Ordinals and cardinals in HOL

Andrei Popescu & Dmitriy Traytel

Contents

Abstract

We develop a basic theory of ordinals and cardinals in Isabelle/HOL, up to the point where some cardinality facts relevant for the "working mathematician" become available. Unlike in set theory, here we do not have at hand canonical notions of ordinal and cardinal. Therefore, here an ordinal is merely a well-order relation and a cardinal is an ordinal minim w.r.t. order embedding on its field.

1 Introduction

In set theory (under formalizations such as Zermelo-Fraenkel or Von Neumann-Bernays-Gödel), an *ordinal* is a special kind of well-order, namely one whose strict version is the restriction of the membership relation to a set. In particular, the field of a set-theoretic ordinal is a transitive set, and the non-strict version of an ordinal relation is set inclusion. Set-theoretic ordinals enjoy the nice properties of membership on transitive sets, while at the same time forming a complete class of representatives for well-orders (since any well-order turns out isomorphic to an ordinal). Moreover, the class of ordinals is itself transitive and well-ordered by membership as the strict relation and inclusion as the non-strict relation. Also knowing that any set can be well-ordered (in the presence of the axiom of choice), one then defines the *cardinal* of a set to be the smallest ordinal isomorphic to a well-order on that set. This makes the class of cardinals a complete set of representatives for the intuitive notion of set cardinality.[1] The ability to produce *canonical well-orders* from the membership relation (having the aforementioned convenient properties) allows for a harmonious development of the theory of cardinals in set-theoretic settings. Non-trivial cardinality results, such as A being equipollent to $A \times A$ for any infinite A, follow rather quickly within this theory.

However, a canonical notion of well-order is *not* available in HOL. Here, one has to do with well-order "as is", but otherwise has all the necessary infrastructure (including Hilbert choice) to "climb" well-orders recursively and to well-oder arbitrary sets.

The current work, formalized in Isabelle/HOL, develops the basic theory of ordinals and cardinals up to the point where there are inferred a collection of non-trivial cardinality facts useful for the "working mathematician", among which:

[1] The "intuitive" cardinality of a set A is the class of all sets equipollent to A, i.e., being in bijection with A.

- Given any two sets (on any two given types)[2], one is injectable in the other.
- If at least one of two sets is infinite, then their sum and their Cartesian product are equipollent to the larger of the two.
- The set of lists (and also the set of finite sets) with element from an infinite set is equipollent with that set.

Our development emulates the standard one from set-theory, but keeps everything *up to order isomorphism.* An (HOL) ordinal is merely a well-order. An *ordinal embedding* is an injective and order-compatible function which maps its source into an initial segment (i.e., order filter) of its target. Now, a *cardinal* (called in this work a *cardinal order*) is an ordinal minim w.r.t. the existence of embeddings among all well-orders on its field. After showing the existence of cardinals on any given set, we define the cardinal of a set A, denoted $|A|$, to be *some* cardinal order on A. This concept is unique only up to order isomorphism (denoted $=o$), but meets its purpose: any two sets A and B (laying at potentially distinct types) are in bijection if and only if $|A| =o |B|$. Moreover, we also show that numeric cardinals assigned to finite sets[3] are *conservatively extended* by our general (order-theoretic) notion of cardinal. We study the interaction of cardinals with standard set-theoretic constructions such as powersets, products, sums and lists. These constructions are shown to preserve the "cardinal identity" $=o$ and also to be monotonic w.r.t. $\leq o$, the ordinal embedding relation. By studying the interaction between these constructions, infinite sets and cardinals, we obtain the aforementioned results for "working mathematicians".

For this development, we did not follow closely any particular textbook, and in fact are not aware of such basic theory of cardinals previously developed in HOL.[4] On the other hand, the set-theoretic versions of the facts proved here are folklore in set theory, and can be found, e.g., in the textbook [1]. Beyond taking care of some locality aspects concerning the spreading of our concepts throughout types, we have not departed much from the techniques used in set theory for establishing these facts – for instance, in the proof of one of our major theorems, *Card-order-Times-same-infinite* from Section 8.4,[5] we have essentially applied the technique described, e.g., in the proof of theorem 1.5.11 from [1], page 47.

Here is the structure of the rest of this document.

[2]Recall that, in HOL, a set on a type α is modeled, just like a predicate, as a function from α to bool.

[3]Numeric cardinals of finite sets are already formalized in Isabelle/HOL.

[4]After writing this formalization, we became aware of Paul Taylor's membership-free development of the theory of ordinals [2].

[5]This theorem states that, for any infinite cardinal r on a set A, $|A \times A|$ is not larger than r.

The next three sections, 2-4, develop some mathematical prerequisites. In Section 2, a large collection of simple facts about injections, bijections, inverses, (in)finite sets and numeric cardinals are proved, making life easier for later, when proving less trivial facts. Section 3 introduces upper and lower bounds operators for order-like relations and studies their basic properties. Section 4 states some useful variations of well-founded recursion and induction principles.

Then come the major sections, 5-8. Section 5 defines and studies, in the context of a well-order relation, the notions of minimum (of a set), maximum (of two elements), supremum, successor (of a set), and order filter (i.e., initial segment, i.e., downward-closed set). Section 6 defines and studies (well-order) embeddings, strict embeddings, isomorphisms, and compatible functions. Section 7 deals with various constructions on well-orders, and with the relations $\leq o$, $<o$ and $=o$ of well-order embedding, strict embedding, and isomorphism, respectively. Section 8 defines and studies cardinal order relations, the cardinal of a set, the connection of cardinals with set-theoretic constructs, the canonical cardinal of natural numbers and finite cardinals, the successor of a cardinal, as well as regular cardinals. (The latter play a crucial role in the development of a new (co)datatype package in HOL.)

Finally, section 9 provides an abstraction of the previous developments on cardinals, to provide a simpler user interface to cardinals, which in most of the cases allows to forget that cardinals are represented by orders and use them through defined arithmetic operators.

More informal details are provided at the beginning of each section, and also at the beginning of some of the subsections.

References

[1] M. Holz, K. Steffens, and E. Weitz. *Introduction to Cardinal Arithmetic*. Birkhäuser, 1999.

[2] Paul Taylor. Intuitionistic sets and ordinals. *J. Symb. Log.*, 61(3):705–744, 1996.

2 More on Injections, Bijections and Inverses

theory *Fun-More*
imports *Main*
begin

2.1 Purely functional properties

lemma *notIn-Un-bij-betw2*:
assumes *NIN*: $b \notin A$ **and** *NIN'*: $b' \notin A'$ **and**

BIJ: *bij-betw f A A'*
shows *bij-betw f* $(A \cup \{b\})$ $(A' \cup \{b'\})$ $= (f\ b = b')$
proof
assume $f\ b = b'$
thus *bij-betw f* $(A \cup \{b\})$ $(A' \cup \{b'\})$
using *assms notIn-Un-bij-betw*[*of b A f A'*] **by** *auto*
next
assume $*$: *bij-betw f* $(A \cup \{b\})$ $(A' \cup \{b'\})$
hence $f\ b \in A' \cup \{b'\}$
unfolding *bij-betw-def* **by** *auto*
moreover
{**assume** $f\ b \in A'$
then obtain *b1* **where** *1*: $b1 \in A$ **and** *2*: $f\ b1 = f\ b$ **using** *BIJ*
by (*auto simp add*: *bij-betw-def*)
hence $b = b1$ **using** $*$
by (*auto simp add*: *bij-betw-def inj-on-def*)
with *1 NIN* **have** *False* **by** *auto*
}
ultimately show $f\ b = b'$ **by** *blast*
qed

lemma *bij-betw-ball*:
assumes *BIJ*: *bij-betw f A B*
shows $(\forall\, b \in B.\ phi\ b) = (\forall\, a \in A.\ phi(f\ a))$
using *assms* **unfolding** *bij-betw-def inj-on-def* **by** *blast*

lemma *bij-betw-diff-singl*:
assumes *BIJ*: *bij-betw f A A'* **and** *IN*: $a \in A$
shows *bij-betw f* $(A - \{a\})$ $(A' - \{f\ a\})$
proof−
let $?B = A - \{a\}$ **let** $?B' = A' - \{f\ a\}$
have $f\ a \in A'$ **using** *IN BIJ* **unfolding** *bij-betw-def* **by** *blast*
hence $a \notin ?B \wedge f\ a \notin ?B' \wedge A = ?B \cup \{a\} \wedge A' = ?B' \cup \{f\ a\}$
using *IN* **by** *blast*
thus *?thesis* **using** *notIn-Un-bij-betw3*[*of a ?B f ?B'*] *BIJ* **by** *simp*
qed

2.2 Properties involving finite and infinite sets

lemma *bij-betw-inv-into-RIGHT*:
assumes *BIJ*: *bij-betw f A A'* **and** *SUB*: $B' \leq A'$
shows $f\ `((inv\text{-}into\ A\ f)\ `B') = B'$
using *assms*
proof(*auto simp add*: *bij-betw-inv-into-right*)
let $?f' = (inv\text{-}into\ A\ f)$
fix a' **assume** $*$: $a' \in B'$
hence $a' \in A'$ **using** *SUB* **by** *auto*

hence $a' = f\ (?f'\ a')$
using *BIJ* **by** (*auto simp add*: *bij-betw-inv-into-right*)
thus $a' \in f\ `\ (?f'\ `\ B')$ **using** $*$ **by** *blast*
qed

lemma *bij-betw-inv-into-RIGHT-LEFT*:
assumes *BIJ*: *bij-betw f A A'* **and** *SUB*: $B' \leq A'$ **and**
IM: $(inv\text{-}into\ A\ f)\ `\ B' = B$
shows $f\ `\ B = B'$
proof−
have $f`((inv\text{-}into\ A\ f)`\ B') = B'$
using *assms bij-betw-inv-into-RIGHT*[*of f A A' B'*] **by** *auto*
thus *?thesis* **using** *IM* **by** *auto*
qed

lemma *bij-betw-inv-into-twice*:
assumes *bij-betw f A A'*
shows $\forall\, a \in A.\ inv\text{-}into\ A'\ (inv\text{-}into\ A\ f)\ a = f\ a$
proof
let $?f' = inv\text{-}into\ A\ f$ **let** $?f'' = inv\text{-}into\ A'\ ?f'$
have *1*: *bij-betw ?f' A' A* **using** *assms*
by (*auto simp add*: *bij-betw-inv-into*)
fix *a* **assume** $*$: $a \in A$
then obtain a' **where** *2*: $a' \in A'$ **and** *3*: $?f'\ a' = a$
using *1* **unfolding** *bij-betw-def* **by** *force*
hence $?f''\ a = a'$
using $*$ *1 3* **by** (*auto simp add*: *bij-betw-inv-into-left*)
moreover have $f\ a = a'$ **using** *assms 2 3*
by (*auto simp add*: *bij-betw-inv-into-right*)
ultimately show $?f''\ a = f\ a$ **by** *simp*
qed

2.3 Properties involving Hilbert choice

lemma *bij-betw-inv-into-LEFT*:
assumes *BIJ*: *bij-betw f A A'* **and** *SUB*: $B \leq A$
shows $(inv\text{-}into\ A\ f)`(f\ `\ B) = B$
using *assms* **unfolding** *bij-betw-def* **using** *inv-into-image-cancel* **by** *force*

lemma *bij-betw-inv-into-LEFT-RIGHT*:
assumes *BIJ*: *bij-betw f A A'* **and** *SUB*: $B \leq A$ **and**
IM: $f\ `\ B = B'$
shows $(inv\text{-}into\ A\ f)\ `\ B' = B$
using *assms bij-betw-inv-into-LEFT*[*of f A A' B*] **by** *fast*

2.4 Other facts

lemma *atLeastLessThan-injective*:

assumes $\{0 \;..< m{::}nat\} = \{0 \;..< n\}$
shows $m = n$
proof−
{**assume** $m < n$
hence $m \in \{0 \;..< n\}$ **by** *auto*
hence $\{0 \;..< m\} < \{0 \;..< n\}$ **by** *auto*
hence *False* **using** *assms* **by** *blast*
}
moreover
{**assume** $n < m$
hence $n \in \{0 \;..< m\}$ **by** *auto*
hence $\{0 \;..< n\} < \{0 \;..< m\}$ **by** *auto*
hence *False* **using** *assms* **by** *blast*
}
ultimately show *?thesis* **by** *force*
qed

lemma *atLeastLessThan-injective2*:
bij-betw f $\{0 \;..< m{::}nat\}$ $\{0 \;..< n\} \Longrightarrow m = n$
using *finite-atLeastLessThan*[*of m*] *finite-atLeastLessThan*[*of n*]
card-atLeastLessThan[*of m*] *card-atLeastLessThan*[*of n*]
bij-betw-iff-card[*of* $\{0 \;..< m\}$ $\{0 \;..< n\}$] **by** *auto*

lemma *atLeastLessThan-less-eq*:
$(\{0..<m\} \leq \{0..<n\}) = ((m{::}nat) \leq n)$
unfolding *ivl-subset* **by** *arith*

lemma *atLeastLessThan-less-eq2*:
assumes *inj-on* f $\{0..<(m{::}nat)\} \wedge f$ ‘ $\{0..<m\} \leq \{0..<n\}$
shows $m \leq n$
using *assms*
using *finite-atLeastLessThan*[*of m*] *finite-atLeastLessThan*[*of n*]
card-atLeastLessThan[*of m*] *card-atLeastLessThan*[*of n*]
card-inj-on-le[*of f* $\{0 \;..< m\}$ $\{0 \;..< n\}$] **by** *fastforce*

lemma *atLeastLessThan-less-eq3*:
$(\exists f.$ *inj-on* f $\{0..<(m{::}nat)\} \wedge f$ ‘ $\{0..<m\} \leq \{0..<n\}) = (m \leq n)$
using *atLeastLessThan-less-eq2*
proof(*auto*)
assume $m \leq n$
hence *inj-on id* $\{0..<m\} \wedge id$ ‘ $\{0..<m\} \subseteq \{0..<n\}$ **unfolding** *inj-on-def* **by** *force*
thus $\exists f.$ *inj-on* f $\{0..<m\} \wedge f$ ‘ $\{0..<m\} \subseteq \{0..<n\}$ **by** *blast*
qed

lemma *atLeastLessThan-less*:
$(\{0..<m\} < \{0..<n\}) = ((m{::}nat) < n)$

proof −
  have $(\{0..<m\} < \{0..<n\}) = (\{0..<m\} \leq \{0..<n\} \wedge \{0..<m\} \;\tilde{}{=}\; \{0..<n\})$
  using *subset-iff-psubset-eq* **by** *blast*
  also have $\ldots = (m \leq n \wedge m \;\tilde{}{=}\; n)$
  using *atLeastLessThan-less-eq atLeastLessThan-injective* **by** *blast*
  also have $\ldots = (m < n)$ **by** *auto*
  finally show *?thesis* **.**
qed

end

3 Basics on Order-Like Relations

theory *Order-Relation-More*
imports *Main*
begin

3.1 The upper and lower bounds operators

lemma *aboveS-subset-above*: *aboveS r a* $\leq$ *above r a*
by(*auto simp add*: *aboveS-def above-def*)

lemma *AboveS-subset-Above*: *AboveS r A* $\leq$ *Above r A*
by(*auto simp add*: *AboveS-def Above-def*)

lemma *UnderS-disjoint*: *A Int* (*UnderS r A*) $= \{\}$
by(*auto simp add*: *UnderS-def*)

lemma *aboveS-notIn*: $a \notin$ *aboveS r a*
by(*auto simp add*: *aboveS-def*)

lemma *Refl-above-in*: $[\![$*Refl r*; $a \in$ *Field r*$]\!] \Longrightarrow a \in$ *above r a*
by(*auto simp add*: *refl-on-def above-def*)

lemma *in-Above-under*: $a \in$ *Field r* $\Longrightarrow a \in$ *Above r* (*under r a*)
by(*auto simp add*: *Above-def under-def*)

lemma *in-Under-above*: $a \in$ *Field r* $\Longrightarrow a \in$ *Under r* (*above r a*)
by(*auto simp add*: *Under-def above-def*)

lemma *in-UnderS-aboveS*: $a \in$ *Field r* $\Longrightarrow a \in$ *UnderS r* (*aboveS r a*)
by(*auto simp add*: *UnderS-def aboveS-def*)

lemma *UnderS-subset-Under*: *UnderS r A* $\leq$ *Under r A*
by(*auto simp add*: *UnderS-def Under-def*)

lemma *subset-Above-Under*: $B \leq$ *Field r* $\Longrightarrow B \leq$ *Above r* (*Under r B*)
by(*auto simp add*: *Above-def Under-def*)

lemma *subset-Under-Above*: $B \leq$ *Field* $r \Longrightarrow B \leq$ *Under* r (*Above* r B)
by(*auto simp add*: *Under-def Above-def*)

lemma *subset-AboveS-UnderS*: $B \leq$ *Field* $r \Longrightarrow B \leq$ *AboveS* r (*UnderS* r B)
by(*auto simp add*: *AboveS-def UnderS-def*)

lemma *subset-UnderS-AboveS*: $B \leq$ *Field* $r \Longrightarrow B \leq$ *UnderS* r (*AboveS* r B)
by(*auto simp add*: *UnderS-def AboveS-def*)

lemma *Under-Above-Galois*:
$[\![B \leq$ *Field* r; $C \leq$ *Field* $r]\!] \Longrightarrow (B \leq$ *Above* r $C) = (C \leq$ *Under* r $B)$
by(*unfold Above-def Under-def*, *blast*)

lemma *UnderS-AboveS-Galois*:
$[\![B \leq$ *Field* r; $C \leq$ *Field* $r]\!] \Longrightarrow (B \leq$ *AboveS* r $C) = (C \leq$ *UnderS* r $B)$
by(*unfold AboveS-def UnderS-def*, *blast*)

lemma *Refl-above-aboveS*:
  assumes *REFL*: *Refl* r **and** *IN*: $a \in$ *Field* r
  shows *above* r a = *aboveS* r $a \cup \{a\}$
proof(*unfold above-def aboveS-def*, *auto*)
  show $(a,a) \in r$ **using** *REFL IN refl-on-def*[*of - r*] **by** *blast*
qed

lemma *Linear-order-under-aboveS-Field*:
  assumes *LIN*: *Linear-order* r **and** *IN*: $a \in$ *Field* r
  shows *Field* r = *under* r $a \cup$ *aboveS* r a
proof(*unfold under-def aboveS-def*, *auto*)
  assume $a \in$ *Field* r $(a, a) \notin r$
  with *LIN IN order-on-defs*[*of - r*] *refl-on-def*[*of - r*]
  show *False* **by** *auto*
next
  fix b **assume** $b \in$ *Field* r $(b, a) \notin r$
  with *LIN IN order-on-defs*[*of Field r r*] *total-on-def*[*of Field r r*]
  have $(a,b) \in r \lor a = b$ **by** *blast*
  thus $(a,b) \in r$
  using *LIN IN order-on-defs*[*of - r*] *refl-on-def*[*of - r*] **by** *auto*
next
  fix b **assume** $(b, a) \in r$
  thus $b \in$ *Field* r
  using *LIN order-on-defs*[*of - r*] *refl-on-def*[*of - r*] **by** *blast*
next
  fix b **assume** $b \neq a$ $(a, b) \in r$
  thus $b \in$ *Field* r
  using *LIN order-on-defs*[*of Field r r*] *refl-on-def*[*of Field r r*] **by** *blast*
qed

lemma *Linear-order-underS-above-Field*:
assumes *LIN*: *Linear-order* r **and** *IN*: $a \in$ *Field* r

shows *Field r* = *underS r a* ∪ *above r a*
proof(*unfold underS-def above-def*, *auto*)
  assume $a \in$ *Field r* $(a, a) \notin r$
  with *LIN IN order-on-defs*[*of - r*] *refl-on-def*[*of - r*]
  show *False* **by** *metis*
next
  fix *b* **assume** $b \in$ *Field r* $(a, b) \notin r$
  with *LIN IN order-on-defs*[*of Field r r*] *total-on-def*[*of Field r r*]
  have $(b,a) \in r \lor b = a$ **by** *blast*
  thus $(b,a) \in r$
  using *LIN IN order-on-defs*[*of - r*] *refl-on-def*[*of - r*] **by** *auto*
next
  fix *b* **assume** $b \neq a$ $(b, a) \in r$
  thus $b \in$ *Field r*
  using *LIN order-on-defs*[*of - r*] *refl-on-def*[*of - r*] **by** *blast*
next
  fix *b* **assume** $(a, b) \in r$
  thus $b \in$ *Field r*
  using *LIN order-on-defs*[*of Field r r*] *refl-on-def*[*of Field r r*] **by** *blast*
qed

lemma *under-empty*: $a \notin$ *Field r* $\Longrightarrow$ *under r a* = {}
unfolding *Field-def under-def* **by** *auto*

lemma *Under-Field*: *Under r A* ≤ *Field r*
by(*unfold Under-def Field-def*, *auto*)

lemma *UnderS-Field*: *UnderS r A* ≤ *Field r*
by(*unfold UnderS-def Field-def*, *auto*)

lemma *above-Field*: *above r a* ≤ *Field r*
by(*unfold above-def Field-def*, *auto*)

lemma *aboveS-Field*: *aboveS r a* ≤ *Field r*
by(*unfold aboveS-def Field-def*, *auto*)

lemma *Above-Field*: *Above r A* ≤ *Field r*
by(*unfold Above-def Field-def*, *auto*)

lemma *Refl-under-Under*:
assumes *REFL*: *Refl r* **and** *NE*: $A \neq \{\}$
shows *Under r A* = $(\bigcap a \in A.$ *under r a*$)$
proof
  show *Under r A* $\subseteq (\bigcap a \in A.$ *under r a*$)$
  by(*unfold Under-def under-def*, *auto*)
next
  show $(\bigcap a \in A.$ *under r a*$) \subseteq$ *Under r A*
  proof(*auto*)
    fix *x*

assume $*$: $\forall$ *xa* $\in$ *A*. *x* $\in$ *under r xa*
hence $\forall$ *xa* $\in$ *A*. (*x*,*xa*) $\in$ *r*
by (*simp add*: *under-def*)
moreover
{from *NE* **obtain** *a* **where** *a* $\in$ *A* **by** *blast*
with $*$ **have** *x* $\in$ *under r a* **by** *simp*
hence *x* $\in$ *Field r*
using *under-Field*[*of r a*] **by** *auto*
}
ultimately show *x* $\in$ *Under r A*
unfolding *Under-def* **by** *auto*
qed
qed

lemma *Refl-underS-UnderS*:
assumes *REFL*: *Refl r* **and** *NE*: *A* $\neq$ {}
shows *UnderS r A* = ($\bigcap$ *a* $\in$ *A*. *underS r a*)
proof
show *UnderS r A* $\subseteq$ ($\bigcap$ *a* $\in$ *A*. *underS r a*)
by(*unfold UnderS-def underS-def*, *auto*)
next
show ($\bigcap$ *a* $\in$ *A*. *underS r a*) $\subseteq$ *UnderS r A*
proof(*auto*)
fix *x*
assume $*$: $\forall$ *xa* $\in$ *A*. *x* $\in$ *underS r xa*
hence $\forall$ *xa* $\in$ *A*. *x* $\neq$ *xa* $\wedge$ (*x*,*xa*) $\in$ *r*
by (*auto simp add*: *underS-def*)
moreover
{from *NE* **obtain** *a* **where** *a* $\in$ *A* **by** *blast*
with $*$ **have** *x* $\in$ *underS r a* **by** *simp*
hence *x* $\in$ *Field r*
using *underS-Field*[*of - r a*] **by** *auto*
}
ultimately show *x* $\in$ *UnderS r A*
unfolding *UnderS-def* **by** *auto*
qed
qed

lemma *Refl-above-Above*:
assumes *REFL*: *Refl r* **and** *NE*: *A* $\neq$ {}
shows *Above r A* = ($\bigcap$ *a* $\in$ *A*. *above r a*)
proof
show *Above r A* $\subseteq$ ($\bigcap$ *a* $\in$ *A*. *above r a*)
by(*unfold Above-def above-def*, *auto*)
next
show ($\bigcap$ *a* $\in$ *A*. *above r a*) $\subseteq$ *Above r A*
proof(*auto*)
fix *x*
assume $*$: $\forall$ *xa* $\in$ *A*. *x* $\in$ *above r xa*

hence $\forall\, xa \in A.\ (xa,x) \in r$
by (*simp add*: *above-def*)
moreover
{from *NE* **obtain** a **where** $a \in A$ **by** *blast*
with $*$ **have** $x \in$ *above* r a **by** *simp*
hence $x \in$ *Field* r
using *above-Field*[*of* r a] **by** *auto*
}
ultimately show $x \in$ *Above* r A
unfolding *Above-def* **by** *auto*
qed
qed

lemma *Refl-aboveS-AboveS*:
assumes *REFL*: *Refl* r **and** *NE*: $A \neq \{\}$
shows *AboveS* r A = $(\bigcap a \in A.$ *aboveS* r $a)$
proof
show *AboveS* r $A \subseteq (\bigcap a \in A.$ *aboveS* r $a)$
by(*unfold AboveS-def aboveS-def*, *auto*)
next
show $(\bigcap a \in A.$ *aboveS* r $a) \subseteq$ *AboveS* r A
proof(*auto*)
fix x
assume $*$: $\forall\, xa \in A.\ x \in$ *aboveS* r xa
hence $\forall\, xa \in A.\ xa \neq x \wedge (xa,x) \in r$
by (*auto simp add*: *aboveS-def*)
moreover
{from *NE* **obtain** a **where** $a \in A$ **by** *blast*
with $*$ **have** $x \in$ *aboveS* r a **by** *simp*
hence $x \in$ *Field* r
using *aboveS-Field*[*of* r a] **by** *auto*
}
ultimately show $x \in$ *AboveS* r A
unfolding *AboveS-def* **by** *auto*
qed
qed

lemma *under-Under-singl*: *under* r a = *Under* r $\{a\}$
by(*unfold Under-def under-def*, *auto simp add*: *Field-def*)

lemma *underS-UnderS-singl*: *underS* r a = *UnderS* r $\{a\}$
by(*unfold UnderS-def underS-def*, *auto simp add*: *Field-def*)

lemma *above-Above-singl*: *above* r a = *Above* r $\{a\}$
by(*unfold Above-def above-def*, *auto simp add*: *Field-def*)

lemma *aboveS-AboveS-singl*: *aboveS* r a = *AboveS* r $\{a\}$
by(*unfold AboveS-def aboveS-def*, *auto simp add*: *Field-def*)

lemma *Under-decr*: $A \leq B \Longrightarrow$ *Under r B* $\leq$ *Under r A*
by(*unfold Under-def*, *auto*)

lemma *UnderS-decr*: $A \leq B \Longrightarrow$ *UnderS r B* $\leq$ *UnderS r A*
by(*unfold UnderS-def*, *auto*)

lemma *Above-decr*: $A \leq B \Longrightarrow$ *Above r B* $\leq$ *Above r A*
by(*unfold Above-def*, *auto*)

lemma *AboveS-decr*: $A \leq B \Longrightarrow$ *AboveS r B* $\leq$ *AboveS r A*
by(*unfold AboveS-def*, *auto*)

lemma *under-incl-iff*:
assumes *TRANS*: *trans r* **and** *REFL*: *Refl r* **and** *IN*: $a \in$ *Field r*
shows (*under r a* $\leq$ *under r b*) = ((a,b) $\in$ r)
proof
  assume (a,b) $\in$ r
  thus *under r a* $\leq$ *under r b* **using** *TRANS*
  by (*auto simp add*: *under-incr*)
next
  assume *under r a* $\leq$ *under r b*
  moreover
  have $a \in$ *under r a* **using** *REFL IN*
  by (*auto simp add*: *Refl-under-in*)
  ultimately show (a,b) $\in$ r
  by (*auto simp add*: *under-def*)
qed

lemma *above-decr*:
assumes *TRANS*: *trans r* **and** *REL*: (a,b) $\in$ r
shows *above r b* $\leq$ *above r a*
proof(*unfold above-def*, *auto*)
  fix x **assume** (b,x) $\in$ r
  with *REL TRANS trans-def*[*of r*]
  show (a,x) $\in$ r **by** *blast*
qed

lemma *aboveS-decr*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
    REL: (a,b) $\in$ r
shows *aboveS r b* $\leq$ *aboveS r a*
proof(*unfold aboveS-def*, *auto*)
  assume $*$: $a \neq b$ **and** $**$: (b,a) $\in$ r
  with *ANTISYM antisym-def*[*of r*] *REL*
  show *False* **by** *auto*
next
  fix x **assume** $x \neq b$ (b,x) $\in$ r
  with *REL TRANS trans-def*[*of r*]
  show (a,x) $\in$ r **by** *blast*

qed

lemma *under-trans*:
assumes *TRANS*: *trans r* **and**
IN1: $a \in$ *under r b* **and** *IN2*: $b \in$ *under r c*
shows $a \in$ *under r c*
proof −
have $(a,b) \in r \wedge (b,c) \in r$
using *IN1 IN2 under-def* **by** *fastforce*
hence $(a,c) \in r$
using *TRANS trans-def*[*of r*] **by** *blast*
thus *?thesis* **unfolding** *under-def* **by** *simp*
qed

lemma *under-underS-trans*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
IN1: $a \in$ *under r b* **and** *IN2*: $b \in$ *underS r c*
shows $a \in$ *underS r c*
proof −
have *0*: $(a,b) \in r \wedge (b,c) \in r$
using *IN1 IN2 under-def underS-def* **by** *fastforce*
hence *1*: $(a,c) \in r$
using *TRANS trans-def*[*of r*] **by** *blast*
have *2*: $b \neq c$ **using** *IN2 underS-def* **by** *force*
have *3*: $a \neq c$
proof
assume $a = c$ **with** *0 2 ANTISYM antisym-def*[*of r*]
show *False* **by** *auto*
qed
from *1 3* **show** *?thesis* **unfolding** *underS-def* **by** *simp*
qed

lemma *underS-under-trans*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
IN1: $a \in$ *underS r b* **and** *IN2*: $b \in$ *under r c*
shows $a \in$ *underS r c*
proof −
have *0*: $(a,b) \in r \wedge (b,c) \in r$
using *IN1 IN2 under-def underS-def* **by** *fast*
hence *1*: $(a,c) \in r$
using *TRANS trans-def*[*of r*] **by** *fast*
have *2*: $a \neq b$ **using** *IN1 underS-def* **by** *force*
have *3*: $a \neq c$
proof
assume $a = c$ **with** *0 2 ANTISYM antisym-def*[*of r*]
show *False* **by** *auto*
qed
from *1 3* **show** *?thesis* **unfolding** *underS-def* **by** *simp*
qed

```
lemma underS-underS-trans:
assumes TRANS: trans r and ANTISYM: antisym r and
        IN1: a ∈ underS r b and IN2: b ∈ underS r c
shows a ∈ underS r c
proof−
  have a ∈ under r b
  using IN1 underS-subset-under by fast
  with assms under-underS-trans show ?thesis by fast
qed

lemma above-trans:
assumes TRANS: trans r and
        IN1: b ∈ above r a and IN2: c ∈ above r b
shows c ∈ above r a
proof−
  have (a,b) ∈ r ∧ (b,c) ∈ r
  using IN1 IN2 above-def by fast
  hence (a,c) ∈ r
  using TRANS trans-def[of r] by blast
  thus ?thesis unfolding above-def by simp
qed

lemma above-aboveS-trans:
assumes TRANS: trans r and ANTISYM: antisym r and
        IN1: b ∈ above r a and IN2: c ∈ aboveS r b
shows c ∈ aboveS r a
proof−
  have 0: (a,b) ∈ r ∧ (b,c) ∈ r
  using IN1 IN2 above-def aboveS-def by fast
  hence 1: (a,c) ∈ r
  using TRANS trans-def[of r] by blast
  have 2: b ≠ c using IN2 aboveS-def by force
  have 3: a ≠ c
  proof
    assume a = c with 0 2 ANTISYM antisym-def[of r]
    show False by auto
  qed
  from 1 3 show ?thesis unfolding aboveS-def by simp
qed

lemma aboveS-above-trans:
assumes TRANS: trans r and ANTISYM: antisym r and
        IN1: b ∈ aboveS r a and IN2: c ∈ above r b
shows c ∈ aboveS r a
proof−
  have 0: (a,b) ∈ r ∧ (b,c) ∈ r
  using IN1 IN2 above-def aboveS-def by fast
  hence 1: (a,c) ∈ r
```

```
  using TRANS trans-def[of r] by blast
  have 2: a ≠ b using IN1 aboveS-def by force
  have 3: a ≠ c
  proof
    assume a = c with 0 2 ANTISYM antisym-def[of r]
    show False by auto
  qed
  from 1 3 show ?thesis unfolding aboveS-def by simp
qed

lemma aboveS-aboveS-trans:
assumes TRANS: trans r and ANTISYM: antisym r and
        IN1: b ∈ aboveS r a and IN2: c ∈ aboveS r b
shows c ∈ aboveS r a
proof−
  have b ∈ above r a
  using IN1 aboveS-subset-above by fast
  with assms above-aboveS-trans show ?thesis by fast
qed

lemma under-Under-trans:
assumes TRANS: trans r and
        IN1: a ∈ under r b and IN2: b ∈ Under r C
shows a ∈ Under r C
proof−
  have b ∈ {u ∈ Field r. ∀ x. x ∈ C ⟶ (u, x) ∈ r}
    using IN2 Under-def by force
  hence (a,b) ∈ r ∧ (∀ c ∈ C. (b,c) ∈ r)
    using IN1 under-def by fast
  hence ∀ c ∈ C. (a,c) ∈ r
  using TRANS trans-def[of r] by blast
  moreover
  have a ∈ Field r using IN1 unfolding Field-def under-def by blast
  ultimately
  show ?thesis unfolding Under-def by blast
qed

lemma underS-Under-trans:
assumes TRANS: trans r and ANTISYM: antisym r and
        IN1: a ∈ underS r b and IN2: b ∈ Under r C
shows a ∈ UnderS r C
proof−
  from IN1 have a ∈ under r b
  using underS-subset-under[of r b] by fast
  with assms under-Under-trans
  have a ∈ Under r C by fast

  moreover
  have a ∉ C
```

```
  proof
    assume *: a ∈ C
    have 1: b ≠ a ∧ (a,b) ∈ r
    using IN1 underS-def[of r b] by auto
    have ∀ c ∈ C. (b,c) ∈ r
    using IN2 Under-def[of r C] by auto
    with * have (b,a) ∈ r by simp
    with 1 ANTISYM antisym-def[of r]
    show False by blast
  qed

  ultimately
  show ?thesis unfolding UnderS-def
  using Under-def by force
qed

lemma underS-UnderS-trans:
assumes TRANS: trans r and ANTISYM: antisym r and
        IN1: a ∈ underS r b and IN2: b ∈ UnderS r C
shows a ∈ UnderS r C
proof−
  from IN2 have b ∈ Under r C
  using UnderS-subset-Under[of r C] by blast
  with underS-Under-trans assms
  show ?thesis by force
qed

lemma above-Above-trans:
assumes TRANS: trans r and
        IN1: a ∈ above r b and IN2: b ∈ Above r C
shows a ∈ Above r C
proof−
  have (b,a) ∈ r ∧ (∀ c ∈ C. (c,b) ∈ r)
    using IN1[unfolded above-def] IN2[unfolded Above-def] by simp
  hence ∀ c ∈ C. (c,a) ∈ r
  using TRANS trans-def[of r] by blast
  moreover
  have a ∈ Field r using IN1[unfolded above-def] Field-def by fast
  ultimately
  show ?thesis unfolding Above-def by auto
qed

lemma aboveS-Above-trans:
assumes TRANS: trans r and ANTISYM: antisym r and
        IN1: a ∈ aboveS r b and IN2: b ∈ Above r C
shows a ∈ AboveS r C
proof−
  from IN1 have a ∈ above r b
  using aboveS-subset-above[of r b] by blast
```

```
  with assms above-Above-trans
  have a ∈ Above r C by fast

  moreover
  have a ∉ C
  proof
    assume ∗: a ∈ C
    have 1: b ≠ a ∧ (b,a) ∈ r
    using IN1 aboveS-def[of r b] by auto
    have ∀ c ∈ C. (c,b) ∈ r
    using IN2 Above-def[of r C] by auto
    with ∗ have (a,b) ∈ r by simp
    with 1 ANTISYM antisym-def[of r]
    show False by blast
  qed

  ultimately
  show ?thesis unfolding AboveS-def
  using Above-def by force
qed

lemma above-AboveS-trans:
assumes TRANS: trans r and ANTISYM: antisym r and
        IN1: a ∈ above r b and IN2: b ∈ AboveS r C
shows a ∈ AboveS r C
proof−
  from IN2 have b ∈ Above r C
  using AboveS-subset-Above[of r C] by blast
  with assms above-Above-trans
  have a ∈ Above r C by force

  moreover
  have a ∉ C
  proof
    assume ∗: a ∈ C
    have 1: (b,a) ∈ r
    using IN1 above-def[of r b] by auto
    have ∀ c ∈ C. b ≠ c ∧ (c,b) ∈ r
    using IN2 AboveS-def[of r C] by auto
    with ∗ have b ≠ a ∧ (a,b) ∈ r by simp
    with 1 ANTISYM antisym-def[of r]
    show False by blast
  qed

  ultimately
  show ?thesis unfolding AboveS-def
  using Above-def by force
qed
```

lemma *aboveS-AboveS-trans*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
IN1: $a \in$ *aboveS r b* **and** *IN2*: $b \in$ *AboveS r C*
shows $a \in$ *AboveS r C*
proof−
from *IN2* **have** $b \in$ *Above r C*
using *AboveS-subset-Above*[*of r C*] **by** *blast*
with *aboveS-Above-trans assms*
show *?thesis* **by** *force*
qed

lemma *under-UnderS-trans*:
assumes *TRANS*: *trans r* **and** *ANTISYM*: *antisym r* **and**
IN1: $a \in$ *under r b* **and** *IN2*: $b \in$ *UnderS r C*
shows $a \in$ *UnderS r C*
proof−
from *IN2* **have** $b \in$ *Under r C*
using *UnderS-subset-Under*[*of r C*] **by** *blast*
with *assms under-Under-trans*
have $a \in$ *Under r C* **by** *force*

moreover
have $a \notin C$
proof
assume $*$: $a \in C$
have *1*: $(a,b) \in r$
using *IN1 under-def*[*of r b*] **by** *auto*
have $\forall c \in C.\ b \neq c \wedge (b,c) \in r$
using *IN2 UnderS-def*[*of r C*] **by** *blast*
with $*$ **have** $b \neq a \wedge (b,a) \in r$ **by** *blast*
with *1 ANTISYM antisym-def*[*of r*]
show *False* **by** *blast*
qed

ultimately
show *?thesis* **unfolding** *UnderS-def Under-def* **by** *fast*
qed

3.2 Properties depending on more than one relation

lemma *under-incr2*:
$r \leq r' \Longrightarrow$ *under r a* $\leq$ *under r' a*
unfolding *under-def* **by** *blast*

lemma *underS-incr2*:
$r \leq r' \Longrightarrow$ *underS r a* $\leq$ *underS r' a*
unfolding *underS-def* **by** *blast*

lemma *above-incr2*:
$r \leq r' \Longrightarrow$ *above* r $a \leq$ *above* r' a
unfolding *above-def* **by** *blast*

lemma *aboveS-incr2*:
$r \leq r' \Longrightarrow$ *aboveS* r $a \leq$ *aboveS* r' a
unfolding *aboveS-def* **by** *blast*

end

4 More on Well-Founded Relations

theory *Wellfounded-More*
imports *Main Order-Relation-More*
begin

4.1 Well-founded recursion via genuine fixpoints

lemma *adm-wf-unique-fixpoint*:
fixes r :: $('a * 'a)$ *set* **and**
  H :: $('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$ **and**
  f :: $'a \Rightarrow 'b$ **and** g :: $'a \Rightarrow 'b$
assumes *WF*: *wf* r **and** *ADM*: *adm-wf* r H **and** *fFP*: $f = H\,f$ **and** *gFP*: $g = H\,g$
shows $f = g$
proof−
 {**fix** x
  have $f\,x = g\,x$
  proof(*rule wf-induct*[*of* r $(\lambda x.\ f\,x = g\,x)$],
    auto simp add: *WF*)
   fix x **assume** $\forall\, y.\ (y, x) \in r \longrightarrow f\,y = g\,y$
   hence $H\,f\,x = H\,g\,x$ **using** *ADM adm-wf-def*[*of* r H] **by** *auto*
   thus $f\,x = g\,x$ **using** *fFP* **and** *gFP* **by** *simp*
  qed
 }
 thus *?thesis* **by** (*simp add*: *ext*)
qed

lemma *wfrec-unique-fixpoint*:
fixes r :: $('a * 'a)$ *set* **and**
  H :: $('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$ **and**
  f :: $'a \Rightarrow 'b$
assumes *WF*: *wf* r **and** *ADM*: *adm-wf* r H **and**
   fp: $f = H\,f$
shows $f =$ *wfrec* r H
proof−
 have H (*wfrec* r H) $=$ *wfrec* r H

using *assms wfrec-fixpoint*[*of r H*] **by** *simp*
thus *?thesis*
using *assms adm-wf-unique-fixpoint*[*of r H wfrec r H*] **by** *simp*
qed

end

5 Well-Order Relations

theory *Wellorder-Relation*
imports *Wellfounded-More*
begin

context *wo-rel*
begin

5.1 Auxiliaries

lemma *PREORD*: *Preorder r*
using *WELL order-on-defs*[*of - r*] **by** *auto*

lemma *PARORD*: *Partial-order r*
using *WELL order-on-defs*[*of - r*] **by** *auto*

lemma *cases-Total2*:
$\bigwedge$ *phi a b*. $[\![\{a,b\} \leq$ *Field r*; $((a,b) \in r - Id \Longrightarrow$ *phi a b*);
$((b,a) \in r - Id \Longrightarrow$ *phi a b*); $(a = b \Longrightarrow$ *phi a b*)$]\!]$
$\Longrightarrow$ *phi a b*
using *TOTALS* **by** *auto*

5.2 Well-founded induction and recursion adapted to non-strict well-order relations

lemma *worec-unique-fixpoint*:
assumes *ADM*: *adm-wo H* **and** *fp*: $f = H f$
shows $f =$ *worec H*
proof−
have *adm-wf* $(r - Id)$ *H*
unfolding *adm-wf-def*
using *ADM adm-wo-def*[*of H*] *underS-def*[*of r*] **by** *auto*
hence $f =$ *wfrec* $(r - Id)$ *H*
using *fp WF wfrec-unique-fixpoint*[*of r − Id H*] **by** *simp*
thus *?thesis* **unfolding** *worec-def* **.**
qed

5.2.1 Properties of max2

lemma *max2-iff*:
assumes $a \in$ *Field r* **and** $b \in$ *Field r*

shows $((max2\ a\ b,\ c) \in r) = ((a,c) \in r \wedge (b,c) \in r)$
proof
  assume $(max2\ a\ b,\ c) \in r$
  thus $(a,c) \in r \wedge (b,c) \in r$
  using *assms max2-greater*[*of a b*] *TRANS trans-def*[*of r*] **by** *blast*
next
  assume $(a,c) \in r \wedge (b,c) \in r$
  thus $(max2\ a\ b,\ c) \in r$
  using *assms max2-among*[*of a b*] **by** *auto*
qed

5.2.2 Properties of minim

lemma *minim-Under*:
$[\![B \le Field\ r;\ B \neq \{\}]\!] \Longrightarrow minim\ B \in Under\ B$
by(*auto simp add*: *Under-def minim-inField minim-least*)

lemma *equals-minim-Under*:
$[\![B \le Field\ r;\ a \in B;\ a \in Under\ B]\!]$
$\Longrightarrow a = minim\ B$
by(*auto simp add*: *Under-def equals-minim*)

lemma *minim-iff-In-Under*:
assumes *SUB*: $B \le Field\ r$ **and** *NE*: $B \neq \{\}$
shows $(a = minim\ B) = (a \in B \wedge a \in Under\ B)$
proof
  assume $a = minim\ B$
  thus $a \in B \wedge a \in Under\ B$
  using *assms minim-in minim-Under* **by** *simp*
next
  assume $a \in B \wedge a \in Under\ B$
  thus $a = minim\ B$
  using *assms equals-minim-Under* **by** *simp*
qed

lemma *minim-Under-under*:
assumes *NE*: $A \neq \{\}$ **and** *SUB*: $A \le Field\ r$
shows $Under\ A = under\ (minim\ A)$
proof−

  have *1*: $minim\ A \in A$
  using *assms minim-in* **by** *auto*
  have *2*: $\forall x \in A.\ (minim\ A,\ x) \in r$
  using *assms minim-least* **by** *auto*

  have $Under\ A \le under\ (minim\ A)$
  proof
    fix x **assume** $x \in Under\ A$
    with *1 Under-def*[*of r*] **have** $(x,minim\ A) \in r$ **by** *auto*

```
    thus x ∈ under(minim A) unfolding under-def by simp
  qed

  moreover

  have under (minim A) ≤ Under A
  proof
    fix x assume x ∈ under(minim A)
    hence 11: (x,minim A) ∈ r unfolding under-def by simp
    hence x ∈ Field r unfolding Field-def by auto
    moreover
    {fix a assume a ∈ A
     with 2 have (minim A, a) ∈ r by simp
     with 11 have (x,a) ∈ r
     using TRANS trans-def[of r] by blast
    }
    ultimately show x ∈ Under A by (unfold Under-def, auto)
  qed

  ultimately show ?thesis by blast
qed

lemma minim-UnderS-underS:
assumes NE: A ≠ {} and SUB: A ≤ Field r
shows UnderS A = underS (minim A)
proof−

  have 1: minim A ∈ A
  using assms minim-in by auto
  have 2: ∀ x ∈ A. (minim A, x) ∈ r
  using assms minim-least by auto

  have UnderS A ≤ underS (minim A)
  proof
    fix x assume x ∈ UnderS A
    with 1 UnderS-def[of r] have x ≠ minim A ∧ (x,minim A) ∈ r by auto
    thus x ∈ underS(minim A) unfolding underS-def by simp
  qed

  moreover

  have underS (minim A) ≤ UnderS A
  proof
    fix x assume x ∈ underS(minim A)
    hence 11: x ≠ minim A ∧ (x,minim A) ∈ r unfolding underS-def by simp
    hence x ∈ Field r unfolding Field-def by auto
    moreover
    {fix a assume a ∈ A
     with 2 have 3: (minim A, a) ∈ r by simp
```

with *11* **have** $(x,a) \in r$
using *TRANS trans-def*[*of r*] **by** *blast*
moreover
have $x \neq a$
proof
assume $x = a$
with *11 3 ANTISYM antisym-def*[*of r*]
show *False* **by** *auto*
qed
ultimately
have $x \neq a \wedge (x,a) \in r$ **by** *simp*
}
ultimately show $x \in$ *UnderS A* **by** (*unfold UnderS-def*, *auto*)
qed

ultimately show *?thesis* **by** *blast*
qed

5.2.3 Properties of supr

lemma *supr-Above*:
assumes *SUB*: $B \leq$ *Field r* **and** *ABOVE*: *Above B* $\neq \{\}$
shows *supr B* $\in$ *Above B*
proof(*unfold supr-def*)
have *Above B* $\leq$ *Field r*
using *Above-Field*[*of r*] **by** *auto*
thus *minim* (*Above B*) $\in$ *Above B*
using *assms* **by** (*simp add*: *minim-in*)
qed

lemma *supr-greater*:
assumes *SUB*: $B \leq$ *Field r* **and** *ABOVE*: *Above B* $\neq \{\}$ **and**
IN: $b \in B$
shows (b, *supr B*) $\in r$
proof−
from *assms supr-Above*
have *supr B* $\in$ *Above B* **by** *simp*
with *IN Above-def*[*of r*] **show** *?thesis* **by** *simp*
qed

lemma *supr-least-Above*:
assumes *SUB*: $B \leq$ *Field r* **and**
ABOVE: $a \in$ *Above B*
shows (*supr B*, a) $\in r$
proof(*unfold supr-def*)
have *Above B* $\leq$ *Field r*
using *Above-Field*[*of r*] **by** *auto*
thus (*minim* (*Above B*), a) $\in r$
using *assms minim-least*

by *simp*
qed

lemma *supr-least*:
$[\![B \leq \mathit{Field}\ r;\ a \in \mathit{Field}\ r;\ (\bigwedge b.\ b \in B \Longrightarrow (b,a) \in r)]\!]$
$\Longrightarrow (\mathit{supr}\ B,\ a) \in r$
by(*auto simp add*: *supr-least-Above Above-def*)

lemma *equals-supr-Above*:
assumes *SUB*: $B \leq \mathit{Field}\ r$ **and** *ABV*: $a \in \mathit{Above}\ B$ **and**
MINIM: $\bigwedge a'.\ a' \in \mathit{Above}\ B \Longrightarrow (a,a') \in r$
shows $a = \mathit{supr}\ B$
proof(*unfold supr-def*)
have $\mathit{Above}\ B \leq \mathit{Field}\ r$
using *Above-Field*[*of r*] **by** *auto*
thus $a = \mathit{minim}\ (\mathit{Above}\ B)$
using *assms equals-minim* **by** *simp*
qed

lemma *equals-supr*:
assumes *SUB*: $B \leq \mathit{Field}\ r$ **and** *IN*: $a \in \mathit{Field}\ r$ **and**
ABV: $\bigwedge b.\ b \in B \Longrightarrow (b,a) \in r$ **and**
MINIM: $\bigwedge a'.\ [\![\ a' \in \mathit{Field}\ r;\ \bigwedge b.\ b \in B \Longrightarrow (b,a') \in r]\!] \Longrightarrow (a,a') \in r$
shows $a = \mathit{supr}\ B$
proof−
have $a \in \mathit{Above}\ B$
unfolding *Above-def* **using** *ABV IN* **by** *simp*
moreover
have $\bigwedge a'.\ a' \in \mathit{Above}\ B \Longrightarrow (a,a') \in r$
unfolding *Above-def* **using** *MINIM* **by** *simp*
ultimately show *?thesis*
using *equals-supr-Above SUB* **by** *auto*
qed

lemma *supr-inField*:
assumes $B \leq \mathit{Field}\ r$ **and** $\mathit{Above}\ B \neq \{\}$
shows $\mathit{supr}\ B \in \mathit{Field}\ r$
proof−
have $\mathit{supr}\ B \in \mathit{Above}\ B$ **using** *supr-Above assms* **by** *simp*
thus *?thesis* **using** *assms Above-Field*[*of r*] **by** *auto*
qed

lemma *supr-above-Above*:
assumes *SUB*: $B \leq \mathit{Field}\ r$ **and** *ABOVE*: $\mathit{Above}\ B \neq \{\}$
shows $\mathit{Above}\ B = \mathit{above}\ (\mathit{supr}\ B)$
proof(*unfold Above-def above-def*, *auto*)
fix *a* **assume** $a \in \mathit{Field}\ r\ \forall b \in B.\ (b,a) \in r$
with *supr-least assms*
show $(\mathit{supr}\ B,\ a) \in r$ **by** *auto*

```
next
  fix b assume (supr B, b) ∈ r
  thus b ∈ Field r
  using REFL refl-on-def[of - r] by auto
next
  fix a b
  assume 1: (supr B, b) ∈ r and 2: a ∈ B
  with assms supr-greater
  have (a,supr B) ∈ r by auto
  thus (a,b) ∈ r
  using 1 TRANS trans-def[of r] by blast
qed

lemma supr-under:
assumes IN: a ∈ Field r
shows a = supr (under a)
proof−
  have under a ≤ Field r
  using under-Field[of r] by auto
  moreover
  have under a ≠ {}
  using IN Refl-under-in[of r] REFL by auto
  moreover
  have a ∈ Above (under a)
  using in-Above-under[of - r] IN by auto
  moreover
  have ∀ a′ ∈ Above (under a). (a,a′) ∈ r
  proof(unfold Above-def under-def, auto)
    fix a′
    assume ∀ aa. (aa, a) ∈ r ⟶ (aa, a′) ∈ r
    hence (a,a) ∈ r ⟶ (a,a′) ∈ r by blast
    moreover have (a,a) ∈ r
    using REFL IN by (auto simp add: refl-on-def)
    ultimately
    show (a, a′) ∈ r by (rule mp)
  qed
  ultimately show ?thesis
  using equals-supr-Above by auto
qed
```

5.2.4 Properties of successor

```
lemma suc-least:
⟦B ≤ Field r; a ∈ Field r; (⋀ b. b ∈ B ⟹ a ≠ b ∧ (b,a) ∈ r)⟧
 ⟹ (suc B, a) ∈ r
by(auto simp add: suc-least-AboveS AboveS-def)

lemma equals-suc:
assumes SUB: B ≤ Field r and IN: a ∈ Field r and
```

ABVS: $\bigwedge$ *b*. $b \in B \Longrightarrow a \neq b \wedge (b,a) \in r$ **and**
MINIM: $\bigwedge$ a'. $[\![a' \in \mathit{Field}\ r;\ \bigwedge\ b.\ b \in B \Longrightarrow a' \neq b \wedge (b,a') \in r]\!] \Longrightarrow (a,a') \in r$
shows $a = \mathit{suc}\ B$
proof−
  have $a \in \mathit{AboveS}\ B$
  unfolding *AboveS-def* **using** *ABVS IN* **by** *simp*
  moreover
  have $\bigwedge$ a'. $a' \in \mathit{AboveS}\ B \Longrightarrow (a,a') \in r$
  unfolding *AboveS-def* **using** *MINIM* **by** *simp*
  ultimately show *?thesis*
  using *equals-suc-AboveS SUB* **by** *auto*
qed

lemma *suc-above-AboveS*:
assumes *SUB*: $B \leq \mathit{Field}\ r$ **and**
    ABOVE: $\mathit{AboveS}\ B \neq \{\}$
shows $\mathit{AboveS}\ B = \mathit{above}\ (\mathit{suc}\ B)$
proof(*unfold AboveS-def above-def*, *auto*)
  fix *a* **assume** $a \in \mathit{Field}\ r\ \forall\, b \in B.\ a \neq b \wedge (b,a) \in r$
  with *suc-least assms*
  show $(\mathit{suc}\ B,a) \in r$ **by** *auto*
next
  fix *b* **assume** $(\mathit{suc}\ B,\ b) \in r$
  thus $b \in \mathit{Field}\ r$
  using *REFL refl-on-def*[*of - r*] **by** *auto*
next
  fix *a b*
  assume *1*: $(\mathit{suc}\ B,\ b) \in r$ **and** *2*: $a \in B$
  with *assms suc-greater*[*of B a*]
  have $(a,\mathit{suc}\ B) \in r$ **by** *auto*
  thus $(a,b) \in r$
  using *1 TRANS trans-def*[*of r*] **by** *blast*
next
  fix *a*
  assume *1*: $(\mathit{suc}\ B,\ a) \in r$ **and** *2*: $a \in B$
  with *assms suc-greater*[*of B a*]
  have $(a,\mathit{suc}\ B) \in r$ **by** *auto*
  moreover have $\mathit{suc}\ B \in \mathit{Field}\ r$
  using *assms suc-inField* **by** *simp*
  ultimately have $a = \mathit{suc}\ B$
  using *1 2 SUB ANTISYM antisym-def*[*of r*] **by** *auto*
  thus *False*
  using *assms suc-greater*[*of B a*] *2* **by** *auto*
qed

lemma *suc-singl-pred*:
assumes *IN*: $a \in \mathit{Field}\ r$ **and** *ABOVE-NE*: $\mathit{aboveS}\ a \neq \{\}$ **and**
    REL: $(a',\mathit{suc}\ \{a\}) \in r$ **and** *DIFF*: $a' \neq \mathit{suc}\ \{a\}$
shows $a' = a \vee (a',a) \in r$

proof−
  have ∗: *suc* {*a*} ∈ *Field r* ∧ *a*′ ∈ *Field r*
  using *WELL REL well-order-on-domain* **by** *metis*
  {**assume** ∗∗: *a*′ ≠ *a*
   hence (*a*,*a*′) ∈ *r* ∨ (*a*′,*a*) ∈ *r*
   using *TOTAL IN* ∗ **by** (*auto simp add*: *total-on-def*)
   moreover
   {**assume** (*a*,*a*′) ∈ *r*
    with ∗∗ ∗ *assms WELL suc-least*[*of* {*a*} *a*′]
    have (*suc* {*a*},*a*′) ∈ *r* **by** *auto*
    with *REL DIFF* ∗ *ANTISYM antisym-def*[*of r*]
    have *False* **by** *simp*
   }
   ultimately have (*a*′,*a*) ∈ *r*
   by *blast*
  }
  thus *?thesis* **by** *blast*
qed

lemma *under-underS-suc*:
assumes *IN*: *a* ∈ *Field r* **and** *ABV*: *aboveS a* ≠ {}
shows *underS* (*suc* {*a*}) = *under a*
proof−
  have *1*: *AboveS* {*a*} ≠ {}
  using *ABV aboveS-AboveS-singl*[*of r*] **by** *auto*
  have *2*: *a* ≠ *suc* {*a*} ∧ (*a*,*suc* {*a*}) ∈ *r*
  using *suc-greater*[*of* {*a*} *a*] *IN 1* **by** *auto*

  have *underS* (*suc* {*a*}) ≤ *under a*
  proof(*unfold underS-def under-def*, *auto*)
    fix *x* **assume** ∗: *x* ≠ *suc* {*a*} **and** ∗∗: (*x*,*suc* {*a*}) ∈ *r*
    with *suc-singl-pred*[*of a x*] *IN ABV*
    have *x* = *a* ∨ (*x*,*a*) ∈ *r* **by** *auto*
    with *REFL refl-on-def*[*of* - *r*] *IN*
    show (*x*,*a*) ∈ *r* **by** *auto*
  qed

  moreover

  have *under a* ≤ *underS* (*suc* {*a*})
  proof(*unfold underS-def under-def*, *auto*)
    assume (*suc* {*a*}, *a*) ∈ *r*
    with *2 ANTISYM antisym-def*[*of r*]
    show *False* **by** *auto*
  next
    fix *x* **assume** ∗: (*x*,*a*) ∈ *r*
    with *2 TRANS trans-def*[*of r*]
    show (*x*,*suc* {*a*}) ∈ *r* **by** *blast*

qed

ultimately show *?thesis* **by** *blast*
qed

5.2.5 Properties of order filters

lemma *ofilter-Under*[*simp*]:
assumes $A \leq$ *Field r*
shows *ofilter*(*Under A*)
proof(*unfold ofilter-def*, *auto*)
fix x **assume** $x \in$ *Under A*
thus $x \in$ *Field r*
using *Under-Field*[*of r*] *assms* **by** *auto*
next
fix a x
assume $a \in$ *Under A* **and** $x \in$ *under a*
thus $x \in$ *Under A*
using *TRANS under-Under-trans*[*of r*] **by** *auto*
qed

lemma *ofilter-UnderS*[*simp*]:
assumes $A \leq$ *Field r*
shows *ofilter*(*UnderS A*)
proof(*unfold ofilter-def*, *auto*)
fix x **assume** $x \in$ *UnderS A*
thus $x \in$ *Field r*
using *UnderS-Field*[*of r*] *assms* **by** *auto*
next
fix a x
assume $a \in$ *UnderS A* **and** $x \in$ *under a*
thus $x \in$ *UnderS A*
using *TRANS ANTISYM under-UnderS-trans*[*of r*] **by** *auto*
qed

lemma *ofilter-Int*[*simp*]: $[\![$*ofilter A*; *ofilter B*$]\!] \Longrightarrow$ *ofilter*(*A Int B*)
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-Un*[*simp*]: $[\![$*ofilter A*; *ofilter B*$]\!] \Longrightarrow$ *ofilter*$(A \cup B)$
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-INTER*:
$[\![I \neq \{\}; \bigwedge i.\ i \in I \Longrightarrow$ *ofilter*$(A\ i)]\!] \Longrightarrow$ *ofilter* $(\bigcap i \in I.\ A\ i)$
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-Inter*:
$[\![S \neq \{\}; \bigwedge A.\ A \in S \Longrightarrow$ *ofilter A*$]\!] \Longrightarrow$ *ofilter* $(\bigcap S)$
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-Union*:
$(\bigwedge A.\ A \in S \Longrightarrow \mathit{ofilter}\ A) \Longrightarrow \mathit{ofilter}\ (\bigcup S)$
unfolding *ofilter-def* **by** *blast*

lemma *ofilter-under-Union*:
$\mathit{ofilter}\ A \Longrightarrow A = \bigcup \{\mathit{under}\ a|\ a.\ a \in A\}$
using *ofilter-under-UNION* [*of A*] **by** *auto*

5.2.6 Other properties

lemma *Trans-Under-regressive*:
assumes *NE*: $A \neq \{\}$ **and** *SUB*: $A \leq \mathit{Field}\ r$
shows $\mathit{Under}(\mathit{Under}\ A) \leq \mathit{Under}\ A$
proof
  let $?a = \mathit{minim}\ A$

  have *1*: $\mathit{minim}\ A \in \mathit{Under}\ A$
  using *assms minim-Under* **by** *auto*
  have *2*: $\forall y \in A.\ (\mathit{minim}\ A,\ y) \in r$
  using *assms minim-least* **by** *auto*

  fix x **assume** $x \in \mathit{Under}(\mathit{Under}\ A)$
  with *1* **have** *1*: $(x,\mathit{minim}\ A) \in r$
  using *Under-def*[*of r*] **by** *auto*
  with *Field-def* **have** $x \in \mathit{Field}\ r$ **by** *fastforce*
  moreover
  {fix y **assume** $*$: $y \in A$
   hence $(x,y) \in r$
   using *1 2 TRANS trans-def*[*of r*] **by** *blast*
   with *Field-def* **have** $(x,y) \in r$ **by** *auto*
  }
  ultimately
  show $x \in \mathit{Under}\ A$ **unfolding** *Under-def* **by** *auto*
qed

lemma *ofilter-suc-Field*:
assumes *OF*: *ofilter A* **and** *NE*: $A \neq \mathit{Field}\ r$
shows $\mathit{ofilter}\ (A \cup \{\mathit{suc}\ A\})$
proof−

  have *1*: $A \leq \mathit{Field}\ r$ **using** *OF ofilter-def* **by** *auto*
  hence *2*: $\mathit{AboveS}\ A \neq \{\}$
  using *ofilter-AboveS-Field NE OF* **by** *blast*
  from *1 2 suc-inField*
  have *3*: $\mathit{suc}\ A \in \mathit{Field}\ r$ **by** *auto*

  show *?thesis*
  proof(*unfold ofilter-def*, *auto simp add*: *1 3*)
    fix *a x*

assume $a \in A$ $x \in$ *under* a $x \notin A$
with *OF ofilter-def* **have** *False* **by** *auto*
thus $x =$ *suc* A **by** *simp*
next
fix x **assume** $*$: $x \in$ *under* (*suc* A) **and** $**$: $x \notin A$
hence $x \in$ *Field* r **using** *under-def Field-def* **by** *fastforce*
with $**$ **have** $x \in$ *AboveS* A
using *ofilter-AboveS-Field*[*of* A] *OF* **by** *auto*
hence (*suc* A,x) $\in r$
using *suc-least-AboveS* **by** *auto*
moreover
have (x,*suc* A) $\in r$ **using** $*$ *under-def*[*of* r] **by** *auto*
ultimately show $x =$ *suc* A
using *ANTISYM antisym-def*[*of* r] **by** *auto*
qed
qed

declare
minim-in[*simp*]
minim-inField[*simp*]
minim-least[*simp*]
under-ofilter[*simp*]
underS-ofilter[*simp*]
Field-ofilter[*simp*]

end

abbreviation *worec* $\equiv$ *wo-rel.worec*
abbreviation *adm-wo* $\equiv$ *wo-rel.adm-wo*
abbreviation *isMinim* $\equiv$ *wo-rel.isMinim*
abbreviation *minim* $\equiv$ *wo-rel.minim*
abbreviation *max2* $\equiv$ *wo-rel.max2*
abbreviation *supr* $\equiv$ *wo-rel.supr*
abbreviation *suc* $\equiv$ *wo-rel.suc*

end

6 Well-Order Embeddings

theory *Wellorder-Embedding*
imports *Fun-More Wellorder-Relation*
begin

6.1 Auxiliaries

lemma *UNION-bij-betw-ofilter*:
assumes *WELL*: *Well-order* r **and**
OF: $\bigwedge i.\ i \in I \Longrightarrow$ *ofilter* r (A i) **and**

BIJ: $\bigwedge i.\ i \in I \Longrightarrow$ *bij-betw f* $(A\ i)\ (A'\ i)$
shows *bij-betw f* $(\bigcup i \in I.\ A\ i)\ (\bigcup i \in I.\ A'\ i)$
proof−
have *wo-rel r* **using** *WELL* **by** (*simp add*: *wo-rel-def*)
hence $\bigwedge i\ j.\ [\![i \in I;\ j \in I]\!] \Longrightarrow A\ i \leq A\ j \vee A\ j \leq A\ i$
using *wo-rel.ofilter-linord*[*of r*] *OF* **by** *blast*
with *WELL BIJ* **show** *?thesis*
by (*auto simp add*: *bij-betw-UNION-chain*)
qed

6.2 (Well-order) embeddings, strict embeddings, isomorphisms and order-compatible functions

lemma *embed-halfcong*:
assumes *EQ*: $\bigwedge a.\ a \in$ *Field r* $\Longrightarrow f\ a = g\ a$ **and**
EMB: *embed r r′ f*
shows *embed r r′ g*
proof(*unfold embed-def*, *auto*)
fix *a* **assume** ∗: $a \in$ *Field r*
hence *bij-betw f* (*under r a*) (*under r′* (*f a*))
using *EMB* **unfolding** *embed-def* **by** *simp*
moreover
{**have** *under r a* $\leq$ *Field r*
by (*auto simp add*: *under-Field*)
hence $\bigwedge b.\ b \in$ *under r a* $\Longrightarrow f\ b = g\ b$
using *EQ* **by** *blast*
}
moreover have $f\ a = g\ a$ **using** ∗ *EQ* **by** *auto*
ultimately show *bij-betw g* (*under r a*) (*under r′* (*g a*))
using *bij-betw-cong*[*of under r a f g under r′* (*f a*)] **by** *auto*
qed

lemma *embed-cong*[*fundef-cong*]:
assumes $\bigwedge a.\ a \in$ *Field r* $\Longrightarrow f\ a = g\ a$
shows *embed r r′ f* = *embed r r′ g*
using *assms embed-halfcong*[*of r f g r′*]
embed-halfcong[*of r g f r′*] **by** *auto*

lemma *embedS-cong*[*fundef-cong*]:
assumes $\bigwedge a.\ a \in$ *Field r* $\Longrightarrow f\ a = g\ a$
shows *embedS r r′ f* = *embedS r r′ g*
unfolding *embedS-def* **using** *assms*
embed-cong[*of r f g r′*] *bij-betw-cong*[*of Field r f g Field r′*] **by** *blast*

lemma *iso-cong*[*fundef-cong*]:
assumes $\bigwedge a.\ a \in$ *Field r* $\Longrightarrow f\ a = g\ a$
shows *iso r r′ f* = *iso r r′ g*
unfolding *iso-def* **using** *assms*
embed-cong[*of r f g r′*] *bij-betw-cong*[*of Field r f g Field r′*] **by** *blast*

lemma *id-compat*: *compat r r id*
by(*auto simp add*: *id-def compat-def*)

lemma *comp-compat*:
$[\![$*compat r r′ f*; *compat r′ r′′ f′*$]\!] \Longrightarrow$ *compat r r′′* (*f′ o f*)
by(*auto simp add*: *comp-def compat-def*)

corollary *one-set-greater*:
($\exists f$::′*a* $\Rightarrow$ ′*a*′. *f* ‘ *A* $\leq$ *A*′ $\wedge$ *inj-on f A*) $\vee$ ($\exists g$::′*a*′ $\Rightarrow$ ′*a*. *g* ‘ *A*′ $\leq$ *A* $\wedge$ *inj-on g A*′)
proof−
  obtain *r* **where** *well-order-on A r* **by** (*fastforce simp add*: *well-order-on*)
  hence *1*: *A* = *Field r* $\wedge$ *Well-order r*
  using *well-order-on-Well-order* **by** *auto*
  obtain *r′* **where** *2*: *well-order-on A′ r′* **by** (*fastforce simp add*: *well-order-on*)
  hence *2*: *A′* = *Field r′* $\wedge$ *Well-order r′*
  using *well-order-on-Well-order* **by** *auto*
  hence ($\exists f$. *embed r r′ f*) $\vee$ ($\exists g$. *embed r′ r g*)
  using *1 2* **by** (*auto simp add*: *wellorders-totally-ordered*)
  moreover
  {fix *f* **assume** *embed r r′ f*
   hence *f*‘*A* $\leq$ *A′* $\wedge$ *inj-on f A*
   using *1 2* **by** (*auto simp add*: *embed-Field embed-inj-on*)
  }
  moreover
  {fix *g* **assume** *embed r′ r g*
   hence *g*‘*A′* $\leq$ *A* $\wedge$ *inj-on g A′*
   using *1 2* **by** (*auto simp add*: *embed-Field embed-inj-on*)
  }
  ultimately show *?thesis* **by** *blast*
qed

corollary *one-type-greater*:
($\exists f$::′*a* $\Rightarrow$ ′*a*′. *inj f*) $\vee$ ($\exists g$::′*a*′ $\Rightarrow$ ′*a*. *inj g*)
using *one-set-greater*[*of UNIV UNIV*] **by** *auto*

6.3 Uniqueness of embeddings

lemma *comp-embedS*:
assumes *WELL*: *Well-order r* **and** *WELL′*: *Well-order r′* **and** *WELL′′*: *Well-order r′′*
      and *EMB*: *embedS r r′ f* **and** *EMB′*: *embedS r′ r′′ f′*
shows *embedS r r′′* (*f′ o f*)
proof−
  have *embed r′ r′′ f′* **using** *EMB′* **unfolding** *embedS-def* **by** *simp*
  thus *?thesis* **using** *assms* **by** (*auto simp add*: *embedS-comp-embed*)
qed

lemma *iso-iff4*:

assumes *WELL*: *Well-order r* **and** *WELL′*: *Well-order r′*
shows *iso r r′ f* = (*embed r r′ f* ∧ *embed r′ r* (*inv-into* (*Field r*) *f*))
using *assms embed-bothWays-iso*
by(*unfold iso-def*, *auto simp add*: *inv-into-Field-embed-bij-betw*)

lemma *embed-embedS-iso*:
embed r r′ f = (*embedS r r′ f* ∨ *iso r r′ f*)
unfolding *embedS-def iso-def* **by** *blast*

lemma *not-embedS-iso*:
¬ (*embedS r r′ f* ∧ *iso r r′ f*)
unfolding *embedS-def iso-def* **by** *blast*

lemma *embed-embedS-iff-not-iso*:
assumes *embed r r′ f*
shows *embedS r r′ f* = (¬ *iso r r′ f*)
using *assms* **unfolding** *embedS-def iso-def* **by** *blast*

lemma *iso-inv-into*:
assumes *WELL*: *Well-order r* **and** *ISO*: *iso r r′ f*
shows *iso r′ r* (*inv-into* (*Field r*) *f*)
using *assms* **unfolding** *iso-def*
using *bij-betw-inv-into inv-into-Field-embed-bij-betw* **by** *blast*

lemma *embedS-or-iso*:
assumes *WELL*: *Well-order r* **and** *WELL′*: *Well-order r′*
shows (∃ *g*. *embedS r r′ g*) ∨ (∃ *h*. *embedS r′ r h*) ∨ (∃ *f*. *iso r r′ f*)
proof−
 {**fix** *f* **assume** ∗: *embed r r′ f*
  {**assume** *bij-betw f* (*Field r*) (*Field r′*)
   hence *?thesis* **using** ∗ **by** (*auto simp add*: *iso-def*)
  }
  moreover
  {**assume** ¬ *bij-betw f* (*Field r*) (*Field r′*)
   hence *?thesis* **using** ∗ **by** (*auto simp add*: *embedS-def*)
  }
  ultimately have *?thesis* **by** *auto*
 }
 moreover
 {**fix** *f* **assume** ∗: *embed r′ r f*
  {**assume** *bij-betw f* (*Field r′*) (*Field r*)
   hence *iso r′ r f* **using** ∗ **by** (*auto simp add*: *iso-def*)
   hence *iso r r′* (*inv-into* (*Field r′*) *f*)
   using *WELL′* **by** (*auto simp add*: *iso-inv-into*)
   hence *?thesis* **by** *blast*
  }
  moreover
  {**assume** ¬ *bij-betw f* (*Field r′*) (*Field r*)
   hence *?thesis* **using** ∗ **by** (*auto simp add*: *embedS-def*)

```
   }
   ultimately have ?thesis by auto
  }
  ultimately show ?thesis using WELL WELL'
  wellorders-totally-ordered[of r r'] by blast
qed

end
```

7 Order Union

```
theory Order-Union
imports Main
begin

definition Osum :: 'a rel ⇒ 'a rel ⇒ 'a rel  (infix Osum 60) where
  r Osum r' = r ∪ r' ∪ {(a, a'). a ∈ Field r ∧ a' ∈ Field r'}

notation Osum  (infix ∪o 60)

lemma Field-Osum: Field (r ∪o r') = Field r ∪ Field r'
  unfolding Osum-def Field-def by blast

lemma Osum-wf:
assumes FLD: Field r Int Field r' = {} and
        WF: wf r and WF': wf r'
shows wf (r Osum r')
unfolding wf-eq-minimal2 unfolding Field-Osum
proof(intro allI impI, elim conjE)
  fix A assume *: A ⊆ Field r ∪ Field r' and **: A ≠ {}
  obtain B where B-def: B = A Int Field r by blast
  show ∃a∈A. ∀a'∈A. (a', a) ∉ r ∪o r'
  proof(cases B = {})
    assume Case1: B ≠ {}
    hence B ≠ {} ∧ B ≤ Field r using B-def by auto
    then obtain a where 1: a ∈ B and 2: ∀a1 ∈ B. (a1,a) ∉ r
    using WF unfolding wf-eq-minimal2 by blast
    hence 3: a ∈ Field r ∧ a ∉ Field r' using B-def FLD by auto

    have ∀a1 ∈ A. (a1,a) ∉ r Osum r'
    proof(intro ballI)
      fix a1 assume **: a1 ∈ A
      {assume Case11: a1 ∈ Field r
       hence (a1,a) ∉ r using B-def ** 2 by auto
       moreover
       have (a1,a) ∉ r' using 3 by (auto simp add: Field-def)
       ultimately have (a1,a) ∉ r Osum r'
       using 3 unfolding Osum-def by auto
      }
```

```
      moreover
      {assume Case12: a1 ∉ Field r
       hence (a1,a) ∉ r unfolding Field-def by auto
       moreover
       have (a1,a) ∉ r' using 3 unfolding Field-def by auto
       ultimately have (a1,a) ∉ r Osum r'
       using 3 unfolding Osum-def by auto
      }
      ultimately show (a1,a) ∉ r Osum r' by blast
    qed
    thus ?thesis using 1 B-def by auto
  next
    assume Case2: B = {}
    hence 1: A ≠ {} ∧ A ≤ Field r' using ∗ ∗∗ B-def by auto
    then obtain a' where 2: a' ∈ A and 3: ∀ a1' ∈ A. (a1',a') ∉ r'
    using WF' unfolding wf-eq-minimal2 by blast
    hence 4: a' ∈ Field r' ∧ a' ∉ Field r using 1 FLD by blast

    have ∀ a1' ∈ A. (a1',a') ∉ r Osum r'
    proof(unfold Osum-def, auto simp add: 3)
      fix a1' assume (a1', a') ∈ r
      thus False using 4 unfolding Field-def by blast
    next
      fix a1' assume a1' ∈ A and a1' ∈ Field r
      thus False using Case2 B-def by auto
    qed
    thus ?thesis using 2 by blast
  qed
qed

lemma Osum-Refl:
assumes FLD: Field r Int Field r' = {} and
        REFL: Refl r and REFL': Refl r'
shows Refl (r Osum r')
using assms
unfolding refl-on-def Field-Osum unfolding Osum-def by blast

lemma Osum-trans:
assumes FLD: Field r Int Field r' = {} and
        TRANS: trans r and TRANS': trans r'
shows trans (r Osum r')
proof(unfold trans-def, auto)
  fix x y z assume ∗: (x, y) ∈ r ∪o r' and ∗∗: (y, z) ∈ r ∪o r'
  show  (x, z) ∈ r ∪o r'
  proof−
    {assume Case1: (x,y) ∈ r
     hence 1: x ∈ Field r ∧ y ∈ Field r unfolding Field-def by auto
     have ?thesis
     proof−
```

{**assume** *Case11*: $(y,z) \in r$
hence $(x,z) \in r$ **using** *Case1 TRANS trans-def*[*of r*] **by** *blast*
hence *?thesis* **unfolding** *Osum-def* **by** *auto*
}
moreover
{**assume** *Case12*: $(y,z) \in r'$
hence $y \in$ *Field* r' **unfolding** *Field-def* **by** *auto*
hence *False* **using** *FLD 1* **by** *auto*
}
moreover
{**assume** *Case13*: $z \in$ *Field* r'
hence *?thesis* **using** *1* **unfolding** *Osum-def* **by** *auto*
}
ultimately show *?thesis* **using** $**$ **unfolding** *Osum-def* **by** *blast*
qed
}
moreover
{**assume** *Case2*: $(x,y) \in r'$
hence *2*: $x \in$ *Field* $r' \wedge y \in$ *Field* r' **unfolding** *Field-def* **by** *auto*
have *?thesis*
proof−
{**assume** *Case21*: $(y,z) \in r$
hence $y \in$ *Field r* **unfolding** *Field-def* **by** *auto*
hence *False* **using** *FLD 2* **by** *auto*
}
moreover
{**assume** *Case22*: $(y,z) \in r'$
hence $(x,z) \in r'$ **using** *Case2 TRANS' trans-def*[*of r'*] **by** *blast*
hence *?thesis* **unfolding** *Osum-def* **by** *auto*
}
moreover
{**assume** *Case23*: $y \in$ *Field r*
hence *False* **using** *FLD 2* **by** *auto*
}
ultimately show *?thesis* **using** $**$ **unfolding** *Osum-def* **by** *blast*
qed
}
moreover
{**assume** *Case3*: $x \in$ *Field* $r \wedge y \in$ *Field* r'
have *?thesis*
proof−
{**assume** *Case31*: $(y,z) \in r$
hence $y \in$ *Field r* **unfolding** *Field-def* **by** *auto*
hence *False* **using** *FLD Case3* **by** *auto*
}
moreover
{**assume** *Case32*: $(y,z) \in r'$
hence $z \in$ *Field* r' **unfolding** *Field-def* **by** *blast*
hence *?thesis* **unfolding** *Osum-def* **using** *Case3* **by** *auto*

```
      }
      moreover
      {assume Case33: y ∈ Field r
       hence False using FLD Case3 by auto
      }
      ultimately show ?thesis using ** unfolding Osum-def by blast
    qed
   }
   ultimately show ?thesis using * unfolding Osum-def by blast
  qed
qed

lemma Osum-Preorder:
⟦Field r Int Field r′ = {}; Preorder r; Preorder r′⟧ ⟹ Preorder (r Osum r′)
unfolding preorder-on-def using Osum-Refl Osum-trans by blast

lemma Osum-antisym:
assumes FLD: Field r Int Field r′ = {} and
        AN: antisym r and AN′: antisym r′
shows antisym (r Osum r′)
proof(unfold antisym-def, auto)
  fix x y assume *: (x, y) ∈ r ∪o r′ and **: (y, x) ∈ r ∪o r′
  show  x = y
  proof−
   {assume Case1: (x,y) ∈ r
    hence 1: x ∈ Field r ∧ y ∈ Field r unfolding Field-def by auto
    have ?thesis
    proof−
      have (y,x) ∈ r ⟹ ?thesis
      using Case1 AN antisym-def[of r] by blast
      moreover
      {assume (y,x) ∈ r′
       hence y ∈ Field r′ unfolding Field-def by auto
       hence False using FLD 1 by auto
      }
      moreover
      have x ∈ Field r′ ⟹ False using FLD 1 by auto
      ultimately show ?thesis using ** unfolding Osum-def by blast
    qed
   }
   moreover
   {assume Case2: (x,y) ∈ r′
    hence 2: x ∈ Field r′ ∧ y ∈ Field r′ unfolding Field-def by auto
    have ?thesis
    proof−
      {assume (y,x) ∈ r
       hence y ∈ Field r unfolding Field-def by auto
       hence False using FLD 2 by auto
      }
```

```
      moreover
      have (y,x) ∈ r′ ⟹ ?thesis
      using Case2 AN′ antisym-def[of r′] by blast
      moreover
      {assume y ∈ Field r
       hence False using FLD 2 by auto
      }
      ultimately show ?thesis using ∗∗ unfolding Osum-def by blast
     qed
    }
    moreover
    {assume Case3: x ∈ Field r ∧ y ∈ Field r′
     have ?thesis
     proof−
      {assume (y,x) ∈ r
       hence y ∈ Field r unfolding Field-def by auto
       hence False using FLD Case3 by auto
      }
      moreover
      {assume Case32: (y,x) ∈ r′
       hence x ∈ Field r′ unfolding Field-def by blast
       hence False using FLD Case3 by auto
      }
      moreover
      have ¬ y ∈ Field r using FLD Case3 by auto
      ultimately show ?thesis using ∗∗ unfolding Osum-def by blast
     qed
    }
    ultimately show ?thesis using ∗ unfolding Osum-def by blast
  qed
qed

lemma Osum-Partial-order:
⟦Field r Int Field r′ = {}; Partial-order r; Partial-order r′⟧ ⟹
 Partial-order (r Osum r′)
unfolding partial-order-on-def using Osum-Preorder Osum-antisym by blast

lemma Osum-Total:
assumes FLD: Field r Int Field r′ = {} and
        TOT: Total r and TOT′: Total r′
shows Total (r Osum r′)
using assms
unfolding total-on-def  Field-Osum unfolding Osum-def by blast

lemma Osum-Linear-order:
⟦Field r Int Field r′ = {}; Linear-order r; Linear-order r′⟧ ⟹
 Linear-order (r Osum r′)
unfolding linear-order-on-def using Osum-Partial-order Osum-Total by blast
```

lemma *Osum-minus-Id1*:
assumes $r \leq Id$
shows $(r\ Osum\ r') - Id \leq (r' - Id) \cup (Field\ r \times Field\ r')$
proof−
  let $?Left = (r\ Osum\ r') - Id$
  let $?Right = (r' - Id) \cup (Field\ r \times Field\ r')$
  {**fix** $a{::}'a$ **and** b **assume** $*$: $(a,b) \notin Id$
   {**assume** $(a,b) \in r$
    with $*$ **have** *False* **using** *assms* **by** *auto*
   }
   moreover
   {**assume** $(a,b) \in r'$
    with $*$ **have** $(a,b) \in r' - Id$ **by** *auto*
   }
   ultimately
   have $(a,b) \in ?Left \Longrightarrow (a,b) \in ?Right$
   unfolding *Osum-def* **by** *auto*
  }
  thus *?thesis* **by** *auto*
qed

lemma *Osum-minus-Id2*:
assumes $r' \leq Id$
shows $(r\ Osum\ r') - Id \leq (r - Id) \cup (Field\ r \times Field\ r')$
proof−
  let $?Left = (r\ Osum\ r') - Id$
  let $?Right = (r - Id) \cup (Field\ r \times Field\ r')$
  {**fix** $a{::}'a$ **and** b **assume** $*$: $(a,b) \notin Id$
   {**assume** $(a,b) \in r'$
    with $*$ **have** *False* **using** *assms* **by** *auto*
   }
   moreover
   {**assume** $(a,b) \in r$
    with $*$ **have** $(a,b) \in r - Id$ **by** *auto*
   }
   ultimately
   have $(a,b) \in ?Left \Longrightarrow (a,b) \in ?Right$
   unfolding *Osum-def* **by** *auto*
  }
  thus *?thesis* **by** *auto*
qed

lemma *Osum-minus-Id*:
assumes *TOT*: *Total* r **and** *TOT'*: *Total* r' **and**
        NID: $\neg\ (r \leq Id)$ **and** *NID'*: $\neg\ (r' \leq Id)$
shows $(r\ Osum\ r') - Id \leq (r - Id)\ Osum\ (r' - Id)$
proof−
  {**fix** $a\ a'$ **assume** $*$: $(a,a') \in (r\ Osum\ r')$ **and** $**$: $a \neq a'$
   have $(a,a') \in (r - Id)\ Osum\ (r' - Id)$

proof−
{**assume** $(a,a') \in r \vee (a,a') \in r'$
with $**$ **have** *?thesis* **unfolding** *Osum-def* **by** *auto*
}
moreover
{**assume** $a \in \mathit{Field}\ r \wedge a' \in \mathit{Field}\ r'$
hence $a \in \mathit{Field}(r - \mathit{Id}) \wedge a' \in \mathit{Field}\ (r' - \mathit{Id})$
using *assms Total-Id-Field* **by** *blast*
hence *?thesis* **unfolding** *Osum-def* **by** *auto*
}
ultimately show *?thesis* **using** $*$ **unfolding** *Osum-def* **by** *fast*
qed
}
thus *?thesis* **by**(*auto simp add*: *Osum-def*)
qed

lemma *wf-Int-Times*:
assumes $A\ \mathit{Int}\ B = \{\}$
shows $\mathit{wf}(A \times B)$
unfolding *wf-def* **using** *assms* **by** *blast*

lemma *Osum-wf-Id*:
assumes TOT: $\mathit{Total}\ r$ **and** TOT': $\mathit{Total}\ r'$ **and**
FLD: $\mathit{Field}\ r\ \mathit{Int}\ \mathit{Field}\ r' = \{\}$ **and**
WF: $\mathit{wf}(r - \mathit{Id})$ **and** WF': $\mathit{wf}(r' - \mathit{Id})$
shows $\mathit{wf}\ ((r\ \mathit{Osum}\ r') - \mathit{Id})$
proof(*cases* $r \le \mathit{Id} \vee r' \le \mathit{Id}$)
assume *Case1*: $\neg(r \le \mathit{Id} \vee r' \le \mathit{Id})$
have $\mathit{Field}(r - \mathit{Id})\ \mathit{Int}\ \mathit{Field}(r' - \mathit{Id}) = \{\}$
using *FLD mono-Field*[*of* $r - \mathit{Id}\ r$] *mono-Field*[*of* $r' - \mathit{Id}\ r'$]
Diff-subset[*of* $r\ \mathit{Id}$] *Diff-subset*[*of* $r'\ \mathit{Id}$] **by** *blast*
thus *?thesis*
using *Case1 Osum-minus-Id*[*of* $r\ r'$] *assms Osum-wf*[*of* $r - \mathit{Id}\ r' - \mathit{Id}$]
wf-subset[*of* $(r - \mathit{Id}) \cup o\ (r' - \mathit{Id})\ (r\ \mathit{Osum}\ r') - \mathit{Id}$] **by** *auto*
next
have *1*: $\mathit{wf}(\mathit{Field}\ r \times \mathit{Field}\ r')$
using *FLD* **by** (*auto simp add*: *wf-Int-Times*)
assume *Case2*: $r \le \mathit{Id} \vee r' \le \mathit{Id}$
moreover
{**assume** *Case21*: $r \le \mathit{Id}$
hence $(r\ \mathit{Osum}\ r') - \mathit{Id} \le (r' - \mathit{Id}) \cup (\mathit{Field}\ r \times \mathit{Field}\ r')$
using *Osum-minus-Id1*[*of* $r\ r'$] **by** *simp*
moreover
{**have** $\mathit{Domain}(\mathit{Field}\ r \times \mathit{Field}\ r')\ \mathit{Int}\ \mathit{Range}(r' - \mathit{Id}) = \{\}$
using *FLD* **unfolding** *Field-def* **by** *blast*
hence $\mathit{wf}((r' - \mathit{Id}) \cup (\mathit{Field}\ r \times \mathit{Field}\ r'))$
using *1* WF' *wf-Un*[*of* $\mathit{Field}\ r \times \mathit{Field}\ r'\ r' - \mathit{Id}$]
by (*auto simp add*: *Un-commute*)
}

ultimately have *?thesis* **using** *wf-subset* **by** *blast*
}
moreover
{assume *Case22*: $r' \leq Id$
hence $(r\ Osum\ r') - Id \leq (r - Id) \cup (Field\ r \times Field\ r')$
using *Osum-minus-Id2*[*of* r' *r*] **by** *simp*
moreover
{have $Range(Field\ r \times Field\ r')\ Int\ Domain(r - Id) = \{\}$
using *FLD* **unfolding** *Field-def* **by** *blast*
hence $wf((r - Id) \cup (Field\ r \times Field\ r'))$
using *1 WF wf-Un*[*of* $r - Id\ Field\ r \times Field\ r'$]
by (*auto simp add*: *Un-commute*)
}
ultimately have *?thesis* **using** *wf-subset* **by** *blast*
}
ultimately show *?thesis* **by** *blast*
qed

lemma *Osum-Well-order*:
assumes *FLD*: $Field\ r\ Int\ Field\ r' = \{\}$ **and**
WELL: *Well-order r* **and** *WELL'*: *Well-order* r'
shows *Well-order* $(r\ Osum\ r')$
proof−
have $Total\ r \wedge Total\ r'$ **using** *WELL WELL'*
by (*auto simp add*: *order-on-defs*)
thus *?thesis* **using** *assms* **unfolding** *well-order-on-def*
using *Osum-Linear-order Osum-wf-Id* **by** *blast*
qed

end

8 Constructions on Wellorders

theory *Wellorder-Constructions*
imports
Wellorder-Embedding Order-Union
begin

unbundle *cardinal-syntax*

declare
ordLeq-Well-order-simp[*simp*]
not-ordLeq-iff-ordLess[*simp*]
not-ordLess-iff-ordLeq[*simp*]
Func-empty[*simp*]
Func-is-emp[*simp*]

lemma *Func-emp2*[*simp*]: $A \neq \{\} \Longrightarrow Func\ A\ \{\} = \{\}$ **by** *auto*

8.1 Restriction to a set

lemma *Restr-incr2*:
$r <= r' \Longrightarrow Restr\ r\ A <= Restr\ r'\ A$
by *blast*

lemma *Restr-incr*:
$[\![r \leq r'; A \leq A']\!] \Longrightarrow Restr\ r\ A \leq Restr\ r'\ A'$
by *blast*

lemma *Restr-Int*:
$Restr\ (Restr\ r\ A)\ B = Restr\ r\ (A\ Int\ B)$
by *blast*

lemma *Restr-iff*: $(a,b) \in Restr\ r\ A = (a \in A \wedge b \in A \wedge (a,b) \in r)$
by (*auto simp add*: *Field-def*)

lemma *Restr-subset1*: $Restr\ r\ A \leq r$
by *auto*

lemma *Restr-subset2*: $Restr\ r\ A \leq A \times A$
by *auto*

lemma *wf-Restr*:
$wf\ r \Longrightarrow wf(Restr\ r\ A)$
using *Restr-subset* **by** (*elim wf-subset*) *simp*

lemma *Restr-incr1*:
$A \leq B \Longrightarrow Restr\ r\ A \leq Restr\ r\ B$
by *blast*

8.2 Order filters versus restrictions and embeddings

lemma *ofilter-Restr*:
assumes *WELL*: *Well-order r* **and**
    OFA: *ofilter r A* **and** *OFB*: *ofilter r B* **and** *SUB*: $A \leq B$
shows *ofilter* (*Restr r B*) *A*
proof−
  let *?rB* = *Restr r B*
  have *Well*: *wo-rel r* **unfolding** *wo-rel-def* **using** *WELL* **.**
  hence *Refl*: *Refl r* **by** (*auto simp add*: *wo-rel.REFL*)
  hence *Field*: *Field ?rB* = *Field r Int B*
  using *Refl-Field-Restr* **by** *blast*
  have *WellB*: *wo-rel ?rB* ∧ *Well-order ?rB* **using** *WELL*
  by (*auto simp add*: *Well-order-Restr wo-rel-def*)

  show *?thesis*
  proof(*auto simp add*: *WellB wo-rel.ofilter-def*)
    fix *a* **assume** $a \in A$
    hence $a \in Field\ r \wedge a \in B$ **using** *assms Well*

by (*auto simp add*: *wo-rel.ofilter-def*)
with *Field* **show** $a \in$ *Field*(*Restr r B*) **by** *auto*
next
fix *a b* **assume** $*$: $a \in A$ **and** $b \in$ *under* (*Restr r B*) *a*
hence $b \in$ *under r a*
using *WELL OFB SUB ofilter-Restr-under*[*of r B a*] **by** *auto*
thus $b \in A$ **using** $*$ *Well OFA* **by**(*auto simp add*: *wo-rel.ofilter-def*)
qed
qed

lemma *ofilter-subset-iso*:
assumes *WELL*: *Well-order r* **and**
OFA: *ofilter r A* **and** *OFB*: *ofilter r B*
shows $(A = B) =$ *iso* (*Restr r A*) (*Restr r B*) *id*
using *assms*
by (*auto simp add*: *ofilter-subset-embedS-iso*)

8.3 Ordering the well-orders by existence of embeddings

corollary *ordLeq-refl-on*: *refl-on* {*r*. *Well-order r*} *ordLeq*
using *ordLeq-reflexive* **unfolding** *ordLeq-def refl-on-def*
by *blast*

corollary *ordLeq-trans*: *trans ordLeq*
using *trans-def*[*of ordLeq*] *ordLeq-transitive* **by** *blast*

corollary *ordLeq-preorder-on*: *preorder-on* {*r*. *Well-order r*} *ordLeq*
by(*auto simp add*: *preorder-on-def ordLeq-refl-on ordLeq-trans*)

corollary *ordIso-refl-on*: *refl-on* {*r*. *Well-order r*} *ordIso*
using *ordIso-reflexive* **unfolding** *refl-on-def ordIso-def*
by *blast*

corollary *ordIso-trans*: *trans ordIso*
using *trans-def*[*of ordIso*] *ordIso-transitive* **by** *blast*

corollary *ordIso-sym*: *sym ordIso*
by (*auto simp add*: *sym-def ordIso-symmetric*)

corollary *ordIso-equiv*: *equiv* {*r*. *Well-order r*} *ordIso*
by (*auto simp add*: *equiv-def ordIso-sym ordIso-refl-on ordIso-trans*)

lemma *ordLess-Well-order-simp*[*simp*]:
assumes $r <o\ r'$
shows *Well-order r* $\wedge$ *Well-order* r'
using *assms* **unfolding** *ordLess-def* **by** *simp*

lemma *ordIso-Well-order-simp*[*simp*]:
assumes $r =o\ r'$

shows *Well-order r* $\wedge$ *Well-order r'*
using *assms* **unfolding** *ordIso-def* **by** *simp*

lemma *ordLess-irrefl*: *irrefl ordLess*
by(*unfold irrefl-def*, *auto simp add*: *ordLess-irreflexive*)

lemma *ordLess-or-ordIso*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'*
shows $r <o\ r' \vee r' <o\ r \vee r =o\ r'$
unfolding *ordLess-def ordIso-def*
using *assms embedS-or-iso*[*of r r'*] **by** *auto*

corollary *ordLeq-ordLess-Un-ordIso*:
ordLeq = *ordLess* $\cup$ *ordIso*
by (*auto simp add*: *ordLeq-iff-ordLess-or-ordIso*)

lemma *not-ordLeq-ordLess*:
$r \leq o\ r' \Longrightarrow \neg\ r' <o\ r$
using *not-ordLess-ordLeq* **by** *blast*

lemma *ordIso-or-ordLess*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'*
shows $r =o\ r' \vee r <o\ r' \vee r' <o\ r$
using *assms ordLess-or-ordLeq ordLeq-iff-ordLess-or-ordIso* **by** *blast*

lemmas *ord-trans* = *ordIso-transitive ordLeq-transitive ordLess-transitive*
ordIso-ordLeq-trans ordLeq-ordIso-trans
ordIso-ordLess-trans ordLess-ordIso-trans
ordLess-ordLeq-trans ordLeq-ordLess-trans

lemma *ofilter-ordLeq*:
assumes *Well-order r* **and** *ofilter r A*
shows *Restr r A* $\leq o$ *r*
proof−
have $A \leq$ *Field r* **using** *assms* **by** (*auto simp add*: *wo-rel-def wo-rel.ofilter-def*)
thus *?thesis* **using** *assms*
by (*simp add*: *ofilter-subset-ordLeq wo-rel.Field-ofilter*
wo-rel-def Restr-Field)
qed

corollary *under-Restr-ordLeq*:
Well-order r $\Longrightarrow$ *Restr r* (*under r a*) $\leq o$ *r*
by (*auto simp add*: *ofilter-ordLeq wo-rel.under-ofilter wo-rel-def*)

8.4 Copy via direct images

lemma *Id-dir-image*: *dir-image Id f* $\leq$ *Id*
unfolding *dir-image-def* **by** *auto*

lemma *Un-dir-image*:
dir-image $(r1 \cup r2)$ f = (*dir-image* $r1$ f) $\cup$ (*dir-image* $r2$ f)
unfolding *dir-image-def* **by** *auto*

lemma *Int-dir-image*:
assumes *inj-on* f (*Field* $r1$ $\cup$ *Field* $r2$)
shows *dir-image* ($r1$ *Int* $r2$) f = (*dir-image* $r1$ f) *Int* (*dir-image* $r2$ f)
proof
  show *dir-image* ($r1$ *Int* $r2$) $f \le$ (*dir-image* $r1$ f) *Int* (*dir-image* $r2$ f)
  using *assms* **unfolding** *dir-image-def inj-on-def* **by** *auto*
next
  show (*dir-image* $r1$ f) *Int* (*dir-image* $r2$ f) $\le$ *dir-image* ($r1$ *Int* $r2$) f
  proof(*clarify*)
    fix a' b'
    assume $(a',b') \in$ *dir-image* $r1$ f $(a',b') \in$ *dir-image* $r2$ f
    then obtain $a1$ $b1$ $a2$ $b2$
    where *1*: $a' = f\,a1 \wedge b' = f\,b1 \wedge a' = f\,a2 \wedge b' = f\,b2$ **and**
          2: $(a1,b1) \in r1 \wedge (a2,b2) \in r2$ **and**
          3: $\{a1,b1\} \le$ *Field* $r1$ $\wedge$ $\{a2,b2\} \le$ *Field* $r2$
    unfolding *dir-image-def Field-def* **by** *blast*
    hence $a1 = a2 \wedge b1 = b2$ **using** *assms* **unfolding** *inj-on-def* **by** *auto*
    hence $a' = f\,a1 \wedge b' = f\,b1 \wedge (a1,b1) \in r1$ *Int* $r2 \wedge (a2,b2) \in r1$ *Int* $r2$
    using *1 2* **by** *auto*
    thus $(a',b') \in$ *dir-image* $(r1 \cap r2)$ f
    unfolding *dir-image-def* **by** *blast*
  qed
qed

lemma *Osum-embed*:
assumes *FLD*: *Field* r *Int* *Field* $r' = \{\}$ **and**
        WELL: *Well-order* r **and** *WELL'*: *Well-order* r'
shows *embed* r (r *Osum* r') *id*
proof−
  have *1*: *Well-order* (r *Osum* r')
  using *assms* **by** (*auto simp add*: *Osum-Well-order*)
  moreover
  have *compat* r (r *Osum* r') *id*
  unfolding *compat-def Osum-def* **by** *auto*
  moreover
  have *inj-on id* (*Field* r) **by** *simp*
  moreover
  have *ofilter* (r *Osum* r') (*Field* r)
  using *1* **proof**(*auto simp add*: *wo-rel-def wo-rel.ofilter-def*
                          Field-Osum under-def)
    fix a b **assume** *2*: $a \in$ *Field* r **and** *3*: $(b,a) \in r$ *Osum* r'
    moreover
    {**assume** $(b,a) \in r'$

hence $a \in$ *Field* r' **using** *Field-def*[*of* r'] **by** *blast*
hence *False* **using** *2 FLD* **by** *blast*
}
moreover
{**assume** $a \in$ *Field* r'
hence *False* **using** *2 FLD* **by** *blast*
}
ultimately
show $b \in$ *Field* r **by** (*auto simp add*: *Osum-def Field-def*)
qed
ultimately show *?thesis*
using *assms* **by** (*auto simp add*: *embed-iff-compat-inj-on-ofilter*)
qed

corollary *Osum-ordLeq*:
assumes *FLD*: *Field* r *Int Field* $r' = \{\}$ **and**
WELL: *Well-order* r **and** *WELL'*: *Well-order* r'
shows $r \leq o\ r$ *Osum* r'
using *assms Osum-embed Osum-Well-order*
unfolding *ordLeq-def* **by** *blast*

lemma *Well-order-embed-copy*:
assumes *WELL*: *well-order-on* $A\ r$ **and**
INJ: *inj-on* $f\ A$ **and** *SUB*: f ‘ $A \leq B$
shows $\exists r'.$ *well-order-on* $B\ r' \wedge r \leq o\ r'$
proof−
have *bij-betw* $f\ A$ (f ‘ A)
using *INJ inj-on-imp-bij-betw* **by** *blast*
then obtain r'' **where** *well-order-on* (f ‘ A) r'' **and** *1*: $r =o\ r''$
using *WELL Well-order-iso-copy* **by** *blast*
hence *2*: *Well-order* $r'' \wedge$ *Field* $r'' = (f$ ‘ $A)$
using *well-order-on-Well-order* **by** *blast*

let *?C* $= B - (f$ ‘ $A)$
obtain r''' **where** *well-order-on ?C* r'''
using *well-order-on* **by** *blast*
hence *3*: *Well-order* $r''' \wedge$ *Field* $r''' =$ *?C*
using *well-order-on-Well-order* **by** *blast*

let *?r'* $= r''$ *Osum* r'''
have *Field* r'' *Int Field* $r''' = \{\}$
using *2 3* **by** *auto*
hence $r'' \leq o$ *?r'* **using** *Osum-ordLeq*[*of* $r''\ r'''$] *2 3* **by** *blast*
hence *4*: $r \leq o$ *?r'* **using** *1 ordIso-ordLeq-trans* **by** *blast*

hence *Well-order ?r'* **unfolding** *ordLeq-def* **by** *auto*
moreover
have *Field ?r'* $= B$ **using** *2 3 SUB* **by** (*auto simp add*: *Field-Osum*)
ultimately show *?thesis* **using** *4* **by** *blast*

qed

8.5 The maxim among a finite set of ordinals

The correct phrasing would be “a maxim of ...", as $\leq o$ is only a preorder.

definition *isOmax* :: $'a$ *rel set* $\Rightarrow$ $'a$ *rel* $\Rightarrow$ *bool*
where
isOmax $R\ r \equiv r \in R \wedge (\forall r' \in R.\ r' \leq o\ r)$

definition *omax* :: $'a$ *rel set* $\Rightarrow$ $'a$ *rel*
where
omax R == *SOME* r. *isOmax* $R\ r$

lemma *exists-isOmax*:
assumes *finite* R **and** $R \neq \{\}$ **and** $\forall\ r \in R.$ *Well-order* r
shows $\exists\ r.$ *isOmax* $R\ r$
proof−
  have *finite* $R \Longrightarrow R \neq \{\} \longrightarrow (\forall\ r \in R.$ *Well-order* $r) \longrightarrow (\exists\ r.$ *isOmax* $R\ r)$
  apply(*erule finite-induct*) **apply**(*simp add*: *isOmax-def*)
  proof(*clarsimp*)
    fix r :: $('a \times 'a)$ *set* **and** R **assume** $*$: *finite* R **and** $**$: $r \notin R$
    and $***$: *Well-order* r **and** $****$: $\forall r{\in}R.$ *Well-order* r
    and *IH*: $R \neq \{\} \longrightarrow (\exists p.$ *isOmax* $R\ p)$
    let $?R'$ = *insert* $r\ R$
    show $\exists p'.$ (*isOmax* $?R'\ p'$)
    proof(*cases* $R = \{\}$)
      assume *Case1*: $R = \{\}$
      thus *?thesis* **unfolding** *isOmax-def* **using** $***$
      by (*simp add*: *ordLeq-reflexive*)
    next
      assume *Case2*: $R \neq \{\}$
      then obtain p **where** p: *isOmax* $R\ p$ **using** *IH* **by** *auto*
      hence *1*: *Well-order* p **using** $****$ **unfolding** *isOmax-def* **by** *simp*
      {assume *Case21*: $r \leq o\ p$
       hence *isOmax* $?R'\ p$ **using** p **unfolding** *isOmax-def* **by** *simp*
       hence *?thesis* **by** *auto*
      }
      moreover
      {assume *Case22*: $p \leq o\ r$
       {fix r' **assume** $r' \in ?R'$
        moreover
        {assume $r' \in R$
         hence $r' \leq o\ p$ **using** p **unfolding** *isOmax-def* **by** *simp*
         hence $r' \leq o\ r$ **using** *Case22* **by**(*rule ordLeq-transitive*)
        }
        moreover have $r \leq o\ r$ **using** $***$ **by**(*rule ordLeq-reflexive*)
        ultimately have $r' \leq o\ r$ **by** *auto*
       }
       hence *isOmax* $?R'\ r$ **unfolding** *isOmax-def* **by** *simp*

```
      hence ?thesis by auto
      }
      moreover have r ≤o p ∨ p ≤o r
      using 1 *** ordLeq-total by auto
      ultimately show ?thesis by blast
    qed
  qed
  thus ?thesis using assms by auto
qed

lemma omax-isOmax:
assumes finite R and R ≠ {} and ∀ r ∈ R. Well-order r
shows isOmax R (omax R)
unfolding omax-def using assms
by(simp add: exists-isOmax someI-ex)

lemma omax-in:
assumes finite R and R ≠ {} and ∀ r ∈ R. Well-order r
shows omax R ∈ R
using assms omax-isOmax unfolding isOmax-def by blast

lemma Well-order-omax:
assumes finite R and R ≠ {} and ∀r∈R. Well-order r
shows Well-order (omax R)
using assms apply − apply(drule omax-in) by auto

lemma omax-maxim:
assumes finite R and ∀ r ∈ R. Well-order r and r ∈ R
shows r ≤o omax R
using assms omax-isOmax unfolding isOmax-def by blast

lemma omax-ordLeq:
assumes finite R and R ≠ {} and *: ∀ r ∈ R. r ≤o p
shows omax R ≤o p
proof−
  have ∀ r ∈ R. Well-order r using * unfolding ordLeq-def by simp
  thus ?thesis using assms omax-in by auto
qed

lemma omax-ordLess:
assumes finite R and R ≠ {} and *: ∀ r ∈ R. r <o p
shows omax R <o p
proof−
  have ∀ r ∈ R. Well-order r using * unfolding ordLess-def by simp
  thus ?thesis using assms omax-in by auto
qed

lemma omax-ordLeq-elim:
assumes finite R and ∀ r ∈ R. Well-order r
```

and *omax* $R \leq o$ p **and** $r \in R$
shows $r \leq o$ p
using *assms omax-maxim*[*of R r*] **apply** *simp*
using *ordLeq-transitive* **by** *blast*

lemma *omax-ordLess-elim*:
assumes *finite R* **and** $\forall$ $r \in R$. *Well-order r*
and *omax* $R < o$ p **and** $r \in R$
shows $r < o$ p
using *assms omax-maxim*[*of R r*] **apply** *simp*
using *ordLeq-ordLess-trans* **by** *blast*

lemma *ordLeq-omax*:
assumes *finite R* **and** $\forall$ $r \in R$. *Well-order r*
and $r \in R$ **and** $p \leq o$ r
shows $p \leq o$ *omax R*
using *assms omax-maxim*[*of R r*] **apply** *simp*
using *ordLeq-transitive* **by** *blast*

lemma *ordLess-omax*:
assumes *finite R* **and** $\forall$ $r \in R$. *Well-order r*
and $r \in R$ **and** $p < o$ r
shows $p < o$ *omax R*
using *assms omax-maxim*[*of R r*] **apply** *simp*
using *ordLess-ordLeq-trans* **by** *blast*

lemma *omax-ordLeq-mono*:
assumes *P*: *finite P* **and** *R*: *finite R*
and *NE-P*: $P \neq \{\}$ **and** *Well-R*: $\forall$ $r \in R$. *Well-order r*
and *LEQ*: $\forall$ $p \in P$. $\exists$ $r \in R$. $p \leq o$ r
shows *omax* $P \leq o$ *omax R*
proof−
  let *?mp* = *omax P* **let** *?mr* = *omax R*
  {fix *p* **assume** $p \in P$
   then obtain *r* **where** *r*: $r \in R$ **and** $p \leq o$ r
   using *LEQ* **by** *blast*
   moreover have *r* <=*o* *?mr*
   using *r R Well-R omax-maxim* **by** *blast*
   ultimately have *p* <=*o* *?mr*
   using *ordLeq-transitive* **by** *blast*
  }
  thus *?mp* <=*o* *?mr*
  using *NE-P P* **using** *omax-ordLeq* **by** *blast*
qed

lemma *omax-ordLess-mono*:
assumes *P*: *finite P* **and** *R*: *finite R*
and *NE-P*: $P \neq \{\}$ **and** *Well-R*: $\forall$ $r \in R$. *Well-order r*
and *LEQ*: $\forall$ $p \in P$. $\exists$ $r \in R$. $p < o$ r

shows *omax P* $<o$ *omax R*
proof−
  let *?mp* = *omax P* **let** *?mr* = *omax R*
  {fix *p* **assume** $p \in P$
   then obtain *r* **where** *r*: $r \in R$ **and** $p <o\ r$
   using *LEQ* **by** *blast*
   moreover have $r <=o$ *?mr*
   using *r R Well-R omax-maxim* **by** *blast*
   ultimately have $p <o$ *?mr*
   using *ordLess-ordLeq-trans* **by** *blast*
  }
  thus *?mp* $<o$ *?mr*
  using *NE-P P omax-ordLess* **by** *blast*
qed

8.6 Limit and succesor ordinals

lemma *embed-underS2*:
assumes *r*: *Well-order r* **and** *g*: *embed r s g* **and** *a*: $a \in$ *Field r*
shows *g* ‘ *underS r a* = *underS s* (*g a*)
  by (*meson a bij-betw-def embed-underS g r*)

lemma *bij-betw-insert*:
assumes $b \notin A$ **and** $f\ b \notin A'$ **and** *bij-betw f A A*′
shows *bij-betw f* (*insert b A*) (*insert* (*f b*) *A*′)
using *notIn-Un-bij-betw*[*OF assms*] **by** *auto*

context *wo-rel*
begin

lemma *underS-induct*:
  assumes $\bigwedge a.\ (\bigwedge a1.\ a1 \in$ *underS a* $\Longrightarrow P\ a1) \Longrightarrow P\ a$
  shows *P a*
  by (*induct rule*: *well-order-induct*) (*rule assms*, *simp add*: *underS-def*)

lemma *suc-underS*:
assumes *B*: $B \subseteq$ *Field r* **and** *A*: *AboveS B* $\neq \{\}$ **and** *b*: $b \in B$
shows $b \in$ *underS* (*suc B*)
using *suc-AboveS*[*OF B A*] *b* **unfolding** *underS-def AboveS-def* **by** *auto*

lemma *underS-supr*:
assumes *bA*: $b \in$ *underS* (*supr A*) **and** *A*: $A \subseteq$ *Field r*
shows $\exists\ a \in A.\ b \in$ *underS a*
proof(*rule ccontr*, *auto*)
  have *bb*: $b \in$ *Field r* **using** *bA* **unfolding** *underS-def Field-def* **by** *auto*
  assume $\forall a{\in}A.\ \ b \notin$ *underS a*
  hence *0*: $\forall a \in A.\ (a,b) \in r$ **using** *A bA* **unfolding** *underS-def*
  by *simp* (*metis REFL in-mono max2-def max2-greater refl-on-domain*)
  have (*supr A*, *b*) $\in r$ **apply**(*rule supr-least*[*OF A bb*]) **using** *0* **by** *auto*

thus *False* **using** *bA* **unfolding** *underS-def* **by** *simp* (*metis ANTISYM antisymD*)
qed

lemma *underS-suc*:
assumes *bA*: $b \in underS\ (suc\ A)$ **and** *A*: $A \subseteq Field\ r$
shows $\exists\ a \in A.\ b \in under\ a$
proof(*rule ccontr*, *auto*)
have *bb*: $b \in Field\ r$ **using** *bA* **unfolding** *underS-def Field-def* **by** *auto*
assume $\forall\, a{\in}A.\ \ b \notin under\ a$
hence *0*: $\forall\, a \in A.\ a \in underS\ b$ **using** *A bA* **unfolding** *underS-def*
by *simp* (*metis* (*lifting*) *bb max2-def max2-greater mem-Collect-eq under-def rev-subsetD*)
have $(suc\ A,\ b) \in r$ **apply**(*rule suc-least*[*OF A bb*]) **using** *0* **unfolding** *underS-def* **by** *auto*
thus *False* **using** *bA* **unfolding** *underS-def* **by** *simp* (*metis ANTISYM antisymD*)
qed

lemma (**in** *wo-rel*) *in-underS-supr*:
assumes *j*: $j \in underS\ i$ **and** *i*: $i \in A$ **and** *A*: $A \subseteq Field\ r$ **and** *AA*: $Above\ A \neq \{\}$
shows $j \in underS\ (supr\ A)$
proof−
have $(i{,}supr\ A) \in r$ **using** *supr-greater*[*OF A AA i*] **.**
thus *?thesis* **using** *j* **unfolding** *underS-def*
by *simp* (*metis REFL TRANS max2-def max2-equals1 refl-on-domain transD*)
qed

lemma *inj-on-Field*:
assumes *A*: $A \subseteq Field\ r$ **and** *f*: $\bigwedge\ a\ b.\ [\![a \in A;\ b \in A;\ a \in underS\ b]\!] \Longrightarrow f\ a \neq f\ b$
shows *inj-on f A*
unfolding *inj-on-def* **proof** *safe*
fix *a b* **assume** *a*: $a \in A$ **and** *b*: $b \in A$ **and** *fab*: $f\ a = f\ b$
{**assume** $a \in underS\ b$
hence *False* **using** *f*[*OF a b*] *fab* **by** *auto*
}
moreover
{**assume** $b \in underS\ a$
hence *False* **using** *f*[*OF b a*] *fab* **by** *auto*
}
ultimately show $a = b$ **using** *TOTALS A a b* **unfolding** *underS-def* **by** *auto*
qed

lemma *in-notinI*:
assumes $(j{,}i) \notin r \lor j = i$ **and** $i \in Field\ r$ **and** $j \in Field\ r$
shows $(i{,}j) \in r$ **by** (*metis assms max2-def max2-greater-among*)

lemma *ofilter-init-seg-of*:
assumes *ofilter F*
shows *Restr r F initial-segment-of r*
using *assms* **unfolding** *ofilter-def init-seg-of-def under-def* **by** *auto*

lemma *underS-init-seg-of-Collect*:
assumes $\bigwedge$*j1 j2*. ⟦*j2* ∈ *underS i*; (*j1*, *j2*) ∈ *r*⟧ ⟹ *R j1 initial-segment-of R j2*
shows {*R j* |*j*. *j* ∈ *underS i*} ∈ *Chains init-seg-of*
unfolding *Chains-def* **proof** *safe*
  fix *j ja* **assume** *jS*: *j* ∈ *underS i* **and** *jaS*: *ja* ∈ *underS i*
  and *init*: (*R ja*, *R j*) ∉ *init-seg-of*
  hence *jja*: {*j*,*ja*} ⊆ *Field r* **and** *j*: *j* ∈ *Field r* **and** *ja*: *ja* ∈ *Field r*
  and *jjai*: (*j*,*i*) ∈ *r* (*ja*,*i*) ∈ *r*
  and *i*: *i* ∉ {*j*,*ja*} **unfolding** *Field-def underS-def* **by** *auto*
  have *jj*: (*j*,*j*) ∈ *r* **and** *jaja*: (*ja*,*ja*) ∈ *r* **using** *j ja* **by** (*metis in-notinI*)+
  show *R j initial-segment-of R ja*
  using *jja init jjai i*
   by (*elim cases-Total3 disjE*) (*auto elim*: *cases-Total3 intro*!: *assms simp*: *underS-def*)
qed

lemma (**in** *wo-rel*) *Field-init-seg-of-Collect*:
assumes $\bigwedge$*j1 j2*. ⟦*j2* ∈ *Field r*; (*j1*, *j2*) ∈ *r*⟧ ⟹ *R j1 initial-segment-of R j2*
shows {*R j* |*j*. *j* ∈ *Field r*} ∈ *Chains init-seg-of*
unfolding *Chains-def* **proof** *safe*
  fix *j ja* **assume** *jS*: *j* ∈ *Field r* **and** *jaS*: *ja* ∈ *Field r*
  and *init*: (*R ja*, *R j*) ∉ *init-seg-of*
  hence *jja*: {*j*,*ja*} ⊆ *Field r* **and** *j*: *j* ∈ *Field r* **and** *ja*: *ja* ∈ *Field r*
  unfolding *Field-def underS-def* **by** *auto*
  have *jj*: (*j*,*j*) ∈ *r* **and** *jaja*: (*ja*,*ja*) ∈ *r* **using** *j ja* **by** (*metis in-notinI*)+
  show *R j initial-segment-of R ja*
  using *jja init*
 by (*elim cases-Total3 disjE*) (*auto elim*: *cases-Total3 intro*!: *assms simp*: *Field-def*)
qed

8.6.1 Successor and limit elements of an ordinal

definition *succ i* ≡ *suc* {*i*}

definition *isSucc i* ≡ ∃ *j*. *aboveS j* ≠ {} ∧ *i* = *succ j*

definition *zero* = *minim* (*Field r*)

definition *isLim i* ≡ ¬ *isSucc i*

lemma *zero-smallest*[*simp*]:
assumes *j* ∈ *Field r* **shows** (*zero*, *j*) ∈ *r*
unfolding *zero-def*
by (*metis AboveS-Field assms subset-AboveS-UnderS subset-antisym subset-refl suc-def*

suc-least-AboveS)

lemma *zero-in-Field*: **assumes** *Field r* $\neq$ {} **shows** *zero* $\in$ *Field r*
using *assms* **unfolding** *zero-def* **by** (*metis Field-ofilter minim-in ofilter-def*)

lemma *leq-zero-imp*[*simp*]:
(*x*, *zero*) $\in$ *r* $\Longrightarrow$ *x* = *zero*
by (*metis ANTISYM WELL antisymD well-order-on-domain zero-smallest*)

lemma *leq-zero*[*simp*]:
assumes *Field r* $\neq$ {} **shows** (*x*, *zero*) $\in$ *r* $\longleftrightarrow$ *x* = *zero*
using *zero-in-Field*[*OF assms*] *in-notinI*[*of x zero*] **by** *auto*

lemma *under-zero*[*simp*]:
assumes *Field r* $\neq$ {} **shows** *under zero* = {*zero*}
using *assms* **unfolding** *under-def* **by** *auto*

lemma *underS-zero*[*simp*,*intro*]: *underS zero* = {}
unfolding *underS-def* **by** *auto*

lemma *isSucc-succ*: *aboveS i* $\neq$ {} $\Longrightarrow$ *isSucc* (*succ i*)
unfolding *isSucc-def succ-def* **by** *auto*

lemma *succ-in-diff*:
assumes *aboveS i* $\neq$ {} **shows** (*i*,*succ i*) $\in$ *r* $\wedge$ *succ i* $\neq$ *i*
using *assms suc-greater*[*of* {*i*}] **unfolding** *succ-def AboveS-def aboveS-def Field-def*
by *auto*

lemmas *succ-in*[*simp*] = *succ-in-diff*[*THEN conjunct1*]
lemmas *succ-diff*[*simp*] = *succ-in-diff*[*THEN conjunct2*]

lemma *succ-in-Field*[*simp*]:
assumes *aboveS i* $\neq$ {} **shows** *succ i* $\in$ *Field r*
using *succ-in*[*OF assms*] **unfolding** *Field-def* **by** *auto*

lemma *succ-not-in*:
assumes *aboveS i* $\neq$ {} **shows** (*succ i*, *i*) $\notin$ *r*
proof
  assume *1*: (*succ i*, *i*) $\in$ *r*
  hence *succ i* $\in$ *Field r* $\wedge$ *i* $\in$ *Field r* **unfolding** *Field-def* **by** *auto*
  hence (*i*, *succ i*) $\in$ *r* $\wedge$ *succ i* $\neq$ *i* **using** *assms* **by** *auto*
  thus *False* **using** *1* **by** (*metis ANTISYM antisymD*)
qed

lemma *not-isSucc-zero*: $\neg$ *isSucc zero*
proof
  assume $*$: *isSucc zero*
  then obtain *i* **where** *aboveS i* $\neq$ {} **and** *1*: *minim* (*Field r*) = *succ i*
  unfolding *isSucc-def zero-def* **by** *auto*

hence *succ i* $\in$ *Field r* **by** *auto*
with $*$ **show** *False*
by (*metis REFL isSucc-def minim-least refl-on-domain subset-refl succ-in succ-not-in zero-def*)
qed

lemma *isLim-zero*[*simp*]: *isLim zero*
by (*metis isLim-def not-isSucc-zero*)

lemma *succ-smallest*:
assumes $(i,j) \in r$ **and** $i \neq j$
shows $(succ\ i,\ j) \in r$
unfolding *succ-def* **apply**(*rule suc-least*)
using *assms* **unfolding** *Field-def* **by** *auto*

lemma *isLim-supr*:
assumes *f*: $i \in$ *Field r* **and** *l*: *isLim i*
shows $i = supr\ (underS\ i)$
proof(*rule equals-supr*)
fix *j* **assume** *j*: $j \in$ *Field r* **and** *1*: $\bigwedge j'.\ j' \in underS\ i \Longrightarrow (j',j) \in r$
show $(i,j) \in r$ **proof**(*intro in-notinI*[*OF - f j*], *safe*)
assume *ji*: $(j,i) \in r\ j \neq i$
hence *a*: *aboveS j* $\neq \{\}$ **unfolding** *aboveS-def* **by** *auto*
hence $i \neq succ\ j$ **using** *l* **unfolding** *isLim-def isSucc-def* **by** *auto*
moreover have $(succ\ j,\ i) \in r$ **using** *succ-smallest*[*OF ji*] **by** *auto*
ultimately have *succ j* $\in$ *underS i* **unfolding** *underS-def* **by** *auto*
hence $(succ\ j,\ j) \in r$ **using** *1* **by** *auto*
thus *False* **using** *succ-not-in*[*OF a*] **by** *simp*
qed
qed(*insert f*, *unfold underS-def Field-def*, *auto*)

definition *pred i* $\equiv$ *SOME j. j* $\in$ *Field r* $\wedge$ *aboveS j* $\neq \{\}$ $\wedge$ *succ j* $=$ *i*

lemma *pred-Field-succ*:
assumes *isSucc i* **shows** *pred i* $\in$ *Field r* $\wedge$ *aboveS* (*pred i*) $\neq \{\}$ $\wedge$ *succ* (*pred i*) $=$ *i*
proof−
obtain *j* **where** *a*: *aboveS j* $\neq \{\}$ **and** *i*: $i = succ\ j$ **using** *assms* **unfolding** *isSucc-def* **by** *auto*
have *1*: $j \in$ *Field r* $j \neq i$ **unfolding** *Field-def i*
using *succ-diff*[*OF a*] *a* **unfolding** *aboveS-def* **by** *auto*
show *?thesis* **unfolding** *pred-def* **apply**(*rule someI-ex*) **using** *1 i a* **by** *auto*
qed

lemmas *pred-Field*[*simp*] = *pred-Field-succ*[*THEN conjunct1*]
lemmas *aboveS-pred*[*simp*] = *pred-Field-succ*[*THEN conjunct2*, *THEN conjunct1*]
lemmas *succ-pred*[*simp*] = *pred-Field-succ*[*THEN conjunct2*, *THEN conjunct2*]

lemma *isSucc-pred-in*:

assumes *isSucc i* **shows** $(pred\ i,\ i) \in r$
proof−
  define *j* **where** $j = pred\ i$
  have *i*: $i = succ\ j$ **using** *assms* **unfolding** *j-def* **by** *simp*
  have *a*: $aboveS\ j \neq \{\}$ **unfolding** *j-def* **using** *assms* **by** *auto*
  show *?thesis* **unfolding** *j-def*[*symmetric*] **unfolding** *i* **using** *succ-in*[*OF a*] **.**
qed

lemma *isSucc-pred-diff*:
assumes *isSucc i* **shows** $pred\ i \neq i$
by (*metis aboveS-pred assms succ-diff succ-pred*)

lemma *succ-inj*[*simp*]:
assumes $aboveS\ i \neq \{\}$ **and** $aboveS\ j \neq \{\}$
shows $succ\ i = succ\ j \longleftrightarrow i = j$
proof *safe*
  assume *s*: $succ\ i = succ\ j$
  {assume $i \neq j$ **and** $(i,j) \in r$
   hence $(succ\ i,\ j) \in r$ **using** *assms* **by** (*metis succ-smallest*)
   hence *False* **using** *s assms* **by** (*metis succ-not-in*)
  }
  moreover
  {assume $i \neq j$ **and** $(j,i) \in r$
   hence $(succ\ j,\ i) \in r$ **using** *assms* **by** (*metis succ-smallest*)
   hence *False* **using** *s assms* **by** (*metis succ-not-in*)
  }
  ultimately show $i = j$ **by** (*metis TOTALS WELL assms*(*1*) *assms*(*2*) *succ-in-diff well-order-on-domain*)
qed

lemma *pred-succ*[*simp*]:
assumes $aboveS\ j \neq \{\}$ **shows** $pred\ (succ\ j) = j$
unfolding *pred-def* **apply**(*rule some-equality*)
using *assms* **apply**(*force simp*: *Field-def aboveS-def*)
by (*metis assms succ-inj*)

lemma *less-succ*[*simp*]:
assumes $aboveS\ i \neq \{\}$
shows $(j,\ succ\ i) \in r \longleftrightarrow (j,i) \in r \lor j = succ\ i$
apply *safe*
 apply (*metis WELL assms in-notinI well-order-on-domain suc-singl-pred succ-def succ-in-diff*)
 apply (*metis* (*opaque-lifting*, *full-types*) *REFL TRANS assms max2-def max2-equals1 refl-on-domain succ-in-Field succ-not-in transD*)
 apply (*metis assms in-notinI succ-in-Field*)
done

lemma *underS-succ*[*simp*]:
assumes *aboveS* $i \neq \{\}$
shows *underS* (*succ* i) = *under* i
unfolding *underS-def under-def* **by** (*auto simp*: *assms succ-not-in*)

lemma *succ-mono*:
assumes *aboveS* $j \neq \{\}$ **and** $(i,j) \in r$
shows (*succ* i, *succ* j) $\in r$
by (*metis* (*full-types*) *assms less-succ succ-smallest*)

lemma *under-succ*[*simp*]:
assumes *aboveS* $i \neq \{\}$
shows *under* (*succ* i) = *insert* (*succ* i) (*under* i)
using *less-succ*[*OF assms*] **unfolding** *under-def* **by** *auto*

definition *mergeSL* :: $('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow (('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b) \Rightarrow$ $'a \Rightarrow 'b$
where
mergeSL S L f i $\equiv$
if isSucc i then S (*pred i*) (*f* (*pred i*))
else L f i

8.6.2 Well-order recursion with (zero), succesor, and limit

definition *worecSL* :: $('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow (('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$
where *worecSL S L* $\equiv$ *worec* (*mergeSL S L*)

definition *adm-woL L* $\equiv \forall f\ g\ i.$ *isLim i* $\wedge$ ($\forall j \in$*underS i*. $f\ j = g\ j$) $\longrightarrow L\ f\ i =$ $L\ g\ i$

lemma *mergeSL*:
assumes *adm-woL L* **shows** *adm-wo* (*mergeSL S L*)
unfolding *adm-wo-def* **proof** *safe*
 fix *f g* :: *'a => 'b* **and** *i* :: *'a*
 assume *1*: $\forall j \in$*underS i*. $f\ j = g\ j$
 show *mergeSL S L f i* = *mergeSL S L g i*
 proof(*cases isSucc i*)
  case *True*
   hence *pred i* $\in$ *underS i* **unfolding** *underS-def* **using** *isSucc-pred-in isSucc-pred-diff* **by** *auto*
  thus *?thesis* **using** *True 1* **unfolding** *mergeSL-def* **by** *auto*
 next
  case *False* **hence** *isLim i* **unfolding** *isLim-def* **by** *auto*
   thus *?thesis* **using** *assms False 1* **unfolding** *mergeSL-def adm-woL-def* **by** *auto*
 qed
qed

lemma *worec-fixpoint1*: *adm-wo H* $\Longrightarrow$ *worec H i* = *H* (*worec H*) *i*

by (*metis worec-fixpoint*)

lemma *worecSL-isSucc*:
assumes *a*: *adm-woL L* **and** *i*: *isSucc i*
shows *worecSL S L i* = *S* (*pred i*) (*worecSL S L* (*pred i*))
proof−
  let *?H* = *mergeSL S L*
  have *worecSL S L i* = *?H* (*worec ?H*) *i*
  unfolding *worecSL-def* **using** *worec-fixpoint1*[*OF mergeSL*[*OF a*]] **.**
  also have ... = *S* (*pred i*) (*worecSL S L* (*pred i*))
  unfolding *worecSL-def mergeSL-def* **using** *i* **by** *simp*
  finally show *?thesis* **.**
qed

lemma *worecSL-succ*:
assumes *a*: *adm-woL L* **and** *i*: *aboveS j* ≠ {}
shows *worecSL S L* (*succ j*) = *S j* (*worecSL S L j*)
proof−
  define *i* **where** *i* = *succ j*
  have *i*: *isSucc i* **by** (*metis i i-def isSucc-succ*)
  have *ij*: *j* = *pred i* **unfolding** *i-def* **using** *assms* **by** *simp*
  have *0*: *succ* (*pred i*) = *i* **using** *i* **by** *simp*
  show *?thesis* **unfolding** *ij* **using** *worecSL-isSucc*[*OF a i*] **unfolding** *0* **.**
qed

lemma *worecSL-isLim*:
assumes *a*: *adm-woL L* **and** *i*: *isLim i*
shows *worecSL S L i* = *L* (*worecSL S L*) *i*
proof−
  let *?H* = *mergeSL S L*
  have *worecSL S L i* = *?H* (*worec ?H*) *i*
  unfolding *worecSL-def* **using** *worec-fixpoint1*[*OF mergeSL*[*OF a*]] **.**
  also have ... = *L* (*worecSL S L*) *i*
  using *i* **unfolding** *worecSL-def mergeSL-def isLim-def* **by** *simp*
  finally show *?thesis* **.**
qed

definition *worecZSL* :: $'b \Rightarrow ('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow (('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$
where *worecZSL Z S L* ≡ *worecSL S* (λ *f a*. *if a* = *zero then Z else L f a*)

lemma *worecZSL-zero*:
assumes *a*: *adm-woL L*
shows *worecZSL Z S L zero* = *Z*
proof−
  let *?L* = λ *f a*. *if a* = *zero then Z else L f a*
  have *worecZSL Z S L zero* = *?L* (*worecZSL Z S L*) *zero*
  unfolding *worecZSL-def* **apply**(*rule worecSL-isLim*)
  using *assms* **unfolding** *adm-woL-def* **by** *auto*
  also have ... = *Z* **by** *simp*

finally show *?thesis* **.**
qed

lemma *worecZSL-succ*:
assumes *a*: *adm-woL L* **and** *i*: *aboveS j* $\neq$ {}
shows *worecZSL Z S L* (*succ j*) = *S j* (*worecZSL Z S L j*)
unfolding *worecZSL-def* **apply**(*rule worecSL-succ*)
using *assms* **unfolding** *adm-woL-def* **by** *auto*

lemma *worecZSL-isLim*:
assumes *a*: *adm-woL L* **and** *isLim i* **and** *i* $\neq$ *zero*
shows *worecZSL Z S L i* = *L* (*worecZSL Z S L*) *i*
proof−
  let *?L* = λ *f a*. *if a* = *zero then Z else L f a*
  have *worecZSL Z S L i* = *?L* (*worecZSL Z S L*) *i*
  unfolding *worecZSL-def* **apply**(*rule worecSL-isLim*)
  using *assms* **unfolding** *adm-woL-def* **by** *auto*
  also have ... = *L* (*worecZSL Z S L*) *i* **using** *assms* **by** *simp*
  finally show *?thesis* **.**
qed

8.6.3 Well-order succ-lim induction

lemma *ord-cases*:
obtains *j* **where** *i* = *succ j* **and** *aboveS j* $\neq$ {} | *isLim i*
by (*metis isLim-def isSucc-def*)

lemma *well-order-inductSL*[*case-names Suc Lim*]:
assumes *SUCC*: ⋀*i*. ⟦*aboveS i* $\neq$ {}; *P i*⟧ $\Longrightarrow$ *P* (*succ i*) **and**
LIM: ⋀*i*. ⟦*isLim i*; ⋀*j*. *j* ∈ *underS i* $\Longrightarrow$ *P j*⟧ $\Longrightarrow$ *P i*
shows *P i*
proof(*induction rule*: *well-order-induct*)
  fix *i* **assume** *0*: ∀ *j*. *j* $\neq$ *i* ∧ (*j*, *i*) ∈ *r* ⟶ *P j*
  show *P i* **proof**(*cases i rule*: *ord-cases*)
    fix *j* **assume** *i*: *i* = *succ j* **and** *j*: *aboveS j* $\neq$ {}
    hence *j* $\neq$ *i* ∧ (*j*, *i*) ∈ *r* **by** (*metis succ-diff succ-in*)
    hence *1*: *P j* **using** *0* **by** *simp*
    show *P i* **unfolding** *i* **apply**(*rule SUCC*) **using** *1 j* **by** *auto*
  qed(*insert 0 LIM*, *unfold underS-def*, *auto*)
qed

lemma *well-order-inductZSL*[*case-names Zero Suc Lim*]:
assumes *ZERO*: *P zero*
and *SUCC*: ⋀*i*. ⟦*aboveS i* $\neq$ {}; *P i*⟧ $\Longrightarrow$ *P* (*succ i*) **and**
LIM: ⋀*i*. ⟦*isLim i*; *i* $\neq$ *zero*; ⋀*j*. *j* ∈ *underS i* $\Longrightarrow$ *P j*⟧ $\Longrightarrow$ *P i*
shows *P i*
apply(*induction rule*: *well-order-inductSL*) **using** *assms* **by** *auto*

definition *isSuccOrd* ≡ ∃ *j* ∈ *Field r*. ∀ *i* ∈ *Field r*. (*i*,*j*) ∈ *r*
definition *isLimOrd* ≡ ¬ *isSuccOrd*

lemma *isLimOrd-succ*:
assumes *isLimOrd* **and** *i* ∈ *Field r*
shows *succ i* ∈ *Field r*
using *assms* **unfolding** *isLimOrd-def isSuccOrd-def*
by (*metis REFL in-notinI refl-on-domain succ-smallest*)

lemma *isLimOrd-aboveS*:
assumes *l*: *isLimOrd* **and** *i*: *i* ∈ *Field r*
shows *aboveS i* ≠ {}
proof−
  obtain *j* **where** *j* ∈ *Field r* **and** (*j*,*i*) ∉ *r*
  using *assms* **unfolding** *isLimOrd-def isSuccOrd-def* **by** *auto*
  hence (*i*,*j*) ∈ *r* ∧ *j* ≠ *i* **by** (*metis i max2-def max2-greater*)
  thus *?thesis* **unfolding** *aboveS-def* **by** *auto*
qed

lemma *succ-aboveS-isLimOrd*:
assumes ∀ *i* ∈ *Field r*. *aboveS i* ≠ {} ∧ *succ i* ∈ *Field r*
shows *isLimOrd*
unfolding *isLimOrd-def isSuccOrd-def* **proof** *safe*
  fix *j* **assume** *j*: *j* ∈ *Field r* ∀ *i*∈*Field r*. (*i*, *j*) ∈ *r*
  hence (*succ j*, *j*) ∈ *r* **using** *assms* **by** *auto*
  moreover have *aboveS j* ≠ {} **using** *assms j* **unfolding** *aboveS-def* **by** *auto*
  ultimately show *False* **by** (*metis succ-not-in*)
qed

lemma *isLim-iff*:
assumes *l*: *isLim i* **and** *j*: *j* ∈ *underS i*
shows ∃ *k*. *k* ∈ *underS i* ∧ *j* ∈ *underS k*
proof−
  have *a*: *aboveS j* ≠ {} **using** *j* **unfolding** *underS-def aboveS-def* **by** *auto*
  show *?thesis* **apply**(*rule exI*[*of - succ j*]) **apply** *safe*
  using *assms a* **unfolding** *underS-def isLim-def*
  apply (*metis* (*lifting*, *full-types*) *isSucc-def mem-Collect-eq succ-smallest*)
  by (*metis* (*lifting*, *full-types*) *a mem-Collect-eq succ-diff succ-in*)
qed

end

abbreviation *zero* ≡ *wo-rel.zero*
abbreviation *succ* ≡ *wo-rel.succ*
abbreviation *pred* ≡ *wo-rel.pred*
abbreviation *isSucc* ≡ *wo-rel.isSucc*
abbreviation *isLim* ≡ *wo-rel.isLim*
abbreviation *isLimOrd* ≡ *wo-rel.isLimOrd*
abbreviation *isSuccOrd* ≡ *wo-rel.isSuccOrd*

abbreviation *adm-woL* ≡ *wo-rel.adm-woL*
abbreviation *worecSL* ≡ *wo-rel.worecSL*
abbreviation *worecZSL* ≡ *wo-rel.worecZSL*

8.7 Projections of wellorders

definition *oproj r s f* ≡ *Field s* ⊆ *f* ' (*Field r*) ∧ *compat r s f*

lemma *oproj-in*:
assumes *oproj r s f* **and** (*a*,*a*′) ∈ *r*
shows (*f a*, *f a*′) ∈ *s*
using *assms* **unfolding** *oproj-def compat-def* **by** *auto*

lemma *oproj-Field*:
assumes *f*: *oproj r s f* **and** *a*: *a* ∈ *Field r*
shows *f a* ∈ *Field s*
using *oproj-in*[*OF f*] *a* **unfolding** *Field-def* **by** *auto*

lemma *oproj-Field2*:
assumes *f*: *oproj r s f* **and** *a*: *b* ∈ *Field s*
shows ∃ *a* ∈ *Field r*. *f a* = *b*
using *assms* **unfolding** *oproj-def* **by** *auto*

lemma *oproj-under*:
assumes *f*: *oproj r s f* **and** *a*: *a* ∈ *under r a*′
shows *f a* ∈ *under s* (*f a*′)
using *oproj-in*[*OF f*] *a* **unfolding** *under-def* **by** *auto*

theorem *embedI*:
assumes *r*: *Well-order r* **and** *s*: *Well-order s*
and *f*: ⋀ *a*. *a* ∈ *Field r* ⟹ *f a* ∈ *Field s* ∧ *f* ' *underS r a* ⊆ *underS s* (*f a*)
shows ∃ *g*. *embed r s g*
proof−
  interpret *r*: *wo-rel r* **by** *unfold-locales* (*rule r*)
  interpret *s*: *wo-rel s* **by** *unfold-locales* (*rule s*)
  let *?G* = λ *g a*. *suc s* (*g* ' *underS r a*)
  define *g* **where** *g* = *worec r ?G*
  have *adm*: *adm-wo r ?G* **unfolding** *r.adm-wo-def image-def* **by** *auto*

  {**fix** *a* **assume** *a* ∈ *Field r*
   hence *bij-betw g* (*under r a*) (*under s* (*g a*)) ∧
        g a ∈ *under s* (*f a*)
   proof(*induction a rule*: *r.underS-induct*)
    case (*1 a*)
    hence *a*: *a* ∈ *Field r*
    and *IH1a*: ⋀ *a1*. *a1* ∈ *underS r a* ⟹ *inj-on g* (*under r a1*)
    and *IH1b*: ⋀ *a1*. *a1* ∈ *underS r a* ⟹ *g* ' *under r a1* = *under s* (*g a1*)
    and *IH2*: ⋀ *a1*. *a1* ∈ *underS r a* ⟹ *g a1* ∈ *under s* (*f a1*)

```
    unfolding underS-def Field-def bij-betw-def by auto
    have fa: f a ∈ Field s using f[OF a] by auto
    have g: g a = suc s (g ` underS r a)
      using r.worec-fixpoint[OF adm] unfolding g-def fun-eq-iff by blast
    have A0: g ` underS r a ⊆ Field s
    using IH1b by (metis IH2 image-subsetI in-mono under-Field)
    {fix a1 assume a1: a1 ∈ underS r a
     from IH2[OF this] have g a1 ∈ under s (f a1) .
     moreover have f a1 ∈ underS s (f a) using f[OF a] a1 by auto
      ultimately have g a1 ∈ underS s (f a) by (metis s.ANTISYM s.TRANS
under-underS-trans)
    }
    hence f a ∈ AboveS s (g ` underS r a) unfolding AboveS-def
    using fa by simp (metis (lifting, full-types) mem-Collect-eq underS-def)
    hence A: AboveS s (g ` underS r a) ≠ {} by auto
    have B: ⋀ a1. a1 ∈ underS r a ⟹ g a1 ∈ underS s (g a)
    unfolding g apply(rule s.suc-underS[OF A0 A]) by auto
    {fix a1 a2 assume a2: a2 ∈ underS r a and 1: a1 ∈ underS r a2
     hence a12: {a1,a2} ⊆ under r a2 and a1 ≠ a2 using r.REFL a
     unfolding underS-def under-def refl-on-def Field-def by auto
     hence g a1 ≠ g a2 using IH1a[OF a2] unfolding inj-on-def by auto
     hence g a1 ∈ underS s (g a2) using IH1b[OF a2] a12
     unfolding underS-def under-def by auto
    } note C = this
    have ga: g a ∈ Field s unfolding g using s.suc-inField[OF A0 A] .
    have aa: a ∈ under r a
    using a r.REFL unfolding under-def underS-def refl-on-def by auto
    show ?case proof safe
       show bij-betw g (under r a) (under s (g a)) unfolding bij-betw-def proof
safe
        show inj-on g (under r a) proof(rule r.inj-on-Field)
          fix a1 a2 assume a1 ∈ under r a and a2: a2 ∈ under r a and a1: a1
∈ underS r a2
          hence a22: a2 ∈ under r a2 and a12: a1 ∈ under r a2 a1 ≠ a2
          using a r.REFL unfolding under-def underS-def refl-on-def by auto
          show g a1 ≠ g a2
          proof(cases a2 = a)
            case False hence a2 ∈ underS r a
            using a2 unfolding underS-def under-def by auto
            from IH1a[OF this] show ?thesis using a12 a22 unfolding inj-on-def
by auto
          qed(insert B a1, unfold underS-def, auto)
        qed(unfold under-def Field-def, auto)
      next
        fix a1 assume a1: a1 ∈ under r a
        show g a1 ∈ under s (g a)
        proof(cases a1 = a)
          case True thus ?thesis
          using ga s.REFL unfolding refl-on-def under-def by auto
```

next
case *False*
hence *a1*: *a1* ∈ *underS r a* **using** *a1* **unfolding** *underS-def under-def* **by** *auto*
thus *?thesis* **using** *B* **unfolding** *underS-def under-def* **by** *auto*
qed
next
fix *b1* **assume** *b1*: *b1* ∈ *under s* (*g a*)
show *b1* ∈ *g* ‘ *under r a*
proof(*cases b1* = *g a*)
case *True* **thus** *?thesis* **using** *aa* **by** *auto*
next
case *False*
hence *b1* ∈ *underS s* (*g a*) **using** *b1* **unfolding** *underS-def under-def* **by** *auto*
from *s.underS-suc*[*OF this*[*unfolded g*] *A0*]
obtain *a1* **where** *a1*: *a1* ∈ *underS r a* **and** *b1*: *b1* ∈ *under s* (*g a1*) **by** *auto*
obtain *a2* **where** *a2* ∈ *under r a1* **and** *b1*: *b1* = *g a2* **using** *IH1b*[*OF a1*] *b1* **by** *auto*
hence *a2* ∈ *under r a* **using** *a1*
by (*metis r.ANTISYM r.TRANS in-mono underS-subset-under under-underS-trans*)
thus *?thesis* **using** *b1* **by** *auto*
qed
qed
next
have (*g a*, *f a*) ∈ *s* **unfolding** *g* **proof**(*rule s.suc-least*[*OF A0*])
fix *b1* **assume** *b1* ∈ *g* ‘ *underS r a*
then obtain *a1* **where** *a1*: *b1* = *g a1* **and** *a1*: *a1* ∈ *underS r a* **by** *auto*
hence *b1* ∈ *underS s* (*f a*)
using *a* **by** (*metis* ‹⋀*a1*. *a1* ∈ *underS r a* ⟹ *g a1* ∈ *underS s* (*f a*)›)
thus *f a* ≠ *b1* ∧ (*b1*, *f a*) ∈ *s* **unfolding** *underS-def* **by** *auto*
qed(*insert fa*, *auto*)
thus *g a* ∈ *under s* (*f a*) **unfolding** *under-def* **by** *auto*
qed
qed
}
thus *?thesis* **unfolding** *embed-def* **by** *auto*
qed

corollary *ordLeq-def2*:
r ≤*o s* ⟷ *Well-order r* ∧ *Well-order s* ∧
(∃ *f*. ∀ *a* ∈ *Field r*. *f a* ∈ *Field s* ∧ *f* ‘ *underS r a* ⊆ *underS s* (*f a*))
using *embed-in-Field*[*of r s*] *embed-underS2*[*of r s*] *embedI*[*of r s*]
unfolding *ordLeq-def* **by** *fast*

lemma *iso-oproj*:
assumes *r*: *Well-order r* **and** *s*: *Well-order s* **and** *f*: *iso r s f*

shows *oproj r s f*
using *assms* **unfolding** *oproj-def*
by (*subst* (*asm*) *iso-iff3*) (*auto simp*: *bij-betw-def*)

theorem *oproj-embed*:
assumes *r*: *Well-order r* **and** *s*: *Well-order s* **and** *f*: *oproj r s f*
shows $\exists$ *g*. *embed s r g*
proof (*rule embedI*[*OF s r*, *of inv-into* (*Field r*) *f*], *unfold underS-def*, *safe*)
fix *b* **assume** *b* $\in$ *Field s*
thus *inv-into* (*Field r*) *f b* $\in$ *Field r* **using** *oproj-Field2*[*OF f*] **by** (*metis imageI inv-into-into*)
next
fix *a b* **assume** *b* $\in$ *Field s a* $\neq$ *b* (*a*, *b*) $\in$ *s*
inv-into (*Field r*) *f a* = *inv-into* (*Field r*) *f b*
with *f* **show** *False* **by** (*auto dest*!: *inv-into-injective simp*: *Field-def oproj-def*)
next
fix *a b* **assume** $*$: *b* $\in$ *Field s a* $\neq$ *b* (*a*, *b*) $\in$ *s*
{ assume (*inv-into* (*Field r*) *f a*, *inv-into* (*Field r*) *f b*) $\notin$ *r*
moreover
from $*$(*3*) **have** *a* $\in$ *Field s* **unfolding** *Field-def* **by** *auto*
with $*$(*1*,*2*) **have**
inv-into (*Field r*) *f a* $\in$ *Field r inv-into* (*Field r*) *f b* $\in$ *Field r*
inv-into (*Field r*) *f a* $\neq$ *inv-into* (*Field r*) *f b*
by (*auto dest*!: *oproj-Field2*[*OF f*] *inv-into-injective intro*!: *inv-into-into*)
ultimately have (*inv-into* (*Field r*) *f b*, *inv-into* (*Field r*) *f a*) $\in$ *r*
using *r* **by** (*auto simp*: *well-order-on-def linear-order-on-def total-on-def*)
with *f*[*unfolded oproj-def compat-def*] $*$(*1*) ‹*a* $\in$ *Field s*›
f-inv-into-f[*of b f Field r*] *f-inv-into-f*[*of a f Field r*]
have (*b*, *a*) $\in$ *s* **by** (*metis in-mono*)
with $*$(*2*,*3*) *s* **have** *False*
by (*auto simp*: *well-order-on-def linear-order-on-def partial-order-on-def antisym-def*)
} thus (*inv-into* (*Field r*) *f a*, *inv-into* (*Field r*) *f b*) $\in$ *r* **by** *blast*
qed

corollary *oproj-ordLeq*:
assumes *r*: *Well-order r* **and** *s*: *Well-order s* **and** *f*: *oproj r s f*
shows *s* $\leq$*o r*
using *oproj-embed*[*OF assms*] *r s* **unfolding** *ordLeq-def* **by** *auto*

end

9 Ordinal Arithmetic

theory *Ordinal-Arithmetic*
imports *Wellorder-Constructions*
begin

definition *osum* :: *'a rel* $\Rightarrow$ *'b rel* $\Rightarrow$ (*'a* + *'b*) *rel* (**infixr** +*o* *70*)

where
  $r +o\ r' = \textit{map-prod Inl Inl}\ `\ r \cup \textit{map-prod Inr Inr}\ `\ r' \cup$
    $\{(\textit{Inl}\ a,\ \textit{Inr}\ a') \mid a\ a'\ .\ a \in \textit{Field}\ r \wedge a' \in \textit{Field}\ r'\}$

lemma *Field-osum*: $\textit{Field}(r +o\ r') = \textit{Inl}\ `\ \textit{Field}\ r \cup \textit{Inr}\ `\ \textit{Field}\ r'$
  unfolding *osum-def Field-def* **by** *auto*

lemma *osum-Refl*:$[\![\textit{Refl}\ r;\ \textit{Refl}\ r']\!] \Longrightarrow \textit{Refl}\ (r +o\ r')$

  unfolding *refl-on-def Field-osum* **unfolding** *osum-def* **by** *blast*

lemma *osum-trans*:
assumes *TRANS*: *trans r* **and** *TRANS′*: *trans r′*
shows *trans* $(r +o\ r')$
proof(*unfold trans-def*, *safe*)
  fix *x y z* **assume** $*$: $(x,\ y) \in r +o\ r'\ (y,\ z) \in r +o\ r'$
  thus $(x,\ z) \in r +o\ r'$
  proof (*cases x y z rule*: *sum.exhaust*[*case-product sum.exhaust sum.exhaust*])
    case (*Inl-Inl-Inl a b c*)
    with $*$ **have** $(a,b) \in r\ (b,c) \in r$ **unfolding** *osum-def* **by** *auto*
    with *TRANS* **have** $(a,c) \in r$ **unfolding** *trans-def* **by** *blast*
    with *Inl-Inl-Inl* **show** *?thesis* **unfolding** *osum-def* **by** *auto*
  next
    case (*Inl-Inl-Inr a b c*)
    with $*$ **have** $a \in \textit{Field}\ r\ c \in \textit{Field}\ r'$ **unfolding** *osum-def Field-def* **by** *auto*
    with *Inl-Inl-Inr* **show** *?thesis* **unfolding** *osum-def* **by** *auto*
  next
    case (*Inl-Inr-Inr a b c*)
    with $*$ **have** $a \in \textit{Field}\ r\ c \in \textit{Field}\ r'$ **unfolding** *osum-def Field-def* **by** *auto*
    with *Inl-Inr-Inr* **show** *?thesis* **unfolding** *osum-def* **by** *auto*
  next
    case (*Inr-Inr-Inr a b c*)
    with $*$ **have** $(a,b) \in r'\ (b,c) \in r'$ **unfolding** *osum-def* **by** *auto*
    with *TRANS′* **have** $(a,c) \in r'$ **unfolding** *trans-def* **by** *blast*
    with *Inr-Inr-Inr* **show** *?thesis* **unfolding** *osum-def* **by** *auto*
  qed (*auto simp*: *osum-def*)
qed

lemma *osum-Preorder*: $[\![\textit{Preorder}\ r;\ \textit{Preorder}\ r']\!] \Longrightarrow \textit{Preorder}\ (r +o\ r')$
  unfolding *preorder-on-def* **using** *osum-Refl osum-trans* **by** *blast*

lemma *osum-antisym*: $[\![\textit{antisym}\ r;\ \textit{antisym}\ r']\!] \Longrightarrow \textit{antisym}\ (r +o\ r')$
  unfolding *antisym-def osum-def* **by** *auto*

lemma *osum-Partial-order*: $[\![\textit{Partial-order}\ r;\ \textit{Partial-order}\ r']\!] \Longrightarrow \textit{Partial-order}$ $(r +o\ r')$
  unfolding *partial-order-on-def* **using** *osum-Preorder osum-antisym* **by** *blast*

lemma *osum-Total*: $[\![\textit{Total}\ r;\ \textit{Total}\ r']\!] \Longrightarrow \textit{Total}\ (r +o\ r')$

unfolding *total-on-def Field-osum* **unfolding** *osum-def* **by** *blast*

lemma *osum-Linear-order*: $[\![Linear\text{-}order\ r;\ Linear\text{-}order\ r']\!] \Longrightarrow Linear\text{-}order\ (r +o\ r')$
unfolding *linear-order-on-def* **using** *osum-Partial-order osum-Total* **by** *blast*

lemma *osum-wf*:
assumes *WF*: *wf r* **and** *WF'*: *wf r'*
shows *wf* $(r +o\ r')$
unfolding *wf-eq-minimal2* **unfolding** *Field-osum*
proof(*intro allI impI*, *elim conjE*)
fix *A* **assume** $*$: $A \subseteq Inl$ ‘ $Field\ r \cup Inr$ ‘ $Field\ r'$ **and** $**$: $A \neq \{\}$
obtain *B* **where** *B-def*: $B = A\ Int\ Inl$ ‘ $Field\ r$ **by** *blast*
show $\exists\, a{\in}A.\ \forall\, a'{\in}A.\ (a',\ a) \notin r +o\ r'$
proof(*cases* $B = \{\}$)
case *False*
hence $B \neq \{\}\ B \leq Inl$ ‘ $Field\ r$ **using** *B-def* **by** *auto*
hence Inl −‘ $B \neq \{\}\ Inl$ −‘ $B \leq Field\ r$ **unfolding** *vimage-def* **by** *auto*
then obtain *a* **where** *1*: $a \in Inl$ −‘ B **and** $\forall\, a1 \in Inl$ −‘ $B.\ (a1,\ a) \notin r$
using *WF* **unfolding** *wf-eq-minimal2* **by** *metis*
hence $\forall\, a1 \in A.\ (a1,\ Inl\ a) \notin r +o\ r'$
unfolding *osum-def* **using** *B-def* $**$ **by** (*auto simp*: *vimage-def Field-def*)
thus *?thesis* **using** *1* **unfolding** *B-def* **by** *auto*
next
case *True*
hence *1*: $A \leq Inr$ ‘ $Field\ r'$ **using** $*$ *B-def* **by** *auto*
with $**$ **have** Inr −‘$A \neq \{\}\ Inr$ −‘ $A \leq Field\ r'$ **unfolding** *vimage-def* **by** *auto*
with $**$ **obtain** *a'* **where** *2*: $a' \in Inr$ −‘ A **and** $\forall\, a1' \in Inr$ −‘ $A.\ (a1',a') \notin r'$
using *WF'* **unfolding** *wf-eq-minimal2* **by** *metis*
hence $\forall\, a1' \in A.\ (a1',\ Inr\ a') \notin r +o\ r'$
unfolding *osum-def* **using** $**$ *1* **by** (*auto simp*: *vimage-def Field-def*)
thus *?thesis* **using** *2* **by** *blast*
qed
qed

lemma *osum-minus-Id*:
assumes *r*: $Total\ r\ \neg\ (r \leq Id)$ **and** *r'*: $Total\ r'\ \neg\ (r' \leq Id)$
shows $(r +o\ r') - Id \leq (r - Id) +o\ (r' - Id)$
unfolding *osum-def Total-Id-Field*[*OF r*] *Total-Id-Field*[*OF r'*] **by** *auto*

lemma *osum-minus-Id1*:
$r \leq Id \Longrightarrow (r +o\ r') - Id \leq (Inl$ ‘ $Field\ r \times Inr$ ‘ $Field\ r') \cup (map\text{-}prod\ Inr\ Inr$ ‘ $(r' - Id))$
unfolding *osum-def* **by** *auto*

lemma *osum-minus-Id2*:
$r' \leq Id \Longrightarrow (r +o\ r') - Id \leq (map\text{-}prod\ Inl\ Inl$ ‘ $(r - Id)) \cup (Inl$ ‘ $Field\ r \times$

Inr ‘ *Field* r')
unfolding *osum-def* **by** *auto*

lemma *osum-wf-Id*:
assumes *TOT*: *Total* r **and** *TOT*$'$: *Total* r' **and** *WF*: $wf(r - Id)$ **and** *WF*$'$: $wf(r' - Id)$
shows *wf* $((r +o\ r') - Id)$
proof(*cases* $r \leq Id \vee r' \leq Id$)
case *False*
thus *?thesis*
using *osum-minus-Id*[*of* r r'] *assms* *osum-wf*[*of* $r - Id$ $r' - Id$]
wf-subset[*of* $(r - Id) +o\ (r' - Id)$ $(r +o\ r') - Id$] **by** *auto*
next
have *1*: *wf* (*Inl* ‘ *Field* r $\times$ *Inr* ‘ *Field* r') **by** (*rule wf-Int-Times*) *auto*
case *True*
thus *?thesis*
proof (*elim disjE*)
assume $r \subseteq Id$
thus *wf* $((r +o\ r') - Id)$
by (*rule wf-subset*[*rotated*, *OF osum-minus-Id1 wf-Un*[*OF 1 wf-map-prod-image*[*OF WF*$'$]]]) *auto*
next
assume $r' \subseteq Id$
thus *wf* $((r +o\ r') - Id)$
by (*rule wf-subset*[*rotated*, *OF osum-minus-Id2 wf-Un*[*OF wf-map-prod-image*[*OF WF*] *1*]]) *auto*
qed
qed

lemma *osum-Well-order*:
assumes *WELL*: *Well-order* r **and** *WELL*$'$: *Well-order* r'
shows *Well-order* $(r +o\ r')$
proof−
have *Total* $r \wedge$ *Total* r' **using** *WELL WELL*$'$ **by** (*auto simp add*: *order-on-defs*)
thus *?thesis* **using** *assms* **unfolding** *well-order-on-def*
using *osum-Linear-order osum-wf-Id* **by** *blast*
qed

lemma *osum-embedL*:
assumes *WELL*: *Well-order* r **and** *WELL*$'$: *Well-order* r'
shows *embed* r $(r +o\ r')$ *Inl*
proof −
have *1*: *Well-order* $(r +o\ r')$ **using** *assms* **by** (*auto simp add*: *osum-Well-order*)
moreover
have *compat* r $(r +o\ r')$ *Inl* **unfolding** *compat-def osum-def* **by** *auto*
moreover
have *ofilter* $(r +o\ r')$ (*Inl* ‘ *Field* r)
unfolding *wo-rel.ofilter-def*[*unfolded wo-rel-def*, *OF 1*] *Field-osum under-def*
unfolding *osum-def Field-def* **by** *auto*

ultimately show *?thesis* **using** *assms* **by** (*auto simp add*: *embed-iff-compat-inj-on-ofilter*)
qed

corollary *osum-ordLeqL*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'*
shows $r \leq o\ r +o\ r'$
using *assms osum-embedL osum-Well-order* **unfolding** *ordLeq-def* **by** *blast*

lemma *dir-image-alt*: *dir-image r f = map-prod f f ‘ r*
unfolding *dir-image-def map-prod-def* **by** *auto*

lemma *map-prod-ordIso*: $[\![$*Well-order r*; *inj-on f* (*Field r*)$]\!] \Longrightarrow$ *map-prod f f ‘ r*
$=o\ r$
unfolding *dir-image-alt*[*symmetric*] **by** (*rule ordIso-symmetric*[*OF dir-image-ordIso*])

definition *oprod* :: *'a rel* ⇒ *'b rel* ⇒ (*'a* × *'b*) *rel* (**infixr** *∗o 80*)
where $r *o\ r' = \{((x1, y1), (x2, y2)).$
$(((y1, y2) \in r' - Id \wedge x1 \in Field\ r \wedge x2 \in Field\ r) \vee$
$((y1, y2) \in Restr\ Id\ (Field\ r') \wedge (x1, x2) \in r))\}$

lemma *Field-oprod*: $Field\ (r *o\ r') = Field\ r \times Field\ r'$
unfolding *oprod-def Field-def* **by** *auto blast+*

lemma *oprod-Refl*:$[\![$*Refl r*; *Refl r'*$]\!] \Longrightarrow$ *Refl* ($r *o\ r'$)
unfolding *refl-on-def Field-oprod* **unfolding** *oprod-def* **by** *auto*

lemma *oprod-trans*:
assumes *trans r trans r' antisym r antisym r'*
shows *trans* ($r *o\ r'$)
proof(*unfold trans-def*, *safe*)
fix *x y z* **assume** ∗: $(x, y) \in r *o\ r'\ (y, z) \in r *o\ r'$
thus $(x, z) \in r *o\ r'$
unfolding *oprod-def*
apply *safe*
apply (*metis assms*(*2*) *transE*)
apply (*metis assms*(*2*) *transE*)
apply (*metis assms*(*2*) *transE*)
apply (*metis assms*(*4*) *antisymD*)
apply (*metis assms*(*4*) *antisymD*)
apply (*metis assms*(*2*) *transE*)
apply (*metis assms*(*4*) *antisymD*)
apply (*metis Field-def Range-iff Un-iff*)
apply (*metis Field-def Range-iff Un-iff*)
apply (*metis Field-def Range-iff Un-iff*)
apply (*metis Field-def Domain-iff Un-iff*)
apply (*metis Field-def Domain-iff Un-iff*)
apply (*metis Field-def Domain-iff Un-iff*)
apply (*metis assms*(*1*) *transE*)
apply (*metis assms*(*1*) *transE*)

apply (*metis assms*(*1*) *transE*)
apply (*metis assms*(*1*) *transE*)
done
qed

lemma *oprod-Preorder*: $[\![$*Preorder r*; *Preorder r'*; *antisym r*; *antisym r'*$]\!] \Longrightarrow$ *Preorder* (*r* $*o$ *r'*)
unfolding *preorder-on-def* **using** *oprod-Refl oprod-trans* **by** *blast*

lemma *oprod-antisym*: $[\![$*antisym r*; *antisym r'*$]\!] \Longrightarrow$ *antisym* (*r* $*o$ *r'*)
unfolding *antisym-def oprod-def* **by** *auto*

lemma *oprod-Partial-order*: $[\![$*Partial-order r*; *Partial-order r'*$]\!] \Longrightarrow$ *Partial-order* (*r* $*o$ *r'*)
unfolding *partial-order-on-def* **using** *oprod-Preorder oprod-antisym* **by** *blast*

lemma *oprod-Total*: $[\![$*Total r*; *Total r'*$]\!] \Longrightarrow$ *Total* (*r* $*o$ *r'*)
unfolding *total-on-def Field-oprod* **unfolding** *oprod-def* **by** *auto*

lemma *oprod-Linear-order*: $[\![$*Linear-order r*; *Linear-order r'*$]\!] \Longrightarrow$ *Linear-order* (*r* $*o$ *r'*)
unfolding *linear-order-on-def* **using** *oprod-Partial-order oprod-Total* **by** *blast*

lemma *oprod-wf*:
assumes *WF*: *wf r* **and** *WF'*: *wf r'*
shows *wf* (*r* $*o$ *r'*)
unfolding *wf-eq-minimal2* **unfolding** *Field-oprod*
proof(*intro allI impI*, *elim conjE*)
fix *A* **assume** $*$: $A \subseteq$ *Field r* $\times$ *Field r'* **and** $**$: $A \neq \{\}$
then obtain *y* **where** *y*: $y \in$ *snd* ‘ A $\forall y' \in$ *snd* ‘ A. $(y', y) \notin r'$
using *spec*[*OF WF'*[*unfolded wf-eq-minimal2*], *of snd* ‘ *A*] **by** *auto*
let $?A$ = *fst* ‘ $A \cap \{x. (x, y) \in A\}$
from $*$ *y* **have** $?A \neq \{\}$ $?A \subseteq$ *Field r* **by** *auto*
then obtain *x* **where** *x*: $x \in ?A$ **and** $\forall x' \in ?A. (x', x) \notin r$
using *spec*[*OF WF*[*unfolded wf-eq-minimal2*], *of ?A*] **by** *auto*
with *y* **have** $\forall a' \in A. (a', (x, y)) \notin r *o\ r'$
unfolding *oprod-def mem-Collect-eq split-beta fst-conv snd-conv Id-def* **by** *auto*
moreover from *x* **have** $(x, y) \in A$ **by** *auto*
ultimately show $\exists a \in A. \forall a' \in A. (a', a) \notin r *o\ r'$ **by** *blast*
qed

lemma *oprod-minus-Id*:
assumes *r*: *Total r* $\neg$ ($r \leq$ *Id*) **and** *r'*: *Total r'* $\neg$ ($r' \leq$ *Id*)
shows $(r *o\ r') - Id \leq (r - Id) *o\ (r' - Id)$
unfolding *oprod-def Total-Id-Field*[*OF r*] *Total-Id-Field*[*OF r'*] **by** *auto*

lemma *oprod-minus-Id1*:
$r \leq Id \Longrightarrow r *o\ r' - Id \leq \{((x,y1), (x,y2)).\ x \in Field\ r \wedge (y1, y2) \in (r' - Id)\}$
unfolding *oprod-def* **by** *auto*

lemma *wf-extend-oprod1*:
assumes *wf r*
shows *wf* $\{((x,y1), (x,y2))\ .\ x \in A \wedge (y1,\ y2) \in r\}$
proof (*unfold wf-eq-minimal2*, *intro allI impI*, *elim conjE*)
fix *B*
assume $*$: $B \subseteq$ *Field* $\{((x,y1), (x,y2))\ .\ x \in A \wedge (y1,\ y2) \in r\}$ **and** $B \neq \{\}$
from *image-mono*[*OF* $*$, *of snd*] **have** *snd* ‘ $B \subseteq$ *Field r* **unfolding** *Field-def* **by** *force*
with ‹$B \neq \{\}$› **obtain** *x* **where** *x*: $x \in$ *snd* ‘ $B\ \forall x' \in$ *snd* ‘ $B.\ (x',\ x) \notin r$
using *spec*[*OF assms*[*unfolded wf-eq-minimal2*], *of snd* ‘ *B*] **by** *auto*
then obtain *a* **where** $(a,\ x) \in B$ **by** *auto*
moreover
from $*$ *x* **have** $\forall a' \in B.\ (a',\ (a,\ x)) \notin \{((x,y1), (x,y2))\ .\ x \in A \wedge (y1,\ y2) \in r\}$ **by** *auto*
ultimately show $\exists ax \in B.\ \forall a' \in B.\ (a',\ ax) \notin \{((x,y1), (x,y2))\ .\ x \in A \wedge (y1,\ y2) \in r\}$ **by** *blast*
qed

lemma *oprod-minus-Id2*:
$r' \leq Id \Longrightarrow r *o\ r' - Id \leq \{((x1,y), (x2,y)).\ (x1,\ x2) \in (r - Id) \wedge y \in Field\ r'\}$
unfolding *oprod-def* **by** *auto*

lemma *wf-extend-oprod2*:
assumes *wf r*
shows *wf* $\{((x1,y), (x2,y))\ .\ (x1,\ x2) \in r \wedge y \in A\}$
proof (*unfold wf-eq-minimal2*, *intro allI impI*, *elim conjE*)
fix *B*
assume $*$: $B \subseteq$ *Field* $\{((x1,\ y), (x2,\ y)).\ (x1,\ x2) \in r \wedge y \in A\}$ **and** $B \neq \{\}$
from *image-mono*[*OF* $*$, *of fst*] **have** *fst* ‘ $B \subseteq$ *Field r* **unfolding** *Field-def* **by** *force*
with ‹$B \neq \{\}$› **obtain** *x* **where** *x*: $x \in$ *fst* ‘ $B\ \forall x' \in$ *fst* ‘ $B.\ (x',\ x) \notin r$
using *spec*[*OF assms*[*unfolded wf-eq-minimal2*], *of fst* ‘ *B*] **by** *auto*
then obtain *a* **where** $(x,\ a) \in B$ **by** *auto*
moreover
from $*$ *x* **have** $\forall a' \in B.\ (a',\ (x,\ a)) \notin \{((x1,\ y),\ x2,\ y).\ (x1,\ x2) \in r \wedge y \in A\}$ **by** *auto*
ultimately show $\exists xa \in B.\ \forall a' \in B.\ (a',\ xa) \notin \{((x1,\ y),\ x2,\ y).\ (x1,\ x2) \in r \wedge y \in A\}$ **by** *blast*
qed

lemma *oprod-wf-Id*:
assumes *TOT*: *Total r* **and** *TOT′*: *Total r′* **and** *WF*: $wf(r - Id)$ **and** *WF′*: $wf(r' - Id)$
shows *wf* $((r *o\ r') - Id)$
proof(*cases* $r \leq Id \vee r' \leq Id$)
case *False*
thus *?thesis*

using *oprod-minus-Id*[*of r r'*] *assms oprod-wf*[*of r* − *Id r'* − *Id*]
wf-subset[*of* (*r* − *Id*) ∗*o* (*r'* − *Id*) (*r* ∗*o r'*) − *Id*] **by** *auto*
next
case *True*
thus *?thesis* **using** *wf-subset*[*OF wf-extend-oprod1*[*OF WF'*] *oprod-minus-Id1*]
wf-subset[*OF wf-extend-oprod2*[*OF WF*] *oprod-minus-Id2*] **by** *auto*
qed

lemma *oprod-Well-order*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'*
shows *Well-order* (*r* ∗*o r'*)
proof−
have *Total r* ∧ *Total r'* **using** *WELL WELL'* **by** (*auto simp add*: *order-on-defs*)
thus *?thesis* **using** *assms* **unfolding** *well-order-on-def*
using *oprod-Linear-order oprod-wf-Id* **by** *blast*
qed

lemma *oprod-embed*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and** *r'* ≠ {}
shows *embed r* (*r* ∗*o r'*) (λ*x*. (*x*, *minim r'* (*Field r'*))) (**is** *embed - - ?f*)
proof −
from *assms*(*3*) **have** *r'*: *Field r'* ≠ {} **unfolding** *Field-def* **by** *auto*
have *minim*[*simp*]: *minim r'* (*Field r'*) ∈ *Field r'*
using *wo-rel.minim-inField*[*unfolded wo-rel-def*, *OF WELL' - r'*] **by** *auto*
{ fix *b*
assume *b*: (*b*, *minim r'* (*Field r'*)) ∈ *r'*
hence *b* ∈ *Field r'* **unfolding** *Field-def* **by** *auto*
hence (*minim r'* (*Field r'*), *b*) ∈ *r'*
using *wo-rel.minim-least*[*unfolded wo-rel-def*, *OF WELL' subset-refl*] *r'* **by**
auto
with *b* **have** *b* = *minim r'* (*Field r'*)
by (*metis WELL' antisym-def linear-order-on-def partial-order-on-def well-order-on-def*)
} note ∗ = *this*
have *1*: *Well-order* (*r* ∗*o r'*) **using** *assms* **by** (*auto simp add*: *oprod-Well-order*)
moreover
from *r'* **have** *compat r* (*r* ∗*o r'*) *?f* **unfolding** *compat-def oprod-def* **by** *auto*
moreover
from ∗ **have** *ofilter* (*r* ∗*o r'*) (*?f* ‘ *Field r*)
unfolding *wo-rel.ofilter-def*[*unfolded wo-rel-def*, *OF 1*] *Field-oprod under-def*
unfolding *oprod-def* **by** *auto* (*auto simp*: *image-iff Field-def*)
moreover have *inj-on ?f* (*Field r*) **unfolding** *inj-on-def* **by** *auto*
ultimately show *?thesis* **using** *assms* **by** (*auto simp add*: *embed-iff-compat-inj-on-ofilter*)
qed

corollary *oprod-ordLeq*: ⟦*Well-order r*; *Well-order r'*; *r'* ≠ {}⟧ ⟹ *r* ≤*o r* ∗*o r'*
using *oprod-embed oprod-Well-order* **unfolding** *ordLeq-def* **by** *blast*

definition *support z A f* = {*x* ∈ *A*. *f x* ≠ *z*}

lemma *support-Un*[*simp*]: *support z* $(A \cup B)$ *f* = *support z A f* $\cup$ *support z B f*
  unfolding *support-def* **by** *auto*

lemma *support-upd*[*simp*]: *support z A* $(f(x := z))$ = *support z A f* $-$ $\{x\}$
  unfolding *support-def* **by** *auto*

lemma *support-upd-subset*[*simp*]: *support z A* $(f(x := y)) \subseteq$ *support z A f* $\cup \{x\}$
  unfolding *support-def* **by** *auto*

lemma *fun-unequal-in-support*:
  assumes $f \neq g$ $f \in$ *Func A B* $g \in$ *Func A C*
  shows (*support z A f* $\cup$ *support z A g*) $\cap$ $\{a.\ f\ a \neq g\ a\} \neq \{\}$ (**is** *?L* $\cap$ *?R* $\neq \{\}$)
proof $-$
  from *assms*(*1*) **obtain** *x* **where** *x*: $f\ x \neq g\ x$ **by** *blast*
  hence $x \in$ *?R* **by** *simp*
  moreover from *assms*(*2*$-$*3*) *x* **have** $x \in A$ **unfolding** *Func-def* **by** *fastforce*
  with *x* **have** $x \in$ *?L* **unfolding** *support-def* **by** *auto*
  ultimately show *?thesis* **by** *auto*
qed

definition *fin-support* **where**
  fin-support z A = $\{f.$ *finite* (*support z A f*)$\}$

lemma *finite-support*: $f \in$ *fin-support z A* $\Longrightarrow$ *finite* (*support z A f*)
  unfolding *support-def fin-support-def* **by** *auto*

lemma *fin-support-Field-osum*:
  $f \in$ *fin-support z* (*Inl* ‘ *A* $\cup$ *Inr* ‘ *B*) $\longleftrightarrow$
  (*f o Inl*) $\in$ *fin-support z A* $\wedge$ (*f o Inr*) $\in$ *fin-support z B* (**is** *?L* $\longleftrightarrow$ *?R1* $\wedge$ *?R2*)
proof *safe*
  assume *?L*
  from ‹*?L*› **show** *?R1* **unfolding** *fin-support-def support-def*
    by (*fastforce simp*: *image-iff elim*: *finite-surj*[*of* - - *case-sum id undefined*])
  from ‹*?L*› **show** *?R2* **unfolding** *fin-support-def support-def*
    by (*fastforce simp*: *image-iff elim*: *finite-surj*[*of* - - *case-sum undefined id*])
next
  assume *?R1 ?R2*
  thus *?L* **unfolding** *fin-support-def support-Un*
    by (*auto simp*: *support-def elim*: *finite-surj*[*of* - - *Inl*] *finite-surj*[*of* - - *Inr*])
qed

lemma *Func-upd*: $[\![f \in$ *Func A B*; $x \in A$; $y \in B]\!] \Longrightarrow f(x := y) \in$ *Func A B*
  unfolding *Func-def* **by** *auto*

context *wo-rel*
begin

definition *isMaxim* :: *'a set* $\Rightarrow$ *'a* $\Rightarrow$ *bool*
where *isMaxim A b* $\equiv$ $b \in A \wedge (\forall a \in A.\ (a,b) \in r)$

definition *maxim* :: *'a set* ⇒ *'a*
where *maxim A* ≡ *THE b. isMaxim A b*

lemma *isMaxim-unique*[*intro*]: ⟦*isMaxim A x*; *isMaxim A y*⟧ ⟹ *x* = *y*
  unfolding *isMaxim-def* **using** *antisymD*[*OF ANTISYM*, *of x y*] **by** *auto*

lemma *maxim-isMaxim*: ⟦*finite A*; *A* ≠ {}; *A* ⊆ *Field r*⟧ ⟹ *isMaxim A* (*maxim A*)
unfolding *maxim-def*
proof (*rule theI'*, *rule ex-ex1I*[*OF - isMaxim-unique*, *rotated*], *assumption*+,
  induct A rule: *finite-induct*)
  case (*insert x A*)
  thus *?case*
  proof (*cases A* = {})
    case *True*
     moreover have *isMaxim* {*x*} *x* **unfolding** *isMaxim-def* **using** *refl-onD*[*OF REFL*] *insert*(*5*) **by** *auto*
    ultimately show *?thesis* **by** *blast*
  next
    case *False*
    with *insert*(*3*,*5*) **obtain** *y* **where** *isMaxim A y* **by** *blast*
    with *insert*(*2*,*5*) **have** *if* (*y*, *x*) ∈ *r then isMaxim* (*insert x A*) *x else isMaxim* (*insert x A*) *y*
      unfolding *isMaxim-def subset-eq* **by** (*metis insert-iff max2-def max2-equals1 max2-iff*)
    thus *?thesis* **by** *metis*
  qed
qed *simp*

lemma *maxim-in*: ⟦*finite A*; *A* ≠ {}; *A* ⊆ *Field r*⟧ ⟹ *maxim A* ∈ *A*
  using *maxim-isMaxim* **unfolding** *isMaxim-def* **by** *auto*

lemma *maxim-greatest*: ⟦*finite A*; *x* ∈ *A*; *A* ⊆ *Field r*⟧ ⟹ (*x*, *maxim A*) ∈ *r*
  using *maxim-isMaxim* **unfolding** *isMaxim-def* **by** *auto*

lemma *isMaxim-zero*: *isMaxim A zero* ⟹ *A* = {*zero*}
  unfolding *isMaxim-def* **by** *auto*

lemma *maxim-insert*:
  assumes *finite A A* ≠ {} *A* ⊆ *Field r x* ∈ *Field r*
  shows *maxim* (*insert x A*) = *max2 x* (*maxim A*)
proof −
   from *assms* **have** ∗: *isMaxim* (*insert x A*) (*maxim* (*insert x A*)) *isMaxim A* (*maxim A*)
    using *maxim-isMaxim* **by** *auto*
  show *?thesis*
  proof (*cases* (*x*, *maxim A*) ∈ *r*)
    case *True*

with ∗(*2*) **have** *isMaxim* (*insert x A*) (*maxim A*) **unfolding** *isMaxim-def*
using *transD*[*OF TRANS*, *of* - *x maxim A*] **by** *blast*
with ∗(*1*) *True* **show** *?thesis* **unfolding** *max2-def* **by** (*metis isMaxim-unique*)
next
case *False*
hence (*maxim A*, *x*) ∈ *r* **by** (*metis* ∗(*2*) *assms*(*3*,*4*) *in-mono in-notinI isMaxim-def*)
with ∗(*2*) *assms*(*4*) **have** *isMaxim* (*insert x A*) *x* **unfolding** *isMaxim-def*
using *transD*[*OF TRANS*, *of* - *maxim A x*] *refl-onD*[*OF REFL*, *of x*] **by** *blast*
with ∗(*1*) *False* **show** *?thesis* **unfolding** *max2-def* **by** (*metis isMaxim-unique*)
qed
qed

lemma *maxim-Un*:
assumes *finite A A* ≠ {} *A* ⊆ *Field r finite B B* ≠ {} *B* ⊆ *Field r*
shows *maxim* (*A* ∪ *B*) = *max2* (*maxim A*) (*maxim B*)
proof −
from *assms* **have** ∗: *isMaxim* (*A* ∪ *B*) (*maxim* (*A* ∪ *B*)) *isMaxim A* (*maxim A*) *isMaxim B* (*maxim B*)
using *maxim-isMaxim* **by** *auto*
show *?thesis*
proof (*cases* (*maxim A*, *maxim B*) ∈ *r*)
case *True*
with ∗(*2*,*3*) **have** *isMaxim* (*A* ∪ *B*) (*maxim B*) **unfolding** *isMaxim-def*
using *transD*[*OF TRANS*, *of* - *maxim A maxim B*] **by** *blast*
with ∗(*1*) *True* **show** *?thesis* **unfolding** *max2-def* **by** (*metis isMaxim-unique*)
next
case *False*
hence (*maxim B*, *maxim A*) ∈ *r* **by** (*metis* ∗(*2*,*3*) *assms*(*3*,*6*) *in-mono in-notinI isMaxim-def*)
with ∗(*2*,*3*) **have** *isMaxim* (*A* ∪ *B*) (*maxim A*) **unfolding** *isMaxim-def*
using *transD*[*OF TRANS*, *of* - *maxim B maxim A*] **by** *blast*
with ∗(*1*) *False* **show** *?thesis* **unfolding** *max2-def* **by** (*metis isMaxim-unique*)
qed
qed

lemma *maxim-insert-zero*:
assumes *finite A A* ≠ {} *A* ⊆ *Field r*
shows *maxim* (*insert zero A*) = *maxim A*
using *assms zero-in-Field maxim-in*[*OF assms*] **by** (*subst maxim-insert*[*unfolded max2-def*]) *auto*

lemma *maxim-equality*: *isMaxim A x* ⟹ *maxim A* = *x*
unfolding *maxim-def* **by** (*rule the-equality*) *auto*

lemma *maxim-singleton*:
x ∈ *Field r* ⟹ *maxim* {*x*} = *x*
using *refl-onD*[*OF REFL*] **by** (*intro maxim-equality*) (*simp add*: *isMaxim-def*)

lemma *maxim-Int*: ⟦*finite* A; $A \neq \{\}$; $A \subseteq$ *Field* r; *maxim* $A \in B$⟧ $\Longrightarrow$ *maxim* $(A \cap B)$ = *maxim* A
by (*rule maxim-equality*) (*auto simp*: *isMaxim-def intro*: *maxim-in maxim-greatest*)

lemma *maxim-mono*: ⟦$X \subseteq Y$; *finite* Y; $X \neq \{\}$; $Y \subseteq$ *Field* r⟧ $\Longrightarrow$ (*maxim* X, *maxim* Y) $\in r$
using *maxim-in*[*OF finite-subset*, *of X Y*] **by** (*auto intro*: *maxim-greatest*)

definition *max-fun-diff f g* $\equiv$ *maxim* ($\{a \in$ *Field* r. $f\ a \neq g\ a\}$)

lemma *max-fun-diff-commute*: *max-fun-diff f g* = *max-fun-diff g f*
unfolding *max-fun-diff-def* **by** *metis*

lemma *zero-under*: $x \in$ *Field* $r \Longrightarrow$ *zero* $\in$ *under* x
unfolding *under-def* **by** (*auto intro*: *zero-smallest*)

end

definition *FinFunc r s* = *Func* (*Field s*) (*Field r*) $\cap$ *fin-support* (*zero r*) (*Field s*)

lemma *FinFuncD*: ⟦$f \in$ *FinFunc r s*; $x \in$ *Field s*⟧ $\Longrightarrow$ $f\ x \in$ *Field r*
unfolding *FinFunc-def Func-def* **by** (*fastforce split*: *option.splits*)

locale *wo-rel2* =
fixes *r s*
assumes *rWELL*: *Well-order r*
and *sWELL*: *Well-order s*
begin

interpretation *r*: *wo-rel r* **by** *unfold-locales* (*rule rWELL*)
interpretation *s*: *wo-rel s* **by** *unfold-locales* (*rule sWELL*)

abbreviation *SUPP* $\equiv$ *support r.zero* (*Field s*)
abbreviation *FINFUNC* $\equiv$ *FinFunc r s*
lemmas *FINFUNCD* = *FinFuncD*[*of - r s*]

lemma *fun-diff-alt*: $\{a \in$ *Field s*. $f\ a \neq g\ a\}$ = (*SUPP f* $\cup$ *SUPP g*) $\cap$ $\{a.\ f\ a \neq g\ a\}$
by (*auto simp*: *support-def*)

lemma *max-fun-diff-alt*:
s.max-fun-diff f g = *s.maxim* ((*SUPP f* $\cup$ *SUPP g*) $\cap$ $\{a.\ f\ a \neq g\ a\}$)
unfolding *s.max-fun-diff-def fun-diff-alt* **..**

lemma *isMaxim-max-fun-diff*: ⟦$f \neq g$; $f \in$ *FINFUNC*; $g \in$ *FINFUNC*⟧ $\Longrightarrow$
s.isMaxim $\{a \in$ *Field s*. $f\ a \neq g\ a\}$ (*s.max-fun-diff f g*)
using *fun-unequal-in-support*[*of f g*] **unfolding** *max-fun-diff-alt fun-diff-alt fun-eq-iff*
by (*intro s.maxim-isMaxim*) (*auto simp*: *FinFunc-def fin-support-def support-def*)

lemma *max-fun-diff-in*: $[\![f \neq g; f \in \mathit{FINFUNC}; g \in \mathit{FINFUNC}]\!] \Longrightarrow$
$s.\mathit{max\text{-}fun\text{-}diff}\ f\ g \in \{a \in \mathit{Field}\ s.\ f\ a \neq g\ a\}$
using *isMaxim-max-fun-diff* **unfolding** *s.isMaxim-def* **by** *blast*

lemma *max-fun-diff-max*: $[\![f \neq g; f \in \mathit{FINFUNC}; g \in \mathit{FINFUNC}; x \in \{a \in \mathit{Field}\ s.\ f\ a \neq g\ a\}]\!] \Longrightarrow$
$(x, s.\mathit{max\text{-}fun\text{-}diff}\ f\ g) \in s$
using *isMaxim-max-fun-diff* **unfolding** *s.isMaxim-def* **by** *blast*

lemma *max-fun-diff*:
$[\![f \neq g; f \in \mathit{FINFUNC}; g \in \mathit{FINFUNC}]\!] \Longrightarrow$
$(\exists\, a\ b.\ a \neq b \wedge a \in \mathit{Field}\ r \wedge b \in \mathit{Field}\ r \wedge$
$f\ (s.\mathit{max\text{-}fun\text{-}diff}\ f\ g) = a \wedge g\ (s.\mathit{max\text{-}fun\text{-}diff}\ f\ g) = b)$
using *isMaxim-max-fun-diff*[*of f g*] **unfolding** *s.isMaxim-def FinFunc-def Func-def* **by** *auto*

lemma *max-fun-diff-le-eq*:
$[\![(s.\mathit{max\text{-}fun\text{-}diff}\ f\ g, x) \in s; f \neq g; f \in \mathit{FINFUNC}; g \in \mathit{FINFUNC}; x \neq s.\mathit{max\text{-}fun\text{-}diff}\ f\ g]\!] \Longrightarrow$
$f\ x = g\ x$
using *max-fun-diff-max*[*of f g x*] *antisymD*[*OF s.ANTISYM*, *of s.max-fun-diff f g x*]
by (*auto simp*: *Field-def*)

lemma *max-fun-diff-max2*:
assumes *ineq*: $s.\mathit{max\text{-}fun\text{-}diff}\ f\ g = s.\mathit{max\text{-}fun\text{-}diff}\ g\ h \longrightarrow$
$f\ (s.\mathit{max\text{-}fun\text{-}diff}\ f\ g) \neq h\ (s.\mathit{max\text{-}fun\text{-}diff}\ g\ h)$ **and**
fg: $f \neq g$ **and** *gh*: $g \neq h$ **and** *fh*: $f \neq h$ **and**
f: $f \in \mathit{FINFUNC}$ **and** *g*: $g \in \mathit{FINFUNC}$ **and** *h*: $h \in \mathit{FINFUNC}$
shows $s.\mathit{max\text{-}fun\text{-}diff}\ f\ h = s.\mathit{max2}\ (s.\mathit{max\text{-}fun\text{-}diff}\ f\ g)\ (s.\mathit{max\text{-}fun\text{-}diff}\ g\ h)$
(**is** *?fh = s.max2 ?fg ?gh*)
proof (*cases ?fg = ?gh*)
case *True*
with *ineq* **have** *f ?fg ≠ h ?fg* **by** *simp*
moreover
{ fix *x* **assume** *x*: $x \in \{a \in \mathit{Field}\ s.\ f\ a \neq h\ a\}$
hence (*x*, *?fg*) ∈ *s*
proof (*cases x = ?fg*)
case *False* **show** *?thesis*
proof (*rule ccontr*)
assume (*x*, *?fg*) ∉ *s*
with *max-fun-diff-in*[*OF fg f g*] *x False* **have** ∗: (*?fg*, *x*) ∈ *s* **by** (*blast intro*: *s.in-notinI*)
hence *f x = g x* **by** (*rule max-fun-diff-le-eq*[*OF - fg f g False*])
moreover have *g x = h x* **using** *max-fun-diff-le-eq*[*OF - gh g h*] *False True* ∗ **by** *simp*
ultimately show *False* **using** *x* **by** *simp*
qed
qed (*simp add*: *refl-onD*[*OF s.REFL*])

}
ultimately have *s.isMaxim* $\{a \in$ *Field s. f a* $\neq$ *h a*$\}$ *?fg*
unfolding *s.isMaxim-def* **using** *max-fun-diff-in*[*OF fg f g*] **by** *simp*
hence *?fh* = *?fg* **using** *isMaxim-max-fun-diff*[*OF fh f h*] **by** *blast*
thus *?thesis* **unfolding** *True s.max2-def* **by** *simp*
next
case *False* **note** ∗ = *this*
show *?thesis*
proof (*cases* (*?fg*, *?gh*) ∈ *s*)
case *True*
hence ∗: *f ?gh* = *g ?gh* **by** (*rule max-fun-diff-le-eq*[*OF - fg f g* ∗[*symmetric*]])
hence *s.isMaxim* $\{a \in$ *Field s. f a* $\neq$ *h a*$\}$ *?gh* **using** *isMaxim-max-fun-diff*[*OF gh g h*]
isMaxim-max-fun-diff[*OF fg f g*] *transD*[*OF s.TRANS - True*]
unfolding *s.isMaxim-def* **by** *auto*
hence *?fh* = *?gh* **using** *isMaxim-max-fun-diff*[*OF fh f h*] **by** *blast*
thus *?thesis* **using** *True* **unfolding** *s.max2-def* **by** *simp*
next
case *False*
with *max-fun-diff-in*[*OF fg f g*] *max-fun-diff-in*[*OF gh g h*] **have** *True*: (*?gh*, *?fg*) ∈ *s*
by (*blast intro*: *s.in-notinI*)
hence ∗: *g ?fg* = *h ?fg* **by** (*rule max-fun-diff-le-eq*[*OF - gh g h* ∗])
hence *s.isMaxim* $\{a \in$ *Field s. f a* $\neq$ *h a*$\}$ *?fg* **using** *isMaxim-max-fun-diff*[*OF gh g h*]
isMaxim-max-fun-diff[*OF fg f g*] *True transD*[*OF s.TRANS*, *of - - ?fg*]
unfolding *s.isMaxim-def* **by** *auto*
hence *?fh* = *?fg* **using** *isMaxim-max-fun-diff*[*OF fh f h*] **by** *blast*
thus *?thesis* **using** *False* **unfolding** *s.max2-def* **by** *simp*
qed
qed

definition *oexp* **where**
oexp = {(*f*, *g*) . *f* ∈ *FINFUNC* ∧ *g* ∈ *FINFUNC* ∧
((*let m* = *s.max-fun-diff f g in* (*f m*, *g m*) ∈ *r*) ∨ *f* = *g*)}

lemma *Field-oexp*: *Field oexp* = *FINFUNC*
unfolding *oexp-def FinFunc-def* **by** (*auto simp*: *Let-def Field-def*)

lemma *oexp-Refl*: *Refl oexp*
unfolding *refl-on-def Field-oexp* **unfolding** *oexp-def* **by** (*auto simp*: *Let-def*)

lemma *oexp-trans*: *trans oexp*
proof (*unfold trans-def*, *safe*)
fix *f g h* :: ′*b* ⇒ ′*a*
let *?fg* = *s.max-fun-diff f g*
and *?gh* = *s.max-fun-diff g h*
and *?fh* = *s.max-fun-diff f h*
assume *oexp*: (*f*, *g*) ∈ *oexp* (*g*, *h*) ∈ *oexp*

thus $(f, h) \in$ *oexp*
proof (*cases* $f = g \vee g = h$)
case *False*
with *oexp* **have** $f \in$ *FINFUNC* $g \in$ *FINFUNC* $h \in$ *FINFUNC*
$(f\ ?fg, g\ ?fg) \in r$ $(g\ ?gh, h\ ?gh) \in r$ **unfolding** *oexp-def Let-def* **by** *auto*
note $* =$ *this False*
show *?thesis*
proof (*cases* $f \neq h$)
case *True*
show *?thesis*
proof (*cases* $?fg = ?gh \longrightarrow f\ ?fg \neq h\ ?gh$)
case *True*
show *?thesis* **using** *max-fun-diff-max2*[*of f g h*, *OF True*] $*$ ‹$f \neq h$› *max-fun-diff-in*
r.max2-iff[*OF FINFUNCD FINFUNCD*] *r.max2-equals1*[*OF FINFUNCD FINFUNCD*] *max-fun-diff-le-eq*
s.in-notinI[*OF disjI1*] **unfolding** *oexp-def Let-def s.max2-def mem-Collect-eq* **by** *safe metis*
next
case *False* **with** $*$ **show** *?thesis* **unfolding** *oexp-def Let-def*
using *antisymD*[*OF r.ANTISYM*, *of g ?gh h ?gh*] *max-fun-diff-in*[*of g h*] **by** *auto*
qed
qed (*auto simp*: *oexp-def* $*(3)$)
qed *auto*
qed

lemma *oexp-Preorder*: *Preorder oexp*
unfolding *preorder-on-def* **using** *oexp-Refl oexp-trans* **by** *blast*

lemma *oexp-antisym*: *antisym oexp*
proof (*unfold antisym-def*, *safe*, *rule ccontr*)
fix *f g* **assume** $(f, g) \in$ *oexp* $(g, f) \in$ *oexp* $g \neq f$
thus *False* **using** *refl-onD*[*OF r.REFL FINFUNCD*] *max-fun-diff-in* **unfolding** *oexp-def Let-def*
by (*auto dest*!: *antisymD*[*OF r.ANTISYM*] *simp*: *s.max-fun-diff-commute*)
qed

lemma *oexp-Partial-order*: *Partial-order oexp*
unfolding *partial-order-on-def* **using** *oexp-Preorder oexp-antisym* **by** *blast*

lemma *oexp-Total*: *Total oexp*
unfolding *total-on-def Field-oexp* **unfolding** *oexp-def* **using** *FINFUNCD max-fun-diff-in*
by (*auto simp*: *Let-def s.max-fun-diff-commute intro*!: *r.in-notinI*)

lemma *oexp-Linear-order*: *Linear-order oexp*
unfolding *linear-order-on-def* **using** *oexp-Partial-order oexp-Total* **by** *blast*

definition *const* $= (\lambda x.$ *if* $x \in$ *Field s then r.zero else undefined*)

lemma *const-in*[*simp*]: $x \in$ *Field s* $\Longrightarrow$ *const* $x =$ *r.zero*
  unfolding *const-def* **by** *auto*

lemma *const-notin*[*simp*]: $x \notin$ *Field s* $\Longrightarrow$ *const* $x =$ *undefined*
  unfolding *const-def* **by** *auto*

lemma *const-Int-Field*[*simp*]: *Field s* $\cap - \{x.$ *const* $x =$ *r.zero*$\} = \{\}$
  by *auto*

lemma *const-FINFUNC*[*simp*]: *Field r* $\neq \{\} \Longrightarrow$ *const* $\in$ *FINFUNC*
  unfolding *FinFunc-def Func-def fin-support-def support-def const-def Int-iff mem-Collect-eq*
  using *r.zero-in-Field* **by** (*metis* (*lifting*) *Collect-empty-eq finite.emptyI*)

lemma *const-least*:
  assumes *Field r* $\neq \{\}$ $f \in$ *FINFUNC*
  shows (*const*, f) $\in$ *oexp*
proof (*cases* $f =$ *const*)
  case *True* **thus** *?thesis* **using** *refl-onD*[*OF oexp-Refl*] *assms*(*2*) **unfolding** *Field-oexp*
by *auto*
next
  case *False*
  with *assms* **show** *?thesis* **using** *max-fun-diff-in*[*of f const*]
    unfolding *oexp-def Let-def* **by** (*auto intro*: *r.zero-smallest FinFuncD simp*:
s.max-fun-diff-commute)
qed

lemma *support-not-const*:
  assumes $F \subseteq$ *FINFUNC* **and** *const* $\notin F$
  shows $\forall f \in F.$ *finite* (*SUPP f*) $\wedge$ *SUPP f* $\neq \{\}$ $\wedge$ *SUPP f* $\subseteq$ *Field s*
proof (*intro ballI conjI*)
  fix f **assume** $f \in F$
  thus *finite* (*SUPP f*) *SUPP f* $\subseteq$ *Field s*
    using *assms*(*1*) **unfolding** *FinFunc-def fin-support-def support-def* **by** *auto*
  show *SUPP f* $\neq \{\}$
  proof (*rule ccontr*, *unfold not-not*)
    assume *SUPP f* $= \{\}$
    moreover from ‹$f \in F$› *assms*(*1*) **have** $f \in$ *FINFUNC* **by** *blast*
    ultimately have $f =$ *const*
      by (*auto simp*: *fun-eq-iff support-def FinFunc-def Func-def const-def*)
    with *assms*(*2*) ‹$f \in F$› **show** *False* **by** *blast*
  qed
qed

lemma *maxim-isMaxim-support*:
  assumes f: $F \subseteq$ *FINFUNC* **and** *const* $\notin F$
  shows $\forall f \in F.$ *s.isMaxim* (*SUPP f*) (*s.maxim* (*SUPP f*))
  using *support-not-const*[*OF assms*] **by** (*auto intro*!: *s.maxim-isMaxim*)

lemma *oexp-empty2*: *Field s* = {} $\Longrightarrow$ *oexp* = {(λx. *undefined*, λx. *undefined*)}
unfolding *oexp-def FinFunc-def fin-support-def support-def* **by** *auto*

lemma *oexp-empty*: $[\![$*Field r* = {}; *Field s* $\neq$ {}$]\!]$ $\Longrightarrow$ *oexp* = {}
unfolding *oexp-def FinFunc-def Let-def* **by** *auto*

lemma *fun-upd-FINFUNC*: $[\![f \in$ *FINFUNC*; $x \in$ *Field s*; $y \in$ *Field r*$]\!]$ $\Longrightarrow f(x := y) \in$ *FINFUNC*
unfolding *FinFunc-def Func-def fin-support-def*
by (*auto intro*: *finite-subset*[*OF support-upd-subset*])

lemma *fun-upd-same-oexp*:
assumes $(f, g) \in$ *oexp* $f\ x = g\ x$ $x \in$ *Field s* $y \in$ *Field r*
shows $(f(x := y), g(x := y)) \in$ *oexp*
proof −
from *assms*(*1*) *fun-upd-FINFUNC*[*OF* - *assms*(*3*,*4*)] **have** *fg*: $f(x := y) \in$ *FINFUNC* $g(x := y) \in$ *FINFUNC*
unfolding *oexp-def* **by** *auto*
moreover from *assms*(*2*) **have** *s.max-fun-diff* $(f(x := y))$ $(g(x := y))$ = *s.max-fun-diff* $f\ g$
unfolding *s.max-fun-diff-def* **by** *auto metis*
ultimately show *?thesis* **using** *assms refl-onD*[*OF r.REFL*] **unfolding** *oexp-def Let-def* **by** *auto*
qed

lemma *fun-upd-smaller-oexp*:
assumes $f \in$ *FINFUNC* $x \in$ *Field s* $y \in$ *Field r* $(y, f\ x) \in r$
shows $(f(x := y), f) \in$ *oexp*
using *assms fun-upd-FINFUNC*[*OF assms*(*1*−*3*)] *s.maxim-singleton*[*of x*]
unfolding *oexp-def FinFunc-def Let-def fin-support-def s.max-fun-diff-def* **by** (*auto simp*: *fun-eq-iff*)

lemma *oexp-wf-Id*: *wf* (*oexp* − *Id*)
proof (*cases Field r* = {} $\vee$ *Field s* = {})
case *True* **thus** *?thesis* **using** *oexp-empty oexp-empty2* **by** *fastforce*
next
case *False*
hence *Fields*: *Field s* $\neq$ {} *Field r* $\neq$ {} **by** *simp-all*
hence [*simp*]: *r.zero* $\in$ *Field r* **by** (*intro r.zero-in-Field*)
have *const*[*simp*]: $\bigwedge F$. $[\![$*const* $\in F$; $F \subseteq$ *FINFUNC*$]\!]$ $\Longrightarrow \exists f0 \in F. \forall f \in F. (f0, f) \in$ *oexp*
using *const-least*[*OF Fields*(*2*)] **by** *auto*
show *?thesis*
unfolding *Linear-order-wf-diff-Id*[*OF oexp-Linear-order*] *Field-oexp*
proof (*intro allI impI*)
fix *A* **assume** *A*: $A \subseteq$ *FINFUNC* $A \neq$ {}
{ **fix** *y F*
have $F \subseteq$ *FINFUNC* $\wedge$ $(\exists f \in F.\ y =$ *s.maxim* (*SUPP f*)) $\longrightarrow$
$(\exists f0 \in F. \forall f \in F. (f0, f) \in$ *oexp*) (**is** *?P F y*)

proof (*induct y arbitrary*: *F rule*: *s.well-order-induct*)
case (*1 y*)
show *?case*
proof (*intro impI*, *elim conjE bexE*)
fix *f* **assume** *F*: *F* ⊆ *FINFUNC f* ∈ *F y* = *s.maxim* (*SUPP f*)
thus ∃*f0*∈*F*. ∀*f*∈*F*. (*f0*, *f*) ∈ *oexp*
proof (*cases const* ∈ *F*)
case *False*
with *F* **have** *maxF*: ∀*f* ∈ *F*. *s.isMaxim* (*SUPP f*) (*s.maxim* (*SUPP f*))
and *SUPPF*: ∀*f* ∈ *F*. *finite* (*SUPP f*) ∧ *SUPP f* ≠ {} ∧ *SUPP f* ⊆ *Field s*
using *maxim-isMaxim-support support-not-const* **by** *auto*
define *z* **where** *z* = *s.minim* {*s.maxim* (*SUPP f*) | *f*. *f* ∈ *F*}
from *F SUPPF maxF* **have** *zmin*: *s.isMinim* {*s.maxim* (*SUPP f*) | *f*. *f* ∈ *F*} *z*
unfolding *z-def* **by** (*intro s.minim-isMinim*) (*auto simp*: *s.isMaxim-def*)
with *F* **have** *zy*: (*z*, *y*) ∈ *s* **unfolding** *s.isMinim-def* **by** *auto*
hence *zField*: *z* ∈ *Field s* **unfolding** *Field-def* **by** *auto*
define *x0* **where** *x0* = *r.minim* {*f z* | *f*. *f* ∈ *F* ∧ *z* = *s.maxim* (*SUPP f*)}
from *F*(*1*,*2*) *maxF*(*1*) *SUPPF zmin*
have *x0min*: *r.isMinim* {*f z* | *f*. *f* ∈ *F* ∧ *z* = *s.maxim* (*SUPP f*)} *x0*
unfolding *x0-def s.isMaxim-def s.isMinim-def*
by (*blast intro*!: *r.minim-isMinim FinFuncD*[*of - r s*])
with *maxF*(*1*) *SUPPF F*(*1*) **have** *x0Field*: *x0* ∈ *Field r*
unfolding *r.isMinim-def s.isMaxim-def* **by** (*auto intro*!: *FINFUNCD*)
from *x0min maxF*(*1*) *SUPPF F*(*1*) **have** *x0notzero*: *x0* ≠ *r.zero*
unfolding *r.isMinim-def s.isMaxim-def FinFunc-def Func-def support-def*
by *fastforce*
define *G* **where** *G* = {*f*(*z* := *r.zero*) | *f*. *f* ∈ *F* ∧ *z* = *s.maxim* (*SUPP f*) ∧ *f z* = *x0*}
from *zmin x0min* **have** *G* ≠ {} **unfolding** *G-def z-def s.isMinim-def r.isMinim-def* **by** *blast*
have *GF*: *G* ⊆ (λ*f*. *f*(*z* := *r.zero*)) ‘ *F* **unfolding** *G-def* **by** *auto*
have *G* ⊆ *fin-support r.zero* (*Field s*)
unfolding *FinFunc-def fin-support-def* **proof** *safe*
fix *g* **assume** *g* ∈ *G*
with *GF* **obtain** *f* **where** *f*: *f* ∈ *F g* = *f*(*z* := *r.zero*) **by** *auto*
with *SUPPF* **have** *finite* (*SUPP f*) **by** *blast*
with *f* **show** *finite* (*SUPP g*)
by (*elim finite-subset*[*rotated*]) (*auto simp*: *support-def*)
qed
moreover from *F GF zField* **have** *G* ⊆ *Func* (*Field s*) (*Field r*)
using *Func-upd*[*of - Field s Field r z r.zero*] **unfolding** *FinFunc-def* **by** *auto*
ultimately have *G*: *G* ⊆ *FINFUNC* **unfolding** *FinFunc-def* **by** *blast*
hence ∃*g0*∈*G*. ∀*g*∈*G*. (*g0*, *g*) ∈ *oexp*
proof (*cases const* ∈ *G*)
case *False*

with G **have** *maxG*: $\forall g \in G.$ *s.isMaxim* (*SUPP* g) (*s.maxim* (*SUPP* g))
and *SUPPG*: $\forall g \in G.$ *finite* (*SUPP* g) $\wedge$ *SUPP* $g \neq \{\}$ $\wedge$ *SUPP* $g \subseteq$ *Field s*
using *maxim-isMaxim-support support-not-const* **by** *auto*
define y' **where** $y' =$ *s.minim* $\{$*s.maxim* (*SUPP* f) $\mid f.\ f \in G\}$
from G *SUPPG maxG* ‹$G \neq \{\}$› **have** y'*min*: *s.isMinim* $\{$*s.maxim* (*SUPP* f) $\mid f.\ f \in G\}$ y'
unfolding y'*-def* **by** (*intro s.minim-isMinim*) (*auto simp*: *s.isMaxim-def*)
moreover
have $\forall g \in G.\ z \notin$ *SUPP* g **unfolding** *support-def G-def* **by** *auto*
moreover
{ fix g **assume** g: $g \in G$
then obtain f **where** $f \in F$ $g = f(z := r.zero)$ **and** z: $z =$ *s.maxim* (*SUPP* f)
unfolding *G-def* **by** *auto*
with *SUPPF bspec*[*OF SUPPG g*] **have** (*s.maxim* (*SUPP* g), z) $\in s$
unfolding z **by** (*intro s.maxim-mono*) *auto*
}
moreover from y'*min* **have** $\bigwedge g.\ g \in G \Longrightarrow (y',$ *s.maxim* (*SUPP* g)) $\in s$
unfolding *s.isMinim-def* **by** *auto*
ultimately have $y' \neq z$ $(y', z) \in s$ **using** *maxG*
unfolding *s.isMinim-def s.isMaxim-def* **by** *auto*
with *zy* **have** $y' \neq y$ $(y', y) \in s$ **using** *antisymD*[*OF s.ANTISYM*] *transD*[*OF s.TRANS*]
by *blast+*
moreover from ‹$G \neq \{\}$› **have** $\exists g \in G.\ y' =$ *wo-rel.maxim s* (*SUPP* g) **using** y'*min*
by (*auto simp*: *G-def s.isMinim-def*)
ultimately show *?thesis* **using** *mp*[*OF spec*[*OF mp*[*OF spec*[*OF 1*]]], *of* y' G] G **by** *auto*
qed *simp*
then obtain *g0* **where** *g0*: $g0 \in G$ $\forall g \in G.\ (g0, g) \in$ *oexp* **by** *blast*
hence *g0z*: $g0\ z = r.zero$ **unfolding** *G-def* **by** *auto*
define *f0* **where** $f0 = g0(z := x0)$
with *x0notzero zField* **have** *SUPP*: *SUPP* $f0 =$ *SUPP* $g0 \cup \{z\}$ **unfolding** *support-def* **by** *auto*
from *g0z* **have** *f0z*: $f0(z := r.zero) = g0$ **unfolding** *f0-def fun-upd-upd* **by** *auto*
have *f0*: $f0 \in F$ **using** *x0min g0*(*1*)
Func-elim[*OF subsetD*[*OF subset-trans*[*OF F*(*1*)[*unfolded FinFunc-def*] *Int-lower1*]] *zField*]
unfolding *f0-def r.isMinim-def G-def* **by** (*force simp*: *fun-upd-idem*)
from *g0*(*1*) *maxF*(*1*) **have** *maxf0*: *s.maxim* (*SUPP* *f0*) $= z$ **unfolding** *SUPP G-def*
by (*intro s.maxim-equality*) (*auto simp*: *s.isMaxim-def*)
show *?thesis*
proof (*intro bexI*[*OF - f0*] *ballI*)

fix *f* **assume** *f*: *f* ∈ *F*
show (*f0*, *f*) ∈ *oexp*
proof (*cases f0* = *f*)
case *True* **thus** *?thesis* **by** (*metis F*(*1*) *Field-oexp f0 in-mono oexp-Refl refl-onD*)
next
case *False*
thus *?thesis*
proof (*cases s.maxim* (*SUPP f*) = *z* ∧ *f z* = *x0*)
case *True*
with *f* **have** *f*(*z* := *r.zero*) ∈ *G* **unfolding** *G-def* **by** *blast*
with *g0*(*2*) *f0z* **have** (*f0*(*z* := *r.zero*), *f*(*z* := *r.zero*)) ∈ *oexp* **by** *auto*
hence *oexp*: (*f0*(*z* := *r.zero*, *z* := *x0*), *f*(*z* := *r.zero*, *z* := *x0*)) ∈ *oexp*
by (*elim fun-upd-same-oexp*[*OF* - - *zField x0Field*]) *simp*
with *f F*(*1*) *x0min True*
have (*f*(*z* := *x0*), *f*) ∈ *oexp* **unfolding** *G-def r.isMinim-def*
by (*intro fun-upd-smaller-oexp*[*OF* - *zField x0Field*]) *auto*
with *oexp* **show** *?thesis* **using** *transD*[*OF oexp-trans*, *of f0 f*(*z* := *x0*) *f*]
unfolding *f0-def* **by** *auto*
next
case *False* **note** *notG* = *this*
thus *?thesis*
proof (*cases s.maxim* (*SUPP f*) = *z*)
case *True*
with *notG* **have** *f0 z* ≠ *f z* **unfolding** *f0-def* **by** *auto*
hence *f0 z* ≠ *f z* **by** *metis*
with *True maxf0 f0 f SUPPF* **have** *s.max-fun-diff f0 f* = *z*
using *s.maxim-Un*[*of SUPP f0 SUPP f*, *unfolded s.max2-def*]
unfolding *max-fun-diff-alt* **by** (*intro trans*[*OF s.maxim-Int*]) *auto*
moreover
from *x0min True f* **have** (*x0*, *f z*) ∈ *r* **unfolding** *r.isMinim-def* **by** *auto*
ultimately show *?thesis* **using** *f f0 F*(*1*) **unfolding** *oexp-def f0-def* **by** *auto*
next
case *False*
with *notG* **have** ∗: (*z*, *s.maxim* (*SUPP f*)) ∈ *s z* ≠ *s.maxim* (*SUPP f*)
using *zmin f* **unfolding** *s.isMinim-def G-def* **by** *auto*
have *f0f*: *f0* (*s.maxim* (*SUPP f*)) = *r.zero*
proof (*rule ccontr*)
assume *f0* (*s.maxim* (*SUPP f*)) ≠ *r.zero*
with *f SUPPF maxF*(*1*) **have** *s.maxim* (*SUPP f*) ∈ *SUPP f0*
unfolding *support-def*[*of* - - *f0*] *s.isMaxim-def* **by** *auto*
with *SUPPF f0* **have** (*s.maxim* (*SUPP f*), *z*) ∈ *s* **unfolding**

```
maxf0[symmetric]
                by (auto intro: s.maxim-greatest)
              with * antisymD[OF s.ANTISYM] show False by simp
            qed
            moreover
            have f (s.maxim (SUPP f)) ≠ r.zero
              using bspec[OF maxF(1) f, unfolded s.isMaxim-def] by (auto
simp: support-def)
            with f0f * f f0 maxf0 SUPPF
              have s.max-fun-diff f0 f = s.maxim (SUPP f0 ∪ SUPP f)
            unfolding max-fun-diff-alt using s.maxim-Un[of SUPP f0 SUPP
f]
              by (intro s.maxim-Int) (auto simp: s.max2-def)
          moreover have s.maxim (SUPP f0 ∪ SUPP f) = s.maxim (SUPP
f)
                using s.maxim-Un[of SUPP f0 SUPP f] * maxf0 SUPPF f0 f
                by (auto simp: s.max2-def)
                 ultimately show ?thesis using f f0 F(1) maxF(1) SUPPF
unfolding oexp-def Let-def
                        by (fastforce simp: s.isMaxim-def intro!: r.zero-smallest
FINFUNCD)
          qed
        qed
      qed
    qed
  qed simp
  qed
 qed
} note * = mp[OF this]
from A(2) obtain f where f: f ∈ A by blast
with A(1) show ∃ a∈A. ∀ a'∈A. (a, a') ∈ oexp
proof (cases f = const)
  case False with f A(1) show ?thesis using maxim-isMaxim-support[of {f}]
    by (intro *[of - s.maxim (SUPP f)]) (auto simp: s.isMaxim-def support-def)
qed simp
qed
qed

lemma oexp-Well-order: Well-order oexp
  unfolding well-order-on-def using oexp-Linear-order oexp-wf-Id by blast

interpretation o: wo-rel oexp by unfold-locales (rule oexp-Well-order)

lemma zero-oexp: Field r ≠ {} ⟹ o.zero = const
  by (rule sym[OF o.leq-zero-imp[OF const-least]])
    (auto intro!: o.zero-in-Field[unfolded Field-oexp] dest!: const-FINFUNC)

end
```

notation *wo-rel2.oexp* (**infixl** ^*o 90*)
lemmas *oexp-def* = *wo-rel2.oexp-def*[*unfolded wo-rel2-def*, *OF conjI*]
lemmas *oexp-Well-order* = *wo-rel2.oexp-Well-order*[*unfolded wo-rel2-def*, *OF conjI*]
lemmas *Field-oexp* = *wo-rel2.Field-oexp*[*unfolded wo-rel2-def*, *OF conjI*]

definition *ozero* = {}

lemma *ozero-Well-order*[*simp*]: *Well-order ozero*
unfolding *ozero-def* **by** *simp*

lemma *ozero-ordIso*[*simp*]: *ozero =o ozero*
unfolding *ozero-def ordIso-def iso-def*[*abs-def*] *embed-def bij-betw-def* **by** *auto*

lemma *Field-ozero*[*simp*]: *Field ozero* = {}
unfolding *ozero-def* **by** *simp*

lemma *iso-ozero-empty*[*simp*]: *r =o ozero* = (*r* = {})
unfolding *ozero-def ordIso-def iso-def*[*abs-def*] *embed-def bij-betw-def*
by (*auto dest*: *well-order-on-domain*)

lemma *ozero-ordLeq*:
assumes *Well-order r* **shows** *ozero* $\leq$*o r*
using *assms* **unfolding** *ozero-def ordLeq-def embed-def*[*abs-def*] *under-def* **by** *auto*

definition *oone* = {((),())}

lemma *oone-Well-order*[*simp*]: *Well-order oone*
unfolding *oone-def* **unfolding** *well-order-on-def linear-order-on-def partial-order-on-def preorder-on-def total-on-def refl-on-def trans-def antisym-def* **by** *auto*

lemma *Field-oone*[*simp*]: *Field oone* = {()}
unfolding *oone-def* **by** *simp*

lemma *oone-ordIso*: *oone =o* {(*x*,*x*)}
unfolding *ordIso-def oone-def well-order-on-def linear-order-on-def partial-order-on-def preorder-on-def total-on-def refl-on-def trans-def antisym-def*
by (*auto simp*: *iso-def embed-def bij-betw-def under-def inj-on-def intro*!: *exI*[*of* - λ-. *x*])

lemma *osum-ordLeqR*: *Well-order r* $\Longrightarrow$ *Well-order s* $\Longrightarrow$ *s* $\leq$*o r* +*o s*
unfolding *ordLeq-def2 underS-def*
by (*auto intro*!: *exI*[*of* - *Inr*] *osum-Well-order*) (*auto simp add*: *osum-def Field-def*)

lemma *osum-congL*:
assumes *r =o s* **and** *t*: *Well-order t*
shows *r* +*o t =o s* +*o t* (**is** *?L =o ?R*)
proof −
from *assms*(*1*) **obtain** *f* **where** *r*: *Well-order r* **and** *s*: *Well-order s* **and** *f*: *iso r s f*

unfolding *ordIso-def* **by** *blast*
let *?f* = *map-sum f id*
from *f* **have** *inj-on ?f* (*Field ?L*)
unfolding *Field-osum iso-def bij-betw-def inj-on-def* **by** *fastforce*
with *f* **have** *bij-betw ?f* (*Field ?L*) (*Field ?R*)
unfolding *Field-osum iso-def bij-betw-def image-image image-Un* **by** *auto*
moreover from *f* **have** *compat ?L ?R ?f*
unfolding *osum-def iso-iff3*[*OF r s*] *compat-def bij-betw-def*
by (*auto simp*: *map-prod-imageI*)
ultimately have *iso ?L ?R ?f* **by** (*subst iso-iff3*) (*auto intro*: *osum-Well-order r s t*)
thus *?thesis* **unfolding** *ordIso-def* **by** (*auto intro*: *osum-Well-order r s t*)
qed

lemma *osum-congR*:
assumes *r =o s* **and** *t*: *Well-order t*
shows *t +o r =o t +o s* (**is** *?L =o ?R*)
proof −
from *assms*(*1*) **obtain** *f* **where** *r*: *Well-order r* **and** *s*: *Well-order s* **and** *f*: *iso r s f*
unfolding *ordIso-def* **by** *blast*
let *?f* = *map-sum id f*
from *f* **have** *inj-on ?f* (*Field ?L*)
unfolding *Field-osum iso-def bij-betw-def inj-on-def* **by** *fastforce*
with *f* **have** *bij-betw ?f* (*Field ?L*) (*Field ?R*)
unfolding *Field-osum iso-def bij-betw-def image-image image-Un* **by** *auto*
moreover from *f* **have** *compat ?L ?R ?f*
unfolding *osum-def iso-iff3*[*OF r s*] *compat-def bij-betw-def*
by (*auto simp*: *map-prod-imageI*)
ultimately have *iso ?L ?R ?f* **by** (*subst iso-iff3*) (*auto intro*: *osum-Well-order r s t*)
thus *?thesis* **unfolding** *ordIso-def* **by** (*auto intro*: *osum-Well-order r s t*)
qed

lemma *osum-cong*:
assumes *t =o u* **and** *r =o s*
shows *t +o r =o u +o s*
using *ordIso-transitive*[*OF osum-congL*[*OF assms*(*1*)] *osum-congR*[*OF assms*(*2*)]]
assms[*unfolded ordIso-def*] **by** *auto*

lemma *Well-order-empty*[*simp*]: *Well-order* {}
unfolding *Field-empty* **by** (*rule well-order-on-empty*)

lemma *well-order-on-singleton*[*simp*]: *well-order-on* {*x*} {(*x*, *x*)}
unfolding *well-order-on-def linear-order-on-def partial-order-on-def preorder-on-def total-on-def*
Field-def refl-on-def trans-def antisym-def **by** *auto*

lemma *oexp-empty*[*simp*]:

assumes *Well-order r*
shows r ^o {} = {($\lambda x.$ *undefined*, $\lambda x.$ *undefined*)}
unfolding *oexp-def*[*OF assms Well-order-empty*] *FinFunc-def fin-support-def support-def* **by** *auto*

lemma *oexp-empty2*[*simp*]:
assumes *Well-order r* $r \neq$ {}
shows {} ^o r = {}
proof −
from *assms*(*2*) **have** *Field r* $\neq$ {} **unfolding** *Field-def* **by** *auto*
thus *?thesis* **unfolding** *oexp-def*[*OF Well-order-empty assms*(*1*)] *FinFunc-def fin-support-def support-def*
by *auto*
qed

lemma *oprod-zero*[*simp*]: {} $*o$ r = {} r $*o$ {} = {}
unfolding *oprod-def* **by** *simp-all*

lemma *oprod-congL*:
assumes r =o s **and** t: *Well-order t*
shows r $*o$ t =o s $*o$ t (**is** *?L* =o *?R*)
proof −
from *assms*(*1*) **obtain** *f* **where** *r*: *Well-order r* **and** *s*: *Well-order s* **and** *f*: *iso r s f*
unfolding *ordIso-def* **by** *blast*
let *?f* = *map-prod f id*
from *f* **have** *inj-on ?f* (*Field ?L*)
unfolding *Field-oprod iso-def bij-betw-def inj-on-def* **by** *fastforce*
with *f* **have** *bij-betw ?f* (*Field ?L*) (*Field ?R*)
unfolding *Field-oprod iso-def bij-betw-def* **by** (*auto intro*!: *map-prod-surj-on*)
moreover from *f* **have** *compat ?L ?R ?f*
unfolding *iso-iff3*[*OF r s*] *compat-def oprod-def bij-betw-def*
by (*auto simp*: *map-prod-imageI*)
ultimately have *iso ?L ?R ?f* **by** (*subst iso-iff3*) (*auto intro*: *oprod-Well-order r s t*)
thus *?thesis* **unfolding** *ordIso-def* **by** (*auto intro*: *oprod-Well-order r s t*)
qed

lemma *oprod-congR*:
assumes r =o s **and** t: *Well-order t*
shows t $*o$ r =o t $*o$ s (**is** *?L* =o *?R*)
proof −
from *assms*(*1*) **obtain** *f* **where** *r*: *Well-order r* **and** *s*: *Well-order s* **and** *f*: *iso r s f*
unfolding *ordIso-def* **by** *blast*
let *?f* = *map-prod id f*
from *f* **have** *inj-on ?f* (*Field ?L*)
unfolding *Field-oprod iso-def bij-betw-def inj-on-def* **by** *fastforce*
with *f* **have** *bij-betw ?f* (*Field ?L*) (*Field ?R*)

unfolding *Field-oprod iso-def bij-betw-def* **by** (*auto intro*!: *map-prod-surj-on*)
moreover from *f well-order-on-domain*[*OF r*] **have** *compat ?L ?R ?f*
unfolding *iso-iff3*[*OF r s*] *compat-def oprod-def bij-betw-def*
by (*auto simp*: *map-prod-imageI dest*: *inj-onD*)
ultimately have *iso ?L ?R ?f* **by** (*subst iso-iff3*) (*auto intro*: *oprod-Well-order r s t*)
thus *?thesis* **unfolding** *ordIso-def* **by** (*auto intro*: *oprod-Well-order r s t*)
qed

lemma *oprod-cong*:
assumes *t =o u* **and** *r =o s*
shows *t *o r =o u *o s*
using *ordIso-transitive*[*OF oprod-congL*[*OF assms*(*1*)] *oprod-congR*[*OF assms*(*2*)]]
assms[*unfolded ordIso-def*] **by** *auto*

lemma *Field-singleton*[*simp*]: *Field* {(*z*,*z*)} = {*z*}
by (*metis well-order-on-Field well-order-on-singleton*)

lemma *zero-singleton*[*simp*]: *zero* {(*z*,*z*)} = *z*
using *wo-rel.zero-in-Field*[*unfolded wo-rel-def*, *of* {(*z*, *z*)}] *well-order-on-singleton*[*of z*]
by *auto*

lemma *FinFunc-singleton*: *FinFunc* {(*z*,*z*)} *s* = {*λx*. *if x* ∈ *Field s then z else undefined*}
unfolding *FinFunc-def Func-def fin-support-def support-def*
by (*auto simp*: *fun-eq-iff split*: *if-split-asm intro*!: *finite-subset*[*of* - {}])

lemma *oone-ordIso-oexp*:
assumes *r =o oone* **and** *s*: *Well-order s*
shows *r ^o s =o oone* (**is** *?L =o ?R*)
proof −
from ‹*r =o oone*› **obtain** *f* **where** ∗: ∀ *x*∈*Field r*. ∀ *y*∈*Field r*. *x* = *y* **and** *f* ‘ *Field r* = {()}
and *r*: *Well-order r*
unfolding *ordIso-def oone-def* **by** (*auto simp add*: *iso-def* [*abs-def*] *bij-betw-def inj-on-def*)
then obtain *x* **where** *x* ∈ *Field r* **by** *auto*
with ∗ **have** *Fr*: *Field r* = {*x*} **by** *auto*
interpret *r*: *wo-rel r* **by** *unfold-locales* (*rule r*)
from *Fr well-order-on-domain*[*OF r*] *refl-onD*[*OF r.REFL*, *of x*] **have** *r-def*: *r* = {(*x*, *x*)} **by** *fast*
interpret *wo-rel2 r s* **by** *unfold-locales* (*rule r*, *rule s*)
have *bij-betw* (*λx*. ()) (*Field ?L*) (*Field ?R*)
unfolding *bij-betw-def Field-oexp* **by** (*auto simp*: *r-def FinFunc-singleton*)
moreover have *compat ?L ?R* (*λx*. ()) **unfolding** *compat-def oone-def* **by** *auto*
ultimately have *iso ?L ?R* (*λx*. ()) **using** *s oone-Well-order*
by (*subst iso-iff3*) (*auto intro*: *oexp-Well-order*)
thus *?thesis* **using** *s oone-Well-order* **unfolding** *ordIso-def* **by** (*auto intro*:

oexp-Well-order)
qed

context
fixes *r s t*
assumes *r*: *Well-order r*
assumes *s*: *Well-order s*
assumes *t*: *Well-order t*
begin

lemma *osum-ozeroL*: *ozero +o r =o r*
using *r* **unfolding** *osum-def ozero-def* **by** (*auto intro*: *map-prod-ordIso*)

lemma *osum-ozeroR*: *r +o ozero =o r*
using *r* **unfolding** *osum-def ozero-def* **by** (*auto intro*: *map-prod-ordIso*)

lemma *osum-assoc*: (*r +o s*) *+o t =o r +o s +o t* (**is** *?L =o ?R*)
proof −
let *?f* =
λ*rst. case rst of Inl* (*Inl r*) ⇒ *Inl r* | *Inl* (*Inr s*) ⇒ *Inr* (*Inl s*) | *Inr t* ⇒ *Inr* (*Inr t*)
have *bij-betw ?f* (*Field ?L*) (*Field ?R*)
unfolding *Field-osum bij-betw-def inj-on-def* **by** (*auto simp*: *image-Un image-iff*)
moreover
have *compat ?L ?R ?f*
proof (*unfold compat-def*, *safe*)
fix *a b*
assume (*a*, *b*) ∈ *?L*
thus (*?f a*, *?f b*) ∈ *?R*
unfolding *osum-def*[*of r +o s t*] *osum-def*[*of r s +o t*] *Field-osum*
unfolding *osum-def Field-osum image-iff image-Un map-prod-def*
by *fastforce*
qed
ultimately have *iso ?L ?R ?f* **using** *r s t* **by** (*subst iso-iff3*) (*auto intro*: *osum-Well-order*)
thus *?thesis* **using** *r s t* **unfolding** *ordIso-def* **by** (*auto intro*: *osum-Well-order*)
qed

lemma *osum-monoR*:
assumes *s <o t*
shows *r +o s <o r +o t* (**is** *?L <o ?R*)
proof −
from *assms* **obtain** *f* **where** *s*: *Well-order s* **and** *t*: *Well-order t* **and** *embedS s t f*
unfolding *ordLess-def* **by** *blast*
hence ∗: *inj-on f* (*Field s*) *compat s t f ofilter t* (*f ‘ Field s*) *f ‘ Field s* ⊂ *Field t*
using *embed-iff-compat-inj-on-ofilter*[*OF s t*, *of f*] *embedS-iff*[*OF s*, *of t f*]

unfolding *embedS-def* **by** *auto*
let *?f* = *map-sum id f*
from $*$(*1*) **have** *inj-on ?f* (*Field ?L*) **unfolding** *Field-osum inj-on-def* **by** *fastforce*
moreover
from $*$(*2*,*4*) **have** *compat ?L ?R ?f* **unfolding** *compat-def osum-def map-prod-def* **by** *fastforce*
moreover
interpret *t*: *wo-rel t* **by** *unfold-locales* (*rule t*)
interpret *rt*: *wo-rel ?R* **by** *unfold-locales* (*rule osum-Well-order*[*OF r t*])
from $*$(*3*) **have** *ofilter ?R* (*?f* ' *Field ?L*)
unfolding *t.ofilter-def rt.ofilter-def Field-osum image-Un image-image under-def*
by (*auto simp*: *osum-def intro*!: *imageI*) (*auto simp*: *Field-def*)
ultimately have *embed ?L ?R ?f* **using** *embed-iff-compat-inj-on-ofilter*[*of ?L ?R ?f*]
by (*auto intro*: *osum-Well-order r s t*)
moreover
from $*$(*4*) **have** *?f* ' *Field ?L* ⊂ *Field ?R* **unfolding** *Field-osum image-Un image-image* **by** *auto*
ultimately have *embedS ?L ?R ?f* **using** *embedS-iff*[*OF osum-Well-order*[*OF r s*], *of ?R ?f*] **by** *auto*
thus *?thesis* **unfolding** *ordLess-def* **by** (*auto intro*: *osum-Well-order r s t*)
qed

lemma *osum-monoL*:
assumes *r* ≤*o s*
shows *r* +*o t* ≤*o s* +*o t*
proof −
from *assms* **obtain** *f* **where** *f*: ∀ *a*∈*Field r*. *f a* ∈ *Field s* ∧ *f* ' *underS r a* ⊆ *underS s* (*f a*)
unfolding *ordLeq-def2* **by** *blast*
let *?f* = *map-sum f id*
from *f* **have** ∀ *a*∈*Field* (*r* +*o t*).
?f a ∈ *Field* (*s* +*o t*) ∧ *?f* ' *underS* (*r* +*o t*) *a* ⊆ *underS* (*s* +*o t*) (*?f a*)
unfolding *Field-osum underS-def* **by** (*fastforce simp*: *osum-def*)
thus *?thesis* **unfolding** *ordLeq-def2* **by** (*auto intro*: *osum-Well-order r s t*)
qed

lemma *oprod-ozeroL*: *ozero* $*$*o r* =*o ozero*
using *ozero-ordIso* **unfolding** *ozero-def* **by** *simp*

lemma *oprod-ozeroR*: *r* $*$*o ozero* =*o ozero*
using *ozero-ordIso* **unfolding** *ozero-def* **by** *simp*

lemma *oprod-ooneR*: *r* $*$*o oone* =*o r* (**is** *?L* =*o ?R*)
proof −
have *bij-betw fst* (*Field ?L*) (*Field ?R*) **unfolding** *Field-oprod bij-betw-def inj-on-def* **by** *simp*

moreover have *compat ?L ?R fst* **unfolding** *compat-def oprod-def* **by** *auto*
ultimately have *iso ?L ?R fst* **using** *r oone-Well-order*
by (*subst iso-iff3*) (*auto intro*: *oprod-Well-order*)
thus *?thesis* **using** *r oone-Well-order* **unfolding** *ordIso-def* **by** (*auto intro*: *oprod-Well-order*)
qed

lemma *oprod-ooneL*: *oone* $*o$ *r* $=o$ *r* (**is** *?L* $=o$ *?R*)
proof −
have *bij-betw snd* (*Field ?L*) (*Field ?R*) **unfolding** *Field-oprod bij-betw-def inj-on-def* **by** *simp*
moreover have *Refl r* **by** (*rule wo-rel.REFL*[*unfolded wo-rel-def*, *OF r*])
hence *compat ?L ?R snd* **unfolding** *compat-def oprod-def refl-on-def* **by** *auto*
ultimately have *iso ?L ?R snd* **using** *r oone-Well-order*
by (*subst iso-iff3*) (*auto intro*: *oprod-Well-order*)
thus *?thesis* **using** *r oone-Well-order* **unfolding** *ordIso-def* **by** (*auto intro*: *oprod-Well-order*)
qed

lemma *oprod-monoR*:
assumes *ozero* $<o$ *r s* $<o$ *t*
shows *r* $*o$ *s* $<o$ *r* $*o$ *t* (**is** *?L* $<o$ *?R*)
proof −
from *assms* **obtain** *f* **where** *s*: *Well-order s* **and** *t*: *Well-order t* **and** *embedS s t f*
unfolding *ordLess-def* **by** *blast*
hence $*$: *inj-on f* (*Field s*) *compat s t f ofilter t* (*f* ‘ *Field s*) *f* ‘ *Field s* ⊂ *Field t*
using *embed-iff-compat-inj-on-ofilter*[*OF s t*, *of f*] *embedS-iff*[*OF s*, *of t f*]
unfolding *embedS-def* **by** *auto*
let *?f* = *map-prod id f*
from $*$(*1*) **have** *inj-on ?f* (*Field ?L*) **unfolding** *Field-oprod inj-on-def* **by** *fastforce*
moreover
from $*$(*2*,*4*) *the-inv-into-f-f*[*OF* $*$(*1*)] **have** *compat ?L ?R ?f* **unfolding** *compat-def oprod-def*
by *auto* (*metis well-order-on-domain t*, *metis well-order-on-domain s*)
moreover
interpret *t*: *wo-rel t* **by** *unfold-locales* (*rule t*)
interpret *rt*: *wo-rel ?R* **by** *unfold-locales* (*rule oprod-Well-order*[*OF r t*])
from $*$(*3*) **have** *ofilter ?R* (*?f* ‘ *Field ?L*)
unfolding *t.ofilter-def rt.ofilter-def Field-oprod under-def*
by (*auto simp*: *oprod-def image-iff*) (*fast* | *metis r well-order-on-domain*)+
ultimately have *embed ?L ?R ?f* **using** *embed-iff-compat-inj-on-ofilter*[*of ?L ?R ?f*]
by (*auto intro*: *oprod-Well-order r s t*)
moreover
from *not-ordLess-ordIso*[*OF assms*(*1*)] **have** *r* ≠ {} **by** (*metis ozero-def ozero-ordIso*)
hence *Field r* ≠ {} **unfolding** *Field-def* **by** *auto*
with $*$(*4*) **have** *?f* ‘ *Field ?L* ⊂ *Field ?R* **unfolding** *Field-oprod*

by *auto* (*metis SigmaD2 SigmaI map-prod-surj-on*)
ultimately have *embedS ?L ?R ?f* **using** *embedS-iff*[*OF oprod-Well-order*[*OF r s*], *of ?R ?f*] **by** *auto*
thus *?thesis* **unfolding** *ordLess-def* **by** (*auto intro*: *oprod-Well-order r s t*)
qed

lemma *oprod-monoL*:
assumes $r \leq o\ s$
shows $r *o\ t \leq o\ s *o\ t$
proof −
from *assms* **obtain** *f* **where** *f*: $\forall a \in Field\ r.\ f\ a \in Field\ s \wedge f\ `\ underS\ r\ a \subseteq underS\ s\ (f\ a)$
unfolding *ordLeq-def2* **by** *blast*
let *?f* = *map-prod f id*
from *f* **have** $\forall a \in Field\ (r *o\ t).$
$?f\ a \in Field\ (s *o\ t) \wedge ?f\ `\ underS\ (r *o\ t)\ a \subseteq underS\ (s *o\ t)\ (?f\ a)$
unfolding *Field-oprod underS-def* **unfolding** *map-prod-def oprod-def* **by** *auto*
thus *?thesis* **unfolding** *ordLeq-def2* **by** (*auto intro*: *oprod-Well-order r s t*)
qed

lemma *oprod-assoc*: $(r *o\ s) *o\ t =o\ r *o\ s *o\ t$ (**is** *?L =o ?R*)
proof −
let $?f = \lambda((a,b),c).\ (a,b,c)$
have *bij-betw ?f* (*Field ?L*) (*Field ?R*)
unfolding *Field-oprod bij-betw-def inj-on-def* **by** (*auto simp*: *image-Un image-iff*)
moreover
have *compat ?L ?R ?f*
proof (*unfold compat-def*, *safe*)
fix *a1 a2 a3 b1 b2 b3*
assume $(((a1, a2), a3), ((b1, b2), b3)) \in ?L$
thus $((a1, a2, a3), (b1, b2, b3)) \in ?R$
unfolding *oprod-def*[*of r *o s t*] *oprod-def*[*of r s *o t*] *Field-oprod*
unfolding *oprod-def Field-oprod image-iff image-Un* **by** *fast*
qed
ultimately have *iso ?L ?R ?f* **using** *r s t* **by** (*subst iso-iff3*) (*auto intro*: *oprod-Well-order*)
thus *?thesis* **using** *r s t* **unfolding** *ordIso-def* **by** (*auto intro*: *oprod-Well-order*)
qed

lemma *oprod-osum*: $r *o\ (s +o\ t) =o\ r *o\ s +o\ r *o\ t$ (**is** *?L =o ?R*)
proof −
let $?f = \lambda(a,bc).\ case\ bc\ of\ Inl\ b \Rightarrow Inl\ (a,\ b) \mid Inr\ c \Rightarrow Inr\ (a,\ c)$
have *bij-betw ?f* (*Field ?L*) (*Field ?R*) **unfolding** *Field-oprod Field-osum bij-betw-def inj-on-def*
by (*fastforce simp*: *image-Un image-iff split*: *sum.splits*)
moreover
have *compat ?L ?R ?f*
proof (*unfold compat-def*, *intro allI impI*)

fix *a b*
assume $(a, b) \in ?L$
thus $(?f\ a, ?f\ b) \in ?R$
unfolding *oprod-def*[*of r s* $+o$ *t*] *osum-def*[*of r* $*o$ *s r* $*o$ *t*] *Field-oprod Field-osum*
unfolding *oprod-def osum-def Field-oprod Field-osum image-iff image-Un* **by** *auto*
qed
ultimately have *iso ?L ?R ?f* **using** *r s t*
by (*subst iso-iff3*) (*auto intro*: *oprod-Well-order osum-Well-order*)
thus *?thesis* **using** *r s t* **unfolding** *ordIso-def* **by** (*auto intro*: *oprod-Well-order osum-Well-order*)
qed

lemma *ozero-oexp*: $\neg$ (*s* $=o$ *ozero*) $\Longrightarrow$ *ozero* $\widehat{\ }o$ *s* $=o$ *ozero*
unfolding *oexp-def*[*OF ozero-Well-order s*] *FinFunc-def*
by *simp* (*metis Func-emp2 bot.extremum-uniqueI emptyE well-order-on-domain s subrelI*)

lemma *oone-oexp*: *oone* $\widehat{\ }o$ *s* $=o$ *oone* (**is** *?L* $=o$ *?R*)
by (*rule oone-ordIso-oexp*[*OF ordIso-reflexive*[*OF oone-Well-order*] *s*])

lemma *oexp-monoR*:
assumes *oone* $<o$ *r s* $<o$ *t*
shows *r* $\widehat{\ }o$ *s* $<o$ *r* $\widehat{\ }o$ *t* (**is** *?L* $<o$ *?R*)
proof $-$
interpret *rs*: *wo-rel2 r s* **by** *unfold-locales* (*rule r*, *rule s*)
interpret *rt*: *wo-rel2 r t* **by** *unfold-locales* (*rule r*, *rule t*)
interpret *rexpt*: *wo-rel r* $\widehat{\ }o$ *t* **by** *unfold-locales* (*rule rt.oexp-Well-order*)
interpret *r*: *wo-rel r* **by** *unfold-locales* (*rule r*)
interpret *s*: *wo-rel s* **by** *unfold-locales* (*rule s*)
interpret *t*: *wo-rel t* **by** *unfold-locales* (*rule t*)
have *Field r* $\neq \{\}$ **by** (*metis assms*(*1*) *internalize-ordLess not-psubset-empty*)
moreover
{ assume *Field r* $= \{r.zero\}$
hence $r = \{(r.zero, r.zero)\}$ **using** *refl-onD*[*OF r.REFL*, *of r.zero*] **unfolding** *Field-def* **by** *auto*
hence *r* $=o$ *oone* **by** (*metis oone-ordIso ordIso-symmetric*)
with *not-ordLess-ordIso*[*OF assms*(*1*)] **have** *False* **by** (*metis ordIso-symmetric*)
}
ultimately obtain *x* **where** *x*: $x \in$ *Field r r.zero* $\in$ *Field r x* $\neq$ *r.zero*
by (*metis insert-iff r.zero-in-Field subsetI subset-singletonD*)
moreover from *assms*(*2*) **obtain** *f* **where** *embedS s t f* **unfolding** *ordLess-def* **by** *blast*
hence $*$: *inj-on f* (*Field s*) *compat s t f ofilter t* (*f* ‘ *Field s*) *f* ‘ *Field s* $\subset$ *Field t*
using *embed-iff-compat-inj-on-ofilter*[*OF s t*, *of f*] *embedS-iff*[*OF s*, *of t f*]
unfolding *embedS-def* **by** *auto*
note *invff* = *the-inv-into-f-f*[*OF* $*$(*1*)] **and** *injfD* = *inj-onD*[*OF* $*$(*1*)]
define *F* **where** [*abs-def*]: *F g z* =

(*if* $z \in f$ ‘ *Field s then g* (*the-inv-into* (*Field s*) *f z*)
else if $z \in$ *Field t then r.zero else undefined*) **for** *g z*
from $*(4)$ *x*(*2*) *the-inv-into-f-eq*[*OF* $*(1)$] **have** *FLR*: *F* ‘ *Field ?L* $\subseteq$ *Field ?R*
unfolding *rt.Field-oexp rs.Field-oexp FinFunc-def Func-def fin-support-def support-def F-def*
by (*fastforce split*: *option.splits if-split-asm elim*!: *finite-surj*[*of - - f*])
have *inj-on F* (*Field ?L*) **unfolding** *rs.Field-oexp inj-on-def fun-eq-iff*
proof *safe*
fix *g h x* **assume** $g \in$ *FinFunc r s* $h \in$ *FinFunc r s* $\forall y.\ F\ g\ y = F\ h\ y$
with *invff* **show** $g\ x = h\ x$ **unfolding** *F-def fun-eq-iff FinFunc-def Func-def*
by *auto* (*metis image-eqI*)
qed
moreover
have *compat ?L ?R F* **unfolding** *compat-def rs.oexp-def rt.oexp-def*
proof (*safe elim*!: *bspec*[*OF iffD1*[*OF image-subset-iff FLR*[*unfolded rs.Field-oexp rt.Field-oexp*]]])
fix *g h* **assume** *gh*: $g \in$ *FinFunc r s* $h \in$ *FinFunc r s* $F\ g \neq F\ h$
let $m =$ *s.max-fun-diff g h in* (*g m*, *h m*) $\in r$
hence $g \neq h$ **by** *auto*
note *max-fun-diff-in* = *rs.max-fun-diff-in*[*OF* ‹$g \neq h$› *gh*(*1*,*2*)]
and *max-fun-diff-max* = *rs.max-fun-diff-max*[*OF* ‹$g \neq h$› *gh*(*1*,*2*)]
with $*(4)$ *invff* $*(2)$ **have** *t.max-fun-diff* (*F g*) (*F h*) = *f* (*s.max-fun-diff g h*)
unfolding *t.max-fun-diff-def compat-def*
by (*intro t.maxim-equality*) (*auto simp*: *t.isMaxim-def F-def dest*: *injfD*)
with *gh invff max-fun-diff-in*
show *let* $m =$ *t.max-fun-diff* (*F g*) (*F h*) *in* (*F g m*, *F h m*) $\in r$
unfolding *F-def Let-def* **by** (*auto simp*: *dest*: *injfD*)
qed
moreover
from *FLR* **have** *ofilter ?R* (*F* ‘ *Field ?L*)
unfolding *rexpt.ofilter-def under-def rs.Field-oexp rt.Field-oexp* **unfolding** *rt.oexp-def*
proof (*safe elim*!: *imageI*)
fix *g h* **assume** *gh*: $g \in$ *FinFunc r s* $h \in$ *FinFunc r t* $F\ g \in$ *FinFunc r t*
let $m =$ *t.max-fun-diff h* (*F g*) *in* (*h m*, *F g m*) $\in r$
thus $h \in F$ ‘ *FinFunc r s*
proof (*cases* $h = F\ g$)
case *False*
hence *max-Field*: *t.max-fun-diff h* (*F g*) $\in \{a \in$ *Field t*. $h\ a \neq F\ g\ a\}$
by (*rule rt.max-fun-diff-in*[*OF - gh*(*2*,*3*)])
{ assume $*$: *t.max-fun-diff h* (*F g*) $\notin f$ ‘ *Field s*
with *max-Field* **have** $**$: *F g* (*t.max-fun-diff h* (*F g*)) = *r.zero* **unfolding** *F-def* **by** *auto*
with $*$ *gh*(*4*) **have** *h* (*t.max-fun-diff h* (*F g*)) = *r.zero* **unfolding** *Let-def* **by** *auto*
with $**$ **have** *False* **using** *max-Field gh*(*2*,*3*) **unfolding** *FinFunc-def Func-def* **by** *auto*
}
hence *max-f-Field*: *t.max-fun-diff h* (*F g*) $\in f$ ‘ *Field s* **by** *blast*
{ fix *z* **assume** *z*: $z \in$ *Field t* $-$ *f* ‘ *Field s*

have $(t.max\text{-}fun\text{-}diff\ h\ (F\ g),\ z) \in t$
proof (*rule ccontr*)
assume $(t.max\text{-}fun\text{-}diff\ h\ (F\ g),\ z) \notin t$
hence $(z,\ t.max\text{-}fun\text{-}diff\ h\ (F\ g)) \in t$ **using** *t.in-notinI*[*of t.max-fun-diff h* (*F g*) *z*]
z max-Field **by** *auto*
hence $z \in f$ ' *Field s* **using** ∗(*3*) *max-f-Field* **unfolding** *t.ofilter-def under-def*
by *fastforce*
with *z* **show** *False* **by** *blast*
qed
hence $h\ z = r.zero$ **using** *rt.max-fun-diff-le-eq*[*OF - False gh*(*2*,*3*), *of z*]
z max-f-Field **unfolding** *F-def* **by** *auto*
} **note** ∗∗ = *this*
with ∗(*3*) *gh*(*2*) **have** $h = F\ (\lambda x.\ if\ x \in Field\ s\ then\ h\ (f\ x)\ else\ undefined)$ **using** *invff*
unfolding *F-def fun-eq-iff FinFunc-def Func-def Let-def t.ofilter-def under-def* **by** *auto*
moreover from *gh*(*2*) ∗(*1*,*3*) **have** $(\lambda x.\ if\ x \in Field\ s\ then\ h\ (f\ x)\ else\ undefined) \in FinFunc\ r\ s$
unfolding *FinFunc-def Func-def fin-support-def support-def t.ofilter-def under-def*
by (*auto intro*: *subset-inj-on elim*!: *finite-imageD*[*OF finite-subset*[*rotated*]])
ultimately show *?thesis* **by** (*rule image-eqI*)
qed *simp*
qed
ultimately have *embed ?L ?R F* **using** *embed-iff-compat-inj-on-ofilter*[*of ?L ?R F*]
by (*auto intro*: *oexp-Well-order r s t*)
moreover
from *FLR* **have** F ' $Field\ ?L \subset Field\ ?R$
proof (*intro psubsetI*)
from ∗(*4*) **obtain** *z* **where** z: $z \in Field\ t\ z \notin f$ ' *Field s* **by** *auto*
define *h* **where** [*abs-def*]: $h\ z' =$
$(if\ z' \in Field\ t\ then\ if\ z' = z\ then\ x\ else\ r.zero\ else\ undefined)$ **for** z'
from *z x*(*3*) **have** $rt.SUPP\ h = \{z\}$ **unfolding** *support-def h-def* **by** *simp*
with *x* **have** $h \in Field\ ?R$ **unfolding** *h-def rt.Field-oexp FinFunc-def Func-def fin-support-def*
by *auto*
moreover
{ **fix** *g*
from *z* **have** $F\ g\ z = r.zero\ h\ z = x$ **unfolding** *support-def h-def F-def* **by** *auto*
with *x*(*3*) **have** $F\ g \neq h$ **unfolding** *fun-eq-iff* **by** *fastforce*
}
hence $h \notin F$ ' *Field ?L* **by** *blast*
ultimately show F ' $Field\ ?L \neq Field\ ?R$ **by** *blast*
qed
ultimately have *embedS ?L ?R F* **using** *embedS-iff*[*OF rs.oexp-Well-order*, *of*

?R F] **by** *auto*
  thus *?thesis* **unfolding** *ordLess-def* **using** *r s t* **by** (*auto intro*: *oexp-Well-order*)
qed

lemma *oexp-monoL*:
  assumes $r \leq o\ s$
  shows $r\ \hat{}o\ t \leq o\ s\ \hat{}o\ t$
proof −
  interpret *rt*: *wo-rel2 r t* **by** *unfold-locales* (*rule r*, *rule t*)
  interpret *st*: *wo-rel2 s t* **by** *unfold-locales* (*rule s*, *rule t*)
  interpret *r*: *wo-rel r* **by** *unfold-locales* (*rule r*)
  interpret *s*: *wo-rel s* **by** *unfold-locales* (*rule s*)
  interpret *t*: *wo-rel t* **by** *unfold-locales* (*rule t*)
  show *?thesis*
  proof (*cases* $t = \{\}$)
    case *True* **thus** *?thesis* **using** *r s* **unfolding** *ordLeq-def2 underS-def* **by** *auto*
  next
    case *False* **thus** *?thesis*
    proof (*cases* $r = \{\}$)
    case *True* **thus** *?thesis* **using** *t* ‹$t \neq \{\}$› *st.oexp-Well-order ozero-ordLeq*[*unfolded ozero-def*]
        by *auto*
    next
      case *False*
      from *assms* **obtain** *f* **where** *f*: *embed r s f* **unfolding** *ordLeq-def* **by** *blast*
      hence *f-underS*: $\forall\ a \in Field\ r.\ f\ a \in Field\ s \wedge f\ `\ underS\ r\ a \subseteq underS\ s\ (f\ a)$
        using *embed-in-Field embed-underS2 rt.rWELL* **by** *fastforce*
      from *f* ‹$t \neq \{\}$› *False* **have** ∗: $Field\ r \neq \{\}$ $Field\ s \neq \{\}$ $Field\ t \neq \{\}$
        unfolding *Field-def embed-def under-def bij-betw-def* **by** *auto*
        with *f* **obtain** *x* **where** $s.zero = f\ x\ x \in Field\ r$ **unfolding** *embed-def bij-betw-def*
        using *s.zero-under subsetD*[*OF under-Field*[*of r*]]
      by (*metis* (*no-types*, *lifting*) *f-inv-into-f f-underS inv-into-into r.zero-in-Field*)
       with *f* **have** *fz*: $f\ r.zero = s.zero$ **and** *inj*: *inj-on f* (*Field r*) **and** *compat*: *compat r s f*
        unfolding *embed-iff-compat-inj-on-ofilter*[*OF r s*] *compat-def*
        by (*fastforce intro*: *s.leq-zero-imp*)+
      let $?f = \lambda g\ x.\ if\ x \in Field\ t\ then\ f\ (g\ x)\ else\ undefined$
      { fix *g* **assume** *g*: $g \in Field\ (r\ \hat{}o\ t)$
        with *fz f-underS* **have** *Field-fg*: $?f\ g \in Field\ (s\ \hat{}o\ t)$
         unfolding *st.Field-oexp rt.Field-oexp FinFunc-def Func-def fin-support-def support-def*
          by (*auto elim*!: *finite-subset*[*rotated*])
        moreover
        have $?f\ `\ underS\ (r\ \hat{}o\ t)\ g \subseteq underS\ (s\ \hat{}o\ t)\ (?f\ g)$
        proof *safe*
          fix *h*
          assume *h-underS*: $h \in underS\ (r\ \hat{}o\ t)\ g$
          hence $h \in Field\ (r\ \hat{}o\ t)$ **unfolding** *underS-def Field-def* **by** *auto*

with *fz f-underS* **have** *Field-fh*: *?f h* ∈ *Field* (*s* ^*o* *t*)
unfolding *st.Field-oexp rt.Field-oexp FinFunc-def Func-def fin-support-def support-def*
by (*auto elim*!: *finite-subset*[*rotated*])
from *h-underS* **have** *h* ≠ *g* **and** *hg*: (*h*, *g*) ∈ *rt.oexp* **unfolding** *underS-def* **by** *auto*
with *f inj* **have** *neq*: *?f h* ≠ *?f g*
unfolding *fun-eq-iff inj-on-def rt.oexp-def map-option-case FinFunc-def Func-def Let-def*
by *simp metis*
with *hg* **have** *t.max-fun-diff* (*?f h*) (*?f g*) = *t.max-fun-diff h g* **unfolding** *rt.oexp-def*
using *rt.max-fun-diff*[*OF* ‹*h* ≠ *g*›] *rt.max-fun-diff-in*[*OF* ‹*h* ≠ *g*›]
by (*subst t.max-fun-diff-def*, *intro t.maxim-equality*)
(*auto simp*: *t.isMaxim-def intro*: *inj-onD*[*OF inj*] *intro*!: *rt.max-fun-diff-max*)
with *Field-fg Field-fh hg fz f-underS compat neq* **have** (*?f h*, *?f g*) ∈ *st.oexp*
using *rt.max-fun-diff*[*OF* ‹*h* ≠ *g*›] *rt.max-fun-diff-in*[*OF* ‹*h* ≠ *g*›] **unfolding** *st.Field-oexp*
unfolding *rt.oexp-def st.oexp-def Let-def compat-def* **by** *auto*
with *neq* **show** *?f h* ∈ *underS* (*s* ^*o* *t*) (*?f g*) **unfolding** *underS-def* **by** *auto*
qed
ultimately have *?f g* ∈ *Field* (*s* ^*o* *t*) ∧ *?f* ‘ *underS* (*r* ^*o* *t*) *g* ⊆ *underS* (*s* ^*o* *t*) (*?f g*)
by *blast*
}
thus *?thesis* **unfolding** *ordLeq-def2* **by** (*fastforce intro*: *oexp-Well-order r s t*)
qed
qed
qed

lemma *ordLeq-oexp2*:
assumes *oone* <*o* *r*
shows *s* ≤*o* *r* ^*o* *s*
proof −
interpret *rs*: *wo-rel2 r s* **by** *unfold-locales* (*rule r*, *rule s*)
interpret *r*: *wo-rel r* **by** *unfold-locales* (*rule r*)
interpret *s*: *wo-rel s* **by** *unfold-locales* (*rule s*)
from *assms well-order-on-domain*[*OF r*] **obtain** *x* **where**
x: *x* ∈ *Field r r.zero* ∈ *Field r x* ≠ *r.zero*
unfolding *ordLess-def oone-def embedS-def*[*abs-def*] *bij-betw-def embed-def under-def*
by (*auto simp*: *image-def*)
(*metis* (*lifting*) *equals0D mem-Collect-eq r.zero-in-Field singletonI*)
let *?f* = λ*a b*. *if b* ∈ *Field s then if b* = *a then x else r.zero else undefined*
from *x*(*3*) **have** *SUPP*: ⋀*y*. *y* ∈ *Field s* ⟹ *rs.SUPP* (*?f y*) = {*y*} **unfolding** *support-def* **by** *auto*
{ fix *y* **assume** *y*: *y* ∈ *Field s*

with *x*(*1*,*2*) *SUPP* **have** *?f y* ∈ *Field* (*r* ^*o s*) **unfolding** *rs.Field-oexp*
by (*auto simp*: *FinFunc-def Func-def fin-support-def*)
moreover
have *?f* ‘ *underS s y* ⊆ *underS* (*r* ^*o s*) (*?f y*)
proof *safe*
fix *z*
assume *z* ∈ *underS s y*
hence *z*: *z* ≠ *y* (*z*, *y*) ∈ *s z* ∈ *Field s* **unfolding** *underS-def Field-def* **by** *auto*
from *x*(*3*) *y z*(*1*,*3*) **have** *?f z* ≠ *?f y* **unfolding** *fun-eq-iff* **by** *auto*
moreover
{ from *x*(*1*,*2*) **have** *?f z* ∈ *FinFunc r s ?f y* ∈ *FinFunc r s*
unfolding *FinFunc-def Func-def fin-support-def* **by** (*auto simp*: *SUPP*[*OF z*(*3*)] *SUPP*[*OF y*])
moreover
from *x*(*3*) *y z*(*1*,*2*) *refl-onD*[*OF s.REFL*] **have** *s.max-fun-diff* (*?f z*) (*?f y*) = *y*
unfolding *rs.max-fun-diff-alt SUPP*[*OF z*(*3*)] *SUPP*[*OF y*]
by (*intro s.maxim-equality*) (*auto simp*: *s.isMaxim-def*)
ultimately have (*?f z*, *?f y*) ∈ *rs.oexp* **using** *y x*(*1*)
unfolding *rs.oexp-def Let-def* **by** *auto*
}
ultimately show *?f z* ∈ *underS* (*r* ^*o s*) (*?f y*) **unfolding** *underS-def* **by** *blast*
qed
ultimately have *?f y* ∈ *Field* (*r* ^*o s*) ∧ *?f* ‘ *underS s y* ⊆ *underS* (*r* ^*o s*) (*?f y*) **by** *blast*
}
thus *?thesis* **unfolding** *ordLeq-def2* **by** (*fast intro*: *oexp-Well-order r s*)
qed

lemma *FinFunc-osum*:
fg ∈ *FinFunc r* (*s* +*o t*) = (*fg o Inl* ∈ *FinFunc r s* ∧ *fg o Inr* ∈ *FinFunc r t*)
(**is** *?L* = (*?R1* ∧ *?R2*))
proof *safe*
assume *?L*
from ‹*?L*› **show** *?R1* **unfolding** *FinFunc-def Field-osum Func-def Int-iff fin-support-Field-osum o-def*
by (*auto split*: *sum.splits*)
from ‹*?L*› **show** *?R2* **unfolding** *FinFunc-def Field-osum Func-def Int-iff fin-support-Field-osum o-def*
by (*auto split*: *sum.splits*)
next
assume *?R1 ?R2*
thus *?L* **unfolding** *FinFunc-def Field-osum Func-def*
by (*auto simp*: *fin-support-Field-osum o-def image-iff split*: *sum.splits*) (*metis sumE*)
qed

lemma *max-fun-diff-eq-Inl*:
assumes *wo-rel.max-fun-diff* $(s +o\ t)$ $(case\text{-}sum\ f1\ g1)$ $(case\text{-}sum\ f2\ g2) = Inl\ x$
case-sum f1 g1 $\neq$ *case-sum f2 g2*
case-sum f1 g1 $\in$ *FinFunc r* $(s +o\ t)$ *case-sum f2 g2* $\in$ *FinFunc r* $(s +o\ t)$
shows *wo-rel.max-fun-diff s f1 f2* $= x$ (**is** *?P*) *g1* $=$ *g2* (**is** *?Q*)
proof −
interpret *st*: *wo-rel s +o t* **by** *unfold-locales* (*rule osum-Well-order*[*OF s t*])
interpret *s*: *wo-rel s* **by** *unfold-locales* (*rule s*)
interpret *rst*: *wo-rel2 r s +o t* **by** *unfold-locales* (*rule r*, *rule osum-Well-order*[*OF s t*])
from *assms*(*1*) **have** $*$: *st.isMaxim* $\{a \in$ *Field* $(s +o\ t)$. *case-sum f1 g1 a* $\neq$ *case-sum f2 g2 a*$\}$ (*Inl x*)
using *rst.isMaxim-max-fun-diff*[*OF assms*(*2*−*4*)] **by** *simp*
hence *s.isMaxim* $\{a \in$ *Field s*. *f1 a* $\neq$ *f2 a*$\}$ *x*
unfolding *st.isMaxim-def s.isMaxim-def Field-osum* **by** (*auto simp*: *osum-def*)
thus *?P* **unfolding** *s.max-fun-diff-def* **by** (*rule s.maxim-equality*)
from *assms*(*3*,*4*) **have** $**$: *g1* $\in$ *FinFunc r t g2* $\in$ *FinFunc r t* **unfolding** *FinFunc-osum*
by (*auto simp*: *o-def*)
show *?Q*
proof
fix *x*
from $*$ $**$ **show** *g1 x* $=$ *g2 x* **unfolding** *st.isMaxim-def Field-osum FinFunc-def Func-def fun-eq-iff*
unfolding *osum-def* **by** (*case-tac x* $\in$ *Field t*) *auto*
qed
qed

lemma *max-fun-diff-eq-Inr*:
assumes *wo-rel.max-fun-diff* $(s +o\ t)$ $(case\text{-}sum\ f1\ g1)$ $(case\text{-}sum\ f2\ g2) = Inr\ x$
case-sum f1 g1 $\neq$ *case-sum f2 g2*
case-sum f1 g1 $\in$ *FinFunc r* $(s +o\ t)$ *case-sum f2 g2* $\in$ *FinFunc r* $(s +o\ t)$
shows *wo-rel.max-fun-diff t g1 g2* $= x$ (**is** *?P*) *g1* $\neq$ *g2* (**is** *?Q*)
proof −
interpret *st*: *wo-rel s +o t* **by** *unfold-locales* (*rule osum-Well-order*[*OF s t*])
interpret *t*: *wo-rel t* **by** *unfold-locales* (*rule t*)
interpret *rst*: *wo-rel2 r s +o t* **by** *unfold-locales* (*rule r*, *rule osum-Well-order*[*OF s t*])
from *assms*(*1*) **have** $*$: *st.isMaxim* $\{a \in$ *Field* $(s +o\ t)$. *case-sum f1 g1 a* $\neq$ *case-sum f2 g2 a*$\}$ (*Inr x*)
using *rst.isMaxim-max-fun-diff*[*OF assms*(*2*−*4*)] **by** *simp*
hence *t.isMaxim* $\{a \in$ *Field t*. *g1 a* $\neq$ *g2 a*$\}$ *x*
unfolding *st.isMaxim-def t.isMaxim-def Field-osum* **by** (*auto simp*: *osum-def*)
thus *?P ?Q* **unfolding** *t.max-fun-diff-def fun-eq-iff*
by (*auto intro*: *t.maxim-equality simp*: *t.isMaxim-def*)
qed

lemma *oexp-osum*: $r\ \hat{}o\ (s +o\ t) =o\ (r\ \hat{}o\ s) *o\ (r\ \hat{}o\ t)$ (**is** *?R =o ?L*)

proof (*rule ordIso-symmetric*)
interpret *rst*: *wo-rel2 r s +o t* **by** *unfold-locales* (*rule r*, *rule osum-Well-order*[*OF s t*])
interpret *rs*: *wo-rel2 r s* **by** *unfold-locales* (*rule r*, *rule s*)
interpret *rt*: *wo-rel2 r t* **by** *unfold-locales* (*rule r*, *rule t*)
let *?f* = λ(*f*, *g*). *case-sum f g*
have *bij-betw ?f* (*Field ?L*) (*Field ?R*)
unfolding *bij-betw-def rst.Field-oexp rs.Field-oexp rt.Field-oexp Field-oprod* **proof** (*intro conjI*)
show *inj-on ?f* (*FinFunc r s* × *FinFunc r t*) **unfolding** *inj-on-def*
by (*auto simp*: *fun-eq-iff split*: *sum.splits*)
show *?f* ‘ (*FinFunc r s* × *FinFunc r t*) = *FinFunc r* (*s +o t*)
proof *safe*
fix *fg* **assume** *fg* ∈ *FinFunc r* (*s +o t*)
thus *fg* ∈ *?f* ‘ (*FinFunc r s* × *FinFunc r t*)
by (*intro image-eqI*[*of - -* (*fg o Inl*, *fg o Inr*)])
(*auto simp*: *FinFunc-osum fun-eq-iff split*: *sum.splits*)
qed (*auto simp*: *FinFunc-osum o-def*)
qed
moreover have *compat ?L ?R ?f*
unfolding *compat-def rst.Field-oexp rs.Field-oexp rt.Field-oexp oprod-def*
unfolding *rst.oexp-def Let-def rs.oexp-def rt.oexp-def*
by (*fastforce simp*: *Field-osum FinFunc-osum o-def split*: *sum.splits*
dest: *max-fun-diff-eq-Inl max-fun-diff-eq-Inr*)
ultimately have *iso ?L ?R ?f* **using** *r s t*
by (*subst iso-iff3*) (*auto intro*: *oexp-Well-order oprod-Well-order osum-Well-order*)
thus *?L =o ?R* **using** *r s t* **unfolding** *ordIso-def*
by (*auto intro*: *oexp-Well-order oprod-Well-order osum-Well-order*)
qed

definition *rev-curr f b* = (*if b* ∈ *Field t then* λ*a*. *f* (*a*, *b*) *else undefined*)

lemma *rev-curr-FinFunc*:
assumes *Field*: *Field r* ≠ {}
shows *rev-curr* ‘ (*FinFunc r* (*s* ∗*o t*)) = *FinFunc* (*r* ^*o s*) *t*
proof *safe*
interpret *rs*: *wo-rel2 r s* **by** *unfold-locales* (*rule r*, *rule s*)
interpret *rst*: *wo-rel2 r* ^*o s t* **by** *unfold-locales* (*rule oexp-Well-order*[*OF r s*], *rule t*)
fix *g* **assume** *g*: *g* ∈ *FinFunc r* (*s* ∗*o t*)
hence *finite* (*rst.SUPP* (*rev-curr g*)) ∀ *x* ∈ *Field t*. *finite* (*rs.SUPP* (*rev-curr g x*))
unfolding *FinFunc-def Field-oprod rs.Field-oexp Func-def fin-support-def support-def*
rs.zero-oexp[*OF Field*] *rev-curr-def* **by** (*auto simp*: *fun-eq-iff rs.const-def elim*!: *finite-surj*)
with *g* **show** *rev-curr g* ∈ *FinFunc* (*r* ^*o s*) *t*
unfolding *FinFunc-def Field-oprod rs.Field-oexp Func-def*
by (*auto simp*: *rev-curr-def fin-support-def*)

next
  interpret *rs*: *wo-rel2 r s* **by** *unfold-locales* (*rule r*, *rule s*)
  interpret *rst*: *wo-rel2 r ^o s t* **by** *unfold-locales* (*rule oexp-Well-order*[*OF r s*], *rule t*)
  fix *fg* **assume** ∗: *fg* ∈ *FinFunc* (*r ^o s*) *t*
  let *?g* = λ(*a*, *b*). *if* (*a*, *b*) ∈ *Field* (*s* ∗*o t*) *then fg b a else undefined*
  show *fg* ∈ *rev-curr* ‘ *FinFunc r* (*s* ∗*o t*)
  proof (*rule image-eqI*[*of - - ?g*])
    show *fg* = *rev-curr ?g*
    proof
      fix *x*
      from ∗ **show** *fg x* = *rev-curr ?g x*
        unfolding *FinFunc-def rs.Field-oexp Func-def rev-curr-def Field-oprod* **by** *auto*
    qed
  next
    have ∗∗: (⋃ *g* ∈ *fg* ‘ *Field t*. *rs.SUPP g*) =
          (⋃ *g* ∈ *fg* ‘ *Field t* − {*rs.const*}. *rs.SUPP g*)
      unfolding *support-def* **by** *auto*
    from ∗ **have** ∗∗∗: ∀ *g* ∈ *fg* ‘ *Field t*. *finite* (*rs.SUPP g*) *finite* (*rst.SUPP fg*)
     unfolding *rs.Field-oexp FinFunc-def Func-def fin-support-def Option.these-def* **by** *force*+
    hence *finite* (*fg* ‘ *Field t* − {*rs.const*}) **using** ∗
      unfolding *support-def rs.zero-oexp*[*OF Field*] *FinFunc-def Func-def*
      by (*elim finite-surj*[*of - - fg*]) (*fastforce simp*: *image-iff Option.these-def*)
    with ∗∗∗ **have** *finite* ((⋃ *g* ∈ *fg* ‘ *Field t*. *rs.SUPP g*) × *rst.SUPP fg*)
      by (*subst* ∗∗) (*auto intro*!: *finite-cartesian-product*)
    with ∗ **show** *?g* ∈ *FinFunc r* (*s* ∗*o t*)
       unfolding *Field-oprod rs.Field-oexp FinFunc-def Func-def fin-support-def Option.these-def*
       support-def rs.zero-oexp[*OF Field*] **by** (*auto elim*!: *finite-subset*[*rotated*])
  qed
qed

lemma *rev-curr-app-FinFunc*[*elim*!]:
  ⟦*f* ∈ *FinFunc r* (*s* ∗*o t*); *z* ∈ *Field t*⟧ ⟹ *rev-curr f z* ∈ *FinFunc r s*
   unfolding *rev-curr-def FinFunc-def Func-def Field-oprod fin-support-def support-def*
  by (*auto elim*: *finite-surj*)

lemma *max-fun-diff-oprod*:
  assumes *Field*: *Field r* ≠ {} **and** *f* ≠ *g f* ∈ *FinFunc r* (*s* ∗*o t*) *g* ∈ *FinFunc r* (*s* ∗*o t*)
  defines *m* ≡ *wo-rel.max-fun-diff t* (*rev-curr f*) (*rev-curr g*)
  shows *wo-rel.max-fun-diff* (*s* ∗*o t*) *f g* =
    (*wo-rel.max-fun-diff s* (*rev-curr f m*) (*rev-curr g m*), *m*)
proof −
  interpret *st*: *wo-rel s* ∗*o t* **by** *unfold-locales* (*rule oprod-Well-order*[*OF s t*])
  interpret *s*: *wo-rel s* **by** *unfold-locales* (*rule s*)

interpret *t*: *wo-rel t* **by** *unfold-locales* (*rule t*)
interpret *r-st*: *wo-rel2 r s* $*o$ *t* **by** *unfold-locales* (*rule r*, *rule oprod-Well-order*[*OF s t*])
interpret *rs*: *wo-rel2 r s* **by** *unfold-locales* (*rule r*, *rule s*)
interpret *rst*: *wo-rel2 r* ^*o s t* **by** *unfold-locales* (*rule oexp-Well-order*[*OF r s*], *rule t*)
from *fun-unequal-in-support*[*OF assms*(*2*), *of Field* (*s* $*o$ *t*) *Field r Field r*] *assms*(*3*,*4*)
have *diff1*: *rev-curr f* ≠ *rev-curr g*
rev-curr f ∈ *FinFunc* (*r* ^*o s*) *t rev-curr g* ∈ *FinFunc* (*r* ^*o s*) *t* **using** *rev-curr-FinFunc*[*OF Field*]
unfolding *fun-eq-iff rev-curr-def*[*abs-def*] *FinFunc-def support-def Field-oprod*
by *auto fast*
hence *diff2*: *rev-curr f m* ≠ *rev-curr g m rev-curr f m* ∈ *FinFunc r s rev-curr g m* ∈ *FinFunc r s*
using *rst.max-fun-diff*[*OF diff1*] *assms*(*3*,*4*) *rst.max-fun-diff-in* **unfolding** *m-def* **by** *auto*
show *?thesis* **unfolding** *st.max-fun-diff-def*
proof (*intro st.maxim-equality*, *unfold st.isMaxim-def Field-oprod*, *safe*)
show *s.max-fun-diff* (*rev-curr f m*) (*rev-curr g m*) ∈ *Field s*
using *rs.max-fun-diff-in*[*OF diff2*] **by** *auto*
next
show *m* ∈ *Field t* **using** *rst.max-fun-diff-in*[*OF diff1*] **unfolding** *m-def* **by** *auto*
next
assume *f* (*s.max-fun-diff* (*rev-curr f m*) (*rev-curr g m*), *m*) =
g (*s.max-fun-diff* (*rev-curr f m*) (*rev-curr g m*), *m*)
(**is** *f* (*?x*, *m*) = *g* (*?x*, *m*))
hence *rev-curr f m ?x* = *rev-curr g m ?x* **unfolding** *rev-curr-def* **by** *auto*
with *rs.max-fun-diff*[*OF diff2*] **show** *False* **by** *auto*
next
fix *x y* **assume** *f* (*x*, *y*) ≠ *g* (*x*, *y*) *x* ∈ *Field s y* ∈ *Field t*
thus ((*x*, *y*), (*s.max-fun-diff* (*rev-curr f m*) (*rev-curr g m*), *m*)) ∈ *s* $*o$ *t*
using *rst.max-fun-diff-in*[*OF diff1*] *rs.max-fun-diff-in*[*OF diff2*] *diff1 diff2*
rst.max-fun-diff-max[*OF diff1*, *of y*] *rs.max-fun-diff-le-eq*[*OF - diff2*, *of x*]
unfolding *oprod-def m-def rev-curr-def fun-eq-iff* **by** (*auto intro*: *s.in-notinI*)
qed
qed

lemma *oexp-oexp*: (*r* ^*o s*) ^*o t* =*o r* ^*o* (*s* $*o$ *t*) (**is** *?R* =*o ?L*)
proof (*cases r* = {})
case *True*
interpret *rs*: *wo-rel2 r s* **by** *unfold-locales* (*rule r*, *rule s*)
interpret *rst*: *wo-rel2 r* ^*o s t* **by** *unfold-locales* (*rule oexp-Well-order*[*OF r s*], *rule t*)
show *?thesis*
proof (*cases s* = {} ∨ *t* = {})
case *True* **with** ‹*r* = {}› **show** *?thesis*
by (*auto simp*: *oexp-empty*[*OF oexp-Well-order*[*OF Well-order-empty s*]]

intro!: *ordIso-transitive*[*OF ordIso-symmetric*[*OF oone-ordIso*] *oone-ordIso*]
ordIso-transitive[*OF oone-ordIso-oexp*[*OF ordIso-symmetric*[*OF oone-ordIso*] *t*] *oone-ordIso*])
next
case *False*
hence $s *o\ t \neq \{\}$ **unfolding** *oprod-def Field-def* **by** *fastforce*
with *False* **show** *?thesis*
using ‹$r = \{\}$› *ozero-ordIso*
by (*auto simp add*: *s t oprod-Well-order ozero-def*)
qed
next
case *False*
hence *Field*: *Field* $r \neq \{\}$ **by** (*metis Field-def Range-empty-iff Un-empty*)
show *?thesis*
proof (*rule ordIso-symmetric*)
interpret *r-st*: *wo-rel2 r s *o t* **by** *unfold-locales* (*rule r*, *rule oprod-Well-order*[*OF s t*])
interpret *rs*: *wo-rel2 r s* **by** *unfold-locales* (*rule r*, *rule s*)
interpret *rst*: *wo-rel2 r ^o s t* **by** *unfold-locales* (*rule oexp-Well-order*[*OF r s*], *rule t*)
have *bij*: *bij-betw rev-curr* (*Field ?L*) (*Field ?R*)
unfolding *bij-betw-def r-st.Field-oexp rst.Field-oexp Field-oprod* **proof** (*intro conjI*)
show *inj-on rev-curr* (*FinFunc r* (*s *o t*))
unfolding *inj-on-def FinFunc-def Func-def Field-oprod rs.Field-oexp rev-curr-def*[*abs-def*]
by (*auto simp*: *fun-eq-iff*) *metis*
show *rev-curr* ‘ (*FinFunc r* (*s *o t*)) = *FinFunc* (*r ^o s*) *t* **by** (*rule rev-curr-FinFunc*[*OF Field*])
qed
moreover
have *compat ?L ?R rev-curr*
unfolding *compat-def* **proof** *safe*
fix *fg1 fg2* **assume** *fg*: (*fg1*, *fg2*) ∈ *r ^o* (*s *o t*)
show (*rev-curr fg1*, *rev-curr fg2*) ∈ *r ^o s ^o t*
proof (*cases fg1 = fg2*)
assume *fg1* ≠ *fg2*
with *fg* **show** *?thesis*
using *rst.max-fun-diff-in*[*of rev-curr fg1 rev-curr fg2*]
max-fun-diff-oprod[*OF Field*, *of fg1 fg2*] *rev-curr-FinFunc*[*OF Field*, *symmetric*]
unfolding *r-st.Field-oexp rs.Field-oexp rst.Field-oexp* **unfolding** *r-st.oexp-def rst.oexp-def*
by (*auto simp*: *rs.oexp-def Let-def*) (*auto simp*: *rev-curr-def*[*abs-def*])
next
assume *fg1* = *fg2*
with *fg bij* **show** *?thesis* **unfolding** *r-st.Field-oexp rs.Field-oexp rst.Field-oexp bij-betw-def*
by (*auto simp*: *r-st.oexp-def rst.oexp-def*)
qed

```
    qed
    ultimately have iso ?L ?R rev-curr using r s t
      by (subst iso-iff3) (auto intro: oexp-Well-order oprod-Well-order)
    thus ?L =o ?R using r s t unfolding ordIso-def
      by (auto intro: oexp-Well-order oprod-Well-order)
  qed
qed

end

end
```

10 Cardinal-Order Relations

```
theory Cardinal-Order-Relation
imports Wellorder-Constructions
begin

declare
  card-order-on-well-order-on[simp]
  card-of-card-order-on[simp]
  card-of-well-order-on[simp]
  Field-card-of[simp]
  card-of-Card-order[simp]
  card-of-Well-order[simp]
  card-of-least[simp]
  card-of-unique[simp]
  card-of-mono1[simp]
  card-of-mono2[simp]
  card-of-cong[simp]
  card-of-Field-ordLess[simp]
  card-of-Field-ordIso[simp]
  card-of-underS[simp]
  ordLess-Field[simp]
  card-of-empty[simp]
  card-of-empty1[simp]
  card-of-image[simp]
  card-of-singl-ordLeq[simp]
  Card-order-singl-ordLeq[simp]
  card-of-Pow[simp]
  Card-order-Pow[simp]
  card-of-Plus1[simp]
  Card-order-Plus1[simp]
  card-of-Plus2[simp]
  Card-order-Plus2[simp]
  card-of-Plus-mono1[simp]
  card-of-Plus-mono2[simp]
  card-of-Plus-mono[simp]
  card-of-Plus-cong2[simp]
```

card-of-Plus-cong[*simp*]
card-of-Un-Plus-ordLeq[*simp*]
card-of-Times1[*simp*]
card-of-Times2[*simp*]
card-of-Times3[*simp*]
card-of-Times-mono1[*simp*]
card-of-Times-mono2[*simp*]
card-of-ordIso-finite[*simp*]
card-of-Times-same-infinite[*simp*]
card-of-Times-infinite-simps[*simp*]
card-of-Plus-infinite1[*simp*]
card-of-Plus-infinite2[*simp*]
card-of-Plus-ordLess-infinite[*simp*]
card-of-Plus-ordLess-infinite-Field[*simp*]
infinite-cartesian-product[*simp*]
cardSuc-Card-order[*simp*]
cardSuc-greater[*simp*]
cardSuc-ordLeq[*simp*]
cardSuc-ordLeq-ordLess[*simp*]
cardSuc-mono-ordLeq[*simp*]
cardSuc-invar-ordIso[*simp*]
card-of-cardSuc-finite[*simp*]
cardSuc-finite[*simp*]
card-of-Plus-ordLeq-infinite-Field[*simp*]
curr-in[*intro*, *simp*]

10.1 Cardinal of a set

lemma *card-of-inj-rel*: **assumes** *INJ*: $\bigwedge x\ y\ y'.\ [\![(x,y) \in R;\ (x,y') \in R]\!] \Longrightarrow y = y'$
shows $|\{y.\ \exists x.\ (x,y) \in R\}|$ <=o $|\{x.\ \exists y.\ (x,y) \in R\}|$
proof−
  let *?Y* = $\{y.\ \exists x.\ (x,y) \in R\}$ **let** *?X* = $\{x.\ \exists y.\ (x,y) \in R\}$
  let *?f* = $\lambda y.$ *SOME* $x.\ (x,y) \in R$
  have *?f* ‘ *?Y* <= *?X* **by** (*auto dest*: *someI*)
  moreover have *inj-on ?f ?Y*
  unfolding *inj-on-def* **proof**(*auto*)
    fix *y1 x1 y2 x2*
    assume ∗: (*x1*, *y1*) ∈ *R* (*x2*, *y2*) ∈ *R* **and** ∗∗: *?f y1* = *?f y2*
    hence (*?f y1*,*y1*) ∈ *R* **using** *someI*[*of* $\lambda x.\ (x,y1) \in R$] **by** *auto*
    moreover have (*?f y2*,*y2*) ∈ *R* **using** ∗ *someI*[*of* $\lambda x.\ (x,y2) \in R$] **by** *auto*
    ultimately show *y1* = *y2* **using** ∗∗ *INJ* **by** *auto*
  qed
  ultimately show |*?Y*| <=o |*?X*| **using** *card-of-ordLeq* **by** *blast*
qed

lemma *card-of-unique2*: $[\![$*card-order-on B r*; *bij-betw f A B*$]\!] \Longrightarrow r$ =o $|A|$
using *card-of-ordIso card-of-unique ordIso-equivalence* **by** *blast*

lemma *internalize-card-of-ordLess*:

$(\ |A|\ <o\ r) = (\exists\, B < \mathit{Field}\ r.\ |A| =o\ |B| \wedge |B| <o\ r)$
proof
  assume $|A| <o\ r$
  then obtain p **where** *1*: $\mathit{Field}\ p < \mathit{Field}\ r \wedge |A| =o\ p \wedge p <o\ r$
  using *internalize-ordLess*[*of* $|A|$ *r*] **by** *blast*
  hence *Card-order p* **using** *card-of-Card-order Card-order-ordIso2* **by** *blast*
  hence $|\mathit{Field}\ p| =o\ p$ **using** *card-of-Field-ordIso* **by** *blast*
  hence $|A| =o\ |\mathit{Field}\ p| \wedge |\mathit{Field}\ p| <o\ r$
  using *1 ordIso-equivalence ordIso-ordLess-trans* **by** *blast*
  thus $\exists\, B < \mathit{Field}\ r.\ |A| =o\ |B| \wedge |B| <o\ r$ **using** *1* **by** *blast*
next
  assume $\exists\, B < \mathit{Field}\ r.\ |A| =o\ |B| \wedge |B| <o\ r$
  thus $|A| <o\ r$ **using** *ordIso-ordLess-trans* **by** *blast*
qed

lemma *internalize-card-of-ordLess2*:
$(\ |A| <o\ |C|\ ) = (\exists\, B < C.\ |A| =o\ |B| \wedge |B| <o\ |C|\ )$
using *internalize-card-of-ordLess*[*of A* $|C|$] *Field-card-of*[*of C*] **by** *auto*

lemma *Card-order-omax*:
assumes *finite R* **and** $R \neq \{\}$ **and** $\forall\, r \in R.$ *Card-order r*
shows *Card-order* (*omax R*)
proof−
  have $\forall\, r \in R.$ *Well-order r*
  using *assms* **unfolding** *card-order-on-def* **by** *simp*
  thus *?thesis* **using** *assms* **apply** − **apply**(*drule omax-in*) **by** *auto*
qed

lemma *Card-order-omax2*:
assumes *finite I* **and** $I \neq \{\}$
shows *Card-order* (*omax* $\{|A\ i|\ |\ i.\ i \in I\}$)
proof−
  let $?R = \{|A\ i|\ |\ i.\ i \in I\}$
  have *finite ?R* **and** $?R \neq \{\}$ **using** *assms* **by** *auto*
  moreover have $\forall\, r \in ?R.$ *Card-order r*
  using *card-of-Card-order* **by** *auto*
  ultimately show *?thesis* **by**(*rule Card-order-omax*)
qed

10.2 Cardinals versus set operations on arbitrary sets

lemma *card-of-set-type*[*simp*]: $|UNIV{::}'a\ set| <o\ |UNIV{::}'a\ set\ set|$
using *card-of-Pow*[*of UNIV::'a set*] **by** *simp*

lemma *card-of-Un1*[*simp*]:
shows $|A| \leq o\ |A \cup B|$
using *inj-on-id*[*of A*] *card-of-ordLeq*[*of A* -] **by** *fastforce*

lemma *card-of-diff*[*simp*]:

shows $|A - B| \leq o\ |A|$
using *inj-on-id*[*of* $A - B$] *card-of-ordLeq*[*of* $A - B$ -] **by** *fastforce*

lemma *subset-ordLeq-strict*:
assumes $A \leq B$ **and** $|A| <o\ |B|$
shows $A < B$
proof−
{**assume** $\neg(A < B)$
hence $A = B$ **using** *assms*(*1*) **by** *blast*
hence *False* **using** *assms*(*2*) *not-ordLess-ordIso card-of-refl* **by** *blast*
}
thus *?thesis* **by** *blast*
qed

corollary *subset-ordLeq-diff*:
assumes $A \leq B$ **and** $|A| <o\ |B|$
shows $B - A \neq \{\}$
using *assms subset-ordLeq-strict* **by** *blast*

lemma *card-of-empty4*:
$|\{\}::'b\ set| <o\ |A::'a\ set| = (A \neq \{\})$
proof(*intro iffI notI*)
assume $*$: $|\{\}::'b\ set| <o\ |A|$ **and** $A = \{\}$
hence $|A| =o\ |\{\}::'b\ set|$
using *card-of-ordIso* **unfolding** *bij-betw-def inj-on-def* **by** *blast*
hence $|\{\}::'b\ set| =o\ |A|$ **using** *ordIso-symmetric* **by** *blast*
with $*$ **show** *False* **using** *not-ordLess-ordIso*[*of* $|\{\}::'b\ set|\ |A|$] **by** *blast*
next
assume $A \neq \{\}$
hence $(\neg\ (\exists f.\ inj\text{-}on\ f\ A \wedge f\ `\ A \subseteq \{\}))$
unfolding *inj-on-def* **by** *blast*
thus $|\ \{\}\ | <o\ |\ A\ |$
using *card-of-ordLess* **by** *blast*
qed

lemma *card-of-empty5*:
$|A| <o\ |B| \Longrightarrow B \neq \{\}$
using *card-of-empty not-ordLess-ordLeq* **by** *blast*

lemma *Well-order-card-of-empty*:
Well-order $r \Longrightarrow |\{\}| \leq o\ r$ **by** *simp*

lemma *card-of-UNIV*[*simp*]:
$|A :: 'a\ set| \leq o\ |UNIV :: 'a\ set|$
using *card-of-mono1*[*of* A] **by** *simp*

lemma *card-of-UNIV2*[*simp*]:
Card-order $r \Longrightarrow (r :: 'a\ rel) \leq o\ |UNIV :: 'a\ set|$
using *card-of-UNIV*[*of Field* r] *card-of-Field-ordIso*

ordIso-symmetric ordIso-ordLeq-trans **by** *blast*

lemma *card-of-Pow-mono*[*simp*]:
assumes $|A| \leq o\ |B|$
shows $|Pow\ A| \leq o\ |Pow\ B|$
proof−
obtain *f* **where** *inj-on f A* $\wedge$ *f* ‘ $A \leq B$
using *assms card-of-ordLeq*[*of A B*] **by** *auto*
hence *inj-on* (*image f*) (*Pow A*) $\wedge$ (*image f*) ‘ (*Pow A*) $\leq$ (*Pow B*)
by (*auto simp add*: *inj-on-image-Pow image-Pow-mono*)
thus *?thesis* **using** *card-of-ordLeq*[*of Pow A*] **by** *metis*
qed

lemma *ordIso-Pow-mono*[*simp*]:
assumes $r \leq o\ r'$
shows $|Pow(Field\ r)| \leq o\ |Pow(Field\ r')|$
using *assms card-of-mono2 card-of-Pow-mono* **by** *blast*

lemma *card-of-Pow-cong*[*simp*]:
assumes $|A| =o\ |B|$
shows $|Pow\ A| =o\ |Pow\ B|$
proof−
obtain *f* **where** *bij-betw f A B*
using *assms card-of-ordIso*[*of A B*] **by** *auto*
hence *bij-betw* (*image f*) (*Pow A*) (*Pow B*)
by (*auto simp add*: *bij-betw-image-Pow*)
thus *?thesis* **using** *card-of-ordIso*[*of Pow A*] **by** *auto*
qed

lemma *ordIso-Pow-cong*[*simp*]:
assumes $r =o\ r'$
shows $|Pow(Field\ r)| =o\ |Pow(Field\ r')|$
using *assms card-of-cong card-of-Pow-cong* **by** *blast*

corollary *Card-order-Plus-empty1*:
Card-order r $\Longrightarrow$ $r =o\ |(Field\ r) <+> \{\}|$
using *card-of-Plus-empty1 card-of-Field-ordIso ordIso-equivalence* **by** *blast*

corollary *Card-order-Plus-empty2*:
Card-order r $\Longrightarrow$ $r =o\ |\{\} <+> (Field\ r)|$
using *card-of-Plus-empty2 card-of-Field-ordIso ordIso-equivalence* **by** *blast*

lemma *Card-order-Un1*:
shows *Card-order r* $\Longrightarrow$ $|Field\ r| \leq o\ |(Field\ r) \cup B|$
using *card-of-Un1 card-of-Field-ordIso ordIso-symmetric ordIso-ordLeq-trans* **by** *auto*

lemma *card-of-Un2*[*simp*]:
shows $|A| \leq o\ |B \cup A|$

using *inj-on-id*[*of A*] *card-of-ordLeq*[*of A* -] **by** *fastforce*

lemma *Card-order-Un2*:
shows *Card-order r* $\Longrightarrow$ $|Field\ r| \leq o\ |A \cup (Field\ r)|$
using *card-of-Un2 card-of-Field-ordIso ordIso-symmetric ordIso-ordLeq-trans* **by** *auto*

lemma *Un-Plus-bij-betw*:
assumes $A\ Int\ B = \{\}$
shows $\exists f.\ bij\text{-}betw\ f\ (A \cup B)\ (A <+> B)$
proof−
  let $?f = \lambda\ c.\ if\ c \in A\ then\ Inl\ c\ else\ Inr\ c$
  have $bij\text{-}betw\ ?f\ (A \cup B)\ (A <+> B)$
  using *assms* **by**(*unfold bij-betw-def inj-on-def*, *auto*)
  thus *?thesis* **by** *blast*
qed

lemma *card-of-Un-Plus-ordIso*:
assumes $A\ Int\ B = \{\}$
shows $|A \cup B| =o\ |A <+> B|$
using *assms card-of-ordIso*[*of* $A \cup B$] *Un-Plus-bij-betw*[*of A B*] **by** *auto*

lemma *card-of-Un-Plus-ordIso1*:
$|A \cup B| =o\ |A <+> (B - A)|$
using *card-of-Un-Plus-ordIso*[*of A* $B - A$] **by** *auto*

lemma *card-of-Un-Plus-ordIso2*:
$|A \cup B| =o\ |(A - B) <+> B|$
using *card-of-Un-Plus-ordIso*[*of* $A - B$ *B*] **by** *auto*

lemma *card-of-Times-singl1*: $|A| =o\ |A \times \{b\}|$
proof−
  have $bij\text{-}betw\ fst\ (A \times \{b\})\ A$ **unfolding** *bij-betw-def inj-on-def* **by** *force*
  thus *?thesis* **using** *card-of-ordIso ordIso-symmetric* **by** *blast*
qed

corollary *Card-order-Times-singl1*:
Card-order r $\Longrightarrow r =o\ |(Field\ r) \times \{b\}|$
using *card-of-Times-singl1*[*of - b*] *card-of-Field-ordIso ordIso-equivalence* **by** *blast*

lemma *card-of-Times-singl2*: $|A| =o\ |\{b\} \times A|$
proof−
  have $bij\text{-}betw\ snd\ (\{b\} \times A)\ A$ **unfolding** *bij-betw-def inj-on-def* **by** *force*
  thus *?thesis* **using** *card-of-ordIso ordIso-symmetric* **by** *blast*
qed

corollary *Card-order-Times-singl2*:
Card-order r $\Longrightarrow r =o\ |\{a\} \times (Field\ r)|$
using *card-of-Times-singl2*[*of - a*] *card-of-Field-ordIso ordIso-equivalence* **by** *blast*

lemma *card-of-Times-assoc*: $|(A \times B) \times C| =o\ |A \times B \times C|$
proof $-$
  let $?f = \lambda((a,b),c).\ (a,(b,c))$
  have $A \times B \times C \subseteq\ ?f\ `\ ((A \times B) \times C)$
  proof
    fix x **assume** $x \in A \times B \times C$
    then obtain $a\ b\ c$ **where** $*$: $a \in A\ b \in B\ c \in C\ x = (a,\ b,\ c)$ **by** *blast*
    let $?x = ((a,\ b),\ c)$
    from $*$ **have** $?x \in (A \times B) \times C\ x =\ ?f\ ?x$ **by** *auto*
    thus $x \in\ ?f\ `\ ((A \times B) \times C)$ **by** *blast*
  qed
  hence *bij-betw* $?f\ ((A \times B) \times C)\ (A \times B \times C)$
  unfolding *bij-betw-def inj-on-def* **by** *auto*
  thus *?thesis* **using** *card-of-ordIso* **by** *blast*
qed

corollary *Card-order-Times3*:
Card-order $r \Longrightarrow |Field\ r| \le o\ |(Field\ r) \times (Field\ r)|$
  by (*rule card-of-Times3*)

lemma *card-of-Times-cong1*[*simp*]:
assumes $|A| =o\ |B|$
shows $|A \times C| =o\ |B \times C|$
using *assms* **by** (*simp add*: *ordIso-iff-ordLeq*)

lemma *card-of-Times-cong2*[*simp*]:
assumes $|A| =o\ |B|$
shows $|C \times A| =o\ |C \times B|$
using *assms* **by** (*simp add*: *ordIso-iff-ordLeq*)

lemma *card-of-Times-mono*[*simp*]:
assumes $|A| \le o\ |B|$ **and** $|C| \le o\ |D|$
shows $|A \times C| \le o\ |B \times D|$
using *assms card-of-Times-mono1*[*of A B C*] *card-of-Times-mono2*[*of C D B*]
    ordLeq-transitive[*of* $|A \times C|$] **by** *blast*

corollary *ordLeq-Times-mono*:
assumes $r \le o\ r'$ **and** $p \le o\ p'$
shows $|(Field\ r) \times (Field\ p)| \le o\ |(Field\ r') \times (Field\ p')|$
using *assms card-of-mono2*[*of r r'*] *card-of-mono2*[*of p p'*] *card-of-Times-mono* **by** *blast*

corollary *ordIso-Times-cong1*[*simp*]:
assumes $r =o\ r'$
shows $|(Field\ r) \times C| =o\ |(Field\ r') \times C|$
using *assms card-of-cong card-of-Times-cong1* **by** *blast*

corollary *ordIso-Times-cong2*:

assumes $r =o\ r'$
shows $|A \times (Field\ r)| =o\ |A \times (Field\ r')|$
using *assms card-of-cong card-of-Times-cong2* **by** *blast*

lemma *card-of-Times-cong*[*simp*]:
assumes $|A| =o\ |B|$ **and** $|C| =o\ |D|$
shows $|A \times C| =o\ |B \times D|$
using *assms*
by (*auto simp add*: *ordIso-iff-ordLeq*)

corollary *ordIso-Times-cong*:
assumes $r =o\ r'$ **and** $p =o\ p'$
shows $|(Field\ r) \times (Field\ p)| =o\ |(Field\ r') \times (Field\ p')|$
using *assms card-of-cong*[*of r r'*] *card-of-cong*[*of p p'*] *card-of-Times-cong* **by** *blast*

lemma *card-of-Sigma-mono2*:
assumes *inj-on f* (I::$'i$ *set*) **and** f ‘ $I \leq$ (J::$'j$ *set*)
shows $|SIGMA\ i : I.\ (A::'j \Rightarrow 'a\ set)\ (f\ i)| \leq o\ |SIGMA\ j : J.\ A\ j|$
proof−
  let $?LEFT = SIGMA\ i : I.\ A\ (f\ i)$
  let $?RIGHT = SIGMA\ j : J.\ A\ j$
  obtain u **where** *u-def*: $u = (\lambda(i::'i,a::'a).\ (f\ i,a))$ **by** *blast*
  have *inj-on u ?LEFT* $\wedge$ u ‘$?LEFT \leq ?RIGHT$
  using *assms* **unfolding** *u-def inj-on-def* **by** *auto*
  thus *?thesis* **using** *card-of-ordLeq* **by** *blast*
qed

lemma *card-of-Sigma-mono*:
assumes *INJ*: *inj-on f I* **and** *IM*: f ‘ $I \leq J$ **and**
    LEQ: $\forall j \in J.\ |A\ j| \leq o\ |B\ j|$
shows $|SIGMA\ i : I.\ A\ (f\ i)| \leq o\ |SIGMA\ j : J.\ B\ j|$
proof−
  have $\forall i \in I.\ |A(f\ i)| \leq o\ |B(f\ i)|$
  using *IM LEQ* **by** *blast*
  hence $|SIGMA\ i : I.\ A\ (f\ i)| \leq o\ |SIGMA\ i : I.\ B\ (f\ i)|$
  using *card-of-Sigma-mono1*[*of I*] **by** *metis*
  moreover have $|SIGMA\ i : I.\ B\ (f\ i)| \leq o\ |SIGMA\ j : J.\ B\ j|$
  using *INJ IM card-of-Sigma-mono2* **by** *blast*
  ultimately show *?thesis* **using** *ordLeq-transitive* **by** *blast*
qed

lemma *ordLeq-Sigma-mono1*:
assumes $\forall i \in I.\ p\ i \leq o\ r\ i$
shows $|SIGMA\ i : I.\ Field(p\ i)| \leq o\ |SIGMA\ i : I.\ Field(r\ i)|$
using *assms* **by** (*auto simp add*: *card-of-Sigma-mono1*)

lemma *ordLeq-Sigma-mono*:
assumes *inj-on f I* **and** f ‘ $I \leq J$ **and**
    $\forall j \in J.\ p\ j \leq o\ r\ j$

shows $|SIGMA\ i : I.\ Field(p(f\ i))| \leq o\ |SIGMA\ j : J.\ Field(r\ j)|$
using *assms card-of-mono2 card-of-Sigma-mono*
  $[of\ f\ I\ J\ \lambda\ i.\ Field(p\ i)\ \lambda\ j.\ Field(r\ j)]$ **by** *metis*

lemma *card-of-Sigma-cong1*:
assumes $\forall i \in I.\ |A\ i| =o\ |B\ i|$
shows $|SIGMA\ i : I.\ A\ i| =o\ |SIGMA\ i : I.\ B\ i|$
using *assms* **by** (*auto simp add*: *card-of-Sigma-mono1 ordIso-iff-ordLeq*)

lemma *card-of-Sigma-cong2*:
assumes *bij-betw f* $(I::'i\ set)\ (J::'j\ set)$
shows $|SIGMA\ i : I.\ (A::'j \Rightarrow 'a\ set)\ (f\ i)| =o\ |SIGMA\ j : J.\ A\ j|$
proof−
 let $?LEFT = SIGMA\ i : I.\ A\ (f\ i)$
 let $?RIGHT = SIGMA\ j : J.\ A\ j$
 obtain *u* **where** *u-def*: $u = (\lambda(i::'i,a::'a).\ (f\ i,a))$ **by** *blast*
 have *bij-betw u ?LEFT ?RIGHT*
 using *assms* **unfolding** *u-def bij-betw-def inj-on-def* **by** *auto*
 thus *?thesis* **using** *card-of-ordIso* **by** *blast*
qed

lemma *card-of-Sigma-cong*:
assumes *BIJ*: *bij-betw f I J* **and**
  ISO: $\forall j \in J.\ |A\ j| =o\ |B\ j|$
shows $|SIGMA\ i : I.\ A\ (f\ i)| =o\ |SIGMA\ j : J.\ B\ j|$
proof−
 have $\forall i \in I.\ |A(f\ i)| =o\ |B(f\ i)|$
 using *ISO BIJ* **unfolding** *bij-betw-def* **by** *blast*
 hence $|SIGMA\ i : I.\ A\ (f\ i)| =o\ |SIGMA\ i : I.\ B\ (f\ i)|$ **by** (*rule card-of-Sigma-cong1*)
 moreover have $|SIGMA\ i : I.\ B\ (f\ i)| =o\ |SIGMA\ j : J.\ B\ j|$
 using *BIJ card-of-Sigma-cong2* **by** *blast*
 ultimately show *?thesis* **using** *ordIso-transitive* **by** *blast*
qed

lemma *ordIso-Sigma-cong1*:
assumes $\forall i \in I.\ p\ i =o\ r\ i$
shows $|SIGMA\ i : I.\ Field(p\ i)| =o\ |SIGMA\ i : I.\ Field(r\ i)|$
using *assms* **by** (*auto simp add*: *card-of-Sigma-cong1*)

lemma *ordLeq-Sigma-cong*:
assumes *bij-betw f I J* **and**
  $\forall j \in J.\ p\ j =o\ r\ j$
shows $|SIGMA\ i : I.\ Field(p(f\ i))| =o\ |SIGMA\ j : J.\ Field(r\ j)|$
using *assms card-of-cong card-of-Sigma-cong*
  $[of\ f\ I\ J\ \lambda\ j.\ Field(p\ j)\ \lambda\ j.\ Field(r\ j)]$ **by** *blast*

lemma *card-of-UNION-Sigma2*:
assumes
$!!\ i\ j.\ [\![\{i,j\} <= I;\ i\ \tilde{}= j]\!] \Longrightarrow A\ i\ Int\ A\ j = \{\}$

shows
$|\bigcup i{\in}I.\ A\ i|$ $=o$ $|Sigma\ I\ A|$
proof−
  let *?L* = $\bigcup i{\in}I.\ A\ i$ **let** *?R* = *Sigma I A*
  have |*?L*| <=*o* |*?R*| **using** *card-of-UNION-Sigma* **.**
  moreover have |*?R*| <=*o* |*?L*|
  proof−
    have *inj-on snd ?R*
    unfolding *inj-on-def* **using** *assms* **by** *auto*
    moreover have *snd ‘ ?R <= ?L* **by** *auto*
    ultimately show *?thesis* **using** *card-of-ordLeq* **by** *blast*
  qed
  ultimately show *?thesis* **by**(*simp add*: *ordIso-iff-ordLeq*)
qed

corollary *Plus-into-Times*:
assumes *A2*: $a1 \neq a2 \wedge \{a1,a2\} \leq A$ **and**
        B2: $b1 \neq b2 \wedge \{b1,b2\} \leq B$
shows ∃*f*. *inj-on f* (*A* <+> *B*) ∧ *f* ‘ (*A* <+> *B*) ≤ *A* × *B*
using *assms* **by** (*auto simp add*: *card-of-Plus-Times card-of-ordLeq*)

corollary *Plus-into-Times-types*:
assumes *A2*: (*a1*::′*a*) ≠ *a2* **and** *B2*: (*b1*::′*b*) ≠ *b2*
shows ∃(*f*::′*a* + ′*b* ⇒ ′*a* ∗ ′*b*). *inj f*
using *assms Plus-into-Times*[*of a1 a2 UNIV b1 b2 UNIV*]
by *auto*

corollary *Times-same-infinite-bij-betw*:
assumes ¬*finite A*
shows ∃*f*. *bij-betw f* (*A* × *A*) *A*
using *assms* **by** (*auto simp add*: *card-of-ordIso*)

corollary *Times-same-infinite-bij-betw-types*:
assumes *INF*: ¬*finite*(*UNIV*::′*a set*)
shows ∃(*f*::(′*a* ∗ ′*a*) => ′*a*). *bij f*
using *assms Times-same-infinite-bij-betw*[*of UNIV*::′*a set*]
by *auto*

corollary *Times-infinite-bij-betw*:
assumes *INF*: ¬*finite A* **and** *NE*: *B* ≠ {} **and** *INJ*: *inj-on g B* ∧ *g* ‘ *B* ≤ *A*
shows (∃*f*. *bij-betw f* (*A* × *B*) *A*) ∧ (∃*h*. *bij-betw h* (*B* × *A*) *A*)
proof−
  have |*B*| ≤*o* |*A*| **using** *INJ card-of-ordLeq* **by** *blast*
  thus *?thesis* **using** *INF NE*
  by (*auto simp add*: *card-of-ordIso card-of-Times-infinite*)
qed

corollary *Times-infinite-bij-betw-types*:
assumes *INF*: ¬*finite*(*UNIV*::′*a set*) **and**

BIJ: $inj(g{::}'b \Rightarrow 'a)$
shows $(\exists (f{::}('b * 'a) => 'a).\ bij\ f) \wedge (\exists (h{::}('a * 'b) => 'a).\ bij\ h)$
using *assms Times-infinite-bij-betw*[*of UNIV*::'*a set UNIV*::'*b set g*]
by *auto*

lemma *card-of-Times-ordLeq-infinite*:
$[\![\neg finite\ C;\ |A| \leq o\ |C|;\ |B| \leq o\ |C|]\!]$
$\Longrightarrow |A \times B| \leq o\ |C|$
by(*simp add*: *card-of-Sigma-ordLeq-infinite*)

corollary *Plus-infinite-bij-betw*:
assumes *INF*: $\neg finite\ A$ **and** *INJ*: *inj-on* $g\ B \wedge g\ `\ B \leq A$
shows $(\exists f.\ bij\text{-}betw\ f\ (A <+> B)\ A) \wedge (\exists h.\ bij\text{-}betw\ h\ (B <+> A)\ A)$
proof−
  have $|B| \leq o\ |A|$ **using** *INJ card-of-ordLeq* **by** *blast*
  thus *?thesis* **using** *INF*
  by (*auto simp add*: *card-of-ordIso*)
qed

corollary *Plus-infinite-bij-betw-types*:
assumes *INF*: $\neg finite(UNIV{::}'a\ set)$ **and**
BIJ: $inj(g{::}'b \Rightarrow 'a)$
shows $(\exists (f{::}('b + 'a) => 'a).\ bij\ f) \wedge (\exists (h{::}('a + 'b) => 'a).\ bij\ h)$
using *assms Plus-infinite-bij-betw*[*of UNIV*::'*a set g UNIV*::'*b set*]
by *auto*

lemma *card-of-Un-infinite*:
assumes *INF*: $\neg finite\ A$ **and** *LEQ*: $|B| \leq o\ |A|$
shows $|A \cup B| =o\ |A| \wedge |B \cup A| =o\ |A|$
proof−
  have $|A \cup B| \leq o\ |A <+> B|$ **by** (*rule card-of-Un-Plus-ordLeq*)
  moreover have $|A <+> B| =o\ |A|$
  using *assms* **by** (*metis card-of-Plus-infinite*)
  ultimately have $|A \cup B| \leq o\ |A|$ **using** *ordLeq-ordIso-trans* **by** *blast*
  hence $|A \cup B| =o\ |A|$ **using** *card-of-Un1 ordIso-iff-ordLeq* **by** *blast*
  thus *?thesis* **using** *Un-commute*[*of B A*] **by** *auto*
qed

lemma *card-of-Un-infinite-simps*[*simp*]:
$[\![\neg finite\ A;\ |B| \leq o\ |A|\ ]\!] \Longrightarrow |A \cup B| =o\ |A|$
$[\![\neg finite\ A;\ |B| \leq o\ |A|\ ]\!] \Longrightarrow |B \cup A| =o\ |A|$
using *card-of-Un-infinite* **by** *auto*

lemma *card-of-Un-diff-infinite*:
assumes *INF*: $\neg finite\ A$ **and** *LESS*: $|B| <o\ |A|$
shows $|A - B| =o\ |A|$
proof−
  obtain C **where** *C-def*: $C = A - B$ **by** *blast*
  have $|A \cup B| =o\ |A|$

using *assms ordLeq-iff-ordLess-or-ordIso card-of-Un-infinite* **by** *blast*
moreover have $C \cup B = A \cup B$ **unfolding** *C-def* **by** *auto*
ultimately have *1*: $|C \cup B| =o |A|$ **by** *auto*

{**assume** $*$: $|C| \leq o |B|$
moreover
{**assume** $**$: *finite B*
hence *finite C*
using *card-of-ordLeq-finite* $*$ **by** *blast*
hence *False* **using** $**$ *INF card-of-ordIso-finite 1* **by** *blast*
}
hence $\neg$*finite B* **by** *auto*
ultimately have *False*
using *card-of-Un-infinite 1 ordIso-equivalence(1,3) LESS not-ordLess-ordIso* **by** *metis*
}
hence *2*: $|B| \leq o |C|$ **using** *card-of-Well-order ordLeq-total* **by** *blast*
{**assume** $*$: *finite C*
hence *finite B* **using** *card-of-ordLeq-finite 2* **by** *blast*
hence *False* **using** $*$ *INF card-of-ordIso-finite 1* **by** *blast*
}
hence $\neg$*finite C* **by** *auto*
hence $|C| =o |A|$
using *card-of-Un-infinite 1 2 ordIso-equivalence(1,3)* **by** *metis*
thus *?thesis* **unfolding** *C-def* **.**
qed

corollary *Card-order-Un-infinite*:
assumes *INF*: $\neg$*finite(Field r)* **and** *CARD*: *Card-order r* **and**
LEQ: $p \leq o\ r$
shows $|$ (*Field r*) $\cup$ (*Field p*) $| =o\ r \wedge |$ (*Field p*) $\cup$ (*Field r*) $| =o\ r$
proof−
have $|$ *Field r* $\cup$ *Field p* $| =o |$ *Field r* $| \wedge$
$|$ *Field p* $\cup$ *Field r* $| =o |$ *Field r* $|$
using *assms* **by** (*auto simp add*: *card-of-Un-infinite*)
thus *?thesis*
using *assms card-of-Field-ordIso*[*of r*]
ordIso-transitive[*of* |*Field r* $\cup$ *Field p*|]
ordIso-transitive[*of* - |*Field r*|] **by** *blast*
qed

corollary *subset-ordLeq-diff-infinite*:
assumes *INF*: $\neg$*finite B* **and** *SUB*: $A \leq B$ **and** *LESS*: $|A| <o |B|$
shows $\neg$*finite* $(B - A)$
using *assms card-of-Un-diff-infinite card-of-ordIso-finite* **by** *blast*

lemma *card-of-Times-ordLess-infinite*[*simp*]:
assumes *INF*: $\neg$*finite C* **and**
LESS1: $|A| <o |C|$ **and** *LESS2*: $|B| <o |C|$

shows $|A \times B| <o |C|$
proof(*cases* $A = \{\} \lor B = \{\}$)
  assume *Case1*: $A = \{\} \lor B = \{\}$
  hence $A \times B = \{\}$ **by** *blast*
  moreover have $C \neq \{\}$ **using**
  LESS1 card-of-empty5 **by** *blast*
  ultimately show *?thesis* **by**(*auto simp add*: *card-of-empty4*)
next
  assume *Case2*: $\neg(A = \{\} \lor B = \{\})$
  {**assume** $*$: $|C| \leq o |A \times B|$
   hence $\neg$*finite* $(A \times B)$ **using** *INF card-of-ordLeq-finite* **by** *blast*
   hence *1*: $\neg$*finite* $A \lor \neg$*finite* B **using** *finite-cartesian-product* **by** *blast*
   {**assume** *Case21*: $|A| \leq o |B|$
    hence $\neg$*finite* B **using** *1 card-of-ordLeq-finite* **by** *blast*
    hence $|A \times B| =o |B|$ **using** *Case2 Case21*
    by (*auto simp add*: *card-of-Times-infinite*)
    hence *False* **using** *LESS2 not-ordLess-ordLeq* $*$ *ordLeq-ordIso-trans* **by** *blast*
   }
   moreover
   {**assume** *Case22*: $|B| \leq o |A|$
    hence $\neg$*finite* A **using** *1 card-of-ordLeq-finite* **by** *blast*
    hence $|A \times B| =o |A|$ **using** *Case2 Case22*
    by (*auto simp add*: *card-of-Times-infinite*)
    hence *False* **using** *LESS1 not-ordLess-ordLeq* $*$ *ordLeq-ordIso-trans* **by** *blast*
   }
   ultimately have *False* **using** *ordLeq-total card-of-Well-order*[*of* A]
   card-of-Well-order[*of* B] **by** *blast*
  }
  thus *?thesis* **using** *ordLess-or-ordLeq*[*of* $|A \times B|$ $|C|$]
  card-of-Well-order[*of* $A \times B$] *card-of-Well-order*[*of* C] **by** *auto*
qed

lemma *card-of-Times-ordLess-infinite-Field*[*simp*]:
assumes *INF*: $\neg$*finite* (*Field* r) **and** r: *Card-order* r **and**
        LESS1: $|A| <o\ r$ **and** *LESS2*: $|B| <o\ r$
shows $|A \times B| <o\ r$
proof$-$
  let *?C* $=$ *Field* r
  have *1*: $r =o |?C| \land |?C| =o\ r$ **using** r *card-of-Field-ordIso*
  ordIso-symmetric **by** *blast*
  hence $|A| <o |?C|$ $|B| <o |?C|$
  using *LESS1 LESS2 ordLess-ordIso-trans* **by** *blast+*
  hence $|A \times B| <o |?C|$ **using** *INF*
  card-of-Times-ordLess-infinite **by** *blast*
  thus *?thesis* **using** *1 ordLess-ordIso-trans* **by** *blast*
qed

lemma *card-of-Un-ordLess-infinite*[*simp*]:
assumes *INF*: $\neg$*finite* C **and**

LESS1: $|A| <o\ |C|$ **and** *LESS2*: $|B| <o\ |C|$
shows $|A \cup B| <o\ |C|$
using *assms card-of-Plus-ordLess-infinite card-of-Un-Plus-ordLeq ordLeq-ordLess-trans* **by** *blast*

lemma *card-of-Un-ordLess-infinite-Field*[*simp*]:
assumes *INF*: $\neg$*finite* (*Field r*) **and** *r*: *Card-order r* **and**
LESS1: $|A| <o\ r$ **and** *LESS2*: $|B| <o\ r$
shows $|A\ Un\ B| <o\ r$
proof−
let *?C* = *Field r*
have *1*: $r =o\ |?C| \wedge |?C| =o\ r$ **using** *r card-of-Field-ordIso ordIso-symmetric* **by** *blast*
hence $|A| <o\ |?C|$ $|B| <o\ |?C|$
using *LESS1 LESS2 ordLess-ordIso-trans* **by** *blast+*
hence $|A\ Un\ B| <o\ |?C|$ **using** *INF card-of-Un-ordLess-infinite* **by** *blast*
thus *?thesis* **using** *1 ordLess-ordIso-trans* **by** *blast*
qed

lemma *ordLeq-finite-Field*:
assumes $r \leq o\ s$ **and** *finite* (*Field s*)
shows *finite* (*Field r*)
by (*metis assms card-of-mono2 card-of-ordLeq-infinite*)

lemma *ordIso-finite-Field*:
assumes $r =o\ s$
shows *finite* (*Field r*) $\longleftrightarrow$ *finite* (*Field s*)
by (*metis assms ordIso-iff-ordLeq ordLeq-finite-Field*)

10.3 Cardinals versus set operations involving infinite sets

lemma *finite-iff-cardOf-nat*:
finite A = ($|A| <o\ |UNIV :: nat\ set|$)
using *infinite-iff-card-of-nat*[*of A*]
not-ordLeq-iff-ordLess[*of* $|A|$ $|UNIV :: nat\ set|$]
by *fastforce*

lemma *finite-ordLess-infinite2*[*simp*]:
assumes *finite A* **and** $\neg$*finite B*
shows $|A| <o\ |B|$
using *assms*
finite-ordLess-infinite[*of* $|A|$ $|B|$]
card-of-Well-order[*of A*] *card-of-Well-order*[*of B*]
Field-card-of[*of A*] *Field-card-of*[*of B*] **by** *auto*

lemma *infinite-card-of-insert*:
assumes $\neg$*finite A*
shows $|insert\ a\ A| =o\ |A|$

proof −
  have *iA*: *insert a A* = *A* ∪ {*a*} **by** *simp*
  show *?thesis*
  using *infinite-imp-bij-betw2*[*OF assms*] **unfolding** *iA*
  by (*metis bij-betw-inv card-of-ordIso*)
qed

lemma *card-of-Un-singl-ordLess-infinite1*:
assumes ¬*finite B* **and** |*A*| <*o* |*B*|
shows |{*a*} *Un A*| <*o* |*B*|
proof −
  have |{*a*}| <*o* |*B*| **using** *assms* **by** *auto*
  thus *?thesis* **using** *assms card-of-Un-ordLess-infinite*[*of B*] **by** *blast*
qed

lemma *card-of-Un-singl-ordLess-infinite*:
assumes ¬*finite B*
shows (|*A*| <*o* |*B*|) = (|{*a*} *Un A*| <*o* |*B*|)
using *assms card-of-Un-singl-ordLess-infinite1*[*of B A*]
proof(*auto*)
  assume |*insert a A*| <*o* |*B*|
  moreover have |*A*| <=*o* |*insert a A*| **using** *card-of-mono1*[*of A insert a A*] **by** *blast*
  ultimately show |*A*| <*o* |*B*| **using** *ordLeq-ordLess-trans* **by** *blast*
qed

10.4 Cardinals versus lists

The next is an auxiliary operator, which shall be used for inductive proofs of facts concerning the cardinality of *List* :

definition *nlists* :: *'a set* ⇒ *nat* ⇒ *'a list set*
where *nlists A n* ≡ {*l*. *set l* ≤ *A* ∧ *length l* = *n*}

lemma *lists-def2*: *lists A* = {*l*. *set l* ≤ *A*}
using *in-listsI* **by** *blast*

lemma *lists-UNION-nlists*: *lists A* = (⋃*n*. *nlists A n*)
unfolding *lists-def2 nlists-def* **by** *blast*

lemma *card-of-lists*: |*A*| ≤*o* |*lists A*|
proof −
  let *?h* = λ *a*. [*a*]
  have *inj-on ?h A* ∧ *?h* ‘ *A* ≤ *lists A*
  unfolding *inj-on-def lists-def2* **by** *auto*
  thus *?thesis* **by** (*metis card-of-ordLeq*)
qed

lemma *nlists-0*: *nlists A 0* = {[]}
unfolding *nlists-def* **by** *auto*

lemma *nlists-not-empty*:
assumes $A \neq \{\}$
shows *nlists* A $n \neq \{\}$
proof(*induct n*, *simp add*: *nlists-0*)
  fix n **assume** *nlists* A $n \neq \{\}$
  then obtain a **and** l **where** $a \in A \wedge l \in$ *nlists* A n **using** *assms* **by** *auto*
  hence a # $l \in$ *nlists* A (*Suc* n) **unfolding** *nlists-def* **by** *auto*
  thus *nlists* A (*Suc* n) $\neq \{\}$ **by** *auto*
qed

lemma *Nil-in-lists*: [] $\in$ *lists* A
unfolding *lists-def2* **by** *auto*

lemma *lists-not-empty*: *lists* $A \neq \{\}$
using *Nil-in-lists* **by** *blast*

lemma *card-of-nlists-Succ*: |*nlists* A (*Suc* n)| =*o* |$A \times$ (*nlists* A n)|
proof−
  let *?B* = $A \times$ (*nlists* A n)   **let** *?h* = $\lambda(a,l)$. a # l
  have *inj-on ?h ?B* $\wedge$ *?h* ‘ *?B* $\leq$ *nlists* A (*Suc* n)
  unfolding *inj-on-def nlists-def* **by** *auto*
  moreover have *nlists* A (*Suc* n) $\leq$ *?h* ‘ *?B*
  proof(*auto*)
    fix l **assume** $l \in$ *nlists* A (*Suc* n)
    hence *1*: *length* l = *Suc* $n \wedge$ *set* $l \leq A$ **unfolding** *nlists-def* **by** *auto*
    then obtain a **and** l' **where** *2*: $l = a$ # l' **by** (*auto simp*: *length-Suc-conv*)
    hence $a \in A \wedge$ *set* $l' \leq A \wedge$ *length* $l' = n$ **using** *1* **by** *auto*
    thus $l \in$ *?h* ‘ *?B* **using** *2* **unfolding** *nlists-def* **by** *auto*
  qed
  ultimately have *bij-betw ?h ?B* (*nlists* A (*Suc* n))
  unfolding *bij-betw-def* **by** *auto*
  thus *?thesis* **using** *card-of-ordIso ordIso-symmetric* **by** *blast*
qed

lemma *card-of-nlists-infinite*:
assumes $\neg$*finite* A
shows |*nlists* A n| $\leq$*o* |A|
proof(*induct n*)
  have $A \neq \{\}$ **using** *assms* **by** *auto*
  thus |*nlists* A *0*| $\leq$*o* |A| **by** (*simp add*: *nlists-0*)
next
  fix n **assume** *IH*: |*nlists* A n| $\leq$*o* |A|
  have |*nlists* A (*Suc* n)| =*o* |$A \times$ (*nlists* A n)|
  using *card-of-nlists-Succ* **by** *blast*
  moreover
  {have *nlists* A $n \neq \{\}$ **using** *assms nlists-not-empty*[*of* A] **by** *blast*
   hence |$A \times$ (*nlists* A n)| =*o* |A|
   using *assms IH* **by** (*auto simp add*: *card-of-Times-infinite*)

}
ultimately show $|nlists\ A\ (Suc\ n)| \leq o\ |A|$
using *ordIso-transitive ordIso-iff-ordLeq* **by** *blast*
qed

lemma *Card-order-lists*: $Card\text{-}order\ r \Longrightarrow r \leq o\ |lists(Field\ r)\ |$
using *card-of-lists card-of-Field-ordIso ordIso-ordLeq-trans ordIso-symmetric* **by** *blast*

lemma *Union-set-lists*: $\bigcup(set\ `\ (lists\ A)) = A$
unfolding *lists-def2*
proof(*auto*)
fix a **assume** $a \in A$
hence $set\ [a] \leq A \wedge a \in set\ [a]$ **by** *auto*
thus $\exists\, l.\ set\ l \leq A \wedge a \in set\ l$ **by** *blast*
qed

lemma *inj-on-map-lists*:
assumes $inj\text{-}on\ f\ A$
shows $inj\text{-}on\ (map\ f)\ (lists\ A)$
using *assms Union-set-lists*[*of A*] *inj-on-mapI*[*of f lists A*] **by** *auto*

lemma *map-lists-mono*:
assumes $f\ `\ A \leq B$
shows $(map\ f)\ `\ (lists\ A) \leq lists\ B$
using *assms* **unfolding** *lists-def2* **by** (*auto*, *blast*)

lemma *map-lists-surjective*:
assumes $f\ `\ A = B$
shows $(map\ f)\ `\ (lists\ A) = lists\ B$
using *assms* **unfolding** *lists-def2*
proof (*auto*, *blast*)
fix l' **assume** $*$: $set\ l' \leq f\ `\ A$
have $set\ l' \leq f\ `\ A \longrightarrow l' \in map\ f\ `\ \{l.\ set\ l \leq A\}$
proof(*induct* l', *auto*)
fix $l\ a$
assume $a \in A$ **and** $set\ l \leq A$ **and**
IH: $f\ `\ (set\ l) \leq f\ `\ A$
hence $set\ (a\ \#\ l) \leq A$ **by** *auto*
hence $map\ f\ (a\ \#\ l) \in map\ f\ `\ \{l.\ set\ l \leq A\}$ **by** *blast*
thus $f\ a\ \#\ map\ f\ l \in map\ f\ `\ \{l.\ set\ l \leq A\}$ **by** *auto*
qed
thus $l' \in map\ f\ `\ \{l.\ set\ l \leq A\}$ **using** $*$ **by** *auto*
qed

lemma *bij-betw-map-lists*:
assumes $bij\text{-}betw\ f\ A\ B$
shows $bij\text{-}betw\ (map\ f)\ (lists\ A)\ (lists\ B)$
using *assms* **unfolding** *bij-betw-def*

by(*auto simp add*: *inj-on-map-lists map-lists-surjective*)

lemma *card-of-lists-mono*[*simp*]:
assumes $|A| \leq o\ |B|$
shows $|lists\ A| \leq o\ |lists\ B|$
proof−
  obtain *f* **where** $inj\text{-}on\ f\ A \wedge f\ `\ A \leq B$
  using *assms card-of-ordLeq*[*of A B*] **by** *auto*
  hence $inj\text{-}on\ (map\ f)\ (lists\ A) \wedge (map\ f)\ `\ (lists\ A) \leq (lists\ B)$
  by (*auto simp add*: *inj-on-map-lists map-lists-mono*)
  thus *?thesis* **using** *card-of-ordLeq*[*of lists A*] **by** *metis*
qed

lemma *ordIso-lists-mono*:
assumes $r \leq o\ r'$
shows $|lists(Field\ r)| \leq o\ |lists(Field\ r')|$
using *assms card-of-mono2 card-of-lists-mono* **by** *blast*

lemma *card-of-lists-cong*[*simp*]:
assumes $|A| =o\ |B|$
shows $|lists\ A| =o\ |lists\ B|$
proof−
  obtain *f* **where** *bij-betw f A B*
  using *assms card-of-ordIso*[*of A B*] **by** *auto*
  hence *bij-betw* (*map f*) (*lists A*) (*lists B*)
  by (*auto simp add*: *bij-betw-map-lists*)
  thus *?thesis* **using** *card-of-ordIso*[*of lists A*] **by** *auto*
qed

lemma *card-of-lists-infinite*[*simp*]:
assumes ¬*finite A*
shows $|lists\ A| =o\ |A|$
proof−
  have $|lists\ A| \leq o\ |A|$ **unfolding** *lists-UNION-nlists*
  by (*rule card-of-UNION-ordLeq-infinite*[*OF assms - ballI*[*OF card-of-nlists-infinite*[*OF assms*]]])
    (*metis infinite-iff-card-of-nat assms*)
  thus *?thesis* **using** *card-of-lists ordIso-iff-ordLeq* **by** *blast*
qed

lemma *Card-order-lists-infinite*:
assumes *Card-order r* **and** ¬*finite*(*Field r*)
shows $|lists(Field\ r)| =o\ r$
using *assms card-of-lists-infinite card-of-Field-ordIso ordIso-transitive* **by** *blast*

lemma *ordIso-lists-cong*:
assumes $r =o\ r'$
shows $|lists(Field\ r)| =o\ |lists(Field\ r')|$
using *assms card-of-cong card-of-lists-cong* **by** *blast*

corollary *lists-infinite-bij-betw*:
assumes $\neg$*finite A*
shows $\exists f.$ *bij-betw f* (*lists A*) *A*
using *assms card-of-lists-infinite card-of-ordIso* **by** *blast*

corollary *lists-infinite-bij-betw-types*:
assumes $\neg$*finite*(*UNIV* :: *'a set*)
shows $\exists (f{::}'a\ list \Rightarrow 'a).\ bij\ f$
using *assms assms lists-infinite-bij-betw*[*of UNIV*::*'a set*]
using *lists-UNIV* **by** *auto*

10.5 Cardinals versus the finite powerset operator

lemma *card-of-Fpow*[*simp*]: $|A| \leq o\ |Fpow\ A|$
proof−
  let $?h = \lambda\ a.\ \{a\}$
  have *inj-on ?h A* $\wedge$ *?h ‘ A* $\leq$ *Fpow A*
  unfolding *inj-on-def Fpow-def* **by** *auto*
  thus *?thesis* **using** *card-of-ordLeq* **by** *metis*
qed

lemma *Card-order-Fpow*: *Card-order r* $\Longrightarrow$ $r \leq o\ |Fpow(Field\ r)\ |$
using *card-of-Fpow card-of-Field-ordIso ordIso-ordLeq-trans ordIso-symmetric* **by** *blast*

lemma *image-Fpow-surjective*:
assumes *f ‘ A = B*
shows (*image f*) ‘ (*Fpow A*) = *Fpow B*
using *assms* **proof**(*unfold Fpow-def*, *auto*)
  fix *Y* **assume** $*$: $Y \leq f$ ‘ *A* **and** $**$: *finite Y*
  hence $\forall\ b \in Y.\ \exists\ a.\ a \in A \wedge f\ a = b$ **by** *auto*
  with *bchoice*[*of Y* $\lambda b\ a.\ a \in A \wedge f\ a = b$]
  obtain *g* **where** *1*: $\forall\ b \in Y.\ g\ b \in A \wedge f(g\ b) = b$ **by** *blast*
  obtain *X* **where** *X-def*: *X = g ‘ Y* **by** *blast*
  have *f ‘ X = Y* $\wedge$ $X \leq A$ $\wedge$ *finite X*
  by(*unfold X-def*, *force simp add*: $**$ *1*)
  thus $Y \in$ (*image f*) ‘ $\{X.\ X \leq A \wedge finite\ X\}$ **by** *auto*
qed

lemma *bij-betw-image-Fpow*:
assumes *bij-betw f A B*
shows *bij-betw* (*image f*) (*Fpow A*) (*Fpow B*)
using *assms* **unfolding** *bij-betw-def*
by (*auto simp add*: *inj-on-image-Fpow image-Fpow-surjective*)

lemma *card-of-Fpow-mono*[*simp*]:
assumes $|A| \leq o\ |B|$
shows $|Fpow\ A| \leq o\ |Fpow\ B|$

proof −
  obtain *f* **where** *inj-on f A* $\wedge$ *f* ‘ *A* $\leq$ *B*
  using *assms card-of-ordLeq*[*of A B*] **by** *auto*
  hence *inj-on* (*image f*) (*Fpow A*) $\wedge$ (*image f*) ‘ (*Fpow A*) $\leq$ (*Fpow B*)
  by (*auto simp add*: *inj-on-image-Fpow image-Fpow-mono*)
  thus *?thesis* **using** *card-of-ordLeq*[*of Fpow A*] **by** *auto*
qed

lemma *ordIso-Fpow-mono*:
assumes *r* $\leq$*o* *r'*
shows |*Fpow*(*Field r*)| $\leq$*o* |*Fpow*(*Field r'*)|
using *assms card-of-mono2 card-of-Fpow-mono* **by** *blast*

lemma *card-of-Fpow-cong*[*simp*]:
assumes |*A*| =*o* |*B*|
shows |*Fpow A*| =*o* |*Fpow B*|
proof −
  obtain *f* **where** *bij-betw f A B*
  using *assms card-of-ordIso*[*of A B*] **by** *auto*
  hence *bij-betw* (*image f*) (*Fpow A*) (*Fpow B*)
  by (*auto simp add*: *bij-betw-image-Fpow*)
  thus *?thesis* **using** *card-of-ordIso*[*of Fpow A*] **by** *auto*
qed

lemma *ordIso-Fpow-cong*:
assumes *r* =*o* *r'*
shows |*Fpow*(*Field r*)| =*o* |*Fpow*(*Field r'*)|
using *assms card-of-cong card-of-Fpow-cong* **by** *blast*

lemma *card-of-Fpow-lists*: |*Fpow A*| $\leq$*o* |*lists A*|
proof −
  have *set* ‘ (*lists A*) = *Fpow A*
  unfolding *lists-def2 Fpow-def* **using** *finite-list finite-set* **by** *blast*
  thus *?thesis* **using** *card-of-ordLeq2*[*of Fpow A*] *Fpow-not-empty*[*of A*] **by** *blast*
qed

lemma *card-of-Fpow-infinite*[*simp*]:
assumes $\neg$*finite A*
shows |*Fpow A*| =*o* |*A*|
using *assms card-of-Fpow-lists card-of-lists-infinite card-of-Fpow*
    ordLeq-ordIso-trans ordIso-iff-ordLeq **by** *blast*

corollary *Fpow-infinite-bij-betw*:
assumes $\neg$*finite A*
shows $\exists$*f*. *bij-betw f* (*Fpow A*) *A*
using *assms card-of-Fpow-infinite card-of-ordIso* **by** *blast*

10.6 The cardinal ω and the finite cardinals

10.6.1 First as well-orders

lemma *Field-natLess*: *Field natLess* = (*UNIV*::*nat set*)
by(*unfold Field-def natLess-def*, *auto*)

lemma *natLeq-well-order-on*: *well-order-on UNIV natLeq*
using *natLeq-Well-order Field-natLeq* **by** *auto*

lemma *natLeq-wo-rel*: *wo-rel natLeq*
unfolding *wo-rel-def* **using** *natLeq-Well-order* **.**

lemma *natLeq-ofilter-less*: *ofilter natLeq* $\{0 \,..{<}\, n\}$
by(*auto simp add*: *natLeq-wo-rel wo-rel.ofilter-def*,
simp add: *Field-natLeq*, *unfold under-def natLeq-def*, *auto*)

lemma *natLeq-ofilter-leq*: *ofilter natLeq* $\{0 \,..\, n\}$
by(*auto simp add*: *natLeq-wo-rel wo-rel.ofilter-def*,
simp add: *Field-natLeq*, *unfold under-def natLeq-def*, *auto*)

lemma *natLeq-UNIV-ofilter*: *wo-rel.ofilter natLeq UNIV*
using *natLeq-wo-rel Field-natLeq wo-rel.Field-ofilter*[*of natLeq*] **by** *auto*

lemma *closed-nat-set-iff*:
assumes $\forall (m::nat)\ n.\ n \in A \wedge m \leq n \longrightarrow m \in A$
shows $A = UNIV \vee (\exists n.\ A = \{0 \,..{<}\, n\})$
proof−
{**assume** $A \neq UNIV$ **hence** $\exists n.\ n \notin A$ **by** *blast*
moreover obtain n **where** *n-def*: $n = (LEAST\ n.\ n \notin A)$ **by** *blast*
ultimately have *1*: $n \notin A \wedge (\forall m.\ m < n \longrightarrow m \in A)$
using *LeastI-ex*[*of* $\lambda\ n.\ n \notin A$] *n-def Least-le*[*of* $\lambda\ n.\ n \notin A$] **by** *fastforce*
have $A = \{0 \,..{<}\, n\}$
proof(*auto simp add*: *1*)
fix m **assume** $*$: $m \in A$
{**assume** $n \leq m$ **with** *assms* $*$ **have** $n \in A$ **by** *blast*
hence *False* **using** *1* **by** *auto*
}
thus $m < n$ **by** *fastforce*
qed
hence $\exists n.\ A = \{0 \,..{<}\, n\}$ **by** *blast*
}
thus *?thesis* **by** *blast*
qed

lemma *natLeq-ofilter-iff*:
ofilter natLeq $A = (A = UNIV \vee (\exists n.\ A = \{0 \,..{<}\, n\}))$
proof(*rule iffI*)
assume *ofilter natLeq A*
hence $\forall m\ n.\ n \in A \wedge m \leq n \longrightarrow m \in A$ **using** *natLeq-wo-rel*

```
  by(auto simp add: natLeq-def wo-rel.ofilter-def under-def)
  thus A = UNIV ∨ (∃ n. A = {0 ..< n}) using closed-nat-set-iff by blast
next
  assume A = UNIV ∨ (∃ n. A = {0 ..< n})
  thus ofilter natLeq A
  by(auto simp add: natLeq-ofilter-less natLeq-UNIV-ofilter)
qed

lemma natLeq-under-leq: under natLeq n = {0 .. n}
unfolding under-def natLeq-def by auto

lemma natLeq-on-ofilter-less-eq:
n ≤ m ⟹ wo-rel.ofilter (natLeq-on m) {0 ..< n}
apply (auto simp add: natLeq-on-wo-rel wo-rel.ofilter-def)
apply (simp add: Field-natLeq-on)
by (auto simp add: under-def)

lemma natLeq-on-ofilter-iff:
wo-rel.ofilter (natLeq-on m) A = (∃ n ≤ m. A = {0 ..< n})
proof(rule iffI)
  assume *: wo-rel.ofilter (natLeq-on m) A
  hence 1: A ≤ {0..<m}
  by (auto simp add: natLeq-on-wo-rel wo-rel.ofilter-def under-def Field-natLeq-on)
  hence ∀ n1 n2. n2 ∈ A ∧ n1 ≤ n2 ⟶ n1 ∈ A
  using * by(fastforce simp add: natLeq-on-wo-rel wo-rel.ofilter-def under-def)
  hence A = UNIV ∨ (∃ n. A = {0 ..< n}) using closed-nat-set-iff by blast
  thus ∃ n ≤ m. A = {0 ..< n} using 1 atLeastLessThan-less-eq by blast
next
  assume (∃ n≤m. A = {0 ..< n})
  thus wo-rel.ofilter (natLeq-on m) A by (auto simp add: natLeq-on-ofilter-less-eq)
qed

corollary natLeq-on-ofilter:
ofilter(natLeq-on n) {0 ..< n}
by (auto simp add: natLeq-on-ofilter-less-eq)

lemma natLeq-on-ofilter-less:
n < m ⟹ ofilter (natLeq-on m) {0 .. n}
by(auto simp add: natLeq-on-wo-rel wo-rel.ofilter-def,
   simp add: Field-natLeq-on, unfold under-def, auto)

lemma natLeq-on-ordLess-natLeq: natLeq-on n <o natLeq
using Field-natLeq Field-natLeq-on[of n]
      finite-ordLess-infinite[of natLeq-on n natLeq]
      natLeq-Well-order natLeq-on-Well-order[of n] by auto

lemma natLeq-on-injective:
natLeq-on m = natLeq-on n ⟹ m = n
using Field-natLeq-on[of m] Field-natLeq-on[of n]
```

atLeastLessThan-injective[*of m n*, *unfolded atLeastLessThan-def*] **by** *blast*

lemma *natLeq-on-injective-ordIso*:
(*natLeq-on m* =*o natLeq-on n*) = (*m* = *n*)
proof(*auto simp add*: *natLeq-on-Well-order ordIso-reflexive*)
assume *natLeq-on m* =*o natLeq-on n*
then obtain *f* **where** *bij-betw f* {*x*. *x*<*m*} {*x*. *x*<*n*}
using *Field-natLeq-on* **unfolding** *ordIso-def iso-def*[*abs-def*] **by** *auto*
thus *m* = *n* **using** *atLeastLessThan-injective2*[*of f m n*]
unfolding *atLeast-0 atLeastLessThan-def lessThan-def Int-UNIV-left* **by** *blast*
qed

10.6.2 Then as cardinals

lemma *ordIso-natLeq-infinite1*:
|*A*| =*o natLeq* $\Longrightarrow$ ¬*finite A*
using *ordIso-symmetric ordIso-imp-ordLeq infinite-iff-natLeq-ordLeq* **by** *blast*

lemma *ordIso-natLeq-infinite2*:
natLeq =*o* |*A*| $\Longrightarrow$ ¬*finite A*
using *ordIso-imp-ordLeq infinite-iff-natLeq-ordLeq* **by** *blast*

lemma *ordIso-natLeq-on-imp-finite*:
|*A*| =*o natLeq-on n* $\Longrightarrow$ *finite A*
unfolding *ordIso-def iso-def*[*abs-def*]
by (*auto simp*: *Field-natLeq-on bij-betw-finite*)

lemma *natLeq-on-Card-order*: *Card-order* (*natLeq-on n*)
proof(*unfold card-order-on-def*,
auto simp add: *natLeq-on-Well-order*, *simp add*: *Field-natLeq-on*)
fix *r* **assume** *well-order-on* {*x*. *x* < *n*} *r*
thus *natLeq-on n* ≤*o r*
using *finite-atLeastLessThan natLeq-on-well-order-on*
finite-well-order-on-ordIso ordIso-iff-ordLeq **by** *blast*
qed

corollary *card-of-Field-natLeq-on*:
|*Field* (*natLeq-on n*)| =*o natLeq-on n*
using *Field-natLeq-on natLeq-on-Card-order*
Card-order-iff-ordIso-card-of[*of natLeq-on n*]
ordIso-symmetric[*of natLeq-on n*] **by** *blast*

corollary *card-of-less*:
|{*0* ..< *n*}| =*o natLeq-on n*
using *Field-natLeq-on card-of-Field-natLeq-on*
unfolding *atLeast-0 atLeastLessThan-def lessThan-def Int-UNIV-left* **by** *auto*

lemma *natLeq-on-ordLeq-less-eq*:
((*natLeq-on m*) ≤*o* (*natLeq-on n*)) = (*m* ≤ *n*)

proof
assume *natLeq-on m* $\leq o$ *natLeq-on n*
then obtain *f* **where** *inj-on f* $\{x.\ x < m\} \wedge f$ ' $\{x.\ x < m\} \leq \{x.\ x < n\}$
unfolding *ordLeq-def* **using**
embed-inj-on[*OF natLeq-on-Well-order*[*of m*], *of natLeq-on n*, *unfolded Field-natLeq-on*]
embed-Field Field-natLeq-on **by** *blast*
thus $m \leq n$ **using** *atLeastLessThan-less-eq2*
unfolding *atLeast-0 atLeastLessThan-def lessThan-def Int-UNIV-left* **by** *blast*
next
assume $m \leq n$
hence *inj-on id* $\{0..<m\} \wedge id$ ' $\{0..<m\} \leq \{0..<n\}$ **unfolding** *inj-on-def* **by** *auto*
hence $|\{0..<m\}| \leq o\ |\{0..<n\}|$ **using** *card-of-ordLeq* **by** *blast*
thus *natLeq-on m* $\leq o$ *natLeq-on n*
using *card-of-less ordIso-ordLeq-trans ordLeq-ordIso-trans ordIso-symmetric* **by** *blast*
qed

lemma *natLeq-on-ordLeq-less*:
((*natLeq-on m*) $<o$ (*natLeq-on n*)) = ($m < n$)
using *not-ordLeq-iff-ordLess*[*OF natLeq-on-Well-order natLeq-on-Well-order*, *of n m*]
natLeq-on-ordLeq-less-eq[*of n m*] **by** *linarith*

lemma *ordLeq-natLeq-on-imp-finite*:
assumes $|A| \leq o$ *natLeq-on n*
shows *finite A*
proof−
have $|A| \leq o\ |\{0\ ..<\ n\}|$
using *assms card-of-less ordIso-symmetric ordLeq-ordIso-trans* **by** *blast*
thus *?thesis* **by** (*auto simp add*: *card-of-ordLeq-finite*)
qed

10.6.3 "Backward compatibility" with the numeric cardinal operator for finite sets

lemma *finite-card-of-iff-card2*:
assumes *FIN*: *finite A* **and** *FIN'*: *finite B*
shows ($|A| \leq o\ |B|$) = (*card A* $\leq$ *card B*)
using *assms card-of-ordLeq*[*of A B*] *inj-on-iff-card-le*[*of A B*] **by** *blast*

lemma *finite-imp-card-of-natLeq-on*:
assumes *finite A*
shows $|A| =o$ *natLeq-on* (*card A*)
proof−
obtain *h* **where** *bij-betw h A* $\{0\ ..<$ *card A*$\}$
using *assms ex-bij-betw-finite-nat* **by** *blast*
thus *?thesis* **using** *card-of-ordIso card-of-less ordIso-equivalence* **by** *blast*
qed

lemma *finite-iff-card-of-natLeq-on*:
finite $A = (\exists\, n.\ |A| =o\ natLeq\text{-}on\ n)$
using *finite-imp-card-of-natLeq-on*[*of A*]
by(*auto simp add*: *ordIso-natLeq-on-imp-finite*)

lemma *finite-card-of-iff-card*:
assumes *FIN*: *finite A* **and** *FIN'*: *finite B*
shows $(\ |A| =o\ |B|\ ) = (card\ A = card\ B)$
using *assms card-of-ordIso*[*of A B*] *bij-betw-iff-card*[*of A B*] **by** *blast*

lemma *finite-card-of-iff-card3*:
assumes *FIN*: *finite A* **and** *FIN'*: *finite B*
shows $(\ |A| <o\ |B|\ ) = (card\ A < card\ B)$
proof−
  have $(\ |A| <o\ |B|\ ) = (\sim\ (\ |B| \leq o\ |A|\ ))$ **by** *simp*
  also have ... $= (\sim\ (card\ B \leq card\ A))$
  using *assms* **by**(*simp add*: *finite-card-of-iff-card2*)
  also have ... $= (card\ A < card\ B)$ **by** *auto*
  finally show *?thesis* **.**
qed

lemma *card-Field-natLeq-on*:
$card(Field(natLeq\text{-}on\ n)) = n$
using *Field-natLeq-on card-atLeastLessThan* **by** *auto*

10.7 The successor of a cardinal

lemma *embed-implies-ordIso-Restr*:
assumes *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and** *EMB*: *embed r' r f*
shows $r' =o\ Restr\ r\ (f\ `\ (Field\ r'))$
using *assms embed-implies-iso-Restr Well-order-Restr* **unfolding** *ordIso-def* **by** *blast*

lemma *cardSuc-Well-order*[*simp*]:
Card-order r $\Longrightarrow$ *Well-order*(*cardSuc r*)
using *cardSuc-Card-order* **unfolding** *card-order-on-def* **by** *blast*

lemma *Field-cardSuc-not-empty*:
assumes *Card-order r*
shows $Field\ (cardSuc\ r) \neq \{\}$
proof
  assume $Field(cardSuc\ r) = \{\}$
  hence $|Field(cardSuc\ r)| \leq o\ r$ **using** *assms Card-order-empty*[*of r*] **by** *auto*
  hence $cardSuc\ r \leq o\ r$ **using** *assms card-of-Field-ordIso*
  cardSuc-Card-order ordIso-symmetric ordIso-ordLeq-trans **by** *blast*
  thus *False* **using** *cardSuc-greater not-ordLess-ordLeq assms* **by** *blast*
qed

lemma *cardSuc-mono-ordLess*[*simp*]:
assumes *CARD*: *Card-order r* **and** *CARD'*: *Card-order r'*
shows (*cardSuc r* $<o$ *cardSuc r'*) = ($r <o r'$)
proof −
have *0*: *Well-order r* $\wedge$ *Well-order r'* $\wedge$ *Well-order*(*cardSuc r*) $\wedge$ *Well-order*(*cardSuc r'*)
using *assms* **by** *auto*
thus *?thesis*
using *not-ordLeq-iff-ordLess not-ordLeq-iff-ordLess*[*of r r'*]
using *cardSuc-mono-ordLeq*[*of r' r*] *assms* **by** *blast*
qed

lemma *cardSuc-natLeq-on-Suc*:
cardSuc(*natLeq-on n*) =*o* *natLeq-on*(*Suc n*)
proof −
obtain *r r' p* **where** *r-def*: *r* = *natLeq-on n* **and**
r'-def: *r'* = *cardSuc*(*natLeq-on n*) **and**
p-def: *p* = *natLeq-on*(*Suc n*) **by** *blast*

have *CARD*: *Card-order r* $\wedge$ *Card-order r'* $\wedge$ *Card-order p* **unfolding** *r-def r'-def p-def*
using *cardSuc-ordLess-ordLeq natLeq-on-Card-order cardSuc-Card-order* **by** *blast*
hence *WELL*: *Well-order r* $\wedge$ *Well-order r'* $\wedge$ *Well-order p*
unfolding *card-order-on-def* **by** *force*
have *FIELD*: *Field r* = $\{0..<n\}$ $\wedge$ *Field p* = $\{0..<(Suc\ n)\}$
unfolding *r-def p-def Field-natLeq-on atLeast-0 atLeastLessThan-def lessThan-def*
by *simp*
hence *FIN*: *finite* (*Field r*) **by** *force*
have $r <o r'$ **using** *CARD* **unfolding** *r-def r'-def* **using** *cardSuc-greater* **by** *blast*
hence |*Field r*| $<o$ *r'* **using** *CARD card-of-Field-ordIso ordIso-ordLess-trans* **by** *blast*
hence *LESS*: |*Field r*| $<o$ |*Field r'*|
using *CARD card-of-Field-ordIso ordLess-ordIso-trans ordIso-symmetric* **by** *blast*

have $r' \leq o\ p$ **using** *CARD* **unfolding** *r-def r'-def p-def*
using *natLeq-on-ordLeq-less cardSuc-ordLess-ordLeq* **by** *blast*
moreover have $p \leq o\ r'$
proof −
{**assume** $r' <o p$
then obtain *f* **where** *0*: *embedS r' p f* **unfolding** *ordLess-def* **by** *force*
let *?q* = *Restr p* (*f* ‘ *Field r'*)
have *1*: *embed r' p f* **using** *0* **unfolding** *embedS-def* **by** *force*
hence *2*: *f* ‘ *Field r'* < $\{0..<(Suc\ n)\}$
using *WELL FIELD 0* **by** (*auto simp add*: *embedS-iff*)
have *wo-rel.ofilter p* (*f* ‘ *Field r'*) **using** *embed-Field-ofilter 1 WELL* **by** *blast*
then obtain *m* **where** $m \leq Suc\ n$ **and** *3*: *f* ‘ (*Field r'*) = $\{0..<m\}$
unfolding *p-def* **by** (*auto simp add*: *natLeq-on-ofilter-iff*)
hence *4*: $m \leq n$ **using** *2* **by** *force*

have *bij-betw f* (*Field r′*) (*f* ‘ (*Field r′*))
using *WELL embed-inj-on*[*OF* - *1*] **unfolding** *bij-betw-def* **by** *blast*
moreover have *finite*(*f* ‘ (*Field r′*)) **using** *3 finite-atLeastLessThan*[*of 0 m*] **by** *force*
ultimately have *5*: *finite* (*Field r′*) ∧ *card*(*Field r′*) = *card* (*f* ‘ (*Field r′*))
using *bij-betw-same-card bij-betw-finite* **by** *metis*
hence *card*(*Field r′*) ≤ *card*(*Field r*) **using** *3 4 FIELD* **by** *force*
hence |*Field r′*| ≤*o* |*Field r*| **using** *FIN 5 finite-card-of-iff-card2* **by** *blast*
hence *False* **using** *LESS not-ordLess-ordLeq* **by** *auto*
}
thus *?thesis* **using** *WELL CARD* **by** *fastforce*
qed
ultimately show *?thesis* **using** *ordIso-iff-ordLeq* **unfolding** *r′-def p-def* **by** *blast*
qed

lemma *card-of-Plus-ordLeq-infinite*[*simp*]:
assumes *C*: ¬*finite C* **and** *A*: |*A*| ≤*o* |*C*| **and** *B*: |*B*| ≤*o* |*C*|
shows |*A* <+> *B*| ≤*o* |*C*|
proof−
let *?r* = *cardSuc* |*C*|
have *Card-order ?r* ∧ ¬*finite* (*Field ?r*) **using** *assms* **by** *simp*
moreover have |*A*| <*o ?r* **and** |*B*| <*o ?r* **using** *A B* **by** *auto*
ultimately have |*A* <+> *B*| <*o ?r*
using *card-of-Plus-ordLess-infinite-Field* **by** *blast*
thus *?thesis* **using** *C* **by** *simp*
qed

lemma *card-of-Un-ordLeq-infinite*[*simp*]:
assumes *C*: ¬*finite C* **and** *A*: |*A*| ≤*o* |*C*| **and** *B*: |*B*| ≤*o* |*C*|
shows |*A Un B*| ≤*o* |*C*|
using *assms card-of-Plus-ordLeq-infinite card-of-Un-Plus-ordLeq ordLeq-transitive* **by** *metis*

10.8 Others

lemma *under-mono*[*simp*]:
assumes *Well-order r* **and** (*i*,*j*) ∈ *r*
shows *under r i* ⊆ *under r j*
using *assms* **unfolding** *under-def order-on-defs trans-def* **by** *blast*

lemma *underS-under*:
assumes *i* ∈ *Field r*
shows *underS r i* = *under r i* − {*i*}
using *assms* **unfolding** *underS-def under-def* **by** *auto*

lemma *relChain-under*:

assumes *Well-order r*
shows *relChain r* (λ *i. under r i*)
using *assms* **unfolding** *relChain-def* **by** *auto*

lemma *card-of-infinite-diff-finite*:
assumes $\neg$*finite A* **and** *finite B*
shows $|A - B| =o |A|$
by (*metis assms card-of-Un-diff-infinite finite-ordLess-infinite2*)

lemma *infinite-card-of-diff-singl*:
assumes $\neg$*finite A*
shows $|A - \{a\}| =o |A|$
by (*metis assms card-of-infinite-diff-finite finite.emptyI finite-insert*)

lemma *card-of-vimage*:
assumes $B \subseteq$ *range f*
shows $|B| \leq o\ |f -` B|$
apply(*rule surj-imp-ordLeq*[*of - f*])
using *assms* **by** (*metis Int-absorb2 image-vimage-eq order-refl*)

lemma *surj-card-of-vimage*:
assumes *surj f*
shows $|B| \leq o\ |f -` B|$
by (*metis assms card-of-vimage subset-UNIV*)

lemma *infinite-Pow*:
assumes $\neg$ *finite A*
shows $\neg$ *finite* (*Pow A*)
proof−
  have $|A| \leq o\ |Pow\ A|$ **by** (*metis card-of-Pow ordLess-imp-ordLeq*)
  thus *?thesis* **by** (*metis assms finite-Pow-iff*)
qed

definition *Bpow* **where**
$Bpow\ r\ A \equiv \{X\ .\ X \subseteq A \wedge |X| \leq o\ r\}$

lemma *Bpow-empty*[*simp*]:
assumes *Card-order r*
shows $Bpow\ r\ \{\} = \{\{\}\}$
using *assms* **unfolding** *Bpow-def* **by** *auto*

lemma *singl-in-Bpow*:
assumes *rc*: *Card-order r*
and *r*: $Field\ r \neq \{\}$ **and** *a*: $a \in A$
shows $\{a\} \in Bpow\ r\ A$
proof−
  have $|\{a\}| \leq o\ r$ **using** *r rc* **by** *auto*
  thus *?thesis* **unfolding** *Bpow-def* **using** *a* **by** *auto*

qed

lemma *ordLeq-card-Bpow*:
assumes *rc*: *Card-order r* **and** *r*: *Field r* $\neq$ {}
shows $|A| \leq o$ $|Bpow\ r\ A|$
proof−
  have *inj-on* (λ *a*. {*a*}) *A* **unfolding** *inj-on-def* **by** *auto*
  moreover have (λ *a*. {*a*}) ‘ *A* $\subseteq$ *Bpow r A*
  using *singl-in-Bpow*[*OF assms*] **by** *auto*
  ultimately show *?thesis* **unfolding** *card-of-ordLeq*[*symmetric*] **by** *blast*
qed

lemma *infinite-Bpow*:
assumes *rc*: *Card-order r* **and** *r*: *Field r* $\neq$ {}
and *A*: ¬*finite A*
shows ¬*finite* (*Bpow r A*)
using *ordLeq-card-Bpow*[*OF rc r*]
by (*metis A card-of-ordLeq-infinite*)

definition *Func-option* **where**
 Func-option A B $\equiv$
  {*f*. ($\forall$ *a*. *f a* $\neq$ *None* $\longleftrightarrow$ *a* $\in$ *A*) $\wedge$ ($\forall$ *a* $\in$ *A*. *case f a of Some b* $\Rightarrow$ *b* $\in$ *B* |*None* $\Rightarrow$ *True*)}

lemma *card-of-Func-option-Func*:
|*Func-option A B*| =*o* |*Func A B*|
proof (*rule ordIso-symmetric*, *unfold card-of-ordIso*[*symmetric*], *intro exI*)
  let *?F* = λ *f a*. *if a* $\in$ *A then Some* (*f a*) *else None*
  show *bij-betw ?F* (*Func A B*) (*Func-option A B*)
  unfolding *bij-betw-def* **unfolding** *inj-on-def* **proof**(*intro conjI ballI impI*)
    fix *f g* **assume** *f*: *f* $\in$ *Func A B* **and** *g*: *g* $\in$ *Func A B* **and** *eq*: *?F f* = *?F g*
    show *f* = *g*
    proof(*rule ext*)
      fix *a*
      show *f a* = *g a*
      proof(*cases a* $\in$ *A*)
        case *True*
        have *Some* (*f a*) = *?F f a* **using** *True* **by** *auto*
        also have ... = *?F g a* **using** *eq* **unfolding** *fun-eq-iff* **by**(*rule allE*)
        also have ... = *Some* (*g a*) **using** *True* **by** *auto*
        finally have *Some* (*f a*) = *Some* (*g a*) **.**
        thus *?thesis* **by** *simp*
      qed(*insert f g*, *unfold Func-def*, *auto*)
    qed
  next
    show *?F* ‘ *Func A B* = *Func-option A B*
    proof *safe*
      fix *f* **assume** *f*: *f* $\in$ *Func-option A B*
     define *g* **where** [*abs-def*]: *g a* = (*case f a of Some b* $\Rightarrow$ *b* | *None* $\Rightarrow$ *undefined*)

```
for a
      have g ∈ Func A B
      using f unfolding g-def Func-def Func-option-def by force+
      moreover have f = ?F g
      proof(rule ext)
        fix a show f a = ?F g a
        using f unfolding Func-option-def g-def by (cases a ∈ A) force+
      qed
      ultimately show f ∈ ?F ‘ (Func A B) by blast
    qed(unfold Func-def Func-option-def, auto)
  qed
qed
```

```
definition Pfunc where
Pfunc A B ≡
 {f. (∀ a. f a ≠ None ⟶ a ∈ A) ∧
     (∀ a. case f a of None ⇒ True | Some b ⇒ b ∈ B)}

lemma Func-Pfunc:
Func-option A B ⊆ Pfunc A B
unfolding Func-option-def Pfunc-def by auto

lemma Pfunc-Func-option:
Pfunc A B = (⋃ A' ∈ Pow A. Func-option A' B)
proof safe
  fix f assume f: f ∈ Pfunc A B
  show f ∈ (⋃ A'∈Pow A. Func-option A' B)
  proof (intro UN-I)
    let ?A' = {a. f a ≠ None}
    show ?A' ∈ Pow A using f unfolding Pow-def Pfunc-def by auto
     show f ∈ Func-option ?A' B using f unfolding Func-option-def Pfunc-def
by auto
  qed
next
  fix f A' assume f ∈ Func-option A' B and A' ⊆ A
  thus f ∈ Pfunc A B unfolding Func-option-def Pfunc-def by auto
qed

lemma card-of-Func-mono:
fixes A1 A2 :: 'a set and B :: 'b set
assumes A12: A1 ⊆ A2 and B: B ≠ {}
shows |Func A1 B| ≤o |Func A2 B|
proof−
  obtain bb where bb: bb ∈ B using B by auto
  define F where [abs-def]: F f1 a =
    (if a ∈ A2 then (if a ∈ A1 then f1 a else bb) else undefined) for f1 :: 'a ⇒ 'b
and a
 show ?thesis unfolding card-of-ordLeq[symmetric] proof(intro exI[of - F] conjI)
```

show *inj-on F* (*Func A1 B*) **unfolding** *inj-on-def* **proof** *safe*
fix *f g* **assume** *f*: $f \in$ *Func A1 B* **and** *g*: $g \in$ *Func A1 B* **and** *eq*: $F\ f = F\ g$
show $f = g$
proof(*rule ext*)
fix *a* **show** $f\ a = g\ a$
proof(*cases* $a \in A1$)
case *True*
thus *?thesis* **using** *eq A12* **unfolding** *F-def fun-eq-iff*
by (*elim allE*[*of - a*]) *auto*
qed(*insert f g*, *unfold Func-def*, *fastforce*)
qed
qed
qed(*insert bb*, *unfold Func-def F-def*, *force*)
qed

lemma *card-of-Func-option-mono*:
fixes *A1 A2* :: *'a set* **and** *B* :: *'b set*
assumes *A12*: $A1 \subseteq A2$ **and** *B*: $B \neq \{\}$
shows $|$*Func-option A1 B*$| \leq o\ |$*Func-option A2 B*$|$
by (*metis card-of-Func-mono*[*OF A12 B*] *card-of-Func-option-Func*
ordIso-ordLeq-trans ordLeq-ordIso-trans ordIso-symmetric)

lemma *card-of-Pfunc-Pow-Func-option*:
assumes $B \neq \{\}$
shows $|$*Pfunc A B*$| \leq o\ |$*Pow A* $\times$ *Func-option A B*$|$
proof−
have $|$*Pfunc A B*$| =o\ |\bigcup A' \in$ *Pow A*. *Func-option A' B*$|$ (**is** - $=o$ *?K*)
unfolding *Pfunc-Func-option* **by**(*rule card-of-refl*)
also have *?K* $\leq o\ |$*Sigma* (*Pow A*) ($\lambda\ A'$. *Func-option A' B*)$|$ **using** *card-of-UNION-Sigma*
.
also have $|$*Sigma* (*Pow A*) ($\lambda\ A'$. *Func-option A' B*)$| \leq o\ |$*Pow A* $\times$ *Func-option*
A B$|$
apply(*rule card-of-Sigma-mono1*) **using** *card-of-Func-option-mono*[*OF - assms*]
by *auto*
finally show *?thesis* **.**
qed

lemma *Bpow-ordLeq-Func-Field*:
assumes *rc*: *Card-order r* **and** *r*: *Field r* $\neq \{\}$ **and** *A*: $\neg$*finite A*
shows $|$*Bpow r A*$| \leq o\ |$*Func* (*Field r*) *A*$|$
proof−
let *?F* = $\lambda\ f$. $\{x \mid x\ a.\ f\ a = x \wedge a \in$ *Field r*$\}$
{fix *X* **assume** $X \in$ *Bpow r A* $- \{\{\}\}$
hence *XA*: $X \subseteq A$ **and** $|X| \leq o\ r$
and *X*: $X \neq \{\}$ **unfolding** *Bpow-def* **by** *auto*
hence $|X| \leq o\ |$*Field r*$|$ **by** (*metis Field-card-of card-of-mono2*)
then obtain *F* **where** *1*: $X = F$ ‘ (*Field r*)
using *card-of-ordLeq2*[*OF X*] **by** *metis*
define *f* **where** [*abs-def*]: $f\ i$ = (*if* $i \in$ *Field r then F i else undefined*) **for** *i*

have ∃ *f* ∈ *Func* (*Field r*) *A*. *X* = *?F f*
apply (*intro bexI*[*of - f*]) **using** *1 XA* **unfolding** *Func-def f-def* **by** *auto*
}
hence *Bpow r A* − {{}} ⊆ *?F* ' (*Func* (*Field r*) *A*) **by** *auto*
hence |*Bpow r A* − {{}}| ≤*o* |*Func* (*Field r*) *A*|
by (*rule surj-imp-ordLeq*)
moreover
{have *2*: ¬*finite* (*Bpow r A*) **using** *infinite-Bpow*[*OF rc r A*] **.**
have |*Bpow r A*| =*o* |*Bpow r A* − {{}}|
by (*metis 2 infinite-card-of-diff-singl ordIso-symmetric*)
}
ultimately show *?thesis* **by** (*metis ordIso-ordLeq-trans*)
qed

lemma *empty-in-Func*[*simp*]:
B ≠ {} ⟹ (λ*x*. *undefined*) ∈ *Func* {} *B*
unfolding *Func-def* **by** *auto*

lemma *Func-mono*[*simp*]:
assumes *B1* ⊆ *B2*
shows *Func A B1* ⊆ *Func A B2*
using *assms* **unfolding** *Func-def* **by** *force*

lemma *Pfunc-mono*[*simp*]:
assumes *A1* ⊆ *A2* **and** *B1* ⊆ *B2*
shows *Pfunc A B1* ⊆ *Pfunc A B2*
using *assms* **unfolding** *Pfunc-def*
apply *safe*
apply (*case-tac x a*, *auto*)
apply (*metis in-mono option.simps*(*5*))
done

lemma *card-of-Func-UNIV-UNIV*:
|*Func* (*UNIV*::'*a set*) (*UNIV*::'*b set*)| =*o* |*UNIV*::('*a* ⇒ '*b*) *set*|
using *card-of-Func-UNIV*[*of UNIV*::'*b set*] **by** *auto*

lemma *ordLeq-Func*:
assumes {*b1*,*b2*} ⊆ *B b1* ≠ *b2*
shows |*A*| ≤*o* |*Func A B*|
unfolding *card-of-ordLeq*[*symmetric*] **proof**(*intro exI conjI*)
let *?F* = λ *aa a*. *if a* ∈ *A then* (*if a* = *aa then b1 else b2*) *else undefined*
show *inj-on ?F A* **using** *assms* **unfolding** *inj-on-def fun-eq-iff* **by** *auto*
show *?F* ' *A* ⊆ *Func A B* **using** *assms* **unfolding** *Func-def* **by** *auto*
qed

lemma *infinite-Func*:
assumes *A*: ¬*finite A* **and** *B*: {*b1*,*b2*} ⊆ *B b1* ≠ *b2*
shows ¬*finite* (*Func A B*)
using *ordLeq-Func*[*OF B*] **by** (*metis A card-of-ordLeq-finite*)

10.9 Infinite cardinals are limit ordinals

lemma *card-order-infinite-isLimOrd*:
assumes *c*: *Card-order r* **and** *i*: ¬*finite* (*Field r*)
shows *isLimOrd r*
proof−
  have *0*: *wo-rel r* **and** *00*: *Well-order r*
  using *c* **unfolding** *card-order-on-def wo-rel-def* **by** *auto*
  hence *rr*: *Refl r* **by** (*metis wo-rel.REFL*)
  show *?thesis* **unfolding** *wo-rel.isLimOrd-def*[*OF 0*] *wo-rel.isSuccOrd-def*[*OF 0*]
proof *safe*
    fix *j* **assume** *j*: *j* ∈ *Field r* **and** *jm*: ∀ *i*∈*Field r*. (*i*, *j*) ∈ *r*
    define *p* **where** *p* = *Restr r* (*Field r* − {*j*})
    have *fp*: *Field p* = *Field r* − {*j*}
    unfolding *p-def* **apply**(*rule Refl-Field-Restr2*[*OF rr*]) **by** *auto*
    have *of*: *ofilter r* (*Field p*) **unfolding** *wo-rel.ofilter-def*[*OF 0*] **proof** *safe*
      fix *a x* **assume** *a*: *a* ∈ *Field p* **and** *x* ∈ *under r a*
      hence *x*: (*x*,*a*) ∈ *r x* ∈ *Field r* **unfolding** *under-def Field-def* **by** *auto*
      moreover have *a*: *a* ≠ *j a* ∈ *Field r* (*a*,*j*) ∈ *r* **using** *a jm* **unfolding** *fp* **by** *auto*
      ultimately have *x* ≠ *j* **using** *j jm* **by** (*metis 0 wo-rel.max2-def wo-rel.max2-equals1*)
      thus *x* ∈ *Field p* **using** *x* **unfolding** *fp* **by** *auto*
    qed(*unfold p-def Field-def*, *auto*)
    have *p* <*o r* **using** *j ofilter-ordLess*[*OF 00 of*] **unfolding** *fp p-def*[*symmetric*] **by** *auto*
    hence *2*: |*Field p*| <*o r* **using** *c* **by** (*metis BNF-Cardinal-Order-Relation.ordLess-Field*)
    have |*Field p*| =*o* |*Field r*| **unfolding** *fp* **using** *i* **by** (*metis infinite-card-of-diff-singl*)
    also have |*Field r*| =*o r*
    using *c* **by** (*metis card-of-unique ordIso-symmetric*)
    finally have |*Field p*| =*o r* **.**
    with *2* **show** *False* **by** (*metis not-ordLess-ordIso*)
  qed
qed

lemma *insert-Chain*:
assumes *Refl r C* ∈ *Chains r* **and** *i* ∈ *Field r* **and** ⋀*j*. *j* ∈ *C* ⟹ (*j*,*i*) ∈ *r* ∨ (*i*,*j*) ∈ *r*
shows *insert i C* ∈ *Chains r*
using *assms* **unfolding** *Chains-def* **by** (*auto dest*: *refl-onD*)

lemma *Collect-insert*: {*R j* |*j*. *j* ∈ *insert j1 J*} = *insert* (*R j1*) {*R j* |*j*. *j* ∈ *J*}
by *auto*

lemma *Field-init-seg-of*[*simp*]:
Field init-seg-of = *UNIV*
unfolding *Field-def init-seg-of-def* **by** *auto*

lemma *refl-init-seg-of*[*intro*, *simp*]: *refl init-seg-of*
unfolding *refl-on-def Field-def* **by** *auto*

lemma *regularCard-all-ex*:
assumes *r*: *Card-order r*   *regularCard r*
and *As*: $\bigwedge$ *i j b*. $b \in B \Longrightarrow (i,j) \in r \Longrightarrow P\ i\ b \Longrightarrow P\ j\ b$
and *Bsub*: $\forall\ b \in B.\ \exists\ i \in$ *Field r*. $P\ i\ b$
and *cardB*: $|B| <o\ r$
shows $\exists\ i \in$ *Field r*. $\forall\ b \in B.\ P\ i\ b$
proof−
  let *?As* $= \lambda i.\ \{b \in B.\ P\ i\ b\}$
  have $\exists\, i \in$ *Field r*. $B \le$ *?As i*
  apply(*rule regularCard-UNION*) **using** *assms* **unfolding** *relChain-def* **by** *auto*
  thus *?thesis* **by** *auto*
qed

lemma *relChain-stabilize*:
assumes *rc*: *relChain r As* **and** *AsB*: $(\bigcup i \in$ *Field r*. *As i*$) \subseteq B$ **and** *Br*: $|B| <o\ r$
and *ir*: $\neg$*finite* (*Field r*) **and** *cr*: *Card-order r*
shows $\exists\ i \in$ *Field r*. *As* (*succ r i*) $=$ *As i*
proof(*rule ccontr*, *auto*)
  have *0*: *wo-rel r* **and** *00*: *Well-order r*
  unfolding *wo-rel-def* **by** (*metis card-order-on-well-order-on cr*)+
  have *L*: *isLimOrd r* **using** *ir cr* **by** (*metis card-order-infinite-isLimOrd*)
  have *AsBs*: $(\bigcup i \in$ *Field r*. *As* (*succ r i*)$) \subseteq B$
  using *AsB L* **apply** *safe* **by** (*metis 0 UN-I subsetD wo-rel.isLimOrd-succ*)
  assume *As-s*: $\forall\, i \in$*Field r*. *As* (*succ r i*) $\neq$ *As i*
  have *1*: $\forall\, i\ j.\ (i,j) \in r \wedge i \neq j \longrightarrow$ *As i* $\subset$ *As j*
  proof *safe*
    fix *i j* **assume** *1*: $(i, j) \in r\ i \neq j$ **and** *Asij*: *As i* $=$ *As j*
    hence *rij*: (*succ r i*, $j) \in r$ **by** (*metis 0 wo-rel.succ-smallest*)
    hence *As* (*succ r i*) $\subseteq$ *As j* **using** *rc* **unfolding** *relChain-def* **by** *auto*
    moreover
    {have (*i*,*succ r i*) $\in r$ **apply**(*rule wo-rel.succ-in*[*OF 0*])
     using *1* **unfolding** *aboveS-def* **by** *auto*
     hence *As i* $\subset$ *As* (*succ r i*) **using** *As-s rc rij* **unfolding** *relChain-def Field-def*
by *auto*
    }
    ultimately show *False* **unfolding** *Asij* **by** *auto*
  qed (*insert rc*, *unfold relChain-def*, *auto*)
  hence $\forall\ i \in$ *Field r*. $\exists\ a.\ a \in$ *As* (*succ r i*) $-$ *As i*
  using *wo-rel.succ-in*[*OF 0*] *AsB*
  by(*metis 0 card-order-infinite-isLimOrd cr ir psubset-imp-ex-mem*
      wo-rel.isLimOrd-aboveS wo-rel.succ-diff)
  then obtain *f* **where** *f*: $\forall\ i \in$ *Field r*. *f i* $\in$ *As* (*succ r i*) $-$ *As i* **by** *metis*
  have *inj-on f* (*Field r*) **unfolding** *inj-on-def* **proof** *safe*
    fix *i j* **assume** *ij*: $i \in$ *Field r* $j \in$ *Field r* **and** *fij*: *f i* $=$ *f j*
    show $i = j$
    proof(*cases rule*: *wo-rel.cases-Total3*[*OF 0*], *safe*)
      assume $(i, j) \in r$ **and** *ijd*: $i \neq j$
      hence *rij*: (*succ r i*, $j) \in r$ **by** (*metis 0 wo-rel.succ-smallest*)
      hence *As* (*succ r i*) $\subseteq$ *As j* **using** *rc* **unfolding** *relChain-def* **by** *auto*

thus $i = j$ **using** *ij ijd fij f* **by** *auto*
next
assume $(j, i) \in r$ **and** *ijd*: $i \neq j$
hence *rij*: $(succ\ r\ j,\ i) \in r$ **by** (*metis 0 wo-rel.succ-smallest*)
hence *As* $(succ\ r\ j) \subseteq As\ i$ **using** *rc* **unfolding** *relChain-def* **by** *auto*
thus $j = i$ **using** *ij ijd fij f* **by** *fastforce*
qed(*insert ij*, *auto*)
qed
moreover have f ‘ (*Field r*) $\subseteq B$ **using** *f AsBs* **by** *auto*
moreover have $|B| <o\ |Field\ r|$ **using** *Br cr* **by** (*metis card-of-unique ord-Less-ordIso-trans*)
ultimately show *False* **unfolding** *card-of-ordLess*[*symmetric*] **by** *auto*
qed

10.10 Regular vs. stable cardinals

definition *stable* :: $'a\ rel \Rightarrow bool$
where
stable r $\equiv \forall (A::'a\ set)\ (F :: 'a \Rightarrow 'a\ set).$
$|A| <o\ r \wedge (\forall\, a \in A.\ |F\ a| <o\ r)$
$\longrightarrow |SIGMA\ a : A.\ F\ a| <o\ r$

lemma *regularCard-stable*:
assumes *cr*: *Card-order r* **and** *ir*: $\neg$*finite* (*Field r*) **and** *reg*: *regularCard r*
shows *stable r*
unfolding *stable-def* **proof** *safe*
fix $A :: 'a\ set$ **and** $F :: 'a \Rightarrow 'a\ set$ **assume** A: $|A| <o\ r$ **and** F: $\forall\, a{\in}A.\ |F\ a| <o\ r$
{**assume** $r \leq o\ |Sigma\ A\ F|$
hence $|Field\ r| \leq o\ |Sigma\ A\ F|$ **using** *card-of-Field-ordIso*[*OF cr*]
by (*metis Field-card-of card-of-cong ordLeq-iff-ordLess-or-ordIso ordLeq-ordLess-trans*)
moreover have *Fi*: *Field r* $\neq \{\}$ **using** *ir* **by** *auto*
ultimately obtain f **where** f: f ‘ *Sigma A F* $=$ *Field r* **using** *card-of-ordLeq2* **by** *metis*
have *r*: *wo-rel r* **using** *cr* **unfolding** *card-order-on-def wo-rel-def* **by** *auto*
{**fix** a **assume** a: $a \in A$
define L **where** $L = \{(a,u)\ |\ u.\ u \in F\ a\}$
have *fL*: f ‘ $L \subseteq$ *Field r* **using** *f a* **unfolding** *L-def* **by** *auto*
have $|L| =o\ |F\ a|$ **unfolding** *L-def card-of-ordIso*[*symmetric*]
apply(*rule exI*[*of - snd*]) **unfolding** *bij-betw-def inj-on-def* **by** (*auto simp*: *image-def*)
hence $|L| <o\ r$ **using** *F a ordIso-ordLess-trans*[*of* $|L|$ $|F\ a|$] **unfolding** *L-def* **by** *auto*
hence $|f$ ‘ $L| <o\ r$ **using** *ordLeq-ordLess-trans*[*OF card-of-image*, *of L*] **unfolding** *L-def* **by** *auto*
hence $\neg$ *cofinal* (f ‘ L) *r* **using** *reg fL* **unfolding** *regularCard-def* **by** (*metis not-ordLess-ordIso*)
then obtain k **where** k: $k \in$ *Field r* **and** $\forall\ l \in L.\ \neg\ (f\ l \neq k \wedge (k, f\ l) \in r)$
unfolding *cofinal-def image-def* **by** *auto*

hence $\exists\ k \in Field\ r.\ \forall\ l \in L.\ (f\ l,\ k) \in r$ **using** r **by** (*metis fL image-subset-iff wo-rel.in-notinI*)
hence $\exists\ k \in Field\ r.\ \forall\ u \in F\ a.\ (f\ (a,u),\ k) \in r$ **unfolding** *L-def* **by** *auto*
}
then obtain *gg* **where** *gg*: $\forall\ a \in A.\ \forall\ u \in F\ a.\ (f\ (a,u),\ gg\ a) \in r$ **by** *metis*
obtain *j0* **where** *j0*: $j0 \in Field\ r$ **using** *Fi* **by** *auto*
define *g* **where** [*abs-def*]: $g\ a = (if\ F\ a \neq \{\}\ then\ gg\ a\ else\ j0)$ **for** *a*
have *g*: $\forall\ a \in A.\ \forall\ u \in F\ a.\ (f\ (a,u),g\ a) \in r$ **using** *gg* **unfolding** *g-def* **by** *auto*
hence *1*: $Field\ r \subseteq (\bigcup a \in A.\ under\ r\ (g\ a))$
using *f*[*symmetric*] **unfolding** *under-def image-def* **by** *auto*
have *gA*: $g\ `\ A \subseteq Field\ r$ **using** *gg j0* **unfolding** *Field-def g-def* **by** *auto*
moreover have *cofinal* $(g\ `\ A)\ r$ **unfolding** *cofinal-def* **proof** *safe*
fix *i* **assume** $i \in Field\ r$
then obtain *j* **where** *ij*: $(i,j) \in r\ i \neq j$ **using** *cr ir* **by** (*metis infinite-Card-order-limit*)
hence $j \in Field\ r$ **by** (*metis card-order-on-def cr well-order-on-domain*)
then obtain *a* **where** *a*: $a \in A$ **and** *j*: $(j,\ g\ a) \in r$ **using** *1* **unfolding** *under-def* **by** *auto*
hence $(i,\ g\ a) \in r$ **using** *ij wo-rel.TRANS*[*OF r*] **unfolding** *trans-def* **by** *blast*
moreover have $i \neq g\ a$
using *ij j r* **unfolding** *wo-rel-def* **unfolding** *well-order-on-def linear-order-on-def partial-order-on-def antisym-def* **by** *auto*
ultimately show $\exists j{\in}g\ `\ A.\ i \neq j \wedge (i,\ j) \in r$ **using** *a* **by** *auto*
qed
ultimately have $|g\ `\ A| =o\ r$ **using** *reg* **unfolding** *regularCard-def* **by** *auto*
moreover have $|g\ `\ A| \leq o\ |A|$ **by** (*metis card-of-image*)
ultimately have *False* **using** *A* **by** (*metis not-ordLess-ordIso ordLeq-ordLess-trans*)
}
thus $|Sigma\ A\ F| <o\ r$
using *cr not-ordLess-iff-ordLeq* **by** (*metis card-of-Well-order card-order-on-well-order-on*)
qed

lemma *stable-regularCard*:
assumes *cr*: *Card-order r* **and** *ir*: $\neg finite\ (Field\ r)$ **and** *st*: *stable r*
shows *regularCard r*
unfolding *regularCard-def* **proof** *safe*
fix *K* **assume** *K*: $K \subseteq Field\ r$ **and** *cof*: *cofinal K r*
have $|K| \leq o\ r$ **using** *K* **by** (*metis card-of-Field-ordIso card-of-mono1 cr ordLeq-ordIso-trans*)
moreover
{assume *Kr*: $|K| <o\ r$
then obtain *f* **where** $\forall\ a \in Field\ r.\ f\ a \in K \wedge a \neq f\ a \wedge (a,\ f\ a) \in r$
using *cof* **unfolding** *cofinal-def* **by** *metis*
hence $Field\ r \subseteq (\bigcup a \in K.\ underS\ r\ a)$ **unfolding** *underS-def* **by** *auto*
hence $r \leq o\ |\bigcup a \in K.\ underS\ r\ a|$ **using** *cr*
by (*metis Card-order-iff-ordLeq-card-of card-of-mono1 ordLeq-transitive*)
also have $|\bigcup a \in K.\ underS\ r\ a| \leq o\ |SIGMA\ a{:}\ K.\ underS\ r\ a|$ **by** (*rule card-of-UNION-Sigma*)

also
{have $\forall\ a \in K.\ |underS\ r\ a| <o\ r$ **using** *K* **by** (*metis card-of-underS cr subsetD*)
hence $|SIGMA\ a{:}\ K.\ underS\ r\ a| <o\ r$ **using** *st Kr* **unfolding** *stable-def* **by** *auto*
}
finally have $r <o\ r$ **.**
hence *False* **by** (*metis ordLess-irreflexive*)
}
ultimately show $|K| =o\ r$ **by** (*metis ordLeq-iff-ordLess-or-ordIso*)
qed

lemma *stable-elim*:
assumes *ST*: *stable r* **and** *A-LESS*: $|A| <o\ r$ **and**
F-LESS: $\bigwedge\ a.\ a \in A \Longrightarrow |F\ a| <o\ r$
shows $|SIGMA\ a : A.\ F\ a| <o\ r$
proof−
obtain A' **where** *1*: $A' < Field\ r \wedge |A'| <o\ r$ **and** *2*: $|A| =o\ |A'|$
using *internalize-card-of-ordLess*[*of A r*] *A-LESS* **by** *blast*
then obtain *G* **where** *3*: *bij-betw G A′ A*
using *card-of-ordIso ordIso-symmetric* **by** *blast*

{fix *a* **assume** $a \in A$
hence $\exists B'.\ B' \leq Field\ r \wedge |F\ a| =o\ |B'| \wedge |B'| <o\ r$
using *internalize-card-of-ordLess*[*of F a r*] *F-LESS* **by** *blast*
}
then obtain F' **where**
temp: $\forall a \in A.\ F'\ a \leq Field\ r \wedge |F\ a| =o\ |F'\ a| \wedge |F'\ a| <o\ r$
using *bchoice*[*of A* $\lambda\ a\ B'.\ B' \leq Field\ r \wedge |F\ a| =o\ |B'| \wedge |B'| <o\ r$] **by** *blast*
hence *4*: $\forall a \in A.\ F'\ a \leq Field\ r \wedge |F'\ a| <o\ r$ **by** *auto*
have *5*: $\forall a \in A.\ |F'\ a| =o\ |F\ a|$ **using** *temp ordIso-symmetric* **by** *auto*

have $\forall a' \in A'.\ F'(G\ a') \leq Field\ r \wedge |F'(G\ a')| <o\ r$
using *3 4 bij-betw-ball*[*of G A′ A*] **by** *auto*
hence $|SIGMA\ a' : A'.\ F'(G\ a')| <o\ r$
using *ST 1* **unfolding** *stable-def* **by** *auto*
moreover have $|SIGMA\ a' : A'.\ F'(G\ a')| =o\ |SIGMA\ a : A.\ F\ a|$
using *card-of-Sigma-cong*[*of G A′ A F′ F*] *5 3* **by** *blast*
ultimately show *?thesis* **using** *ordIso-symmetric ordIso-ordLess-trans* **by** *blast*
qed

lemma *stable-natLeq*: *stable natLeq*
proof(*unfold stable-def*, *safe*)
fix *A* :: *′a set* **and** *F* :: $'a \Rightarrow 'a\ set$
assume $|A| <o\ natLeq$ **and** $\forall a{\in}A.\ |F\ a| <o\ natLeq$
hence $finite\ A \wedge (\forall a \in A.\ finite(F\ a))$
by (*auto simp add*: *finite-iff-ordLess-natLeq*)
hence *finite*(*Sigma A F*) **by** (*simp only*: *finite-SigmaI*[*of A F*])
thus $|Sigma\ A\ F\ | <o\ natLeq$

by (*auto simp add*: *finite-iff-ordLess-natLeq*)
qed

corollary *regularCard-natLeq*: *regularCard natLeq*
using *stable-regularCard*[*OF natLeq-Card-order - stable-natLeq*] *Field-natLeq* **by** *simp*

lemma *stable-cardSuc*:
assumes *CARD*: *Card-order r* **and** *INF*: $\neg$*finite* (*Field r*)
shows *stable*(*cardSuc r*)
using *infinite-cardSuc-regularCard regularCard-stable*
by (*metis CARD INF cardSuc-Card-order cardSuc-finite*)

lemma *stable-UNION*:
assumes *ST*: *stable r* **and** *A-LESS*: $|A| <o\ r$ **and**
F-LESS: $\bigwedge a.\ a \in A \Longrightarrow |F\ a| <o\ r$
shows $|\bigcup a \in A.\ F\ a| <o\ r$
proof−
have $|\bigcup a \in A.\ F\ a| \le o\ |SIGMA\ a : A.\ F\ a|$
using *card-of-UNION-Sigma* **by** *blast*
thus *?thesis* **using** *assms stable-elim ordLeq-ordLess-trans* **by** *blast*
qed

lemma *stable-ordIso1*:
assumes *ST*: *stable r* **and** *ISO*: $r' =o\ r$
shows *stable* r'
proof(*unfold stable-def*, *auto*)
fix A::$'b$ *set* **and** F::$'b \Rightarrow 'b$ *set*
assume $|A| <o\ r'$ **and** $\forall a \in A.\ |F\ a| <o\ r'$
hence $(\ |A| <o\ r) \wedge (\forall a \in A.\ |F\ a| <o\ r)$
using *ISO ordLess-ordIso-trans* **by** *blast*
hence $|SIGMA\ a : A.\ F\ a| <o\ r$ **using** *assms stable-elim* **by** *blast*
thus $|SIGMA\ a : A.\ F\ a| <o\ r'$
using *ISO ordIso-symmetric ordLess-ordIso-trans* **by** *blast*
qed

lemma *stable-ordIso2*:
assumes *ST*: *stable r* **and** *ISO*: $r =o\ r'$
shows *stable* r'
using *assms stable-ordIso1 ordIso-symmetric* **by** *blast*

lemma *stable-ordIso*:
assumes $r =o\ r'$
shows *stable* r = *stable* r'
using *assms stable-ordIso1 stable-ordIso2* **by** *blast*

lemma *stable-nat*: *stable* $|UNIV$::*nat set*$|$
using *stable-natLeq card-of-nat stable-ordIso* **by** *auto*

Below, the type of "A" is not important – we just had to choose an appropriate type to make "A" possible. What is important is that arbitrarily large infinite sets of stable cardinality exist.

lemma *infinite-stable-exists*:
assumes *CARD*: $\forall r \in R$. *Card-order* (*r*::'*a rel*)
shows $\exists$(*A* :: (*nat* + '*a set*)*set*).
    $\neg$*finite A* $\wedge$ *stable* |*A*| $\wedge$ ($\forall r \in R$. *r* <*o* |*A*|)
proof−
  have $\exists$(*A* :: (*nat* + '*a set*)*set*).
    $\neg$*finite A* $\wedge$ *stable* |*A*| $\wedge$ |*UNIV*::'*a set*| <*o* |*A*|
  proof(*cases finite* (*UNIV*::'*a set*))
    assume *Case1*: *finite* (*UNIV*::'*a set*)
    let *?B* = *UNIV*::*nat set*
    have $\neg$*finite*(*?B* <+> {}) **using** *finite-Plus-iff* **by** *blast*
    moreover
    have *stable* |*?B*| **using** *stable-natLeq card-of-nat stable-ordIso1* **by** *blast*
    hence *stable* |*?B* <+> {}| **using** *stable-ordIso card-of-Plus-empty1* **by** *blast*
    moreover
    have $\neg$*finite*(*Field* |*?B*|) $\wedge$ *finite*(*Field* |*UNIV*::'*a set*|) **using** *Case1* **by** *simp*
    hence |*UNIV*::'*a set*| <*o* |*?B*| **by** (*simp add*: *finite-ordLess-infinite*)
    hence |*UNIV*::'*a set*| <*o* |*?B* <+> {}| **using** *card-of-Plus-empty1 ordLess-ordIso-trans*
by *blast*
    ultimately show *?thesis* **by** *blast*
  next
    assume *Case1*: $\neg$*finite* (*UNIV*::'*a set*)
    hence $*$: $\neg$*finite*(*Field* |*UNIV*::'*a set*|) **by** *simp*
    let *?B* = *Field*(*cardSuc* |*UNIV*::'*a set*|)
    have *0*: |*?B*| =*o* |{} <+> *?B*| **using** *card-of-Plus-empty2* **by** *blast*
    have *1*: $\neg$*finite ?B* **using** *Case1 card-of-cardSuc-finite* **by** *blast*
    hence *2*: $\neg$*finite*({} <+> *?B*) **using** *0 card-of-ordIso-finite* **by** *blast*
    have |*?B*| =*o cardSuc* |*UNIV*::'*a set*|
    using *card-of-Card-order cardSuc-Card-order card-of-Field-ordIso* **by** *blast*
    moreover have *stable*(*cardSuc* |*UNIV*::'*a set*|)
    using *stable-cardSuc* $*$ *card-of-Card-order* **by** *blast*
    ultimately have *stable* |*?B*| **using** *stable-ordIso* **by** *blast*
    hence *3*: *stable* |{} <+> *?B*| **using** *stable-ordIso 0* **by** *blast*
    have |*UNIV*::'*a set*| <*o cardSuc* |*UNIV*::'*a set*|
    using *card-of-Card-order cardSuc-greater* **by** *blast*
    moreover have |*?B*| =*o* *cardSuc* |*UNIV*::'*a set*|
    using *card-of-Card-order cardSuc-Card-order card-of-Field-ordIso* **by** *blast*
    ultimately have |*UNIV*::'*a set*| <*o* |*?B*|
    using *ordIso-symmetric ordLess-ordIso-trans* **by** *blast*
    hence |*UNIV*::'*a set*| <*o* |{} <+> *?B*| **using** *0 ordLess-ordIso-trans* **by** *blast*
    thus *?thesis* **using** *2 3* **by** *blast*
  qed
  thus *?thesis* **using** *CARD card-of-UNIV2 ordLeq-ordLess-trans* **by** *blast*
qed

corollary *infinite-regularCard-exists*:
assumes *CARD*: $\forall r \in R$. *Card-order* (*r*::'*a rel*)

shows $\exists (A :: (nat + 'a\ set)set)$.
  $\neg$*finite* $A \wedge$ *regularCard* $|A| \wedge (\forall r \in R.\ r <o\ |A|)$
using *infinite-stable-exists*[*OF CARD*] *stable-regularCard* **by** (*metis Field-card-of card-of-card-order-on*)

end

11 Cardinal Arithmetic

theory *Cardinal-Arithmetic*
imports *Cardinal-Order-Relation*
begin

11.1 Binary sum

lemma *csum-Cnotzero2*:
  Cnotzero r2 $\Longrightarrow$ *Cnotzero* (*r1* +*c* *r2*)
unfolding *csum-def*
by (*metis Cnotzero-imp-not-empty Field-card-of Plus-eq-empty-conv card-of-card-order-on czeroE*)

lemma *single-cone*:
  $|\{x\}|$ =*o* *cone*
proof –
  let *?f* = $\lambda x.$ ()
  have *bij-betw ?f* $\{x\}$ $\{()\}$ **unfolding** *bij-betw-def* **by** *auto*
  thus *?thesis* **unfolding** *cone-def* **using** *card-of-ordIso* **by** *blast*
qed

lemma *cone-Cnotzero*: *Cnotzero cone*
by (*simp add*: *cone-not-czero Card-order-cone*)

lemma *cone-ordLeq-ctwo*: *cone* $\leq$*o* *ctwo*
unfolding *cone-def ctwo-def card-of-ordLeq*[*symmetric*] **by** *auto*

lemma *csum-czero1*: *Card-order r* $\Longrightarrow$ *r* +*c* *czero* =*o* *r*
  unfolding *czero-def csum-def Field-card-of*
  by (*rule ordIso-transitive*[*OF ordIso-symmetric*[*OF card-of-Plus-empty1*] *card-of-Field-ordIso*])

lemma *csum-czero2*: *Card-order r* $\Longrightarrow$ *czero* +*c* *r* =*o* *r*
  unfolding *czero-def csum-def Field-card-of*
  by (*rule ordIso-transitive*[*OF ordIso-symmetric*[*OF card-of-Plus-empty2*] *card-of-Field-ordIso*])

11.2 Product

lemma *Times-cprod*: $|A \times B|$ =*o* $|A|$ **c* $|B|$
by (*simp only*: *cprod-def Field-card-of card-of-refl*)

lemma *card-of-Times-singleton*:

fixes $A :: 'a\ set$
shows $|A \times \{x\}| =o\ |A|$
proof −
define $f :: 'a \times 'b \Rightarrow 'a$ **where** $f = (\lambda(a, b).\ a)$
have $A \subseteq f\ `\ (A \times \{x\})$ **unfolding** *f-def* **by** (*auto simp*: *image-iff*)
hence *bij-betw* $f\ (A \times \{x\})\ A$ **unfolding** *bij-betw-def inj-on-def f-def* **by** *fastforce*
thus *?thesis* **using** *card-of-ordIso* **by** *blast*
qed

lemma *cprod-assoc*: $(r *c\ s) *c\ t =o\ r *c\ s *c\ t$
unfolding *cprod-def Field-card-of* **by** (*rule card-of-Times-assoc*)

lemma *cprod-czero*: $r *c\ czero =o\ czero$
unfolding *cprod-def czero-def Field-card-of* **by** (*simp add*: *card-of-empty-ordIso*)

lemma *cprod-cone*: *Card-order* $r \Longrightarrow r *c\ cone =o\ r$
unfolding *cprod-def cone-def Field-card-of*
by (*drule card-of-Field-ordIso*) (*erule ordIso-transitive*[*OF card-of-Times-singleton*])

lemma *ordLeq-cprod1*: ⟦*Card-order p1*; *Cnotzero p2*⟧ $\Longrightarrow$ $p1 \leq o\ p1 *c\ p2$
unfolding *cprod-def* **by** (*metis Card-order-Times1 czeroI*)

11.3 Exponentiation

lemma *cexp-czero*: $r\ \widehat{}c\ czero =o\ cone$
unfolding *cexp-def czero-def Field-card-of Func-empty* **by** (*rule single-cone*)

lemma *Pow-cexp-ctwo*:
$|Pow\ A| =o\ ctwo\ \widehat{}c\ |A|$
unfolding *ctwo-def cexp-def Field-card-of* **by** (*rule card-of-Pow-Func*)

lemma *Cnotzero-cexp*:
assumes *Cnotzero q Card-order r*
shows *Cnotzero* $(q\ \widehat{}c\ r)$
proof (*cases* $r =o\ czero$)
case *False* **thus** *?thesis*
apply −
apply (*rule Cnotzero-mono*)
apply (*rule assms*(*1*))
apply (*rule Card-order-cexp*)
apply (*rule ordLeq-cexp1*)
apply (*rule conjI*)
apply (*rule notI*)
apply (*erule notE*)
apply (*rule ordIso-transitive*)
apply *assumption*
apply (*rule czero-ordIso*)
apply (*rule assms*(*2*))

apply (*rule conjunct2*)
apply (*rule assms*(*1*))
done
next
case *True* **thus** *?thesis*
apply −
apply (*rule Cnotzero-mono*)
apply (*rule cone-Cnotzero*)
apply (*rule Card-order-cexp*)
apply (*rule ordIso-imp-ordLeq*)
apply (*rule ordIso-symmetric*)
apply (*rule ordIso-transitive*)
apply (*rule cexp-cong2*)
apply *assumption*
apply (*rule conjunct2*)
apply (*rule assms*(*1*))
apply (*rule assms*(*2*))
apply (*rule cexp-czero*)
done
qed

lemma *Cinfinite-ctwo-cexp*:
Cinfinite r $\Longrightarrow$ *Cinfinite* (*ctwo* ^*c r*)
unfolding *ctwo-def cexp-def cinfinite-def Field-card-of*
by (*rule conjI*, *rule infinite-Func*, *auto*)

lemma *cone-ordLeq-iff-Field*:
assumes *cone* $\leq$*o r*
shows *Field r* $\neq$ {}
proof (*rule ccontr*)
assume $\neg$ *Field r* $\neq$ {}
hence *Field r* = {} **by** *simp*
thus *False* **using** *card-of-empty3*
card-of-mono2[*OF assms*] *Cnotzero-imp-not-empty*[*OF cone-Cnotzero*] **by** *auto*
qed

lemma *cone-ordLeq-cexp*: *cone* $\leq$*o r1* $\Longrightarrow$ *cone* $\leq$*o r1* ^*c r2*
by (*simp add*: *cexp-def cone-def Func-non-emp cone-ordLeq-iff-Field*)

lemma *Card-order-czero*: *Card-order czero*
by (*simp only*: *card-of-Card-order czero-def*)

lemma *cexp-mono2''*:
assumes *2*: *p2* $\leq$*o r2*
and *n1*: *Cnotzero q*
and *n2*: *Card-order p2*
shows *q* ^*c p2* $\leq$*o q* ^*c r2*
proof (*cases p2* =*o* (*czero* :: *'a rel*))
case *True*

hence q ^c $p2$ $=o$ q ^c ($czero$ :: $'a$ rel) **using** $n1$ $n2$ *cexp-cong2 Card-order-czero* **by** *blast*
also have q ^c ($czero$ :: $'a$ rel) $=o$ $cone$ **using** *cexp-czero* **by** *blast*
also have $cone \leq o$ q ^c $r2$ **using** *cone-ordLeq-cexp cone-ordLeq-Cnotzero n1* **by** *blast*
finally show *?thesis* **.**
next
case *False* **thus** *?thesis* **using** *assms cexp-mono2' czeroI* **by** *metis*
qed

lemma *csum-cexp*: $[\![Cinfinite\ r1;\ Cinfinite\ r2;\ Card\text{-}order\ q;\ ctwo \leq o\ q]\!] \Longrightarrow$
q ^c $r1$ $+c$ q ^c $r2$ $\leq o$ q ^c ($r1$ $+c$ $r2$)
apply (*rule csum-cinfinite-bound*)
apply (*rule cexp-mono2*)
apply (*rule ordLeq-csum1*)
apply (*erule conjunct2*)
apply *assumption*
apply (*rule notE*)
apply (*rule cinfinite-not-czero*[*of r1*])
apply (*erule conjunct1*)
apply *assumption*
apply (*erule conjunct2*)
apply (*rule cexp-mono2*)
apply (*rule ordLeq-csum2*)
apply (*erule conjunct2*)
apply *assumption*
apply (*rule notE*)
apply (*rule cinfinite-not-czero*[*of r2*])
apply (*erule conjunct1*)
apply *assumption*
apply (*erule conjunct2*)
apply (*rule Card-order-cexp*)
apply (*rule Card-order-cexp*)
apply (*rule Cinfinite-cexp*)
apply *assumption*
apply (*rule Cinfinite-csum*)
by (*rule disjI1*)

lemma *csum-cexp'*: $[\![Cinfinite\ r;\ Card\text{-}order\ q;\ ctwo \leq o\ q]\!] \Longrightarrow q +c\ r \leq o\ q$ ^c r
apply (*rule csum-cinfinite-bound*)
apply (*metis Cinfinite-Cnotzero ordLeq-cexp1*)
apply (*metis ordLeq-cexp2*)
apply *blast+*
by (*metis Cinfinite-cexp*)

lemma *card-of-Sigma-ordLeq-Cinfinite*:
$[\![Cinfinite\ r;\ |I| \leq o\ r;\ \forall\, i \in I.\ |A\ i| \leq o\ r]\!] \Longrightarrow |SIGMA\ i : I.\ A\ i| \leq o\ r$
unfolding *cinfinite-def* **by** (*blast intro*: *card-of-Sigma-ordLeq-infinite-Field*)

```
lemma card-order-cexp:
  assumes card-order r1 card-order r2
  shows card-order (r1 ^c r2)
proof -
  have Field r1 = UNIV Field r2 = UNIV using assms card-order-on-Card-order
by auto
  thus ?thesis unfolding cexp-def Func-def by simp
qed

lemma Cinfinite-ordLess-cexp:
  assumes r: Cinfinite r
  shows r <o r ^c r
proof -
  have r <o ctwo ^c r using r by (simp only: ordLess-ctwo-cexp)
  also have ctwo ^c r ≤o r ^c r
    by (rule cexp-mono1[OF ctwo-ordLeq-Cinfinite]) (auto simp: r ctwo-not-czero
Card-order-ctwo)
  finally show ?thesis .
qed

lemma infinite-ordLeq-cexp:
  assumes Cinfinite r
  shows r ≤o r ^c r
by (rule ordLess-imp-ordLeq[OF Cinfinite-ordLess-cexp[OF assms]])

lemma czero-cexp: Cnotzero r ⟹ czero ^c r =o czero
 by (drule Cnotzero-imp-not-empty) (simp add: cexp-def czero-def card-of-empty-ordIso)

lemma Func-singleton:
fixes x :: 'b and A :: 'a set
shows |Func A {x}| =o |{x}|
proof (rule ordIso-symmetric)
  define f where [abs-def]: f y a = (if y = x ∧ a ∈ A then x else undefined) for
y a
  have Func A {x} ⊆ f ` {x} unfolding f-def Func-def by (force simp: fun-eq-iff)
 hence bij-betw f {x} (Func A {x}) unfolding bij-betw-def inj-on-def f-def Func-def
    by (auto split: if-split-asm)
  thus |{x}| =o |Func A {x}| using card-of-ordIso by blast
qed

lemma cone-cexp: cone ^c r =o cone
  unfolding cexp-def cone-def Field-card-of by (rule Func-singleton)

lemma card-of-Func-squared:
  fixes A :: 'a set
  shows |Func (UNIV :: bool set) A| =o |A × A|
proof (rule ordIso-symmetric)
  define f where f = (λ(x::'a,y) b. if A = {} then undefined else if b then x else
y)
```

have *Func* (*UNIV* :: *bool set*) $A \subseteq f$ ‘ $(A \times A)$ **unfolding** *f-def Func-def*
by (*auto simp*: *image-iff fun-eq-iff split*: *option.splits if-split-asm*) *blast*
hence *bij-betw f* $(A \times A)$ (*Func* (*UNIV* :: *bool set*) A)
unfolding *bij-betw-def inj-on-def f-def Func-def* **by** (*auto simp*: *fun-eq-iff*)
thus $|A \times A| =o$ |*Func* (*UNIV* :: *bool set*) A| **using** *card-of-ordIso* **by** *blast*
qed

lemma *cexp-ctwo*: r ^*c* *ctwo* $=o$ r $*c$ r
unfolding *cexp-def ctwo-def cprod-def Field-card-of* **by** (*rule card-of-Func-squared*)

lemma *card-of-Func-Plus*:
fixes A :: $'a$ *set* **and** B :: $'b$ *set* **and** C :: $'c$ *set*
shows |*Func* (A <+> B) C| $=o$ |*Func* A C $\times$ *Func* B C|
proof (*rule ordIso-symmetric*)
define f **where** $f = (\lambda(g :: 'a => 'c, h::'b \Rightarrow 'c)\ ab.$ *case ab of Inl a* $\Rightarrow$ $g\ a$ | *Inr b* $\Rightarrow$ $h\ b$)
define f' **where** $f' = (\lambda(f :: ('a + 'b) \Rightarrow 'c).\ (\lambda a.\ f\ (Inl\ a), \lambda b.\ f\ (Inr\ b)))$
have f ‘ (*Func* A C $\times$ *Func* B C) $\subseteq$ *Func* (A <+> B) C
unfolding *Func-def f-def* **by** (*force split*: *sum.splits*)
moreover have f' ‘ *Func* (A <+> B) $C \subseteq$ *Func* A C $\times$ *Func* B C **unfolding** *Func-def* f'*-def* **by** *force*
moreover have $\forall a \in$ *Func* A C $\times$ *Func* B C. f' $(f\ a) = a$ **unfolding** f'*-def* *f-def Func-def* **by** *auto*
moreover have $\forall a' \in$ *Func* (A <+> B) C. f $(f'\ a') = a'$ **unfolding** f'*-def* *f-def Func-def*
by (*auto split*: *sum.splits*)
ultimately have *bij-betw f* (*Func* A C $\times$ *Func* B C) (*Func* (A <+> B) C)
by (*intro bij-betw-byWitness*[*of -* f' f])
thus |*Func* A C $\times$ *Func* B C| $=o$ |*Func* (A <+> B) C| **using** *card-of-ordIso* **by** *blast*
qed

lemma *cexp-csum*: r ^*c* $(s +c\ t)$ $=o$ r ^*c* s $*c$ r ^*c* t
unfolding *cexp-def cprod-def csum-def Field-card-of* **by** (*rule card-of-Func-Plus*)

11.4 Powerset

definition *cpow* **where** *cpow* r = |*Pow* (*Field* r)|

lemma *card-order-cpow*: *card-order* $r \Longrightarrow$ *card-order* (*cpow* r)
by (*simp only*: *cpow-def Field-card-order Pow-UNIV card-of-card-order-on*)

lemma *cpow-greater-eq*: *Card-order* $r \Longrightarrow r \le o$ *cpow* r
by (*rule ordLess-imp-ordLeq*) (*simp only*: *cpow-def Card-order-Pow*)

lemma *Cinfinite-cpow*: *Cinfinite* $r \Longrightarrow$ *Cinfinite* (*cpow* r)
unfolding *cpow-def cinfinite-def* **by** (*metis Field-card-of card-of-Card-order infinite-Pow*)

lemma *Card-order-cpow*: *Card-order* (*cpow r*)
unfolding *cpow-def* **by** (*rule card-of-Card-order*)

lemma *cardSuc-ordLeq-cpow*: *Card-order r* $\Longrightarrow$ *cardSuc r* $\leq$*o cpow r*
unfolding *cpow-def* **by** (*metis Card-order-Pow cardSuc-ordLess-ordLeq card-of-Card-order*)

lemma *cpow-cexp-ctwo*: *cpow r* =*o ctwo* ^*c r*
unfolding *cpow-def ctwo-def cexp-def Field-card-of* **by** (*rule card-of-Pow-Func*)

11.5 Inverse image

lemma *vimage-ordLeq*:
assumes |*A*| $\leq$*o k* **and** $\forall$ *a* $\in$ *A*. |*vimage f* {*a*}| $\leq$*o k* **and** *Cinfinite k*
shows |*vimage f A*| $\leq$*o k*
proof−
  have *vimage f A* = ($\bigcup$ *a* $\in$ *A*. *vimage f* {*a*}) **by** *auto*
  also have |$\bigcup$ *a* $\in$ *A*. *vimage f* {*a*}| $\leq$*o k*
  using *UNION-Cinfinite-bound*[*OF assms*] **.**
  finally show *?thesis* **.**
qed

11.6 Maximum

definition *cmax* **where**
  cmax r s =
    (*if cinfinite r* $\vee$ *cinfinite s then czero* +*c r* +*c s*
     else natLeq-on (*max* (*card* (*Field r*)) (*card* (*Field s*))) +*c czero*)

lemma *cmax-com*: *cmax r s* =*o cmax s r*
  unfolding *cmax-def*
  by (*auto simp*: *max.commute intro*: *csum-cong2*[*OF csum-com*] *csum-cong2*[*OF czero-ordIso*])

lemma *cmax1*:
  assumes *Card-order r Card-order s s* $\leq$*o r*
  shows *cmax r s* =*o r*
unfolding *cmax-def* **proof** (*split if-splits*, *intro conjI impI*)
  assume *cinfinite r* $\vee$ *cinfinite s*
  hence *Cinf*: *Cinfinite r* **using** *assms*(*1*,*3*) **by** (*metis cinfinite-mono*)
  have *czero* +*c r* +*c s* =*o r* +*c s* **by** (*rule csum-czero2*[*OF Card-order-csum*])
  also have *r* +*c s* =*o r* **by** (*rule csum-absorb1*[*OF Cinf assms*(*3*)])
  finally show *czero* +*c r* +*c s* =*o r* **.**
next
  assume ¬ (*cinfinite r* $\vee$ *cinfinite s*)
  hence *fin*: *finite* (*Field r*) **and** *finite* (*Field s*) **unfolding** *cinfinite-def* **by** *simp-all*
  moreover
  { from *assms*(*2*) **have** |*Field s*| =*o s* **by** (*rule card-of-Field-ordIso*)
    also from *assms*(*3*) **have** *s* $\leq$*o r* **.**
    also from *assms*(*1*) **have** *r* =*o* |*Field r*| **by** (*rule ordIso-symmetric*[*OF card-of-Field-ordIso*])
    finally have |*Field s*| $\leq$*o* |*Field r*| **.**

}
ultimately have *card* (*Field s*) ≤ *card* (*Field r*) **by** (*subst sym*[*OF finite-card-of-iff-card2*])
hence *max* (*card* (*Field r*)) (*card* (*Field s*)) = *card* (*Field r*) **by** (*rule max-absorb1*)
hence *natLeq-on* (*max* (*card* (*Field r*)) (*card* (*Field s*))) +*c czero* =
natLeq-on (*card* (*Field r*)) +*c czero* **by** *simp*
also have . . . =*o natLeq-on* (*card* (*Field r*)) **by** (*rule csum-czero1*[*OF natLeq-on-Card-order*])
also have *natLeq-on* (*card* (*Field r*)) =*o* |*Field r*|
by (*rule ordIso-symmetric*[*OF finite-imp-card-of-natLeq-on*[*OF fin*]])
also from *assms*(*1*) **have** |*Field r*| =*o r* **by** (*rule card-of-Field-ordIso*)
finally show *natLeq-on* (*max* (*card* (*Field r*)) (*card* (*Field s*))) +*c czero* =*o r* **.**
qed

lemma *cmax2*:
assumes *Card-order r Card-order s r* ≤*o s*
shows *cmax r s* =*o s*
by (*metis assms cmax1 cmax-com ordIso-transitive*)

lemma *csum-absorb2*: *Cinfinite r2* ⟹ *r1* ≤*o r2* ⟹ *r1* +*c r2* =*o r2*
by (*metis csum-absorb2'*)

lemma *cprod-infinite2'*: ⟦*Cnotzero r1*; *Cinfinite r2*; *r1* ≤*o r2*⟧ ⟹ *r1* ∗*c r2* =*o r2*
unfolding *ordIso-iff-ordLeq*
by (*intro conjI cprod-cinfinite-bound ordLeq-cprod2 ordLeq-refl*)
(*auto dest*!: *ordIso-imp-ordLeq not-ordLeq-ordLess simp*: *czero-def Card-order-empty*)

context
fixes *r s*
assumes *r*: *Cinfinite r*
and *s*: *Cinfinite s*
begin

lemma *cmax-csum*: *cmax r s* =*o r* +*c s*
proof (*cases r* ≤*o s*)
case *True*
hence *cmax r s* =*o s* **by** (*metis cmax2 r s*)
also have *s* =*o r* +*c s* **by** (*metis True csum-absorb2 ordIso-symmetric s*)
finally show *?thesis* **.**
next
case *False*
hence *s* ≤*o r* **by** (*metis ordLeq-total r s card-order-on-def*)
hence *cmax r s* =*o r* **by** (*metis cmax1 r s*)
also have *r* =*o r* +*c s* **by** (*metis* ‹*s* ≤*o r*› *csum-absorb1 ordIso-symmetric r*)
finally show *?thesis* **.**
qed

lemma *cmax-cprod*: *cmax r s* =*o r* ∗*c s*
proof (*cases r* ≤*o s*)
case *True*

hence *cmax r s =o s* **by** (*metis cmax2 r s*)
also have *s =o r ∗c s* **by** (*metis Cinfinite-Cnotzero True cprod-infinite2′ ordIso-symmetric r s*)
finally show *?thesis* **.**
next
case *False*
hence *s ≤o r* **by** (*metis ordLeq-total r s card-order-on-def*)
hence *cmax r s =o r* **by** (*metis cmax1 r s*)
also have *r =o r ∗c s* **by** (*metis Cinfinite-Cnotzero ‹s ≤o r› cprod-infinite1′ ordIso-symmetric r s*)
finally show *?thesis* **.**
qed

end

lemma *Card-order-cmax*:
assumes *r*: *Card-order r* **and** *s*: *Card-order s*
shows *Card-order* (*cmax r s*)
unfolding *cmax-def* **by** (*auto simp*: *Card-order-csum*)

lemma *ordLeq-cmax*:
assumes *r*: *Card-order r* **and** *s*: *Card-order s*
shows *r ≤o cmax r s ∧ s ≤o cmax r s*
proof−
{**assume** *r ≤o s*
hence *?thesis* **by** (*metis cmax2 ordIso-iff-ordLeq ordLeq-transitive r s*)
}
moreover
{**assume** *s ≤o r*
hence *?thesis* **using** *cmax-com* **by** (*metis cmax2 ordIso-iff-ordLeq ordLeq-transitive r s*)
}
ultimately show *?thesis* **using** *r s ordLeq-total* **unfolding** *card-order-on-def* **by** *auto*
qed

lemmas *ordLeq-cmax1* = *ordLeq-cmax*[*THEN conjunct1*] **and**
ordLeq-cmax2 = *ordLeq-cmax*[*THEN conjunct2*]

lemma *finite-cmax*:
assumes *r*: *Card-order r* **and** *s*: *Card-order s*
shows *finite* (*Field* (*cmax r s*)) ⟷ *finite* (*Field r*) ∧ *finite* (*Field s*)
proof−
{**assume** *r ≤o s*
hence *?thesis* **by** (*metis cmax2 ordIso-finite-Field ordLeq-finite-Field r s*)
}
moreover
{**assume** *s ≤o r*
hence *?thesis* **by** (*metis cmax1 ordIso-finite-Field ordLeq-finite-Field r s*)

}
ultimately show *?thesis* **using** *r s ordLeq-total* **unfolding** *card-order-on-def*
by *auto*
qed

end

12 Extending Well-founded Relations to Wellorders

theory *Wellorder-Extension*
imports *Main Order-Union*
begin

12.1 Extending Well-founded Relations to Wellorders

A *downset* (also lower set, decreasing set, initial segment, or downward closed set) is closed w.r.t. smaller elements.

definition *downset-on* **where**
$downset\text{-}on\ A\ r = (\forall\, x\ y.\ (x,\ y) \in r \wedge y \in A \longrightarrow x \in A)$

lemma *downset-onI*:
$(\bigwedge x\ y.\ (x,\ y) \in r \Longrightarrow y \in A \Longrightarrow x \in A) \Longrightarrow downset\text{-}on\ A\ r$
by (*auto simp*: *downset-on-def*)

lemma *downset-onD*:
$downset\text{-}on\ A\ r \Longrightarrow (x,\ y) \in r \Longrightarrow y \in A \Longrightarrow x \in A$
unfolding *downset-on-def* **by** *blast*

Extensions of relations w.r.t. a given set.

definition *extension-on* **where**
$extension\text{-}on\ A\ r\ s = (\forall\, x{\in}A.\ \forall\, y{\in}A.\ (x,\ y) \in s \longrightarrow (x,\ y) \in r)$

lemma *extension-onI*:
$(\bigwedge x\ y.\ [\![x \in A;\ y \in A;\ (x,\ y) \in s]\!] \Longrightarrow (x,\ y) \in r) \Longrightarrow extension\text{-}on\ A\ r\ s$
by (*auto simp*: *extension-on-def*)

lemma *extension-onD*:
$extension\text{-}on\ A\ r\ s \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow (x,\ y) \in s \Longrightarrow (x,\ y) \in r$
by (*auto simp*: *extension-on-def*)

lemma *downset-on-Union*:
assumes $\bigwedge r.\ r \in R \Longrightarrow downset\text{-}on\ (Field\ r)\ p$
shows $downset\text{-}on\ (Field\ (\bigcup R))\ p$
using *assms* **by** (*auto intro*: *downset-onI dest*: *downset-onD*)

lemma *chain-subset-extension-on-Union*:

assumes $chain_{\subseteq}$ R **and** $\bigwedge r.\ r \in R \Longrightarrow$ *extension-on* (*Field* r) r p
shows *extension-on* (*Field* ($\bigcup R$)) ($\bigcup R$) p
using *assms*
by (*simp add*: *chain-subset-def extension-on-def*)
(*metis* (*no-types*) *mono-Field subsetD*)

lemma *downset-on-empty* [*simp*]: *downset-on* {} *p*
by (*auto simp*: *downset-on-def*)

lemma *extension-on-empty* [*simp*]: *extension-on* {} *p q*
by (*auto simp*: *extension-on-def*)

Every well-founded relation can be extended to a wellorder.

theorem *well-order-extension*:
assumes *wf p*
shows $\exists\, w.\ p \subseteq w \wedge$ *Well-order* w
proof −
let $?K = \{r.$ *Well-order* $r \wedge$ *downset-on* (*Field* r) $p \wedge$ *extension-on* (*Field* r) r
$p\}$
define I **where** $I =$ *init-seg-of* $\cap\ ?K \times ?K$
have *I-init*: $I \subseteq$ *init-seg-of* **by** (*simp add*: *I-def*)
then have *subch*: $\bigwedge R.\ R \in$ *Chains* $I \Longrightarrow chain_{\subseteq}\ R$
by (*auto simp*: *init-seg-of-def chain-subset-def Chains-def*)
have *Chains-wo*: $\bigwedge R\ r.\ R \in$ *Chains* $I \Longrightarrow r \in R \Longrightarrow$
Well-order $r \wedge$ *downset-on* (*Field* r) $p \wedge$ *extension-on* (*Field* r) r p
by (*simp add*: *Chains-def I-def*) *blast*
have *FI*: *Field* $I = ?K$ **by** (*auto simp*: *I-def init-seg-of-def Field-def*)
then have *0*: *Partial-order* I
by (*auto simp*: *partial-order-on-def preorder-on-def antisym-def antisym-init-seg-of*
refl-on-def
trans-def I-def elim: *trans-init-seg-of*)
have $\bigcup R \in ?K \wedge (\forall\, r{\in}R.\ (r,\bigcup R) \in I)$ **if** $R \in$ *Chains* I **for** R
proof −
from *that* **have** *Ris*: $R \in$ *Chains init-seg-of* **using** *mono-Chains* [*OF I-init*]
by *blast*
have *subch*: $chain_{\subseteq}\ R$ **using** ‹$R \in$ *Chains* I› *I-init*
by (*auto simp*: *init-seg-of-def chain-subset-def Chains-def*)
have $\forall\, r{\in}R.$ *Refl* r **and** $\forall\, r{\in}R.$ *trans* r **and** $\forall\, r{\in}R.$ *antisym* r **and**
$\forall\, r{\in}R.$ *Total* r **and** $\forall\, r{\in}R.$ *wf* $(r - Id)$ **and**
$\bigwedge r.\ r \in R \Longrightarrow$ *downset-on* (*Field* r) p **and**
$\bigwedge r.\ r \in R \Longrightarrow$ *extension-on* (*Field* r) r p
using *Chains-wo* [*OF* ‹$R \in$ *Chains* I›] **by** (*simp-all add*: *order-on-defs*)
have *Refl* ($\bigcup R$) **using** ‹$\forall\, r{\in}R.$ *Refl* r› **unfolding** *refl-on-def* **by** *fastforce*
moreover have *trans* ($\bigcup R$)
by (*rule chain-subset-trans-Union* [*OF subch* ‹$\forall\, r{\in}R.$ *trans* r›])
moreover have *antisym* ($\bigcup R$)
by (*rule chain-subset-antisym-Union* [*OF subch* ‹$\forall\, r{\in}R.$ *antisym* r›])
moreover have *Total* ($\bigcup R$)
by (*rule chain-subset-Total-Union* [*OF subch* ‹$\forall\, r{\in}R.$ *Total* r›])

moreover have *wf* ((⋃ *R*) − *Id*)
proof −
have (⋃ *R*) − *Id* = ⋃{*r* − *Id* | *r*. *r* ∈ *R*} **by** *blast*
with ‹∀ *r*∈*R*. *wf* (*r* − *Id*)› *wf-Union-wf-init-segs* [*OF Chains-inits-DiffI* [*OF Ris*]]
show *?thesis* **by** *fastforce*
qed
ultimately have *Well-order* (⋃ *R*) **by** (*simp add*: *order-on-defs*)
moreover have ∀ *r*∈*R*. *r initial-segment-of* ⋃ *R* **using** *Ris*
by (*simp add*: *Chains-init-seg-of-Union*)
moreover have *downset-on* (*Field* (⋃ *R*)) *p*
by (*rule downset-on-Union* [*OF* ‹⋀*r*. *r* ∈ *R* ⟹ *downset-on* (*Field r*) *p*›])
moreover have *extension-on* (*Field* (⋃ *R*)) (⋃ *R*) *p*
by (*rule chain-subset-extension-on-Union* [*OF subch* ‹⋀*r*. *r* ∈ *R* ⟹ *extension-on* (*Field r*) *r p*›])
ultimately show *?thesis*
using *mono-Chains* [*OF I-init*] **and** ‹*R* ∈ *Chains I*›
by (*simp* (*no-asm*) *add*: *I-def del*: *Field-Union*) (*metis Chains-wo*)
qed
then have *1*: ∃ *u*∈*Field I*. ∀ *r*∈*R*. (*r*, *u*) ∈ *I* **if** *R*∈*Chains I* **for** *R*
using *that* **by** (*subst FI*) *blast*

Zorn's Lemma yields a maximal wellorder m.

from *Zorns-po-lemma* [*OF 0 1*] **obtain** *m* :: ('*a* × '*a*) *set*
where *Well-order m* **and** *downset-on* (*Field m*) *p* **and** *extension-on* (*Field m*) *m p* **and**
max: ∀ *r*. *Well-order r* ∧ *downset-on* (*Field r*) *p* ∧ *extension-on* (*Field r*) *r p* ∧
(*m*, *r*) ∈ *I* ⟶ *r* = *m*
by (*auto simp*: *FI*)
have *Field p* ⊆ *Field m*
proof (*rule ccontr*)
let *?Q* = *Field p* − *Field m*
assume ¬ (*Field p* ⊆ *Field m*)
with *assms* [*unfolded wf-eq-minimal*, *THEN spec*, *of ?Q*]
obtain *x* **where** *x* ∈ *Field p* **and** *x* ∉ *Field m* **and**
min: ∀ *y*. (*y*, *x*) ∈ *p* ⟶ *y* ∉ *?Q* **by** *blast*

Add *x* as topmost element to *m*.

let *?s* = {(*y*, *x*) | *y*. *y* ∈ *Field m*}
let *?m* = *insert* (*x*, *x*) *m* ∪ *?s*
have *Fm*: *Field ?m* = *insert x* (*Field m*) **by** (*auto simp*: *Field-def*)
have *Refl m* **and** *trans m* **and** *antisym m* **and** *Total m* **and** *wf* (*m* − *Id*)
using ‹*Well-order m*› **by** (*simp-all add*: *order-on-defs*)

We show that the extension is a wellorder.

have *Refl ?m* **using** ‹*Refl m*› *Fm* **by** (*auto simp*: *refl-on-def*)
moreover have *trans ?m* **using** ‹*trans m*› ‹*x* ∉ *Field m*›
unfolding *trans-def Field-def Domain-unfold Domain-converse* [*symmetric*] **by** *blast*

```
    moreover have antisym ?m using ‹antisym m› ‹x ∉ Field m›
     unfolding antisym-def Field-def Domain-unfold Domain-converse [symmetric]
by blast
   moreover have Total ?m using ‹Total m› Fm by (auto simp: Relation.total-on-def)
    moreover have wf (?m − Id)
    proof −
      have wf ?s using ‹x ∉ Field m›
          by (simp add: wf-eq-minimal Field-def Domain-unfold Domain-converse
[symmetric]) metis
      thus ?thesis using ‹wf (m − Id)› ‹x ∉ Field m›
        wf-subset [OF ‹wf ?s› Diff-subset]
        by (fastforce intro!: wf-Un simp add: Un-Diff Field-def)
    qed
    ultimately have Well-order ?m by (simp add: order-on-defs)
    moreover have extension-on (Field ?m) ?m p
      using ‹extension-on (Field m) m p› ‹downset-on (Field m) p›
      by (subst Fm) (auto simp: extension-on-def dest: downset-onD)
    moreover have downset-on (Field ?m) p
      apply (subst Fm)
      using ‹downset-on (Field m) p› and min
      unfolding downset-on-def Field-def by blast
    moreover have (m, ?m) ∈ I
      using ‹Well-order m› and ‹Well-order ?m› and
      ‹downset-on (Field m) p› and ‹downset-on (Field ?m) p› and
      ‹extension-on (Field m) m p› and ‹extension-on (Field ?m) ?m p› and
      ‹Refl m› and ‹x ∉ Field m›
      by (auto simp: I-def init-seg-of-def refl-on-def)
    ultimately
    — This contradicts maximality of m:
    show False using max and ‹x ∉ Field m› unfolding Field-def by blast
  qed
  have p ⊆ m
    using ‹Field p ⊆ Field m› and ‹extension-on (Field m) m p›
    unfolding Field-def extension-on-def by auto fast
  with ‹Well-order m› show ?thesis by blast
qed
```

Every well-founded relation can be extended to a total wellorder.

```
corollary total-well-order-extension:
  assumes wf p
  shows ∃ w. p ⊆ w ∧ Well-order w ∧ Field w = UNIV
proof −
  from well-order-extension [OF assms] obtain w
    where p ⊆ w and wo: Well-order w by blast
  let ?A = UNIV − Field w
  from well-order-on [of ?A] obtain w′ where wo′: well-order-on ?A w′ ..
  have [simp]: Field w′ = ?A using well-order-on-Well-order [OF wo′] by simp
  have ∗: Field w ∩ Field w′ = {} by simp
  let ?w = w ∪o w′
```

have *p* ⊆ *?w* **using** ‹*p* ⊆ *w*› **by** (*auto simp*: *Osum-def*)
moreover have *Well-order ?w* **using** *Osum-Well-order* [*OF* ∗ *wo*] **and** *wo′* **by** *simp*
moreover have *Field ?w* = *UNIV* **by** (*simp add*: *Field-Osum*)
ultimately show *?thesis* **by** *blast*
qed

corollary *well-order-on-extension*:
assumes *wf p* **and** *Field p* ⊆ *A*
shows ∃ *w*. *p* ⊆ *w* ∧ *well-order-on A w*
proof −
from *total-well-order-extension* [*OF* ‹*wf p*›] **obtain** *r*
where *p* ⊆ *r* **and** *wo*: *Well-order r* **and** *univ*: *Field r* = *UNIV* **by** *blast*
let *?r* = {(*x*, *y*). *x* ∈ *A* ∧ *y* ∈ *A* ∧ (*x*, *y*) ∈ *r*}
from ‹*p* ⊆ *r*› **have** *p* ⊆ *?r* **using** ‹*Field p* ⊆ *A*› **by** (*auto simp*: *Field-def*)
have *Refl r trans r antisym r Total r wf* (*r* − *Id*)
using ‹*Well-order r*› **by** (*simp-all add*: *order-on-defs*)
have *refl-on A ?r* **using** ‹*Refl r*› **by** (*auto simp*: *refl-on-def univ*)
moreover have *trans ?r* **using** ‹*trans r*›
unfolding *trans-def* **by** *blast*
moreover have *antisym ?r* **using** ‹*antisym r*›
unfolding *antisym-def* **by** *blast*
moreover have *total-on A ?r* **using** ‹*Total r*› **by** (*simp add*: *total-on-def univ*)
moreover have *wf* (*?r* − *Id*) **by** (*rule wf-subset* [*OF* ‹*wf*(*r* − *Id*)›]) *blast*
ultimately have *well-order-on A ?r* **by** (*simp add*: *order-on-defs*)
with ‹*p* ⊆ *?r*› **show** *?thesis* **by** *blast*
qed

end

13 Theory of Ordinals and Cardinals

theory *Cardinals*
imports *Ordinal-Arithmetic Cardinal-Arithmetic Wellorder-Extension*
begin

end

14 Sets Strictly Bounded by an Infinite Cardinal

theory *Bounded-Set*
imports *Cardinals*
begin

typedef (*′a*, *′k*) *bset* (- *set*[-] [*22*, *21*] *21*) =
{*A* :: *′a set*. |*A*| <*o natLeq* +*c* |*UNIV* :: *′k set*|}
morphisms *set-bset Abs-bset*
by (*rule exI*[*of* - {}]) (*auto simp*: *card-of-empty4 csum-def*)

setup-lifting *type-definition-bset*

lift-definition *map-bset* ::
 $('a \Rightarrow 'b) \Rightarrow 'a\ set['k] \Rightarrow 'b\ set['k]$ **is** *image*
 using *card-of-image ordLeq-ordLess-trans* **by** *blast*

lift-definition *rel-bset* ::
 $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a\ set['k] \Rightarrow 'b\ set['k] \Rightarrow bool$ **is** *rel-set*
 .

lift-definition *bempty* :: $'a\ set['k]$ **is** $\{\}$
 by (*auto simp*: *card-of-empty4 csum-def*)

lift-definition *binsert* :: $'a \Rightarrow 'a\ set['k] \Rightarrow 'a\ set['k]$ **is** *insert*
 using *infinite-card-of-insert ordIso-ordLess-trans Field-card-of Field-natLeq UNIV-Plus-UNIV csum-def finite-Plus-UNIV-iff finite-insert finite-ordLess-infinite2 infinite-UNIV-nat*
by *metis*

definition *bsingleton* **where**
 bsingleton x = *binsert x bempty*

lemma *set-bset-to-set-bset*: $|A|$ *<o natLeq +c* $|UNIV :: 'k\ set| \Longrightarrow$
 set-bset (*the-inv set-bset A* :: $'a\ set['k]$) = *A*
 apply (*rule f-the-inv-into-f*[*unfolded inj-on-def*])
 apply (*simp add*: *set-bset-inject range-eqI Abs-bset-inverse*[*symmetric*])
 apply (*rule range-eqI Abs-bset-inverse*[*symmetric*] *CollectI*)+
 .

lemma *rel-bset-aux-infinite*:
 fixes *a* :: $'a\ set['k]$ **and** *b* :: $'b\ set['k]$
 shows ($\forall t \in$ *set-bset a*. $\exists u \in$ *set-bset b*. *R t u*) $\wedge$ ($\forall u \in$ *set-bset b*. $\exists t \in$ *set-bset a*. *R t u*) $\longleftrightarrow$
 ((*BNF-Def.Grp* {*a*. *set-bset a* $\subseteq$ {(*a*, *b*). *R a b*}} (*map-bset fst*))$^{-1-1}$ *OO*
 BNF-Def.Grp {*a*. *set-bset a* $\subseteq$ {(*a*, *b*). *R a b*}} (*map-bset snd*)) *a b* (**is** *?L* $\longleftrightarrow$ *?R*)
proof
 assume *?L*
 define R' :: $('a \times 'b)\ set['k]$
 where R' = *the-inv set-bset* (*Collect* (*case-prod R*) $\cap$ (*set-bset a* $\times$ *set-bset b*))
 (**is** - = *the-inv set-bset ?L'*)
 have $|?L'|$ *<o natLeq +c* $|UNIV :: 'k\ set|$
 unfolding *csum-def Field-natLeq*
 by (*intro ordLeq-ordLess-trans*[*OF card-of-mono1*[*OF Int-lower2*]]
 card-of-Times-ordLess-infinite)
 (*simp*, (*transfer*, *simp add*: *csum-def Field-natLeq*)+)
 hence $*$: *set-bset* R' = $?L'$ **unfolding** R'*-def* **by** (*intro set-bset-to-set-bset*)
 show *?R* **unfolding** *Grp-def relcompp.simps conversep.simps*
 proof (*intro CollectI case-prodI exI*[*of - a*] *exI*[*of - b*] *exI*[*of - R'*] *conjI refl*)

from $*$ **show** $a = $ *map-bset fst* R' **using** *conjunct1*[*OF* ‹*?L*›]
by (*transfer*, *auto simp add*: *image-def Int-def split*: *prod.splits*)
from $*$ **show** $b = $ *map-bset snd* R' **using** *conjunct2*[*OF* ‹*?L*›]
by (*transfer*, *auto simp add*: *image-def Int-def split*: *prod.splits*)
qed (*auto simp add*: $*$)
next
assume *?R* **thus** *?L* **unfolding** *Grp-def relcompp.simps conversep.simps*
by *transfer force*
qed

bnf *'a set*['*k*]
map: *map-bset*
sets: *set-bset*
bd: *natLeq* +*c* |*UNIV* :: '*k set*|
wits: *bempty*
rel: *rel-bset*
proof −
show *map-bset id* = *id* **by** (*rule ext*, *transfer*) *simp*
next
fix *f g*
show *map-bset* (*f o g*) = *map-bset f o map-bset g* **by** (*rule ext*, *transfer*) *auto*
next
fix *X f g*
assume $\bigwedge z.\ z \in$ *set-bset* $X \Longrightarrow f\ z = g\ z$
then show *map-bset f X* = *map-bset g X* **by** *transfer force*
next
fix *f*
show *set-bset* ∘ *map-bset f* = (‘) *f* ∘ *set-bset* **by** (*rule ext*, *transfer*) *auto*
next
fix *X* :: '*a set*['*k*]
show |*set-bset X*| ≤*o natLeq* +*c* |*UNIV* :: '*k set*|
by *transfer* (*blast dest*: *ordLess-imp-ordLeq*)
next
fix *R S*
show *rel-bset R OO rel-bset S* ≤ *rel-bset* (*R OO S*)
by (*rule predicate2I*, *transfer*) (*auto simp*: *rel-set-OO*[*symmetric*])
next
fix *R* :: '*a* ⇒ '*b* ⇒ *bool*
show *rel-bset R* = ((λ*x y*. ∃ *z*. *set-bset z* ⊆ {(*x*, *y*). *R x y*} ∧
map-bset fst z = *x* ∧ *map-bset snd z* = *y*) :: '*a set*['*k*] ⇒ '*b set*['*k*] ⇒ *bool*)
by (*simp add*: *rel-bset-def map-fun-def o-def rel-set-def*
rel-bset-aux-infinite[*unfolded OO-Grp-alt*])
next
fix *x*
assume *x* ∈ *set-bset bempty*
then show *False* **by** *transfer simp*
qed (*simp-all add*: *card-order-csum natLeq-card-order cinfinite-csum natLeq-cinfinite*)

lemma *map-bset-bempty*[*simp*]: *map-bset f bempty* = *bempty*
  by *transfer auto*

lemma *map-bset-binsert*[*simp*]: *map-bset f* (*binsert x X*) = *binsert* (*f x*) (*map-bset f X*)
  by *transfer auto*

lemma *map-bset-bsingleton*: *map-bset f* (*bsingleton x*) = *bsingleton* (*f x*)
  unfolding *bsingleton-def* **by** *simp*

lemma *bempty-not-binsert*: *bempty* $\neq$ *binsert x X binsert x X* $\neq$ *bempty*
  by (*transfer*, *auto*)+

lemma *bempty-not-bsingleton*[*simp*]: *bempty* $\neq$ *bsingleton x bsingleton x* $\neq$ *bempty*
  unfolding *bsingleton-def* **by** (*simp-all add*: *bempty-not-binsert*)

lemma *bsingleton-inj*[*simp*]: *bsingleton x* = *bsingleton y* $\longleftrightarrow$ *x* = *y*
  unfolding *bsingleton-def* **by** *transfer auto*

lemma *rel-bsingleton*[*simp*]:
  rel-bset R (*bsingleton x1*) (*bsingleton x2*) = *R x1 x2*
  unfolding *bsingleton-def*
  by *transfer* (*auto simp*: *rel-set-def*)

lemma *rel-bset-bsingleton*[*simp*]:
  rel-bset R (*bsingleton x1*) = (λ*X*. *X* $\neq$ *bempty* $\wedge$ ($\forall$*x2*$\in$*set-bset X*. *R x1 x2*))
  rel-bset R X (*bsingleton x2*) = (*X* $\neq$ *bempty* $\wedge$ ($\forall$*x1*$\in$*set-bset X*. *R x1 x2*))
  unfolding *bsingleton-def fun-eq-iff*
  by (*transfer*, *force simp add*: *rel-set-def*)+

lemma *rel-bset-bempty*[*simp*]:
  rel-bset R bempty X = (*X* = *bempty*)
  rel-bset R Y bempty = (*Y* = *bempty*)
  by (*transfer*, *simp add*: *rel-set-def*)+

definition *bset-of-option* **where**
  bset-of-option = *case-option bempty bsingleton*

lift-definition *bgraph* :: ($'a \Rightarrow\ 'b$ *option*) $\Rightarrow$ ($'a \times\ 'b$) *set*[$'a$ *set*] **is**
  λf. {(*a*, *b*). *f a* = *Some b*}
proof −
  fix *f* :: $'a \Rightarrow\ 'b$ *option*
  have |{(*a*, *b*). *f a* = *Some b*}| $\leq o$ |*UNIV* :: $'a$ *set*|
    by (*rule surj-imp-ordLeq*[*of* - λx. (*x*, *the* (*f x*))]) *auto*
  also have |*UNIV* :: $'a$ *set*| $<o$ |*UNIV* :: $'a$ *set set*|
    by *simp*
  also have |*UNIV* :: $'a$ *set set*| $\leq o$ *natLeq* $+c$ |*UNIV* :: $'a$ *set set*|
    by (*rule ordLeq-csum2*) *simp*
  finally show |{(*a*, *b*). *f a* = *Some b*}| $<o$ *natLeq* $+c$ |*UNIV* :: $'a$ *set set*| **.**

qed

lemma *rel-bset-False*[*simp*]: *rel-bset* ($\lambda x\ y$. *False*) *x y* = (*x* = *bempty* $\wedge$ *y* = *bempty*)
  by *transfer* (*auto simp*: *rel-set-def*)

lemma *rel-bset-of-option*[*simp*]:
  rel-bset R (*bset-of-option x1*) (*bset-of-option x2*) = *rel-option R x1 x2*
  unfolding *bset-of-option-def bsingleton-def*[*abs-def*]
  by *transfer* (*auto simp*: *rel-set-def split*: *option.splits*)

lemma *rel-bgraph*[*simp*]:
  rel-bset (*rel-prod* (=) *R*) (*bgraph f1*) (*bgraph f2*) = *rel-fun* (=) (*rel-option R*) *f1 f2*
  apply *transfer*
  apply (*auto simp*: *rel-fun-def rel-option-iff rel-set-def split*: *option.splits*)
  using *option.collapse* **apply** *fastforce*+
  done

lemma *set-bset-bsingleton*[*simp*]:
  set-bset (*bsingleton x*) = {*x*}
  unfolding *bsingleton-def* **by** *transfer auto*

lemma *binsert-absorb*[*simp*]: *binsert a* (*binsert a x*) = *binsert a x*
  by *transfer simp*

lemma *map-bset-eq-bempty-iff*[*simp*]: *map-bset f X* = *bempty* $\longleftrightarrow$ *X* = *bempty*
  by *transfer auto*

lemma *map-bset-eq-bsingleton-iff*[*simp*]:
  map-bset f X = *bsingleton x* $\longleftrightarrow$ (*set-bset X* $\neq$ {} $\wedge$ ($\forall$ *y* $\in$ *set-bset X*. *f y* = *x*))
  unfolding *bsingleton-def* **by** *transfer auto*

lift-definition *bCollect* :: (*'a* $\Rightarrow$ *bool*) $\Rightarrow$ *'a set*[*'a set*] **is** *Collect*
  apply (*rule ordLeq-ordLess-trans*[*OF card-of-mono1*[*OF subset-UNIV*]])
  apply (*rule ordLess-ordLeq-trans*[*OF card-of-set-type*])
  apply (*rule ordLeq-csum2*[*OF card-of-Card-order*])
  done

lift-definition *bmember* :: *'a* $\Rightarrow$ *'a set*[*'k*] $\Rightarrow$ *bool* **is** ($\in$) **.**

lemma *bmember-bCollect*[*simp*]: *bmember a* (*bCollect P*) = *P a*
  by *transfer simp*

lemma *bset-eq-iff*: *A* = *B* $\longleftrightarrow$ ($\forall$ *a*. *bmember a A* = *bmember a B*)
  by *transfer auto*

locale *bset-lifting*
begin

declare *bset.rel-eq*[*relator-eq*]
declare *bset.rel-mono*[*relator-mono*]
declare *bset.rel-compp*[*symmetric*, *relator-distr*]
declare *bset.rel-transfer*[*transfer-rule*]

lemma *bset-quot-map*[*quot-map*]: *Quotient R Abs Rep T* $\Longrightarrow$
Quotient (*rel-bset R*) (*map-bset Abs*) (*map-bset Rep*) (*rel-bset T*)
unfolding *Quotient-alt-def5 bset.rel-Grp*[*of UNIV*, *simplified*, *symmetric*]
bset.rel-conversep[*symmetric*] *bset.rel-compp*[*symmetric*]
by (*auto elim*: *bset.rel-mono-strong*)

lemma *set-relator-eq-onp* [*relator-eq-onp*]:
rel-bset (*eq-onp P*) = *eq-onp* (λA. *Ball* (*set-bset A*) *P*)
unfolding *fun-eq-iff eq-onp-def* **by** *transfer* (*auto simp*: *rel-set-def*)

end

end