fps\_expansion fps\_cos } 1$
using has_fps_expansion_cos[of 1] by simp

lemma has_fps_expansion_shift [fps_expansion_intros]:
fixes $F :: 'a :: \{\text{banach}, \text{real\_normed\_field}\}$ fps
assumes $f \text{ has\_fps\_expansion } F$ and $n \leq \text{subdegree } F$
assumes $c = \text{fps\_nth } F \ n$
shows $(\lambda x. \text{if } x = 0 \text{ then } c \text{ else } f \ x / x ^ n) \text{ has\_fps\_expansion } (\text{fps\_shift } n \ F)$
proof –
have eventually $(\lambda x. x \in \text{eball } 0 \ (\text{fps\_conv\_radius } F)) \ (\text{nhds } 0)$
using assms by (intro eventually_nhds_in_open) (auto simp: has_fps_expansion_def
zero_ereal_def)

moreover have *eventually* $(\lambda x. \text{eval\_fps } F \ x = f \ x) \ (nhds \ 0)$
using *assms* **by** $(\text{auto simp: has\_fps\_expansion\_def})$
ultimately have *eventually* $(\lambda x. \text{eval\_fps } (\text{fps\_shift } n \ F) \ x =$
 $(\text{if } x = 0 \text{ then } c \text{ else } f \ x / x \wedge n)) \ (nhds \ 0)$
by *eventually_elim* $(\text{auto simp: eval\_fps\_shift assms})$
with *assms* **show** *?thesis* **by** $(\text{auto simp: has\_fps\_expansion\_def})$
qed

lemma *has_fps_expansion_divide* [*fps_expansion_intros*]:
fixes $F \ G :: 'a :: \{\text{banach, real\_normed\_field}\}$ *fps*
assumes *f* *has_fps_expansion* F **and** *g* *has_fps_expansion* G **and**
 $\text{subdegree } G \leq \text{subdegree } F \ G \neq 0$
 $c = \text{fps\_nth } F \ (\text{subdegree } G) / \text{fps\_nth } G \ (\text{subdegree } G)$
shows $(\lambda x. \text{if } x = 0 \text{ then } c \text{ else } f \ x / g \ x) \text{ has\_fps\_expansion } (F / G)$
proof –
define n **where** $n = \text{subdegree } G$
define F' **and** G' **where** $F' = \text{fps\_shift } n \ F$ **and** $G' = \text{fps\_shift } n \ G$
have $F = F' * \text{fps\_X} \wedge n \ G = G' * \text{fps\_X} \wedge n$ **unfolding** $F'\_def \ G'\_def \ n\_def$
by $(\text{rule fps\_shift\_times\_fps\_X\_power } [\text{symmetric}] \ \text{le\_refl} \mid \text{fact}) +$
moreover from *assms* **have** $\text{fps\_nth } G' \ 0 \neq 0$
by $(\text{simp add: } G'\_def \ n\_def)$
ultimately have $FG: F / G = F' * \text{inverse } G'$
by $(\text{simp add: fps\_divide\_unit})$
have $(\lambda x. (\text{if } x = 0 \text{ then } \text{fps\_nth } F \ n \text{ else } f \ x / x \wedge n) * \text{inverse } (\text{if } x = 0 \text{ then } \text{fps\_nth } G \ n \text{ else } g \ x / x \wedge n)) \text{ has\_fps\_expansion } F / G$
 $(\text{is } ?h \text{ has\_fps\_expansion } \_) \text{ unfolding } FG \ F'\_def \ G'\_def \ n\_def \text{ using } \langle G \neq 0 \rangle$
by $(\text{intro has\_fps\_expansion\_mult has\_fps\_expansion\_inverse has\_fps\_expansion\_shift assms}) \text{ auto}$
also have $?h = (\lambda x. \text{if } x = 0 \text{ then } c \text{ else } f \ x / g \ x)$
using *assms*(5) **unfolding** n_def
by $(\text{intro ext}) \ (\text{auto split: if\_splits simp: field\_simps})$
finally show *?thesis* .
qed

lemma *has_fps_expansion_divide'* [*fps_expansion_intros*]:
fixes $F \ G :: 'a :: \{\text{banach, real\_normed\_field}\}$ *fps*
assumes *f* *has_fps_expansion* F **and** *g* *has_fps_expansion* G **and** $\text{fps\_nth } G \ 0 \neq 0$
shows $(\lambda x. f \ x / g \ x) \text{ has\_fps\_expansion } (F / G)$
proof –
have $(\lambda x. \text{if } x = 0 \text{ then } \text{fps\_nth } F \ 0 / \text{fps\_nth } G \ 0 \text{ else } f \ x / g \ x) \text{ has\_fps\_expansion } (F / G)$
 $(\text{is } ?h \text{ has\_fps\_expansion } \_) \text{ using } \text{assms}(3) \text{ by } (\text{intro has\_fps\_expansion\_divide assms}) \text{ auto}$
also from *assms* **have** $\text{fps\_nth } F \ 0 = f \ 0 \ \text{fps\_nth } G \ 0 = g \ 0$

```

  by (auto simp: has_fps_expansion_def eval_fps_at_0 dest: eventually_nhds_x_imp_x)
  hence ?h = ( $\lambda x. f\ x / g\ x$ ) by auto
  finally show ?thesis .
qed

```

```

lemma has_fps_expansion_tan [fps_expansion_intros]:
  fixes c :: 'a :: {banach, real_normed_field, field_char_0}
  shows ( $\lambda x. \tan (c * x)$ ) has_fps_expansion fps_tan c
proof -
  have ( $\lambda x. \sin (c * x) / \cos (c * x)$ ) has_fps_expansion fps_sin c / fps_cos c
  by (intro fps_expansion_intros) auto
  thus ?thesis by (simp add: tan_def fps_tan_def)
qed

```

```

lemma has_fps_expansion_tan' [fps_expansion_intros]:
  tan has_fps_expansion fps_tan (1 :: 'a :: {banach, real_normed_field, field_char_0})
  using has_fps_expansion_tan[of 1] by simp

```

```

lemma has_fps_expansion_imp_holomorphic:
  assumes f has_fps_expansion F
  obtains s where open s 0  $\in s$  f holomorphic_on s  $\wedge z. z \in s \implies f\ z = \text{eval\_fps } F\ z$ 
proof -
  from assms obtain s where s: open s 0  $\in s$   $\wedge z. z \in s \implies \text{eval\_fps } F\ z = f\ z$ 
  unfolding has_fps_expansion_def eventually_nhds by blast
  let ?s' = eball 0 (fps_conv_radius F)  $\cap s$ 
  have eval_fps F holomorphic_on ?s'
    by (intro holomorphic_intros) auto
  also have ?this  $\longleftrightarrow f$  holomorphic_on ?s'
    using s by (intro holomorphic_cong) auto
  finally show ?thesis using s assms
    by (intro that[of ?s']) (auto simp: has_fps_expansion_def zero_ereal_def)
qed
end

```

6.51 Smooth paths

```

theory Smooth_Paths
  imports
    Retracts
begin

```

6.51.1 Homeomorphisms of arc images

```

lemma path_connected_arc_complement:
  fixes  $\gamma :: \text{real} \Rightarrow 'a :: \text{euclidean\_space}$ 
  assumes arc  $\gamma\ 2 \leq \text{DIM}('a)$ 
  shows path_connected( $\neg \text{path\_image } \gamma$ )

```

```

proof –
  have path_image  $\gamma$  homeomorphic  $\{0..1::\text{real}\}$ 
    by (simp add: assms homeomorphic_arc_image_interval)
  then
  show ?thesis
    apply (rule path_connected_complement_homeomorphic_convex_compact)
    apply (auto simp: assms)
    done
qed

lemma connected_arc_complement:
  fixes  $\gamma :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
  assumes arc  $\gamma$   $2 \leq \text{DIM}('a)$ 
  shows connected(– path_image  $\gamma$ )
  by (simp add: assms path_connected_arc_complement path_connected_imp_connected)

lemma inside_arc_empty:
  fixes  $\gamma :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
  assumes arc  $\gamma$ 
  shows inside(path_image  $\gamma$ ) = {}
proof (cases  $\text{DIM}('a) = 1$ )
  case True
  then show ?thesis
    using assms connected_arc_image connected_convex_1_gen inside_convex by
    blast
  next
  case False
  show ?thesis
  proof (rule inside_bounded_complement_connected_empty)
    show connected (– path_image  $\gamma$ )
      apply (rule connected_arc_complement [OF assms])
      using False
    by (metis DIM_ge_Suc0 One_nat_def Suc_1 not_less_eq_eq order_class.order.antisym)
    show bounded (path_image  $\gamma$ )
      by (simp add: assms bounded_arc_image)
  qed
qed

lemma inside_simple_curve_imp_closed:
  fixes  $\gamma :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
  shows  $\llbracket \text{simple\_path } \gamma; x \in \text{inside}(\text{path\_image } \gamma) \rrbracket \Longrightarrow \text{pathfinish } \gamma = \text{pathstart } \gamma$ 
  using arc_simple_path inside_arc_empty by blast

```

6.51.2 Piecewise differentiability of paths

```

lemma continuous_on_joinpaths_D1:
  continuous_on  $\{0..1\}$  ( $g1 \text{ +++ } g2$ )  $\Longrightarrow \text{continuous\_on } \{0..1\} \ g1$ 
  apply (rule continuous_on_eq [of _ ( $g1 \text{ +++ } g2$ )  $\circ ((*)(\text{inverse } 2))$ ])

```



```

apply (rule continuous_intros | simp)+
apply (auto elim!: continuous_on_subset simp: joinpaths_def)
done

```

lemma continuous_on_joinpaths_D2:

```

   $\llbracket \text{continuous\_on } \{0..1\} (g1 +++ g2); \text{pathfinish } g1 = \text{pathstart } g2 \rrbracket \implies \text{continuous\_on } \{0..1\} g2$ 
apply (rule continuous_on_eq [of _ (g1 +++ g2)  $\circ (\lambda x. \text{inverse } 2 * x + 1/2)$ ])
apply (rule continuous_intros | simp)+
apply (auto elim!: continuous_on_subset simp add: joinpaths_def pathfinish_def
  pathstart_def Ball_def)
done

```

lemma piecewise_differentiable_D1:

```

  assumes (g1 +++ g2) piecewise_differentiable_on {0..1}
  shows g1 piecewise_differentiable_on {0..1}
proof -
  obtain S where cont: continuous_on {0..1} g1 and finite S
    and S:  $\bigwedge x. x \in \{0..1\} - S \implies g1 +++ g2 \text{ differentiable at } x \text{ within } \{0..1\}$ 
    using assms unfolding piecewise_differentiable_on_def
    by (blast dest!: continuous_on_joinpaths_D1)
  show ?thesis
    unfolding piecewise_differentiable_on_def
  proof (intro exI conjI ballI cont)
    show finite (insert 1 (((*)2) ' S))
      by (simp add: finite_S)
    show g1 differentiable at x within {0..1} if  $x \in \{0..1\} - \text{insert } 1 ((*)2 ' S)$ 
  for x
    proof (rule_tac d=dist (x/2) (1/2) in differentiable_transform_within)
      have g1 +++ g2 differentiable at (x / 2) within {0..1/2}
        by (rule differentiable_subset [OF S [of x/2]] | use that in force)+
      then show g1 +++ g2  $\circ (*) (\text{inverse } 2)$  differentiable at x within {0..1}
        using image_affinity_atLeastAtMost_div [of 2 0 0::real 1]
        by (auto intro: differentiable_chain_within)
    qed (use that in auto simp: joinpaths_def)
  qed
qed

```

lemma piecewise_differentiable_D2:

```

  assumes (g1 +++ g2) piecewise_differentiable_on {0..1} and eq: pathfinish g1
    = pathstart g2
  shows g2 piecewise_differentiable_on {0..1}
proof -
  have [simp]: g1 1 = g2 0
    using eq by (simp add: pathfinish_def pathstart_def)
  obtain S where cont: continuous_on {0..1} g2 and finite S
    and S:  $\bigwedge x. x \in \{0..1\} - S \implies g1 +++ g2 \text{ differentiable at } x \text{ within } \{0..1\}$ 
    using assms unfolding piecewise_differentiable_on_def
    by (blast dest!: continuous_on_joinpaths_D2)

```

```

show ?thesis
  unfolding piecewise_differentiable_on_def
proof (intro exI conjI ballI cont)
  show finite (insert 0 ((λx. 2*x-1)‘S))
    by (simp add: ⟨finite S⟩)
  show g2 differentiable at x within {0..1} if x ∈ {0..1} - insert 0 ((λx.
2*x-1)‘S) for x
  proof (rule_tac d=dist ((x+1)/2) (1/2) in differentiable_transform_within)
    have x2: (x + 1) / 2 ∉ S
      using that
      apply (clarsimp simp: image_iff)
      by (metis add.commute add_diff_cancel_left' mult_2 field_sum_of_halves)
    have g1 +++ g2 ∘ (λx. (x+1) / 2) differentiable at x within {0..1}
      by (rule differentiable_chain_within differentiable_subset [OF S [of (x+1)/2]])
  | use x2 that in force)+
    then show g1 +++ g2 ∘ (λx. (x+1) / 2) differentiable at x within {0..1}
      by (auto intro: differentiable_chain_within)
    show (g1 +++ g2 ∘ (λx. (x + 1) / 2)) x' = g2 x' if x' ∈ {0..1} dist x' x
    < dist ((x + 1) / 2) (1/2) for x'
    proof -
      have [simp]: (2*x'+2)/2 = x'+1
        by (simp add: field_split_simps)
      show ?thesis
        using that by (auto simp: joinpaths_def)
    qed
  qed (use that in ⟨auto simp: joinpaths_def⟩)
qed
qed
qed

lemma piecewise_C1_differentiable_D1:
  fixes g1 :: real ⇒ 'a::real_normed_field
  assumes (g1 +++ g2) piecewise_C1_differentiable_on {0..1}
  shows g1 piecewise_C1_differentiable_on {0..1}
proof -
  obtain S where finite S
    and co12: continuous_on ({0..1} - S) (λx. vector_derivative (g1 +++
g2) (at x))
    and g12D: ∀ x∈{0..1} - S. g1 +++ g2 differentiable at x
  using assms by (auto simp: piecewise_C1_differentiable_on_def C1_differentiable_on_eq)
  have g1D: g1 differentiable at x if x ∈ {0..1} - insert 1 ((*) 2 ‘ S) for x
proof (rule differentiable_transform_within)
  show g1 +++ g2 ∘ (*) (inverse 2) differentiable at x
    using that g12D
    apply (simp only: joinpaths_def)
    by (rule differentiable_chain_at derivative_intros | force)+
  show ∧x'. [|dist x' x < dist (x/2) (1/2)|]
    ⇒ (g1 +++ g2 ∘ (*) (inverse 2)) x' = g1 x'
    using that by (auto simp: dist_real_def joinpaths_def)
qed (use that in ⟨auto simp: dist_real_def⟩)

```

```

have [simp]: vector_derivative (g1 ∘ (*) 2) (at (x/2)) = 2 *R vector_derivative
g1 (at x)
  if x ∈ {0..1} - insert 1 ((*) 2 ' S) for x
  apply (subst vector_derivative_chain_at)
  using that
  apply (rule derivative_eq_intros g1D | simp)+
  done
have continuous_on ({0..1/2} - insert (1/2) S) (λx. vector_derivative (g1
+++ g2) (at x))
  using co12 by (rule continuous_on_subset) force
then have coDhalf: continuous_on ({0..1/2} - insert (1/2) S) (λx. vec-
tor_derivative (g1 ∘ (*) 2) (at x))
  proof (rule continuous_on_eq [OF _ vector_derivative_at])
    show (g1 +++ g2 has_vector_derivative vector_derivative (g1 ∘ (*) 2) (at
x)) (at x)
      if x ∈ {0..1/2} - insert (1/2) S for x
      proof (rule has_vector_derivative_transform_within)
        show (g1 ∘ (*) 2 has_vector_derivative vector_derivative (g1 ∘ (*) 2) (at
x)) (at x)
          using that
          by (force intro: g1D differentiable_chain_at simp: vector_derivative_works
[symmetric])
        show  $\bigwedge x'. \llbracket \text{dist } x' x < \text{dist } x (1/2) \rrbracket \implies (g1 \circ (*) 2) x' = (g1 +++ g2) x'$ 
          using that by (auto simp: dist_norm joinpaths_def)
        qed (use that in <auto simp: dist_norm>)
      qed
    have continuous_on ({0..1} - insert 1 ((*) 2 ' S))
      ((λx. 1/2 * vector_derivative (g1 ∘ (*) 2) (at x)) ∘ (*) (1/2))
      apply (rule continuous_intros)+
      using coDhalf
      apply (simp add: scaleR_conv_of_real image_set_diff image_image)
      done
    then have con_g1: continuous_on ({0..1} - insert 1 ((*) 2 ' S)) (λx. vec-
tor_derivative g1 (at x))
      by (rule continuous_on_eq) (simp add: scaleR_conv_of_real)
    have continuous_on {0..1} g1
      using continuous_on_joinpaths_D1 assms piecewise_C1_differentiable_on_def
by blast
  with <finite S> show ?thesis
    apply (clarsimp simp add: piecewise_C1_differentiable_on_def C1_differentiable_on_eq)
    apply (rule_tac x=insert 1 (((*) 2)' S) in exI)
    apply (simp add: g1D con_g1)
    done
qed

lemma piecewise_C1_differentiable_D2:
  fixes g2 :: real ⇒ 'a::real_normed_field
  assumes (g1 +++ g2) piecewise_C1_differentiable_on {0..1} pathfinish g1 =
pathstart g2

```

```

    shows  $g2$  piecewise_C1_differentiable_on  $\{0..1\}$ 
  proof -
    obtain  $S$  where finite  $S$ 
      and  $co12$ : continuous_on  $(\{0..1\} - S)$   $(\lambda x. \text{vector\_derivative } (g1 +++ g2) \text{ (at } x))$ 
      and  $g12D$ :  $\forall x \in \{0..1\} - S. g1 +++ g2$  differentiable at  $x$ 
    using assms by (auto simp: piecewise_C1_differentiable_on_def C1_differentiable_on_eq)
    have  $g2D$ :  $g2$  differentiable at  $x$  if  $x \in \{0..1\} - \text{insert } 0 ((\lambda x. 2*x-1) ' S)$  for
    x
  proof (rule differentiable_transform_within)
    show  $g1 +++ g2 \circ (\lambda x. (x+1) / 2)$  differentiable at  $x$ 
      using  $g12D$  that
      apply (simp only: joinpaths_def)
      apply (drule_tac  $x = (x+1) / 2$  in bspec, force simp: field_split_simps)
      apply (rule differentiable_chain_at derivative_intros | force)+
      done
    show  $\bigwedge x'. \text{dist } x' x < \text{dist } ((x+1) / 2) (1/2) \implies (g1 +++ g2 \circ (\lambda x. (x+1) / 2)) x' = g2 x'$ 
      using that by (auto simp: dist_real_def joinpaths_def field_simps)
      qed (use that in <auto simp: dist_norm>)
    have [simp]:  $\text{vector\_derivative } (g2 \circ (\lambda x. 2*x-1)) \text{ (at } ((x+1)/2)) = 2 *_{\mathbb{R}} \text{vector\_derivative } g2 \text{ (at } x)$ 
      if  $x \in \{0..1\} - \text{insert } 0 ((\lambda x. 2*x-1) ' S)$  for  $x$ 
      using that by (auto simp: vector_derivative_chain_at field_split_simps  $g2D$ )
    have continuous_on  $(\{1/2..1\} - \text{insert } (1/2) S)$   $(\lambda x. \text{vector\_derivative } (g1 +++ g2) \text{ (at } x))$ 
      using  $co12$  by (rule continuous_on_subset) force
    then have  $coDhalf$ : continuous_on  $(\{1/2..1\} - \text{insert } (1/2) S)$   $(\lambda x. \text{vector\_derivative } (g2 \circ (\lambda x. 2*x-1)) \text{ (at } x))$ 
    proof (rule continuous_on_eq [OF _ vector_derivative_at])
      show  $(g1 +++ g2 \text{ has\_vector\_derivative } \text{vector\_derivative } (g2 \circ (\lambda x. 2 * x - 1)) \text{ (at } x))$ 
        (at  $x$ )
      if  $x \in \{1 / 2..1\} - \text{insert } (1 / 2) S$  for  $x$ 
      proof (rule_tac  $f = g2 \circ (\lambda x. 2*x-1)$  and  $d = \text{dist } (3/4) ((x+1)/2)$  in has_vector_derivative_transformation)
        show  $(g2 \circ (\lambda x. 2 * x - 1) \text{ has\_vector\_derivative } \text{vector\_derivative } (g2 \circ (\lambda x. 2 * x - 1)) \text{ (at } x))$ 
          (at  $x$ )
        using that by (force intro:  $g2D$  differentiable_chain_at simp: vector_derivative_works [symmetric])
      show  $\bigwedge x'. \llbracket \text{dist } x' x < \text{dist } (3 / 4) ((x+1) / 2) \rrbracket \implies (g2 \circ (\lambda x. 2 * x - 1)) x' = (g1 +++ g2) x'$ 
        using that by (auto simp: dist_norm joinpaths_def add_divide_distrib)
      qed (use that in <auto simp: dist_norm>)
    qed
    have [simp]:  $((\lambda x. (x+1) / 2) ' (\{0..1\} - \text{insert } 0 ((\lambda x. 2 * x - 1) ' S))) = (\{1/2..1\} - \text{insert } (1/2) S)$ 
    apply (simp add: image_set_diff inj_on_def image_image)
    apply (auto simp: image_affinity_atLeastAtMost_div add_divide_distrib)

```

```

done
have continuous_on ({0..1} - insert 0 ((λx. 2*x-1) ' S))
  ((λx. 1/2 * vector_derivative (g2 ∘ (λx. 2*x-1)) (at x)) ∘ (λx.
(x+1)/2))
by (rule continuous_intros | simp add: coDhalf)+
then have con_g2: continuous_on ({0..1} - insert 0 ((λx. 2*x-1) ' S)) (λx.
vector_derivative g2 (at x))
by (rule continuous_on_eq) (simp add: scaleR_conv_of_real)
have continuous_on {0..1} g2
using continuous_on_joinpaths_D2 assms piecewise_C1_differentiable_on_def
by blast
with ⟨finite S⟩ show ?thesis
apply (clarsimp simp add: piecewise_C1_differentiable_on_def C1_differentiable_on_eq)
apply (rule_tac x=insert 0 ((λx. 2 * x - 1) ' S) in exI)
apply (simp add: g2D con_g2)
done
qed

```

6.51.3 Valid paths, and their start and finish

definition *valid_path* :: (real $\Rightarrow$ 'a :: real_normed_vector) $\Rightarrow$ bool
where *valid_path* $f \equiv f$ piecewise_C1_differentiable_on {0..1::real}

definition *closed_path* :: (real $\Rightarrow$ 'a :: real_normed_vector) $\Rightarrow$ bool
where *closed_path* $g \equiv g\ 0 = g\ 1$

In particular, all results for paths apply

lemma *valid_path_imp_path*: *valid_path* $g \implies$ *path* g
by (simp add: path_def piecewise_C1_differentiable_on_def valid_path_def)

lemma *connected_valid_path_image*: *valid_path* $g \implies$ *connected*(*path_image* g)
by (metis *connected_path_image* *valid_path_imp_path*)

lemma *compact_valid_path_image*: *valid_path* $g \implies$ *compact*(*path_image* g)
by (metis *compact_path_image* *valid_path_imp_path*)

lemma *bounded_valid_path_image*: *valid_path* $g \implies$ *bounded*(*path_image* g)
by (metis *bounded_path_image* *valid_path_imp_path*)

lemma *closed_valid_path_image*: *valid_path* $g \implies$ *closed*(*path_image* g)
by (metis *closed_path_image* *valid_path_imp_path*)

lemma *valid_path_compose*:
assumes *valid_path* g
and *der*: $\bigwedge x. x \in \text{path\_image } g \implies f \text{ field\_differentiable } (at\ x)$
and *con*: *continuous_on* (*path_image* g) (*deriv* f)
shows *valid_path* ($f \circ g$)

proof –

obtain S **where** *finite* S **and** *g_diff*: g C1_differentiable_on {0..1} – S

```

    using ⟨valid_path g⟩ unfolding valid_path_def piecewise_C1_differentiable_on_def
  by auto
  have f ∘ g differentiable at t when t ∈ {0..1} - S for t
  proof (rule differentiable_chain_at)
    show g differentiable at t using ⟨valid_path g⟩
    by (meson C1_differentiable_on_eq ⟨g C1_differentiable_on {0..1} - S⟩
that)
  next
    have g t ∈ path_image g using that DiffD1_image_eqI path_image_def by
metis
    then show f differentiable at (g t)
    using der[THEN field_differentiable_imp_differentiable] by auto
  qed
  moreover have continuous_on ({0..1} - S) (λx. vector_derivative (f ∘ g) (at
x))
  proof (rule continuous_on_eq [where f = λx. vector_derivative g (at x) *
deriv f (g x)],
    rule continuous_intros)
    show continuous_on ({0..1} - S) (λx. vector_derivative g (at x))
    using g_diff C1_differentiable_on_eq by auto
  next
    have continuous_on {0..1} (λx. deriv f (g x))
    using continuous_on_compose[OF _ con[unfolded path_image_def], unfolded
comp_def]
    ⟨valid_path g⟩ piecewise_C1_differentiable_on_def valid_path_def
    by blast
    then show continuous_on ({0..1} - S) (λx. deriv f (g x))
    using continuous_on_subset by blast
  next
    show vector_derivative g (at t) * deriv f (g t) = vector_derivative (f ∘ g)
(at t)
    when t ∈ {0..1} - S for t
    proof (rule vector_derivative_chain_at_general[symmetric])
      show g differentiable at t by (meson C1_differentiable_on_eq g_diff that)
    next
      have g t ∈ path_image g using that DiffD1_image_eqI path_image_def by
metis
      then show f field_differentiable at (g t) using der by auto
    qed
  qed
  ultimately have f ∘ g C1_differentiable_on {0..1} - S
  using C1_differentiable_on_eq by blast
  moreover have path (f ∘ g)
  apply (rule path_continuous_image[OF valid_path_imp_path[OF ⟨valid_path
g⟩]])
  using der
  by (simp add: continuous_at_imp_continuous_on field_differentiable_imp_continuous_at)
  ultimately show ?thesis unfolding valid_path_def piecewise_C1_differentiable_on_def
path_def

```

```

    using ⟨finite S⟩ by auto
qed

```

```

lemma valid_path_uminus_comp[simp]:
  fixes g::real ⇒ 'a ::real_normed_field
  shows valid_path (uminus ∘ g) ⟷ valid_path g
proof
  show valid_path g ⟹ valid_path (uminus ∘ g) for g::real ⇒ 'a
    by (auto intro!: valid_path_compose derivative_intros)
  then show valid_path g when valid_path (uminus ∘ g)
    by (metis fun.map_comp group_add_class.minus_comp_minus id_comp that)
qed

```

```

lemma valid_path_offset[simp]:
  shows valid_path (λt. g t - z) ⟷ valid_path g
proof
  show *: valid_path (g::real ⇒ 'a) ⟹ valid_path (λt. g t - z) for g z
    unfolding valid_path_def
    by (fastforce intro:derivative_intros C1_differentiable_imp_piecewise piecewise_C1_differentiable_diff)
  show valid_path (λt. g t - z) ⟹ valid_path g
    using *[of λt. g t - z, simplified] .
qed

```

```

lemma valid_path_imp_reverse:
  assumes valid_path g
  shows valid_path(reversepath g)
proof -
  obtain S where finite S and S: g C1_differentiable_on ({0..1} - S)
  using assms by (auto simp: valid_path_def piecewise_C1_differentiable_on_def)
  then have finite ((-) 1 ‘ S)
    by auto
  moreover have (reversepath g C1_differentiable_on ({0..1} - (-) 1 ‘ S))
    unfolding reversepath_def
    apply (rule C1_differentiable_compose [of λx::real. 1-x _ g, unfolded o_def])
    using S
    by (force simp: finite_vimageI inj_on_def C1_differentiable_on_eq elim!: continuous_on_subset)+
  ultimately show ?thesis using assms
    by (auto simp: valid_path_def piecewise_C1_differentiable_on_def path_def [symmetric])
qed

```

```

lemma valid_path_reversepath [simp]: valid_path(reversepath g) ⟷ valid_path g
  using valid_path_imp_reverse by force

```

```

lemma valid_path_join:
  assumes valid_path g1 valid_path g2 pathfinish g1 = pathstart g2

```

```

    shows valid_path(g1 +++ g2)
  proof -
    have g1 1 = g2 0
      using assms by (auto simp: pathfinish_def pathstart_def)
    moreover have (g1 ∘ (λx. 2*x)) piecewise_C1_differentiable_on {0..1/2}
      apply (rule piecewise_C1_differentiable_compose)
      using assms
      apply (auto simp: valid_path_def piecewise_C1_differentiable_on_def continuous_on_joinpaths)
      apply (force intro: finite_vimageI [where h = (*)2] inj_onI)
      done
    moreover have (g2 ∘ (λx. 2*x-1)) piecewise_C1_differentiable_on {1/2..1}
      apply (rule piecewise_C1_differentiable_compose)
      using assms unfolding valid_path_def piecewise_C1_differentiable_on_def
      by (auto intro!: continuous_intros finite_vimageI [where h = (λx. 2*x - 1)] inj_onI
        simp: image_affinity_atLeastAtMost_diff continuous_on_joinpaths)
    ultimately show ?thesis
      apply (simp only: valid_path_def continuous_on_joinpaths joinpaths_def)
      apply (rule piecewise_C1_differentiable_cases)
      apply (auto simp: o_def)
      done
  qed

```

```

lemma valid_path_join_D1:
  fixes g1 :: real ⇒ 'a::real_normed_field
  shows valid_path (g1 +++ g2) ⇒ valid_path g1
  unfolding valid_path_def
  by (rule piecewise_C1_differentiable_D1)

```

```

lemma valid_path_join_D2:
  fixes g2 :: real ⇒ 'a::real_normed_field
  shows [valid_path (g1 +++ g2); pathfinish g1 = pathstart g2] ⇒ valid_path g2
  unfolding valid_path_def
  by (rule piecewise_C1_differentiable_D2)

```

```

lemma valid_path_join_eq [simp]:
  fixes g2 :: real ⇒ 'a::real_normed_field
  shows pathfinish g1 = pathstart g2 ⇒ (valid_path(g1 +++ g2) ⟷ valid_path g1 ∧ valid_path g2)
  using valid_path_join_D1 valid_path_join_D2 valid_path_join by blast

```

```

lemma valid_path_shiftpath [intro]:
  assumes valid_path g pathfinish g = pathstart g a ∈ {0..1}
  shows valid_path(shiftpath a g)
  using assms
  apply (auto simp: valid_path_def shiftpath_alt_def)
  apply (rule piecewise_C1_differentiable_cases)

```



```

  apply (auto simp: algebra_simps)
  apply (rule piecewise_C1_differentiable_affine [of g 1 a, simplified o_def scaleR_one])
  apply (auto simp: pathfinish_def pathstart_def elim: piecewise_C1_differentiable_on_subset)
  apply (rule piecewise_C1_differentiable_affine [of g 1 a-1, simplified o_def
scaleR_one algebra_simps])
  apply (auto simp: pathfinish_def pathstart_def elim: piecewise_C1_differentiable_on_subset)
done

```

lemma *vector_derivative_linepath_within*:

```

   $x \in \{0..1\} \implies \text{vector\_derivative} (\text{linepath } a \ b) (\text{at } x \text{ within } \{0..1\}) = b - a$ 
  apply (rule vector_derivative_within_cbox [of 0 1::real, simplified])
  apply (auto simp: has_vector_derivative_linepath_within)
done

```

lemma *vector_derivative_linepath_at* [simp]: $\text{vector\_derivative} (\text{linepath } a \ b) (\text{at } x) = b - a$

```

  by (simp add: has_vector_derivative_linepath_within vector_derivative_at)

```

lemma *valid_path_linepath* [iff]: $\text{valid\_path} (\text{linepath } a \ b)$

```

  apply (simp add: valid_path_def piecewise_C1_differentiable_on_def C1_differentiable_on_eq
continuous_on_linepath)
  apply (rule_tac x={} in exI)
  apply (simp add: differentiable_on_def differentiable_def)
  using has_vector_derivative_def has_vector_derivative_linepath_within
  apply (fastforce simp add: continuous_on_eq continuous_within)
done

```

lemma *valid_path_subpath*:

```

  fixes  $g :: \text{real} \Rightarrow 'a :: \text{real\_normed\_vector}$ 
  assumes  $\text{valid\_path } g \ u \in \{0..1\} \ v \in \{0..1\}$ 
  shows  $\text{valid\_path} (\text{subpath } u \ v \ g)$ 
proof (cases  $v=u$ )
case True
  then show ?thesis
    unfolding valid_path_def subpath_def
    by (force intro: C1_differentiable_on_const C1_differentiable_imp_piecewise)
next
case False
  have  $(g \circ (\lambda x. ((v-u) * x + u))) \text{ piecewise\_C1\_differentiable\_on } \{0..1\}$ 
  apply (rule piecewise_C1_differentiable_compose)
  apply (simp add: C1_differentiable_imp_piecewise)
  apply (simp add: image_affinity_atLeastAtMost)
  using assms False
  apply (auto simp: algebra_simps valid_path_def piecewise_C1_differentiable_on_subset)
  apply (subst Int_commute)
  apply (auto simp: inj_on_def algebra_simps crossproduct_eq finite_vimage_IntI)
  done
  then show ?thesis
    by (auto simp: o_def valid_path_def subpath_def)

```

3350

qed

lemma *valid_path_rectpath* [*simp*, *intro*]: *valid_path* (*rectpath* *a b*)
 by (*simp* add: *Let_def* *rectpath_def*)

end

6.52 Neighbourhood bases and Locally path-connected spaces

theory *Locally*
imports
Path_Connected Function_Topology
begin

6.52.1 Neighbourhood Bases

Useful for "local" properties of various kinds

definition *neighbourhood_base_at* **where**
neighbourhood_base_at *x P X* $\equiv$
 $\forall W. \text{openin } X \ W \wedge x \in W$
 $\longrightarrow (\exists U \ V. \text{openin } X \ U \wedge P \ V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W)$

definition *neighbourhood_base_of* **where**
neighbourhood_base_of *P X* $\equiv$
 $\forall x \in \text{topspace } X. \text{neighbourhood_base_at } x \ P \ X$

lemma *neighbourhood_base_of*:
neighbourhood_base_of *P X* $\longleftrightarrow$
 $(\forall W \ x. \text{openin } X \ W \wedge x \in W$
 $\longrightarrow (\exists U \ V. \text{openin } X \ U \wedge P \ V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W))$
unfolding *neighbourhood_base_at_def* *neighbourhood_base_of_def*
using *openin_subset* **by** *blast*

lemma *neighbourhood_base_at_mono*:
 $\llbracket \text{neighbourhood_base_at } x \ P \ X; \bigwedge S. \llbracket P \ S; x \in S \rrbracket \Longrightarrow Q \ S \rrbracket \Longrightarrow \text{neighbourhood_base_at } x \ Q \ X$
unfolding *neighbourhood_base_at_def*
by (*meson subset_eq*)

lemma *neighbourhood_base_of_mono*:
 $\llbracket \text{neighbourhood_base_of } P \ X; \bigwedge S. P \ S \Longrightarrow Q \ S \rrbracket \Longrightarrow \text{neighbourhood_base_of } Q \ X$
unfolding *neighbourhood_base_of_def*
using *neighbourhood_base_at_mono* **by** *force*

lemma *open_neighbourhood_base_at*:
 $(\bigwedge S. \llbracket P \ S; x \in S \rrbracket \Longrightarrow \text{openin } X \ S)$

$\implies \text{neighbourhood\_base\_at } x \ P \ X \longleftrightarrow (\forall W. \text{openin } X \ W \wedge x \in W \longrightarrow (\exists U. P \ U \wedge x \in U \wedge U \subseteq W))$
unfolding *neighbourhood_base_at_def*
by (*meson subsetD*)

lemma *open_neighbourhood_base_of*:
 $(\forall S. P \ S \longrightarrow \text{openin } X \ S)$
 $\implies \text{neighbourhood\_base\_of } P \ X \longleftrightarrow (\forall W \ x. \text{openin } X \ W \wedge x \in W \longrightarrow (\exists U. P \ U \wedge x \in U \wedge U \subseteq W))$
apply (*simp add: neighbourhood_base_of, safe, blast*)
by *meson*

lemma *neighbourhood_base_of_open_subset*:
 $\llbracket \text{neighbourhood\_base\_of } P \ X; \text{openin } X \ S \rrbracket$
 $\implies \text{neighbourhood\_base\_of } P \ (\text{subtopology } X \ S)$
apply (*clarsimp simp add: neighbourhood_base_of openin_subtopology_alt image_def*)
apply (*rename_tac x V*)
apply (*drule_tac x=S \cap V in spec*)
apply (*simp add: Int_ac*)
by (*metis IntI le_infI1 openin_Int*)

lemma *neighbourhood_base_of_topology_base*:
 $\text{openin } X = \text{arbitrary union\_of } (\lambda W. W \in \mathcal{B})$
 $\implies \text{neighbourhood\_base\_of } P \ X \longleftrightarrow$
 $(\forall W \ x. W \in \mathcal{B} \wedge x \in W \longrightarrow (\exists U \ V. \text{openin } X \ U \wedge P \ V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W))$
apply (*auto simp: openin_topology_base_unique neighbourhood_base_of*)
by (*meson subset_trans*)

lemma *neighbourhood_base_at_unlocalized*:
assumes $\bigwedge S \ T. \llbracket P \ S; \text{openin } X \ T; x \in T; T \subseteq S \rrbracket \implies P \ T$
shows $\text{neighbourhood\_base\_at } x \ P \ X$
 $\longleftrightarrow (x \in \text{topspace } X \longrightarrow (\exists U \ V. \text{openin } X \ U \wedge P \ V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq \text{topspace } X))$
(is ?lhs = ?rhs)

proof
assume *R: ?rhs* **show** *?lhs*
unfolding *neighbourhood_base_at_def*
proof *clarify*
fix *W*
assume $\text{openin } X \ W \ x \in W$
then have $x \in \text{topspace } X$
using *openin_subset* **by** *blast*
with *R* **obtain** *U V* **where** $\text{openin } X \ U \ P \ V \ x \in U \ U \subseteq V \ V \subseteq \text{topspace } X$
by *blast*
then show $\exists U \ V. \text{openin } X \ U \wedge P \ V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W$
by (*metis IntI <openin X W> <x \in W> assms inf_le1 inf_le2 openin_Int*)
qed

qed (*simp add: neighbourhood_base_at_def*)

lemma *neighbourhood_base_at_with_subset*:

$\llbracket \text{openin } X \ U; x \in U \rrbracket$
 $\implies (\text{neighbourhood\_base\_at } x \ P \ X \longleftrightarrow \text{neighbourhood\_base\_at } x \ (\lambda T. T$
 $\subseteq U \wedge P \ T) \ X)$
 apply (*simp add: neighbourhood_base_at_def*)
 apply (*metis IntI Int_subset_iff openin_Int*)
 done

lemma *neighbourhood_base_of_with_subset*:

$\text{neighbourhood\_base\_of } P \ X \longleftrightarrow \text{neighbourhood\_base\_of } (\lambda t. t \subseteq \text{topspace } X$
 $\wedge P \ t) \ X$
 using *neighbourhood_base_at_with_subset*
 by (*fastforce simp add: neighbourhood_base_of_def*)

6.52.2 Locally path-connected spaces

definition *weakly_locally_path_connected_at*

where *weakly_locally_path_connected_at* $x \ X \equiv \text{neighbourhood\_base\_at } x \ (\text{path\_connectedin } X) \ X$

definition *locally_path_connected_at*

where *locally_path_connected_at* $x \ X \equiv$
 $\text{neighbourhood\_base\_at } x \ (\lambda U. \text{openin } X \ U \wedge \text{path\_connectedin } X \ U) \ X$

definition *locally_path_connected_space*

where *locally_path_connected_space* $X \equiv \text{neighbourhood\_base\_of } (\text{path\_connectedin } X) \ X$

lemma *locally_path_connected_space_alt*:

$\text{locally\_path\_connected\_space } X \longleftrightarrow \text{neighbourhood\_base\_of } (\lambda U. \text{openin } X \ U$
 $\wedge \text{path\_connectedin } X \ U) \ X$
 (is ?P = ?Q)

and *locally_path_connected_space_eq_open_path_component_of*:

$\text{locally\_path\_connected\_space } X \longleftrightarrow$
 $(\forall U \ x. \text{openin } X \ U \wedge x \in U \longrightarrow \text{openin } X \ (\text{Collect } (\text{path\_component\_of}$
 $(\text{subtopology } X \ U) \ x)))$
 (is ?P = ?R)

proof –

have *?P if ?Q*

using *locally_path_connected_space_def neighbourhood_base_of_mono that*

by *auto*

moreover have *?R if P: ?P*

proof –

show *?thesis*

proof *clarify*

fix $U \ y$

assume *openin* $X \ U \ y \in U$

```

      have  $\exists T. \text{openin } X \ T \wedge x \in T \wedge T \subseteq \text{Collect } (\text{path\_component\_of } (\text{subtopology } X \ U) \ y)$ 
      if  $\text{path\_component\_of } (\text{subtopology } X \ U) \ y \ x$  for  $x$ 

    proof -
      have  $x \in U$ 
      using  $\text{path\_component\_of\_equiv that topspace\_subtopology}$  by fastforce
      then have  $\exists Ua. \text{openin } X \ Ua \wedge (\exists V. \text{path\_connectedin } X \ V \wedge x \in Ua \wedge Ua \subseteq V \wedge V \subseteq U)$ 
      using  $P$ 
      by (simp add:  $\langle \text{openin } X \ U \rangle \text{locally\_path\_connected\_space\_def neighbourhood\_base\_of}$ )
      then show ?thesis
      by (metis  $\text{dual\_order.trans path\_component\_of\_equiv path\_component\_of\_maximal path\_connectedin\_subtopology subset\_iff that}$ )
      qed
      then show  $\text{openin } X \ (\text{Collect } (\text{path\_component\_of } (\text{subtopology } X \ U) \ y))$ 
      using  $\text{openin\_subopen}$  by force
      qed
    qed
  moreover have ?Q if ?R
  using that
  apply (simp add:  $\text{open\_neighbourhood\_base\_of}$ )
  by (metis  $\text{mem\_Collect\_eq openin\_subset path\_component\_of\_refl path\_connectedin\_path\_component\_of path\_connectedin\_subtopology that topspace\_subtopology\_subset}$ )
  ultimately show  $?P = ?Q \ ?P = ?R$ 
  by blast+
qed

lemma locally_path_connected_space:
  locally_path_connected_space  $X \longleftrightarrow (\forall V \ x. \text{openin } X \ V \wedge x \in V \longrightarrow (\exists U. \text{openin } X \ U \wedge \text{path\_connectedin } X \ U \wedge x \in U \wedge U \subseteq V))$ 
  by (simp add:  $\text{locally\_path\_connected\_space\_alt open\_neighbourhood\_base\_of}$ )

lemma locally_path_connected_space_open_path_components:
  locally_path_connected_space  $X \longleftrightarrow (\forall U \ c. \text{openin } X \ U \wedge c \in \text{path\_components\_of } (\text{subtopology } X \ U) \longrightarrow \text{openin } X \ c)$ 
  apply (auto simp:  $\text{locally\_path\_connected\_space\_eq\_open\_path\_component\_of path\_components\_of\_def}$ )
  by (metis  $\text{imageI inf.absorb\_iff2 openin\_closedin\_eq}$ )

lemma openin_path_component_of_locally_path_connected_space:
  locally_path_connected_space  $X \implies \text{openin } X \ (\text{Collect } (\text{path\_component\_of } X \ x))$ 
  apply (auto simp:  $\text{locally\_path\_connected\_space\_eq\_open\_path\_component\_of}$ )
  by (metis  $\text{openin\_empty openin\_topspace path\_component\_of\_eq\_empty subtopology\_topspace}$ )

```

lemma *openin_path_components_of_locally_path_connected_space*:

[[*locally_path_connected_space* X ; $c \in \text{path\_components\_of } X$]] $\implies \text{openin } X \ c$
apply (*auto simp: locally_path_connected_space_eq_open_path_component_of*)
by (*metis (no_types, lifting) imageE openin_topspace path_components_of_def subtopology_topspace*)

lemma *closedin_path_components_of_locally_path_connected_space*:

[[*locally_path_connected_space* X ; $c \in \text{path\_components\_of } X$]] $\implies \text{closedin } X \ c$

by (*simp add: closedin_def complement_path_components_of_Union openin_path_components_of_lo
openin_clauses(3) path_components_of_subset*)

lemma *closedin_path_component_of_locally_path_connected_space*:

assumes *locally_path_connected_space* X
shows *closedin* X (*Collect* (*path_component_of* X x))
proof (*cases* $x \in \text{topspace } X$)
case *True*
then show *?thesis*
by (*simp add: asms closedin_path_components_of_locally_path_connected_space
path_component_in_path_components_of*)
next
case *False*
then show *?thesis*
by (*metis closedin_empty_path_component_of_eq_empty*)
qed

lemma *weakly_locally_path_connected_at*:

weakly_locally_path_connected_at $x \ X \longleftrightarrow$
 $(\forall V. \text{openin } X \ V \wedge x \in V$
 $\longrightarrow (\exists U. \text{openin } X \ U \wedge x \in U \wedge U \subseteq V \wedge$
 $(\forall y \in U. \exists C. \text{path\_connectedin } X \ C \wedge C \subseteq V \wedge x \in C \wedge y \in C)))$
 $(\text{is } ?lhs = ?rhs)$

proof

assume *?lhs* **then show** *?rhs*

apply (*simp add: weakly_locally_path_connected_at_def neighbourhood_base_at_def*)
by (*meson order_trans subsetD*)

next

have $*$: $\exists V. \text{path\_connectedin } X \ V \wedge U \subseteq V \wedge V \subseteq W$
if $(\forall y \in U. \exists C. \text{path\_connectedin } X \ C \wedge C \subseteq W \wedge x \in C \wedge y \in C)$
for $W \ U$

proof (*intro exI conjI*)

let $?V = (\text{Collect} (\text{path\_component\_of} (\text{subtopology } X \ W) \ x))$
show *path_connectedin* X (*Collect* (*path_component_of* (*subtopology* $X \ W$)

x))

by (*meson path_connectedin_path_component_of_path_connectedin_subtopology*)
show $U \subseteq ?V$

by (*auto simp: path_component_of_path_connectedin_subtopology that*)
show $?V \subseteq W$

```

    by (meson path_connectedin_path_component_of_path_connectedin_subtopology)
  qed
  assume ?rhs
  then show ?lhs
    unfolding weakly_locally_path_connected_at_def neighbourhood_base_at_def
  by (metis *)
  qed

lemma locally_path_connected_space_im_kleinen:
  locally_path_connected_space X  $\longleftrightarrow$ 
  ( $\forall V x. \text{openin } X \ V \wedge x \in V$ 
 $\longrightarrow (\exists U. \text{openin } X \ U \wedge$ 
 $x \in U \wedge U \subseteq V \wedge$ 
 $(\forall y \in U. \exists c. \text{path\_connectedin } X \ c \wedge$ 
 $c \subseteq V \wedge x \in c \wedge y \in c)))$ )

apply (simp add: locally_path_connected_space_def neighbourhood_base_of_def)
apply (simp add: weakly_locally_path_connected_at_flip: weakly_locally_path_connected_at_def)
using openin_subset apply force
done

lemma locally_path_connected_space_open_subset:
   $\llbracket \text{locally\_path\_connected\_space } X; \text{openin } X \ s \rrbracket$ 
 $\implies \text{locally\_path\_connected\_space } (\text{subtopology } X \ s)$ 
  apply (simp add: locally_path_connected_space_def neighbourhood_base_of_openin_open_subtopology
    path_connectedin_subtopology)
  by (meson order_trans)

lemma locally_path_connected_space_quotient_map_image:
  assumes f: quotient_map X Y f and X: locally_path_connected_space X
  shows locally_path_connected_space Y
  unfolding locally_path_connected_space_open_path_components
  proof clarify
    fix V C
    assume V: openin Y V and c: C  $\in$  path_components_of (subtopology Y V)
    then have sub: C  $\subseteq$  topspace Y
    using path_connectedin_path_components_of_path_connectedin_subset_topspace
    path_connectedin_subtopology by blast
    have openin X {x  $\in$  topspace X. f x  $\in$  C}
    proof (subst openin_subopen, clarify)
      fix x
      assume x: x  $\in$  topspace X and f x  $\in$  C
      let ?T = Collect (path_component_of (subtopology X {z  $\in$  topspace X. f z  $\in$ 
V}) x)
      show  $\exists T. \text{openin } X \ T \wedge x \in T \wedge T \subseteq \{x \in \text{topspace } X. f x \in C\}$ 
      proof (intro exI conjI)
        have  $\exists U. \text{openin } X \ U \wedge ?T \in \text{path\_components\_of } (\text{subtopology } X \ U)$ 
        proof (intro exI conjI)
          show openin X ({z  $\in$  topspace X. f z  $\in$  V})
          using V f openin_subset quotient_map_def by fastforce

```

```

show Collect (path_component_of (subtopology X {z ∈ topspace X. f z ∈
V}) x)
  ∈ path_components_of (subtopology X {z ∈ topspace X. f z ∈ V})
    by (metis (no_types, lifting) Int_iff ⟨f x ∈ C⟩ c mem_Collect_eq
path_component_in_path_components_of path_components_of_subset topspace_subtopology
topspace_subtopology_subset x)
qed
with X show openin X ?T
  using locally_path_connected_space_open_path_components by blast
show x: x ∈ ?T
  using V ⟨f x ∈ C⟩ c openin_subset path_component_of_equiv path_components_of_subset
topspace_subtopology topspace_subtopology_subset x
    by fastforce
have f ‘ ?T ⊆ C
proof (rule path_components_of_maximal)
  show C ∈ path_components_of (subtopology Y V)
    by (simp add: c)
  show path_connectedin (subtopology Y V) (f ‘ ?T)
proof –
  have quotient_map (subtopology X {a ∈ topspace X. f a ∈ V}) (subtopology
Y V) f
    by (simp add: V f quotient_map_restriction)
  then show ?thesis
    by (meson path_connectedin_continuous_map_image path_connectedin_path_component_of
quotient_imp_continuous_map)
qed
show ¬ disjoint C (f ‘ ?T)
  by (metis (no_types, lifting) ⟨f x ∈ C⟩ x disjoint_iff image_eqI)
qed
then show ?T ⊆ {x ∈ topspace X. f x ∈ C}
  by (force simp: path_component_of_equiv)
qed
qed
then show openin Y C
  using f sub by (simp add: quotient_map_def)
qed

lemma homeomorphic_locally_path_connected_space:
  assumes X homeomorphic_space Y
  shows locally_path_connected_space X ⟷ locally_path_connected_space Y
proof –
  obtain f g where homeomorphic_maps X Y f g
    using assms homeomorphic_space_def by fastforce
  then show ?thesis
    by (metis (no_types) homeomorphic_map_def homeomorphic_maps_map lo-
cally_path_connected_space_quotient_map_image)
qed

lemma locally_path_connected_space_retraction_map_image:

```



```

    [[retraction_map X Y r; locally_path_connected_space X]]
      ==> locally_path_connected_space Y
  using Abstract_Topology.retraction_imp_quotient_map locally_path_connected_space_quotient_map_image
  by blast

```

lemma *locally_path_connected_space_euclideanreal*: *locally_path_connected_space euclideanreal*

```

  unfolding locally_path_connected_space_def neighbourhood_base_of
  proof clarsimp
    fix W and x :: real
    assume open W x ∈ W
    then obtain e where e > 0 and e:  $\bigwedge x'. |x' - x| < e \longrightarrow x' \in W$ 
      by (auto simp: open_real)
    then show  $\exists U. \text{open } U \wedge (\exists V. \text{path\_connected } V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W)$ 
      by (force intro!: convex_imp_path_connected exI [where x = {x-e<.. $x$ +e}])
  qed

```

lemma *locally_path_connected_space_discrete_topology*:

```

  locally_path_connected_space (discrete_topology U)
  using locally_path_connected_space_open_path_components by fastforce

```

lemma *path_component_eq_connected_component_of*:

```

  assumes locally_path_connected_space X
  shows (path_component_of_set X x = connected_component_of_set X x)
  proof (cases x ∈ topspace X)
    case True
    then show ?thesis
      using connectedin_connected_component_of [of X x]
      apply (clarsimp simp add: connectedin_def connected_space_clopen_in topspace_subtopology_subset
        cong: conj_cong)
      apply (drule_tac x=path_component_of_set X x in spec)
      by (metis assms closedin_closed_subtopology closedin_connected_component_of
        closedin_path_component_of locally_path_connected_space inf.absorb_iff2 inf.orderE
        openin_path_component_of locally_path_connected_space openin_subtopology path_component_of_eq_empty
        path_component_subset_connected_component_of)
    next
    case False
    then show ?thesis
      using connected_component_of_eq_empty path_component_of_eq_empty by
  fastforce
  qed

```

lemma *path_components_eq_connected_components_of*:

```

  locally_path_connected_space X ==> (path_components_of X = connected_components_of X)
  by (simp add: path_components_of_def connected_components_of_def image_def
    path_component_eq_connected_component_of)

```

```

lemma path_connected_eq_connected_space:
  locally_path_connected_space  $X$ 
     $\implies$  path_connected_space  $X \longleftrightarrow$  connected_space  $X$ 
  by (metis connected_components_of_subset_sing path_components_eq_connected_components_of
path_components_of_subset_singleton)

lemma locally_path_connected_space_product_topology:
  locally_path_connected_space(product_topology  $X$   $I$ )  $\longleftrightarrow$ 
    topspace(product_topology  $X$   $I$ ) =  $\{\}$   $\vee$ 
    finite  $\{i. i \in I \wedge \sim \text{path\_connected\_space}(X\ i)\} \wedge$ 
     $(\forall i \in I. \text{locally\_path\_connected\_space}(X\ i))$ 
    (is  $?lhs \longleftrightarrow ?empty \vee ?rhs$ )
proof (cases  $?empty$ )
  case True
    then show  $?thesis$ 
    by (simp add: locally_path_connected_space_def neighbourhood_base_of_openin_closedin_eq)
  next
    case False
    then obtain  $z$  where  $z: z \in (\prod_E i \in I. \text{topspace}(X\ i))$ 
    by auto
    have  $?rhs$  if  $L: ?lhs$ 
    proof –
      obtain  $U\ C$  where  $U: \text{openin}(\text{product\_topology}\ X\ I)\ U$ 
      and  $V: \text{path\_connectedin}(\text{product\_topology}\ X\ I)\ C$ 
      and  $z \in U\ U \subseteq C$  and  $C_{\text{sub}}: C \subseteq (\prod_E i \in I. \text{topspace}(X\ i))$ 
      using  $L$  apply (clarsimp simp add: locally_path_connected_space_def neigh-
bourhood_base_of)
      by (metis openin_tospace_product_topology  $z$ )
      then obtain  $V$  where  $\text{fin}V: \text{finite}\ \{i \in I. V\ i \neq \text{topspace}(X\ i)\}$ 
      and  $XV: \bigwedge i. i \in I \implies \text{openin}(X\ i)\ (V\ i)$  and  $z \in \text{Pi}_E\ I\ V$  and  $\text{sub}U: \text{Pi}_E$ 
 $I\ V \subseteq U$ 
      by (force simp: openin_product_topology_alt)
      show  $?thesis$ 
      proof (intro conjI ballI)
        have path_connected_space  $(X\ i)$  if  $i \in I\ V\ i = \text{topspace}(X\ i)$  for  $i$ 
        proof –
          have  $pc: \text{path\_connectedin}(X\ i)\ ((\lambda x. x\ i) \text{‘ } C)$ 
          apply (rule path_connectedin_continuous_map_image [OF  $V$ ])
          by (simp add: continuous_map_product_projection  $\langle i \in I \rangle$ )
          moreover have  $((\lambda x. x\ i) \text{‘ } C) = \text{topspace}(X\ i)$ 
          proof
            show  $(\lambda x. x\ i) \text{‘ } C \subseteq \text{topspace}(X\ i)$ 
            by (simp add: pc path_connectedin_subset_tospace)
            have  $V\ i \subseteq (\lambda x. x\ i) \text{‘ } (\prod_E i \in I. V\ i)$ 
            by (metis  $\langle z \in \text{Pi}_E\ I\ V \rangle$  empty_iff_image_projection_PiE order_refl
that( $I$ ))
            also have  $\dots \subseteq (\lambda x. x\ i) \text{‘ } U$ 
            using  $\text{sub}U$  by blast
            finally show topspace  $(X\ i) \subseteq (\lambda x. x\ i) \text{‘ } C$ 

```

```

      using  $\langle U \subseteq C \rangle$  that by blast
    qed
    ultimately show ?thesis
      by (simp add: path_connectedin_topspace)
    qed
    then have  $\{i \in I. \neg \text{path\_connected\_space } (X\ i)\} \subseteq \{i \in I. \forall i \neq \text{topspace } (X\ i)\}$ 
      by blast
    with finV show finite  $\{i \in I. \neg \text{path\_connected\_space } (X\ i)\}$ 
      using finite_subset by blast
  next
    show locally_path_connected_space (X i) if  $i \in I$  for i
      apply (rule locally_path_connected_space_quotient_map_image [OF _ L,
where  $f = \lambda x. x\ i$ ])
        by (metis False Abstract_Topology.retraction_imp_quotient_map retraction_map_product_projection that)
      qed
    qed
    moreover have ?lhs if R: ?rhs
      proof (clarsimp simp add: locally_path_connected_space_def neighbourhood_base_of)
        fix F z
        assume openin (product_topology X I) F and  $z \in F$ 
        then obtain W where finW: finite  $\{i \in I. W\ i \neq \text{topspace } (X\ i)\}$ 
          and opeW:  $\bigwedge i. i \in I \implies \text{openin } (X\ i) (W\ i)$  and  $z \in \text{Pi}_E\ I\ W\ \text{Pi}_E\ I$ 
        W  $\subseteq F$ 
        by (auto simp: openin_product_topology_alt)
        have  $\forall i \in I. \exists U\ C. \text{openin } (X\ i)\ U \wedge \text{path\_connectedin } (X\ i)\ C \wedge z\ i \in U \wedge U \subseteq C \wedge C \subseteq W\ i \wedge$ 
           $(W\ i = \text{topspace } (X\ i) \wedge \text{path\_connected\_space } (X\ i) \longrightarrow U = \text{topspace } (X\ i) \wedge C = \text{topspace } (X\ i))$ 
          (is  $\forall i \in I. ?\Phi\ i$ )
        proof
          fix i assume  $i \in I$ 
          have locally_path_connected_space (X i)
            by (simp add: R  $\langle i \in I \rangle$ )
          moreover have openin (X i) (W i)  $z\ i \in W\ i$ 
            using  $\langle z \in \text{Pi}_E\ I\ W \rangle$  opeW  $\langle i \in I \rangle$  by auto
          ultimately obtain U C where UC: openin (X i) U path_connectedin (X i)  $C\ z\ i \in U\ U \subseteq C\ C \subseteq W\ i$ 
            using  $\langle i \in I \rangle$  by (force simp: locally_path_connected_space_def neighbourhood_base_of)
          show ? $\Phi\ i$ 
          proof (cases  $W\ i = \text{topspace } (X\ i) \wedge \text{path\_connected\_space } (X\ i)$ )
            case True
            then show ?thesis
              using  $\langle z\ i \in W\ i \rangle$  path_connectedin_topspace by blast
          next
            case False

```

```

    then show ?thesis
      by (meson UC)
    qed
  qed
  then obtain U C where
    *:  $\bigwedge i. i \in I \implies \text{openin } (X \ i) \ (U \ i) \wedge \text{path\_connectedin } (X \ i) \ (C \ i) \wedge z \ i \in$ 
     $(U \ i) \wedge (U \ i) \subseteq (C \ i) \wedge (C \ i) \subseteq W \ i \wedge$ 
     $(W \ i = \text{topspace } (X \ i) \wedge \text{path\_connected\_space } (X \ i)$ 
     $\longrightarrow (U \ i) = \text{topspace } (X \ i) \wedge (C \ i) = \text{topspace } (X \ i))$ 
    by metis
  let ?A =  $\{i \in I. \neg \text{path\_connected\_space } (X \ i)\} \cup \{i \in I. W \ i \neq \text{topspace } (X$ 
   $i)\}$ 
  have  $\{i \in I. U \ i \neq \text{topspace } (X \ i)\} \subseteq ?A$ 
    by (clarsimp simp add: *)
  moreover have finite ?A
    by (simp add: that finW)
  ultimately have finite  $\{i \in I. U \ i \neq \text{topspace } (X \ i)\}$ 
    using finite_subset by auto
  then have openin (product_topology X I) (PiE I U)
    using * by (simp add: openin_PiE_gen)
  then show  $\exists U. \text{openin } (\text{product\_topology } X \ I) \ U \wedge$ 
     $(\exists V. \text{path\_connectedin } (\text{product\_topology } X \ I) \ V \wedge z \in U \wedge U \subseteq V \wedge$ 
     $V \subseteq F)$ 
    apply (rule_tac x=PiE I U in exI, simp)
    apply (rule_tac x=PiE I C in exI)
    using  $\langle z \in \text{PiE } I \ W \rangle \langle \text{PiE } I \ W \subseteq F \rangle *$ 
    apply (simp add: path_connectedin_PiE_subset_PiE_PiE_iff_PiE_mono
    dual_order.trans)
  done
  qed
  ultimately show ?thesis
    using False by blast
  qed
end

```

6.53 Euclidean space and n-spheres, as subtopologies of n-dimensional space

```

theory Abstract_Euclidean_Space
imports Homotopy Locally
begin

```

6.53.1 Euclidean spaces as abstract topologies

```

definition Euclidean_space :: nat  $\Rightarrow$  (nat  $\Rightarrow$  real) topology
  where Euclidean_space n  $\equiv$  subtopology (powertop_real UNIV)  $\{x. \forall i \geq n. x \ i = 0\}$ 

```

lemma *topspace_Euclidean_space*:

$\text{topspace}(\text{Euclidean\_space } n) = \{x. \forall i \geq n. x\ i = 0\}$

by (*simp add: Euclidean_space_def*)

lemma *nonempty_Euclidean_space*: $\text{topspace}(\text{Euclidean\_space } n) \neq \{\}$

by (*force simp: topspace_Euclidean_space*)

lemma *subset_Euclidean_space* [*simp*]:

$\text{topspace}(\text{Euclidean\_space } m) \subseteq \text{topspace}(\text{Euclidean\_space } n) \longleftrightarrow m \leq n$

apply (*simp add: topspace_Euclidean_space subset_iff, safe*)

apply (*drule_tac x=($\lambda i. \text{if } i < m \text{ then } 1 \text{ else } 0$) in spec*)

apply *auto*

using *not_less by fastforce*

lemma *topspace_Euclidean_space_alt*:

$\text{topspace}(\text{Euclidean\_space } n) = (\bigcap i \in \{n.. \}. \{x. x \in \text{topspace}(\text{powertop\_real UNIV}) \wedge x\ i \in \{0\}\})$

by (*auto simp: topspace_Euclidean_space*)

lemma *closedin_Euclidean_space*:

$\text{closedin}(\text{powertop\_real UNIV}) (\text{topspace}(\text{Euclidean\_space } n))$

proof –

have $\text{closedin}(\text{powertop\_real UNIV}) \{x. x\ i = 0\}$ **if** $n \leq i$ **for** i

proof –

have $\text{closedin}(\text{powertop\_real UNIV}) \{x \in \text{topspace}(\text{powertop\_real UNIV}). x\ i \in \{0\}\}$

proof (*rule closedin_continuous_map_preimage*)

show $\text{continuous\_map}(\text{powertop\_real UNIV}) \text{euclideanreal } (\lambda x. x\ i)$

by (*metis UNIV_I continuous_map_product_coordinates*)

show $\text{closedin euclideanreal } \{0\}$

by *simp*

qed

then show *?thesis*

by *auto*

qed

then show *?thesis*

unfolding *topspace_Euclidean_space_alt*

by *force*

qed

lemma *closedin_Euclidean_imp_closed*: $\text{closedin}(\text{Euclidean\_space } m) S \implies \text{closed } S$

by (*metis Euclidean_space_def closed_closedin closedin_Euclidean_space closedin_closed_subtopology euclidean_product_topology topspace_Euclidean_space*)

lemma *closedin_Euclidean_space_iff*:

$\text{closedin}(\text{Euclidean\_space } m) S \longleftrightarrow \text{closed } S \wedge S \subseteq \text{topspace}(\text{Euclidean\_space } m)$

```

(is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
    using closedin_closed_subtopology topspace_Euclidean_space
    by (fastforce simp: topspace_Euclidean_space_alt closedin_Euclidean_imp_closed)
  show ?rhs  $\implies$  ?lhs
    apply (simp add: closedin_subtopology Euclidean_space_def)
    by (metis (no_types) closed_closedin euclidean_product_topology inf.orderE)
qed

```

```

lemma continuous_map_componentwise_Euclidean_space:
  continuous_map X (Euclidean_space n) ( $\lambda x\ i.$  if  $i < n$  then  $f\ x\ i$  else 0)  $\longleftrightarrow$ 
  ( $\forall i < n.$  continuous_map X euclideanreal ( $\lambda x.$   $f\ x\ i$ ))
proof -
  have *: continuous_map X euclideanreal ( $\lambda x.$  if  $k < n$  then  $f\ x\ k$  else 0)
    if  $\bigwedge i. i < n \implies$  continuous_map X euclideanreal ( $\lambda x.$   $f\ x\ i$ ) for  $k$ 
    by (intro continuous_intros that)
  show ?thesis
    unfolding Euclidean_space_def continuous_map_in_subtopology
    by (fastforce simp: continuous_map_componentwise_UNIV * elim: continuous_map_eq)
qed

```

```

lemma continuous_map_Euclidean_space_add [continuous_intros]:
   $\llbracket$  continuous_map X (Euclidean_space n)  $f$ ; continuous_map X (Euclidean_space
  n)  $g$   $\rrbracket$ 
 $\implies$  continuous_map X (Euclidean_space n) ( $\lambda x\ i.$   $f\ x\ i + g\ x\ i$ )
  unfolding Euclidean_space_def continuous_map_in_subtopology
  by (fastforce simp add: continuous_map_componentwise_UNIV continuous_map_add)

```

```

lemma continuous_map_Euclidean_space_diff [continuous_intros]:
   $\llbracket$  continuous_map X (Euclidean_space n)  $f$ ; continuous_map X (Euclidean_space
  n)  $g$   $\rrbracket$ 
 $\implies$  continuous_map X (Euclidean_space n) ( $\lambda x\ i.$   $f\ x\ i - g\ x\ i$ )
  unfolding Euclidean_space_def continuous_map_in_subtopology
  by (fastforce simp add: continuous_map_componentwise_UNIV continuous_map_diff)

```

```

lemma continuous_map_Euclidean_space_iff:
  continuous_map (Euclidean_space m) euclidean g
  = continuous_on (topspace (Euclidean_space m)) g
proof
  assume continuous_map (Euclidean_space m) euclidean g
  then have continuous_map (top_of_set { $f. \forall n \geq m. f\ n = 0$ }) euclidean g
    by (simp add: Euclidean_space_def euclidean_product_topology)
  then show continuous_on (topspace (Euclidean_space m)) g
    by (metis continuous_map_subtopology_eu subtopology_topspace topspace_Euclidean_space)
next
  assume continuous_on (topspace (Euclidean_space m)) g
  then have continuous_map (top_of_set { $f. \forall n \geq m. f\ n = 0$ }) euclidean g

```

```

  by (metis (lifting) continuous_map_into_fulltopology continuous_map_subtopology_eu
order_refl topspace_Euclidean_space)
  then show continuous_map (Euclidean_space m) euclidean g
  by (simp add: Euclidean_space_def euclidean_product_topology)
qed

```

lemma *cm_Euclidean_space_iff_continuous_on*:

```

  continuous_map (subtopology (Euclidean_space m) S) (Euclidean_space n) f
   $\longleftrightarrow$  continuous_on (topspace (subtopology (Euclidean_space m) S)) f  $\wedge$ 
  f ' (topspace (subtopology (Euclidean_space m) S))  $\subseteq$  topspace (Euclidean_space
n)
  (is ?P  $\longleftrightarrow$  ?Q  $\wedge$  ?R)

```

proof –

have ?Q if ?P

proof –

have $\bigwedge n. \text{Euclidean\_space } n = \text{top\_of\_set } \{f. \forall m \geq n. f\ m = 0\}$

by (simp add: Euclidean_space_def euclidean_product_topology)

with that show ?thesis

by (simp add: subtopology_subtopology)

qed

moreover

have ?R if ?P

using that by (simp add: image_subset_iff continuous_map_def)

moreover

have ?P if ?Q ?R

proof –

```

  have continuous_map (top_of_set (topspace (subtopology (subtopology (powertop_real
UNIV) {f.  $\forall n \geq m. f\ n = 0$ }) S))) (top_of_set (topspace (subtopology (powertop_real
UNIV) {f.  $\forall na \geq n. f\ na = 0$ }))) f

```

using Euclidean_space_def that by auto

then show ?thesis

```

  by (simp add: Euclidean_space_def euclidean_product_topology subtopol-
ogy_subtopology)

```

qed

ultimately show ?thesis

by auto

qed

lemma *homeomorphic_Euclidean_space_product_topology*:

```

  Euclidean_space n homeomorphic_space product_topology ( $\lambda i. \text{euclideanreal}$ )
 $\{.. $n$ \}$ 

```

proof –

have cm: continuous_map (product_topology ($\lambda i. \text{euclideanreal}$) $\{.. n \}$)

euclideanreal ($\lambda x. \text{if } k < n \text{ then } x\ k \text{ else } 0$) **for** k

by (auto intro: continuous_map_if_continuous_map_product_projection)

show ?thesis

unfolding homeomorphic_space_def homeomorphic_maps_def

apply (rule_tac x= $\lambda f. \text{restrict } f\ \{.. n \}$ **in** exI)

apply (rule_tac x= $\lambda f\ i. \text{if } i < n \text{ then } f\ i \text{ else } 0$ **in** exI)

```

    apply (simp add: Euclidean_space_def continuous_map_in_subtopology)
    apply (intro conjI continuous_map_from_subtopology)
    apply (force simp: continuous_map_componentwise cm intro: continu-
ous_map_product_projection)+
  done
qed

```

```

lemma contractible_Euclidean_space [simp]: contractible_space (Euclidean_space
n)
  using homeomorphic_Euclidean_space_product_topology contractible_space_euclideanreal
    contractible_space_product_topology homeomorphic_space_contractibility by
blast

```

```

lemma path_connected_Euclidean_space: path_connected_space (Euclidean_space
n)
  by (simp add: contractible_imp_path_connected_space)

```

```

lemma connected_Euclidean_space: connected_space (Euclidean_space n)
  by (simp add: contractible_imp_connected_space)

```

```

lemma locally_path_connected_Euclidean_space:
  locally_path_connected_space (Euclidean_space n)
  apply (simp add: homeomorphic_locally_path_connected_space [OF homeomor-
phic_Euclidean_space_product_topology [of n]]
    locally_path_connected_space_product_topology)
  using locally_path_connected_space_euclideanreal by auto

```

```

lemma compact_Euclidean_space:
  compact_space (Euclidean_space n)  $\longleftrightarrow$   $n = 0$ 
  by (auto simp: homeomorphic_compact_space [OF homeomorphic_Euclidean_space_product_topology]
    compact_space_product_topology)

```

6.53.2 n-dimensional spheres

definition *nsphere where*

$nsphere\ n \equiv subtopology\ (Euclidean\_space\ (Suc\ n))\ \{x.\ (\sum\ i \leq n.\ x\ i^2) = 1\}$

lemma *nsphere:*

```

  nsphere n = subtopology (powertop_real UNIV)
    {x. ( $\sum\ i \leq n.\ x\ i^2$ ) = 1  $\wedge$  ( $\forall\ i > n.\ x\ i = 0$ ) }
  by (simp add: nsphere_def Euclidean_space_def subtopology_subtopology Suc_le_eq
Collect_conj_eq Int_commute)

```

lemma *continuous_map_nsphere_projection:* $continuous\_map\ (nsphere\ n)\ euclideanreal\ (\lambda x.\ x\ k)$

unfolding *nsphere*

by (*blast intro: continuous_map_from_subtopology [OF continuous_map_product_projection]*)

lemma *in_topspace_nsphere:* $(\lambda n.\ if\ n = 0\ then\ 1\ else\ 0) \in topspace\ (nsphere\ n)$

by (simp add: nsphere_def topspace_Euclidean_space power2_eq_square if_distrib
[where $f = \lambda x. x * \_$] cong: if_cong)

lemma nonempty_nsphere [simp]: $\sim (topspace(nsphere\ n) = \{\})$
using in_topspace_nsphere by auto

lemma subtopology_nsphere_equator:
subtopology (nsphere (Suc n)) $\{x. x(Suc\ n) = 0\} = nsphere\ n$
proof -
have $(\{x. (\sum_{i \leq n. (x\ i)^2}) + (x\ (Suc\ n))^2 = 1 \wedge (\forall i > Suc\ n. x\ i = 0)\} \cap \{x. x\ (Suc\ n) = 0\})$
= $\{x. (\sum_{i \leq n. (x\ i)^2}) = 1 \wedge (\forall i > n. x\ i = (0::real))\}$
using Suc_lessI [of n] by (fastforce simp: set_eq_iff)
then show ?thesis
by (simp add: nsphere_subtopology_subtopology)
qed

lemma topspace_nsphere_minus1:
assumes $x: x \in topspace\ (nsphere\ n)$ and $x\ n = 0$
shows $x \in topspace\ (nsphere\ (n - Suc\ 0))$
proof (cases $n = 0$)
case True
then show ?thesis
using x by auto
next
case False
have subt_eq: $nsphere\ (n - Suc\ 0) = subtopology\ (nsphere\ n)\ \{x. x\ n = 0\}$
by (metis False Suc_pred le_zero_eq not_le subtopology_nsphere_equator)
with x show ?thesis
by (simp add: assms)
qed

lemma continuous_map_nsphere_reflection:
continuous_map (nsphere n) (nsphere n) $(\lambda x\ i. \text{if } i = k \text{ then } -x\ i \text{ else } x\ i)$
proof -
have cm: continuous_map (powertop_real UNIV) euclideanreal $(\lambda x. \text{if } j = k \text{ then } -x\ j \text{ else } x\ j)$ for j
proof (cases $j=k$)
case True
then show ?thesis
by simp (metis UNIV_I continuous_map_product_projection)
next
case False
then show ?thesis
by (auto intro: continuous_map_product_projection)
qed
have eq: $(\text{if } i = k \text{ then } x\ k * x\ k \text{ else } x\ i * x\ i) = x\ i * x\ i$ for i and $x :: nat \Rightarrow real$
by simp

```

show ?thesis
apply (simp add: nsphere continuous_map_in_subtopology continuous_map_componentwise_UNIV
           continuous_map_from_subtopology cm)
apply (intro conjI allI impI continuous_intros continuous_map_from_subtopology
       continuous_map_product_projection)
      apply (auto simp: power2_eq_square if_distrib [where f =  $\lambda x. x * \_$ ] eq
             cong: if_cong)
    done
qed

```

proposition *contractible_space_upper_hemisphere:*

```

assumes  $k \leq n$ 
shows contractible_space(subtopology (nsphere n) { $x. x k \geq 0$ })
proof -
  define p:: nat  $\Rightarrow$  real where  $p \equiv \lambda i. \text{if } i = k \text{ then } 1 \text{ else } 0$ 
  have  $p \in \text{topspace}(\text{nsphere } n)$ 
    using assms
  by (simp add: nsphere p_def power2_eq_square if_distrib [where f =  $\lambda x. x * \_$ ]
      cong: if_cong)
  let ?g =  $\lambda x i. x i / \text{sqrt}(\sum j \leq n. x j^2)$ 
  let ?h =  $\lambda(t,q) i. (1 - t) * q i + t * p i$ 
  let ?Y = subtopology (Euclidean_space (Suc n)) { $x. 0 \leq x k \wedge (\exists i \leq n. x i \neq 0)$ }
  have continuous_map (prod_topology (top_of_set { $0..1$ }) (subtopology (nsphere
n) { $x. 0 \leq x k$ }))
    (subtopology (nsphere n) { $x. 0 \leq x k$ }) (?g  $\circ$  ?h))
  proof (rule continuous_map_compose)
    have *:  $\llbracket 0 \leq b k; (\sum i \leq n. (b i)^2) = 1; \forall i > n. b i = 0; 0 \leq a; a \leq 1 \rrbracket$ 
       $\implies \exists i. (i = k \longrightarrow (1 - a) * b k + a \neq 0) \wedge$ 
         $(i \neq k \longrightarrow i \leq n \wedge a \neq 1 \wedge b i \neq 0)$  for  $a::\text{real}$  and  $b$ 
    apply (cases  $a \neq 1 \wedge b k = 0$ ; simp)
    apply (metis (no_types, lifting) atMost_iff sum.neutral zero_power2)
    by (metis add.commute add_le_same_cancel2 diff_ge_0_iff_ge diff_zero
        less_eq_real_def mult_eq_0_iff mult_nonneg_nonneg not_le numeral_One zero_neq_numeral)
    show continuous_map (prod_topology (top_of_set { $0..1$ }) (subtopology (nsphere
n) { $x. 0 \leq x k$ })) ?Y ?h
      using assms
    apply (auto simp: * nsphere continuous_map_componentwise_UNIV
           prod_topology_subtopology_subtopology_subtopology case_prod_unfold
           continuous_map_in_subtopology Euclidean_space_def p_def if_distrib
           [where f =  $\lambda x. \_ * x$ ] cong: if_cong)
      apply (intro continuous_map_prod_snd continuous_intros continuous_map_from_subtopology)
        apply auto
      done
  next
    have 1:  $\bigwedge x i. \llbracket i \leq n; x i \neq 0 \rrbracket \implies (\sum i \leq n. (x i / \text{sqrt}(\sum j \leq n. (x j)^2))^2) = 1$ 
      by (force simp: sum_nonneg sum_nonneg_eq_0_iff field_split_simps simp
          flip: sum_divide_distrib)

```

```

have cm: continuous_map ?Y (nsphere n) ( $\lambda x i. x i / \text{sqrt} (\sum j \leq n. (x j)^2)$ )
unfolding Euclidean_space_def nsphere subtopology_subtopology continuous_map_in_subtopology
proof (intro continuous_intros conjI)
  show continuous_map
    ( $\text{subtopology} (\text{powertop\_real UNIV}) (\{x. \forall i \geq \text{Suc } n. x i = 0\} \cap \{x. 0 \leq x k \wedge (\exists i \leq n. x i \neq 0)\})$ )
    ( $\text{powertop\_real UNIV} (\lambda x i. x i / \text{sqrt} (\sum j \leq n. (x j)^2)$ )
  unfolding continuous_map_componentwise
  by (intro continuous_intros conjI ballI) (auto simp: sum_nonneg_eq_0_iff)
qed (auto simp: 1)
show continuous_map ?Y ( $\text{subtopology} (\text{nsphere } n) \{x. 0 \leq x k\}$ ) ( $\lambda x i. x i / \text{sqrt} (\sum j \leq n. (x j)^2)$ )
  by (force simp: cm sum_nonneg continuous_map_in_subtopology if_distrib
[where  $f = \lambda x. \_ * x$ ] cong: if_cong)
qed
moreover have (?g  $\circ$  ?h) (0, x) = x
  if  $x \in \text{topspace} (\text{subtopology} (\text{nsphere } n) \{x. 0 \leq x k\})$  for x
  using that
  by (simp add: assms nsphere)
moreover
have (?g  $\circ$  ?h) (1, x) = p
  if  $x \in \text{topspace} (\text{subtopology} (\text{nsphere } n) \{x. 0 \leq x k\})$  for x
  by (force simp: assms p_def power2_eq_square if_distrib [where  $f = \lambda x. x *$ 
_] cong: if_cong)
ultimately
show ?thesis
  apply (simp add: contractible_space_def homotopic_with)
  apply (rule_tac x=p in exI)
  apply (rule_tac x=?g  $\circ$  ?h in exI, force)
  done
qed

```

corollary contractible_space_lower_hemisphere:

```

  assumes  $k \leq n$ 
  shows contractible_space ( $\text{subtopology} (\text{nsphere } n) \{x. x k \leq 0\}$ )
proof -
  have contractible_space ( $\text{subtopology} (\text{nsphere } n) \{x. 0 \leq x k\}$ ) = ?thesis
  proof (rule homeomorphic_space_contractibility)
    show  $\text{subtopology} (\text{nsphere } n) \{x. 0 \leq x k\}$  homeomorphic_space subtopology
    ( $\text{nsphere } n$ )  $\{x. x k \leq 0\}$ 
    unfolding homeomorphic_space_def homeomorphic_maps_def
    apply (rule_tac x= $\lambda x i. \text{if } i = k \text{ then } -(x i) \text{ else } x i$  in exI)+
    apply (auto simp: continuous_map_in_subtopology continuous_map_from_subtopology
      continuous_map_nsphere_reflection)
    done
  qed
then show ?thesis

```

using *contractible_space_upper_hemisphere* [*OF assms*] by *metis*
 qed

proposition *nullhomotopic_nonsurjective_sphere_map*:

assumes *f*: *continuous_map* (*nsphere p*) (*nsphere p*) *f*

and *fim*: $f \text{ ' } (topspace(nsphere\ p)) \neq topspace(nsphere\ p)$

obtains *a* where *homotopic_with* ($\lambda x. True$) (*nsphere p*) (*nsphere p*) *f* ($\lambda x. a$)

proof –

obtain *a* where *a*: $a \in topspace(nsphere\ p)$ $a \notin f \text{ ' } (topspace(nsphere\ p))$

using *fim* *continuous_map_image_subset_topspace* *f* by *blast*

then have *a1*: $(\sum i \leq p. (a\ i)^2) = 1$ and *a0*: $\bigwedge i. i > p \implies a\ i = 0$

by (*simp_all* add: *nsphere*)

have *f1*: $(\sum j \leq p. (f\ x\ j)^2) = 1$ if $x \in topspace\ (nsphere\ p)$ for *x*

proof –

have $f\ x \in topspace\ (nsphere\ p)$

using *continuous_map_image_subset_topspace* *f* that by *blast*

then show *?thesis*

by (*simp* add: *nsphere*)

qed

show *thesis*

proof

let *?g* = $\lambda x\ i. x\ i / \text{sqrt}(\sum j \leq p. x\ j^2)$

let *?h* = $\lambda(t,x)\ i. (1 - t) * f\ x\ i - t * a\ i$

let *?Y* = *subtopology* (*Euclidean_space*(*Suc p*)) ($-\ \{\lambda i. 0\}$)

let *?T01* = *top_of_set* $\{0..1::\text{real}\}$

have *1*: *continuous_map* (*prod_topology* *?T01* (*nsphere p*)) (*nsphere p*) (*?g* $\circ$ *?h*)

proof (*rule* *continuous_map_compose*)

have *continuous_map* (*prod_topology* *?T01* (*nsphere p*)) *euclideanreal* (($\lambda x. f\ x\ k$) $\circ$ *snd*) for *k*

unfolding *nsphere*

apply (*simp* add: *continuous_map_of_snd*)

apply (*rule* *continuous_map_compose* [*of_ nsphere p f*, *unfolded o_def*])

using *f* apply (*simp* add: *nsphere*)

by (*simp* add: *continuous_map_nsphere_projection*)

then have *continuous_map* (*prod_topology* *?T01* (*nsphere p*)) *euclideanreal* ($\lambda r. ?h\ r\ k$)

for *k*

unfolding *case_prod_unfold o_def*

by (*intro* *continuous_map_into_fulltopology* [*OF* *continuous_mapfst*] *continuous_intros*) *auto*

moreover have $?h \text{ ' } (\{0..1\} \times topspace\ (nsphere\ p)) \subseteq \{x. \forall i \geq Suc\ p. x\ i = 0\}$

using *continuous_map_image_subset_topspace* [*OF* *f*]

by (*auto* *simp*: *nsphere_image_subset_iff* *a0*)

moreover have $(\lambda i. 0) \notin ?h \text{ ' } (\{0..1\} \times topspace\ (nsphere\ p))$

proof *clarify*

fix *t b*

```

    assume eq:  $(\lambda i. 0) = (\lambda i. (1 - t) * f\ b\ i - t * a\ i)$  and  $t \in \{0..1\}$  and
  b:  $b \in \text{topspace } (\text{nsphere } p)$ 
  have  $(1 - t)^2 = (\sum i \leq p. ((1 - t) * f\ b\ i)^2)$ 
    using f1 [OF b] by (simp add: power_mult_distrib flip: sum_distrib_left)
  also have  $\dots = (\sum i \leq p. (t * a\ i)^2)$ 
    using eq by (simp add: fun_eq_iff)
  also have  $\dots = t^2$ 
    using a1 by (simp add: power_mult_distrib flip: sum_distrib_left)
  finally have  $1 - t = t$ 
    by (simp add: power2_eq_iff)
  then have  $*: t = 1/2$ 
    by simp
  have fba:  $f\ b \neq a$ 
    using a(2) b by blast
  then show False
    using eq unfolding * by (simp add: fun_eq_iff)
qed
ultimately show continuous_map (prod_topology ?T01 (nsphere p)) ?Y ?h
  by (simp add: Euclidean_space_def continuous_map_in_subtopology continuous_map_componentwise_UNIV)
next
  have *:  $\llbracket \forall i \geq \text{Suc } p. x\ i = 0; x \neq (\lambda i. 0) \rrbracket \implies (\sum j \leq p. (x\ j)^2) \neq 0$  for  $x :: \text{nat} \Rightarrow \text{real}$ 
    by (force simp: fun_eq_iff not_less_eq_eq sum_nonneg_eq_0_iff)
  show continuous_map ?Y (nsphere p) ?g
    apply (simp add: Euclidean_space_def continuous_map_in_subtopology continuous_map_componentwise_UNIV
      nsphere continuous_map_componentwise_subtopology_subtopology)
    apply (intro conjI allI continuous_intros continuous_map_from_subtopology [OF continuous_map_product_projection])
    apply (simp_all add: *)
    apply (force simp: sum_nonneg fun_eq_iff not_less_eq_eq sum_nonneg_eq_0_iff power_divide simp flip: sum_divide_distrib)
  done
qed
have 2:  $(?g \circ ?h)\ (0, x) = f\ x$  if  $x \in \text{topspace } (\text{nsphere } p)$  for  $x$ 
  using that f1 by simp
have 3:  $(?g \circ ?h)\ (1, x) = (\lambda i. - a\ i)$  for  $x$ 
  using a by (force simp: field_split_simps nsphere)
then show homotopic_with  $(\lambda x. \text{True})$  (nsphere p) (nsphere p)  $f\ (\lambda x. (\lambda i. - a\ i))$ 
  by (force simp: homotopic_with intro: 1 2 3)
qed
qed

lemma Hausdorff_Euclidean_space:
  Hausdorff_space (Euclidean_space n)
  unfolding Euclidean_space_def
  by (rule Hausdorff_space_subtopology) (metis Hausdorff_space_euclidean Haus-
```

dorff_space_product_topology)

end

6.54 Metrics on product spaces

```
theory Function_Metric
  imports
    Function_Topology
    Elementary_Metric_Spaces
begin
```

In general, the product topology is not metrizable, unless the index set is countable. When the index set is countable, essentially any (convergent) combination of the metrics on the factors will do. We use below the simplest one, based on L^1 , but L^2 would also work, for instance.

What is not completely trivial is that the distance thus defined induces the same topology as the product topology. This is what we have to prove to show that we have an instance of *metric_space*.

The proofs below would work verbatim for general countable products of metric spaces. However, since distances are only implemented in terms of type classes, we only develop the theory for countable products of the same space.

```
instantiation fun :: (countable, metric_space) metric_space
begin
```

```
definition dist_fun_def:
```

$$\text{dist } x \ y = (\sum n. (1/2)^n * \min (\text{dist } (x \text{ (from\_nat } n)) \ (y \text{ (from\_nat } n))) \ 1)$$

```
definition uniformity_fun_def:
```

$$(\text{uniformity}::('a \Rightarrow 'b) \times ('a \Rightarrow 'b)) \text{ filter} = (\text{INF } e \in \{0 < ..\}. \text{principal } \{(x, y). \text{dist } (x::('a \Rightarrow 'b)) \ y < e\})$$

Except for the first one, the auxiliary lemmas below are only useful when proving the instance: once it is proved, they become trivial consequences of the general theory of metric spaces. It would thus be desirable to hide them once the instance is proved, but I do not know how to do this.

```
lemma dist_fun_le_dist_first_terms:
```

$$\text{dist } x \ y \leq 2 * \text{Max } \{\text{dist } (x \text{ (from\_nat } n)) \ (y \text{ (from\_nat } n)) \mid n. n \leq N\} + (1/2)^N$$

```
proof -
```

$$\text{have } (\sum n. (1 / 2)^{(n+\text{Suc } N)} * \min (\text{dist } (x \text{ (from\_nat } (n+\text{Suc } N))) \ (y \text{ (from\_nat } (n+\text{Suc } N)))) \ 1)$$

$$= (\sum n. (1 / 2)^{(\text{Suc } N)} * ((1/2)^n * \min (\text{dist } (x \text{ (from\_nat } (n+\text{Suc } N))) \ (y \text{ (from\_nat } (n+\text{Suc } N)))) \ 1)$$

```
  by (rule suminf_cong, simp add: power_add)
```

```

also have ... = (1/2)^(Suc N) * (∑ n. (1 / 2) ^ n * min (dist (x (from_nat
(n+Suc N))) (y (from_nat (n+Suc N)))) 1)
  apply (rule suminf_mult)
  by (rule summable_comparison_test'[of λn. (1/2)^n], auto simp add: summable_geometric_iff)
also have ... ≤ (1/2)^(Suc N) * (∑ n. (1 / 2) ^ n)
  apply (simp, rule suminf_le, simp)
  by (rule summable_comparison_test'[of λn. (1/2)^n], auto simp add: summable_geometric_iff)
also have ... = (1/2)^(Suc N) * 2
  using suminf_geometric[of 1/2] by auto
also have ... = (1/2)^N
  by auto
finally have *: (∑ n. (1 / 2) ^ (n+Suc N) * min (dist (x (from_nat (n+Suc
N))) (y (from_nat (n+Suc N)))) 1) ≤ (1/2)^N
  by simp

define M where M = Max {dist (x (from_nat n)) (y (from_nat n)) | n. n ≤ N}
have dist (x (from_nat 0)) (y (from_nat 0)) ≤ M
  unfolding M_def by (rule Max_ge, auto)
then have [simp]: M ≥ 0 by (meson dual_order.trans zero_le_dist)
have dist (x (from_nat n)) (y (from_nat n)) ≤ M if n ≤ N for n
  unfolding M_def apply (rule Max_ge) using that by auto
then have i: min (dist (x (from_nat n)) (y (from_nat n))) 1 ≤ M if n ≤ N for
n
  using that by force
have (∑ n < Suc N. (1 / 2) ^ n * min (dist (x (from_nat n)) (y (from_nat
n))) 1) ≤
  (∑ n < Suc N. M * (1 / 2) ^ n)
  by (rule sum_mono, simp add: i)
also have ... = M * (∑ n < Suc N. (1 / 2) ^ n)
  by (rule sum_distrib_left[symmetric])
also have ... ≤ M * (∑ n. (1 / 2) ^ n)
  by (rule mult_left_mono, rule sum_le_suminf, auto simp add: summable_geometric_iff)
also have ... = M * 2
  using suminf_geometric[of 1/2] by auto
finally have **: (∑ n < Suc N. (1 / 2) ^ n * min (dist (x (from_nat n)) (y
(from_nat n))) 1) ≤ 2 * M
  by simp

have dist x y = (∑ n. (1 / 2) ^ n * min (dist (x (from_nat n)) (y (from_nat
n))) 1)
  unfolding dist_fun_def by simp
also have ... = (∑ n. (1 / 2) ^ (n+Suc N) * min (dist (x (from_nat (n+Suc
N))) (y (from_nat (n+Suc N)))) 1)
  + (∑ n < Suc N. (1 / 2) ^ n * min (dist (x (from_nat n)) (y
(from_nat n))) 1)
  apply (rule suminf_split_initial_segment)
  by (rule summable_comparison_test'[of λn. (1/2)^n], auto simp add: summable_geometric_iff)
also have ... ≤ 2 * M + (1/2)^N
  using * ** by auto

```

finally show *?thesis unfolding M_def by simp*
qed

```

lemma open_fun_contains_ball_aux:
  assumes open (U::('a  $\Rightarrow$  'b) set))
    x  $\in$  U
  shows  $\exists e > 0. \{y. \text{dist } x \ y < e\} \subseteq U$ 
proof -
  have *: openin (product_topology ( $\lambda i. \text{euclidean}$ ) UNIV) U
    using open_fun_def assms by auto
  obtain X where H:  $\text{Pi}_E \text{ UNIV } X \subseteq U$ 
     $\bigwedge i. \text{openin euclidean } (X \ i)$ 
    finite  $\{i. X \ i \neq \text{topspace euclidean}\}$ 
    x  $\in \text{Pi}_E \text{ UNIV } X$ 
  using product_topology_open_contains_basis[OF *  $\langle x \in U \rangle$ ] by auto
  define I where  $I = \{i. X \ i \neq \text{topspace euclidean}\}$ 
  have finite I unfolding I_def using H(3) by auto
  {
    fix i
    have x i  $\in X \ i$  using  $\langle x \in U \rangle$  H by auto
    then have  $\exists e. e > 0 \wedge \text{ball } (x \ i) \ e \subseteq X \ i$ 
      using  $\langle \text{openin euclidean } (X \ i) \rangle$  open_openin open_contains_ball by blast
    then obtain e where  $e > 0 \wedge \text{ball } (x \ i) \ e \subseteq X \ i$  by blast
    define f where  $f = \min e \ (1/2)$ 
    have  $f > 0 \wedge f < 1$  unfolding f_def using  $\langle e > 0 \rangle$  by auto
    moreover have  $\text{ball } (x \ i) \ f \subseteq X \ i$  unfolding f_def using  $\langle \text{ball } (x \ i) \ e \subseteq X \ i \rangle$ 
    i) by auto
    ultimately have  $\exists f. f > 0 \wedge f < 1 \wedge \text{ball } (x \ i) \ f \subseteq X \ i$  by auto
  } note * = this
  have  $\exists e. \forall i. e \ i > 0 \wedge e \ i < 1 \wedge \text{ball } (x \ i) \ (e \ i) \subseteq X \ i$ 
    by (rule choice, auto simp add: *)
  then obtain e where  $\bigwedge i. e \ i > 0 \wedge \bigwedge i. e \ i < 1 \wedge \bigwedge i. \text{ball } (x \ i) \ (e \ i) \subseteq X \ i$ 
    by blast
  define m where  $m = \text{Min } \{(1/2)^\sim \text{to\_nat } i * e \ i \mid i. i \in I\}$ 
  have  $m > 0$  if  $I \neq \{\}$ 
    unfolding m_def Min_gr_iff using  $\langle \text{finite } I \rangle \langle I \neq \{\} \rangle \langle \bigwedge i. e \ i > 0 \rangle$  by auto
  moreover have  $\{y. \text{dist } x \ y < m\} \subseteq U$ 
  proof (auto)
    fix y assume  $\text{dist } x \ y < m$ 
    have  $y \ i \in X \ i$  if  $i \in I$  for i
    proof -
      have *: summable ( $\lambda n. (1/2)^\sim n * \min (\text{dist } (x \ (\text{from\_nat } n)) \ (y \ (\text{from\_nat } n))) \ 1$ )
        by (rule summable_comparison_test' [of  $\lambda n. (1/2)^\sim n$ ], auto simp add:
        summable_geometric_iff)
      define n where  $n = \text{to\_nat } i$ 
      have  $(1/2)^\sim n * \min (\text{dist } (x \ (\text{from\_nat } n)) \ (y \ (\text{from\_nat } n))) \ 1 < m$ 
        using  $\langle \text{dist } x \ y < m \rangle$  unfolding dist_fun_def using sum_le_suminf[OF
        *, of {n}] by auto
    end
  end

```



```

    then have  $(1/2)^\wedge (to\_nat\ i) * \min (dist\ (x\ i)\ (y\ i))\ 1 < m$ 
      using  $\langle n = to\_nat\ i \rangle$  by auto
    also have  $\dots \leq (1/2)^\wedge (to\_nat\ i) * e\ i$ 
      unfolding  $m\_def$  apply (rule  $Min\_le$ ) using  $\langle finite\ I \rangle\ \langle i \in I \rangle$  by auto
    ultimately have  $\min (dist\ (x\ i)\ (y\ i))\ 1 < e\ i$ 
      by (auto simp add:  $field\_split\_simps$ )
    then have  $dist\ (x\ i)\ (y\ i) < e\ i$ 
      using  $\langle e\ i < 1 \rangle$  by auto
    then show  $y\ i \in X\ i$  using  $\langle ball\ (x\ i)\ (e\ i) \subseteq X\ i \rangle$  by auto
  qed
  then have  $y \in Pi_E\ UNIV\ X$  using  $H(1)$  unfolding  $I\_def\ tospace\_euclidean$ 
by (auto simp add:  $PiE\_iff$ )
  then show  $y \in U$  using  $\langle Pi_E\ UNIV\ X \subseteq U \rangle$  by auto
  qed
  ultimately have  $*: \exists m > 0. \{y. dist\ x\ y < m\} \subseteq U$  if  $I \neq \{\}$  using that by
auto

  have  $U = UNIV$  if  $I = \{\}$ 
    using that  $H(1)$  unfolding  $I\_def\ tospace\_euclidean$  by (auto simp add:
 $PiE\_iff$ )
  then have  $\exists m > 0. \{y. dist\ x\ y < m\} \subseteq U$  if  $I = \{\}$  using that  $zero\_less\_one$ 
by blast
  then show  $\exists m > 0. \{y. dist\ x\ y < m\} \subseteq U$  using  $*$  by blast
  qed

lemma  $ball\_fun\_contains\_open\_aux$ :
  fixes  $x::('a \Rightarrow 'b)$  and  $e::real$ 
  assumes  $e > 0$ 
  shows  $\exists U. open\ U \wedge x \in U \wedge U \subseteq \{y. dist\ x\ y < e\}$ 
proof -
  have  $\exists N::nat. 2^\wedge N > 8/e$ 
    by (simp add:  $real\_arch\_pow$ )
  then obtain  $N::nat$  where  $2^\wedge N > 8/e$  by auto
  define  $f$  where  $f = e/4$ 
  have  $[simp]: e > 0 \wedge f > 0$  unfolding  $f\_def$  using  $assms$  by auto
  define  $X::('a \Rightarrow 'b\ set)$  where  $X = (\lambda i. if\ (\exists n \leq N. i = from\_nat\ n)\ then\ \{z. dist\ (x\ i)\ z < f\}$ 
else  $UNIV)$ 
  have  $\{i. X\ i \neq UNIV\} \subseteq from\_nat\ \{0..N\}$ 
    unfolding  $X\_def$  by auto
  then have  $finite\ \{i. X\ i \neq tospace\ euclidean\}$ 
    unfolding  $toSPACE\_euclidean$  using  $finite\_surj$  by blast
  have  $\bigwedge i. open\ (X\ i)$ 
    unfolding  $X\_def$  using  $metric\_space\_class.open\_ball\ open\_UNIV$  by auto
  then have  $\bigwedge i. openin\ euclidean\ (X\ i)$ 
    using  $open\_openin$  by auto
  define  $U$  where  $U = Pi_E\ UNIV\ X$ 
  have  $open\ U$ 
    unfolding  $open\_fun\_def\ product\_topology\_def$  apply (rule  $topology\_generated\_by\_Basis$ )
    unfolding  $U\_def$  using  $\langle \bigwedge i. openin\ euclidean\ (X\ i) \rangle\ \langle finite\ \{i. X\ i \neq tospace$ 

```

```

euclidean}⟩
  by auto
moreover have  $x \in U$ 
  unfolding  $U\_def$   $X\_def$  by (auto simp add:  $PiE\_iff$ )
moreover have  $dist\ x\ y < e$  if  $y \in U$  for  $y$ 
proof -
  have *:  $dist\ (x\ (from\_nat\ n))\ (y\ (from\_nat\ n)) \leq f$  if  $n \leq N$  for  $n$ 
    using  $\langle y \in U \rangle$  unfolding  $U\_def$  apply (auto simp add:  $PiE\_iff$ )
    unfolding  $X\_def$  using that by (metis less_imp_le mem_Collect_eq)
  have **:  $Max\ \{dist\ (x\ (from\_nat\ n))\ (y\ (from\_nat\ n)) \mid n. n \leq N\} \leq f$ 
    apply (rule Max.boundedI) using * by auto

  have  $dist\ x\ y \leq 2 * Max\ \{dist\ (x\ (from\_nat\ n))\ (y\ (from\_nat\ n)) \mid n. n \leq N\}$ 
+  $(1/2)^N$ 
    by (rule dist_fun_le_dist_first_terms)
  also have  $\dots \leq 2 * f + e / 8$ 
  apply (rule add_mono) using **  $\langle 2^N > 8/e \rangle$  by (auto simp add: field_split_simps)
  also have  $\dots \leq e/2 + e/8$ 
    unfolding  $f\_def$  by auto
  also have  $\dots < e$ 
    by auto
  finally show  $dist\ x\ y < e$  by simp
qed
ultimately show ?thesis by auto
qed

lemma fun_open_ball_aux:
  fixes  $U::('a \Rightarrow 'b)$  set
  shows  $open\ U \longleftrightarrow (\forall x \in U. \exists e > 0. \forall y. dist\ x\ y < e \longrightarrow y \in U)$ 
proof (auto)
  assume open  $U$ 
  fix  $x$  assume  $x \in U$ 
  then show  $\exists e > 0. \forall y. dist\ x\ y < e \longrightarrow y \in U$ 
    using open_fun_contains_ball_aux[OF  $\langle open\ U \rangle\ \langle x \in U \rangle$ ] by auto
next
  assume  $H: \forall x \in U. \exists e > 0. \forall y. dist\ x\ y < e \longrightarrow y \in U$ 
  define  $K$  where  $K = \{V. open\ V \wedge V \subseteq U\}$ 
  {
    fix  $x$  assume  $x \in U$ 
    then obtain  $e$  where  $e: e > 0\ \{y. dist\ x\ y < e\} \subseteq U$  using  $H$  by blast
    then obtain  $V$  where  $V: open\ V\ x \in V\ V \subseteq \{y. dist\ x\ y < e\}$ 
      using ball_fun_contains_open_aux[OF  $\langle e > 0 \rangle, of\ x$ ] by auto
    have  $V \in K$ 
      unfolding  $K\_def$  using  $e(2)\ V(1)\ V(3)$  by auto
    then have  $x \in (\bigcup K)$  using  $\langle x \in V \rangle$  by auto
  }
  then have  $(\bigcup K) = U$ 
    unfolding  $K\_def$  by auto
  moreover have  $open\ (\bigcup K)$ 

```

```

    unfolding K_def by auto
    ultimately show open U by simp
qed

```

```

instance proof

```

```

  fix x y::'a  $\Rightarrow$  'b show (dist x y = 0) = (x = y)
  proof
    assume x = y
    then show dist x y = 0 unfolding dist_fun_def using ‹x = y› by auto
  next
    assume dist x y = 0
    have *: summable ( $\lambda n. (1/2)^n * \min (dist (x (from\_nat n)) (y (from\_nat n))) 1$ )
    by (rule summable_comparison_test' [of  $\lambda n. (1/2)^n$ ], auto simp add: summable_geometric_iff)
    have  $(1/2)^n * \min (dist (x (from\_nat n)) (y (from\_nat n))) 1 = 0$  for n
    using ‹dist x y = 0› unfolding dist_fun_def by (simp add: * sum-
inf_eq_zero_iff)
    then have dist (x (from\_nat n)) (y (from\_nat n)) = 0 for n
    by (metis div_0 min_def nonzero_mult_div_cancel_left power_eq_0_iff
zero_eq_1_divide_iff zero_neq_numeral)
    then have x (from\_nat n) = y (from\_nat n) for n
    by auto
    then have x i = y i for i
    by (metis from_nat_to_nat)
    then show x = y
    by auto
  qed
next

```

The proof of the triangular inequality is trivial, modulo the fact that we are dealing with infinite series, hence we should justify the convergence at each step. In this respect, the following simplification rule is extremely handy.

```

  have [simp]: summable ( $\lambda n. (1/2)^n * \min (dist (u (from\_nat n)) (v (from\_nat n))) 1$ ) for u v::'a  $\Rightarrow$  'b
  by (rule summable_comparison_test' [of  $\lambda n. (1/2)^n$ ], auto simp add: summable_geometric_iff)
  fix x y z::'a  $\Rightarrow$  'b
  {
    fix n
    have *: dist (x (from\_nat n)) (y (from\_nat n))  $\leq$ 
      dist (x (from\_nat n)) (z (from\_nat n)) + dist (y (from\_nat n)) (z
(from\_nat n))
    by (simp add: dist_triangle2)
    have  $0 \leq dist (y (from\_nat n)) (z (from\_nat n))$ 
    using zero_le_dist by blast
  moreover
  {
    assume min (dist (y (from\_nat n)) (z (from\_nat n))) 1  $\neq$  dist (y (from\_nat
n)) (z (from\_nat n))
    then have  $1 \leq \min (dist (x (from\_nat n)) (z (from\_nat n))) 1 + \min (dist$ 

```

```

(y (from_nat n)) (z (from_nat n))) 1
  by (metis (no_types) diff_le_eq diff_self min_def zero_le_dist zero_le_one)
}
ultimately have min (dist (x (from_nat n)) (y (from_nat n))) 1 ≤
  min (dist (x (from_nat n)) (z (from_nat n))) 1 + min (dist (y (from_nat
n)) (z (from_nat n))) 1
  using * by linarith
} note ineq = this
have dist x y = (∑ n. (1/2)^n * min (dist (x (from_nat n)) (y (from_nat n)))
1)
  unfolding dist_fun_def by simp
also have ... ≤ (∑ n. (1/2)^n * min (dist (x (from_nat n)) (z (from_nat n)))
1
  + (1/2)^n * min (dist (y (from_nat n)) (z (from_nat n))) 1)
  apply (rule suminf_le)
  using ineq apply (metis (no_types, opaque_lifting) add.right_neutral dis-
trib_left
  le_divide_eq_numeral1(1) mult_2_right mult_left_mono zero_le_one zero_le_power)
  by (auto simp add: summable_add)
also have ... = (∑ n. (1/2)^n * min (dist (x (from_nat n)) (z (from_nat n)))
1)
  + (∑ n. (1/2)^n * min (dist (y (from_nat n)) (z (from_nat n)))
1)
  by (rule suminf_add[symmetric], auto)
also have ... = dist x z + dist y z
  unfolding dist_fun_def by simp
finally show dist x y ≤ dist x z + dist y z
  by simp
next

```

Finally, we show that the topology generated by the distance and the product topology coincide. This is essentially contained in Lemma *fun_open_ball_aux*, except that the condition to prove is expressed with filters. To deal with this, we copy some mumbo jumbo from Lemma *eventually_uniformity_metric* in `~/src/HOL/Real_Vector_Spaces.thy`

```

fix U::('a ⇒ 'b) set
have eventually P uniformity ⟷ (∃ e>0. ∀ x (y::('a ⇒ 'b)). dist x y < e ⟶
P (x, y)) for P
  unfolding uniformity_fun_def apply (subst eventually_INF_base)
  by (auto simp: eventually_principal subset_eq intro: bexI[of _ min _ _])
then show open U = (∀ x∈U. ∀_F (x', y) in uniformity. x' = x ⟶ y ∈ U)
  unfolding fun_open_ball_aux by simp
qed (simp add: uniformity_fun_def)

```

end

Nice properties of spaces are preserved under countable products. In addition to first countability, second countability and metrizability, as we have seen above, completeness is also preserved, and therefore being Polish.

```

instance fun :: (countable, complete_space) complete_space
proof
  fix u::nat  $\Rightarrow$  ('a  $\Rightarrow$  'b) assume Cauchy u
  have Cauchy ( $\lambda n. u\ n\ i$ ) for i
  unfolding cauchy_def proof (intro impI allI)
    fix e assume e>(0::real)
    obtain k where i = from_nat k
    using from_nat_to_nat[of i] by metis
    have  $(1/2)^k * \min\ e\ 1 > 0$  using e>0 by auto
    then have  $\exists N. \forall m\ n. N \leq m \wedge N \leq n \longrightarrow \text{dist}\ (u\ m)\ (u\ n) < (1/2)^k * \min\ e\ 1$ 
    using Cauchy u unfolding cauchy_def by blast
    then obtain N::nat where C:  $\forall m\ n. N \leq m \wedge N \leq n \longrightarrow \text{dist}\ (u\ m)\ (u\ n) < (1/2)^k * \min\ e\ 1$ 
    by blast
    have  $\forall m\ n. N \leq m \wedge N \leq n \longrightarrow \text{dist}\ (u\ m\ i)\ (u\ n\ i) < e$ 
    proof (auto)
      fix m n::nat assume m  $\geq N$  n  $\geq N$ 
      have  $(1/2)^k * \min\ (\text{dist}\ (u\ m\ i)\ (u\ n\ i))\ 1$ 
        = sum ( $\lambda p. (1/2)^p * \min\ (\text{dist}\ (u\ m\ (\text{from\_nat}\ p))\ (u\ n\ (\text{from\_nat}\ p)))\ 1$ ) {k}
      using i = from_nat k by auto
      also have ...  $\leq (\sum p. (1/2)^p * \min\ (\text{dist}\ (u\ m\ (\text{from\_nat}\ p))\ (u\ n\ (\text{from\_nat}\ p)))\ 1)$ 
      apply (rule sum_le_suminf)
      by (rule summable_comparison_test'[of  $\lambda n. (1/2)^n$ ], auto simp add:
        summable_geometric_iff)
      also have ... = dist (u m) (u n)
      unfolding dist_fun_def by auto
      also have ...  $< (1/2)^k * \min\ e\ 1$ 
      using C m  $\geq N$  n  $\geq N$  by auto
      finally have  $\min\ (\text{dist}\ (u\ m\ i)\ (u\ n\ i))\ 1 < \min\ e\ 1$ 
      by (auto simp add: field_split_simps)
      then show dist (u m i) (u n i)  $< e$  by auto
    qed
    then show  $\exists N. \forall m\ n. N \leq m \wedge N \leq n \longrightarrow \text{dist}\ (u\ m\ i)\ (u\ n\ i) < e$ 
    by blast
  qed
  then have  $\exists x. (\lambda n. u\ n\ i) \longrightarrow x$  for i
  using Cauchy_convergent_convergent_def by auto
  then have  $\exists x. \forall i. (\lambda n. u\ n\ i) \longrightarrow x\ i$ 
  using choice by force
  then obtain x where *:  $\bigwedge i. (\lambda n. u\ n\ i) \longrightarrow x\ i$  by blast
  have u  $\longrightarrow x$ 
  proof (rule metric_LIMSEQ_I)
    fix e assume [simp]: e>(0::real)
    have i:  $\exists K. \forall n \geq K. \text{dist}\ (u\ n\ i)\ (x\ i) < e/4$  for i
    by (rule metric_LIMSEQ_D, auto simp add: *)
    have  $\exists K. \forall i. \forall n \geq K\ i. \text{dist}\ (u\ n\ i)\ (x\ i) < e/4$ 

```

```

    apply (rule choice) using i by auto
  then obtain K where K:  $\bigwedge i n. n \geq K \implies \text{dist } (u \ n \ i) \ (x \ i) < e/4$ 
    by blast

  have  $\exists N::nat. 2^N > 4/e$ 
    by (simp add: real_arch_pow)
  then obtain N::nat where  $2^N > 4/e$  by auto
  define L where  $L = \text{Max } \{K \ (\text{from\_nat } n) \mid n. n \leq N\}$ 
  have  $\text{dist } (u \ k) \ x < e$  if  $k \geq L$  for k
  proof -
    have *:  $\text{dist } (u \ k \ (\text{from\_nat } n)) \ (x \ (\text{from\_nat } n)) \leq e/4$  if  $n \leq N$  for n
    proof -
      have  $K \ (\text{from\_nat } n) \leq L$ 
        unfolding L_def apply (rule Max_ge) using  $\langle n \leq N \rangle$  by auto
      then have  $k \geq K \ (\text{from\_nat } n)$  using  $\langle k \geq L \rangle$  by auto
      then show ?thesis using K less_imp_le by auto
    qed
  have **:  $\text{Max } \{\text{dist } (u \ k \ (\text{from\_nat } n)) \ (x \ (\text{from\_nat } n)) \mid n. n \leq N\} \leq e/4$ 
    apply (rule Max.boundedI) using * by auto
  have  $\text{dist } (u \ k) \ x \leq 2 * \text{Max } \{\text{dist } (u \ k \ (\text{from\_nat } n)) \ (x \ (\text{from\_nat } n)) \mid n. n \leq N\} + (1/2)^N$ 
    using dist_fun_le_dist_first_terms by auto
  also have  $\dots \leq 2 * e/4 + e/4$ 
    apply (rule add_mono)
    using **  $\langle 2^N > 4/e \rangle$  less_imp_le by (auto simp add: field_split_simps)
  also have  $\dots < e$  by auto
  finally show ?thesis by simp
qed
then show  $\exists L. \forall k \geq L. \text{dist } (u \ k) \ x < e$  by blast
qed
then show convergent u using convergent_def by blast
qed

instance fun :: (countable, polish_space) polish_space
  by standard

end
theory Analysis
imports

  Convex
  Determinants

  Connected
  Abstract_Limits

  Elementary_Normed_Spaces
  Norm_Arith

```

Convex_Euclidean_Space
Operator_Norm

Line_Segment
Derivative
Cartesian_Euclidean_Space
Weierstrass_Theorems

Ball_Volume
Integral_Test
Improper_Integral
Equivalence_Measurable_On_Borel
Lebesgue_Integral_Substitution
Embed_Measure
Complete_Measure
Radon_Nikodym
Fashoda_Theorem
Cross3
Homeomorphism
Bounded_Continuous_Function
Abstract_Topology
Product_Topology
Lindelof_Spaces
Infinite_Products
Infinite_Sum
Infinite_Set_Sum
Polytope
Jordan_Curve
Poly_Roots
Generalised_Binomial_Theorem
Gamma_Function
Change_Of_Vars
Multivariate_Analysis
Simplex_Content
FPS_Convergence
Smooth_Paths
Abstract_Euclidean_Space
Function_Metric

begin

end

Bibliography

- [1]
- [2] J. B. Conway. *A course in functional analysis*, volume 96. Springer Science & Business Media, 2013.
- [3] J. Dugundji. An extension of Tietze’s theorem. *Pacific J. Math.*, 1(3):353–367, 1951.
- [4] M. Maggesi. A formalization of metric spaces in HOL light. *J. Autom. Reasoning*, 60(2):237–254, 2018.